UNIVERSIDAD AUSTRAL

INGENIERÍA

Navegación Autónoma de Drones Urbanos con Visión Monocular y Small Language Model (SLM)

Tesis para Magister en Ciencia de Datos

Maestría en Ciencia de Datos - 2024/2025

Directores:

- DEL ROSSO, Rodrigo
- NUSKE, Ezequiel

Alumno:

- NICOLAU, Jorge Enrique

Resumen

La navegación autónoma de cuadricópteros eVTOL en entornos urbanos densos —como el de Buenos Aires— enfrenta severos desafíos derivados de la alta densidad de obstáculos, la degradación de señales satelitales y las restricciones presupuestarias que descartan sensores de alto costo como LiDAR. Para abordar esta problemática bajo estrictas restricciones de hardware y en conformidad con las normativas locales (ANAC Res. 880/2019), esta tesis presenta un sistema integral de planificación previa, percepción monocular y decisión táctica basado exclusivamente en **visión monocular** y **modelos de lenguaje pequeños (SLM)** locales.

La arquitectura desacopla un módulo de planificación deliberativa en tierra (estación de control WebDCS, que compila directivas en lenguaje natural a un manifiesto inmutable de navegación) de un lazo táctico a bordo de alta frecuencia. El módulo de percepción implementa un pipeline ligero de **flujo óptico denso derotado** mediante telemetría de actitud, mapeo de ocupación espacial (`ObstacleField`) y estimación de **Tiempo-a-Colisión (TTC)** a partir de la divergencia del campo de flujo, alcanzando un área bajo la curva ROC de 0.96–0.97 en su validación frente a canales de profundidad de referencia. En la capa de decisión a bordo, un grafo de estados separa dos lazos: un lazo rápido de guiado y evasión por flujo óptico, único responsable de los comandos de velocidad, y un lazo lento asíncrono que juzga qué sectores del fotograma son volables y traduce la respuesta a sub-metas en coordenadas del mundo ancladas a la pose del fotograma observado, de modo que la latencia no las invalida. El lazo lento se diseñó para un modelo de visión-lenguaje pequeño (LFM2.5-VL-1.6B, ejecutado en la misma GPU que la simulación con decodificación estructurada `json_schema`); medido contra etiquetas de referencia, el modelo no distinguió sectores libres de bloqueados (AUC 0.53), mientras que una red de profundidad monocular de ~25 M de parámetros (Depth Anything V2) sí lo hizo (AUC 0.83–0.87) en ~110 ms. La configuración final usa esa red en el lazo lento y en la resolución de atascos, y reserva el modelo de visión-lenguaje para la planificación de ruta sobre el mapa cenital antes del vuelo, la única tarea en la que aportó una señal medible (AUC 0.81).

El desarrollo y la experimentación se realizan en el simulador **Cosys-AirSim** sobre **Unreal Engine 5.5**, en tres entornos de dificultad creciente (`MiniSim`, `TownSim` con vegetación densa y `CitySim`, basado en *Sample City*), y calibrando la dinámica de vuelo contra conjuntos de datos de telemetría de drones comerciales reales. El trabajo documenta una evaluación comparativa sistemática del brazo SLM frente a Máquinas de Estados Finitos (FSM) y control puramente reactivo, analizando métricas de tasa de éxito de misión, tiempo de reacción y longitud de trayectoria ponderada por éxito (SPL). Asimismo, se identifican y analizan los modos de falla estructurales propios de sistemas robóticos asistidos por modelos de lenguaje, demostrando que los errores críticos en la toma de decisiones se originan predominantemente en degradaciones silenciosas del contrato de datos de la interfaz de percepción antes que en el razonamiento del modelo.

Palabras clave

Navegación Autónoma de Drones | Visión Monocular | Flujo Óptico | Tiempo-a-Colisión (TTC) | Small Language Models (SLM) | Máquinas de Estados Finitos (FSM) | Control Táctico Híbrido | Simulación en Unreal Engine / Cosys-AirSim | Robótica Urbana de Bajo Costo

Índice

Carátula y Resumen

1. Introducción

2. Estado del Arte

3. Entorno de Simulación

4. Planificación de Misión (WebDCS)

5. Arquitectura del Lazo Táctico

6. Percepción Monocular

7. Estimación TTC

8. Decisiones SLM

9. Modos de Falla LLM

10. Metodología Experimental

11. Resultados

12. Conclusiones

13. Referencias

Anexos

1. Introducción

1.1 Motivación y contexto

Los drones autónomos han cruzado en los últimos años la frontera que separa la plataforma experimental de la infraestructura real. En el frente civil, el mercado global de UAVs comerciales alcanzó los USD 28,6 mil millones en 2025 con una proyección de superar los USD 52,1 mil millones en 2029, impulsado por entregas de última milla, movilidad aérea urbana, inspección de infraestructura crítica y agricultura de precisión (Dataintelo, 2025). En el frente militar, la guerra en Ucrania consolidó al dron autónomo como el activo táctico más transformador del conflicto moderno: en 2024, Ucrania fabricó cerca de dos millones de drones y reentrenó modelos de inteligencia artificial con datos de combate real, multiplicando por tres o cuatro la tasa de impacto de sus sistemas aéreos (Breaking Defense, 2025). En ambos escenarios, el denominador técnico común es la misma exigencia: un vehículo de bajo costo debe percibir su entorno, tomar decisiones en milisegundos y actuar sin intervención humana directa en el lazo, con hardware de borde limitado y a menudo en ausencia de conectividad.

Argentina enfrenta esta convergencia tecnológica con cuatro urgencias concretas y simultáneas. La primera es la vigilancia del litoral marítimo: la Zona Económica Exclusiva comprende aproximadamente 1,6 millones de km² —incluyendo áreas críticas para la pesca comercial, la seguridad marítima y la soberanía territorial en el Atlántico Sur y Malvinas—, un espacio imposible de cubrir de forma continua con medios tripulados a costo razonable. En mayo de 2026, Argentina formalizó un acuerdo con los Estados Unidos para incorporar drones militares VTOL Shield AI V-Bat y aeronaves de patrulla bajo la *Maritime Domain Situational Awareness Initiative* (Infobae, 2026), y una propuesta legislativa de julio de 2026 impulsa la creación de un Sistema Nacional de Vigilancia Marítima con UAVs (El Estratégico, 2026).

La segunda es la respuesta temprana a incendios forestales: los incendios en la Patagonia y el litoral fluvial son uno de los riesgos ambientales y económicos más severos del país; el Ministerio de Ambiente ya incorporó 17 UAVs autónomos en parques nacionales de Neuquén y Chubut, y la Provincia de Misiones avanza en despacho autónomo de drones inmediatamente tras la detección de una columna de humo.

La tercera es la agricultura de precisión, el dominio de adopción más masiva y de crecimiento más acelerado del agro argentino: la superficie tratada con drones pasó de 50.000 hectáreas en 2024 a un millón de hectáreas en julio de 2025, con una proyección de superar los dos millones al cierre de la campaña gruesa 2026 (Dataportuaria, 2026), impulsada por una flota que creció de 90 equipos importados en 2023 a 600 en agosto de 2024 y una proyección de 2.000 unidades operativas para 2025 (Infocampo, 2025). El uso dominante es la pulverización dirigida de fitosanitarios y la fertilización —con rendimientos de hasta 35 ha/hora—, pero la Red de Drones del INTA, con más de 65 investigadores y 22 pilotos licenciados distribuidos en el país, ya despliega la tecnología en fenotipado de cultivos extensivos, detección temprana de enfermedades en frutales y vid, y monitoreo ganadero (INTA, 2025). Es, además, un dominio donde la autonomía de decisión ya se explora activamente: INTA y la Universidad Nacional de Catamarca ensayan drones con reconocimiento por IA para detectar y corregir en tiempo real fallas de plantación de caña de azúcar en el NOA (Agrolatam, 2025). El crecimiento no está acompañado por un marco regulatorio específico: el SENASA todavía no estableció un régimen propio para la aplicación de fitosanitarios con drones, pese a la actualización de las normas de aviación no tripulada (Infocampo, 2025) —el mismo tipo de vacío regulatorio que atraviesa los otros tres dominios de este trabajo—.

La cuarta urgencia es la que articula directamente el entorno experimental de este trabajo: la seguridad y la logística aérea urbana en Buenos Aires. La Ciudad Autónoma de Buenos Aires abrió en 2025 el marco para la entrega comercial con drones (iProfesional, 2025), y la ANAC publicó la Resolución 319/2025 —que aprueba la Parte 100 del Reglamento Aeronáutico Civil Argentino— y la Resolución 550/2025, que desregulan y simplifican la operación de aeronaves no tripuladas en el espacio aéreo nacional. Pero la operación de UAVs en ese mismo tejido urbano ya no es una promesa regulatoria sino una práctica establecida: el Escuadrón de Drones de la Policía de la Ciudad —integrado a un sistema de videovigilancia con 850 puntos de captura, 3.400 cámaras y 82 % de cobertura territorial a julio de 2025, sostenido por una inversión cercana a los USD 48 millones (Perfil, 2026)— vuela unidades DJI Matrice 200 con seguimiento automático de personas y vehículos y sensores capaces de detectar cables de un milímetro de diámetro, en misiones de apoyo a persecuciones, localización de focos de incendio y búsqueda de personas extraviadas, incluso sobre la Reserva Ecológica y el Riachuelo (Buenos Aires Ciudad, 2023). Buenos Aires concentra en su tejido urbano los desafíos técnicos más exigentes del dominio: densidad edilicia extrema en el Microcentro y Palermo con cañones de 8–12 pisos, degradación severa del GPS dentro de esos cañones, cables de alta y baja tensión a baja altura, interferencia electromagnética y proximidad al espacio controlado de Aeroparque. En los cuatro dominios, el problema técnico de fondo es idéntico: decidir, en tiempo real y con percepción limitada, qué hacer a continuación.

La navegación autónoma en entornos urbanos densos agrega capas de dificultad que no aparecen en espacios abiertos: alta densidad de obstáculos fijos y móviles, degradación de la señal GPS por la altura de los edificios (*urban canyon*), variaciones extremas de iluminación y, en el contexto latinoamericano en particular, restricciones de presupuesto que descartan sensores costosos como el LiDAR y la obligación de operar bajo marcos regulatorios específicos (Resolución ANAC 880/2019; Decreto 663/2024). Un dron sin percepción confiable y sin capacidad de decisión robusta representa un riesgo tangible: puede colisionar, invadir zonas sensibles o interferir con otras operaciones, con consecuencias humanas y económicas concretas (Samy et al., 2019; Aldao et al., 2022).

Este trabajo se enmarca en esa tensión: cómo dotar a un cuadricóptero eVTOL de navegación autónoma reactiva usando exclusivamente visión monocular y hardware de bajo costo —sin redes neuronales de detección, sin LiDAR, sin GPU dedicada de alto costo—, manteniendo rigor metodológico y un diseño de seguridad explícito.

1.2 Objetivos

Objetivo general. Desarrollar un sistema de visión por computadora basado en percepción monocular con capacidad de navegación autónoma de cuadricópteros eVTOL, aplicable a logística urbana, mantenimiento de infraestructura, agricultura de precisión, vigilancia de áreas extensas y atención de emergencias en escenarios representativos del territorio argentino.

Objetivos específicos:

1. Implementar un pipeline reproducible sobre el simulador Cosys-AirSim (basado en Unreal Engine 5.5) que permita validar el desempeño del sistema de navegación en condiciones controladas, repetibles y con telemetría calibrada frente a vuelos reales.
2. Integrar detección de obstáculos y navegación reactiva a partir de percepción monocular ligera — flujo óptico denso derotado, estimación de TTC, mapa de ocupación espacial— optimizada para hardware de bajo costo, de modo de sostener un caso de negocio viable para economías con recursos limitados.
3. Evaluar el rendimiento de un modelo de lenguaje pequeño (SLM) como capa de decisión táctica frente a una máquina de estados finitos (FSM) —el estándar de facto en pilotos automáticos— y frente a un controlador puramente reactivo, usando tasa de éxito de misión, tiempo de reacción y longitud de trayectoria ponderada por éxito (SPL) como métricas de comparación en entornos urbanos de complejidad creciente.
4. Documentar con evidencia medida los modos de falla estructurales de la integración SLM–lazo de control, identificando patrones transferibles a los dominios de aplicación de alta pertinencia local (vigilancia marítima, respuesta a incendios forestales, agricultura de precisión) donde los recursos computacionales de borde y la ausencia de conectividad imponen las mismas restricciones estudiadas en el entorno experimental.

1.3 Alcance y diseño de la arquitectura

El lazo rápido del sistema implementa una percepción monocular ligera **sin redes neuronales de detección**, basada en flujo óptico denso derotado mediante telemetría de actitud y estimación física del Tiempo-a-Colisión (TTC). Esta elección de diseño responde a fundamentos técnicos y computacionales concretos:

1. **Eficiencia y presupuesto de cómputo:** En plataformas robóticas de bajo costo que comparten CPU con la inferencia local de un modelo de lenguaje, la estimación geométrica por flujo óptico reduce drásticamente la latencia por ciclo frente a detectores de objetos basados en aprendizaje profundo.

2. **Robustez fuera de distribución (OOD):** A diferencia de modelos supervisados dependientes de categorías predefinidas de obstáculos, el cálculo de divergencia del campo de flujo modela directamente el fenómeno físico de aproximación: reacciona ante cualquier geometría sin importar su textura o clase semántica, una propiedad crítica en entornos no estructurados como el litoral marítimo o un área de incendio.
3. **Adecuación a la geometría del vuelo urbano:** Para un cuadricóptero con cámara frontal en cañón urbano, el campo visual está dominado por estructuras verticales y fachadas. El flujo óptico traslacional permite estimar la ocupación espacial (`ObstacleField`) en cuadrantes discretos de forma directa y determinista sin ningún entrenamiento previo.

La capa de decisión es un grafo LangGraph con dos lazos desacoplados: un lazo rápido de guiado y evasión por flujo óptico, que es el único que comanda velocidades, y un lazo lento que, de forma asíncrona, propone sub-metas en coordenadas del mundo ancladas a la pose del fotograma que observó. El lazo lento decide qué sectores son volables con una red de profundidad monocular estimada desde el mismo RGB (Depth Anything V2, ~25 M de parámetros). El modelo de visión-lenguaje pequeño (LFM2.5-VL-1.6B, cuantizado Q4_K_M, ~1.6 GB con el codificador visual, en la misma GPU que la simulación, con decodificación estructurada mediante `json_schema`) trabaja en la planificación de ruta previa al vuelo, sobre el mapa cenital: evaluado contra etiquetas de referencia (cap. 11, §11.0), no distingue sectores volables de bloqueados en la imagen de a bordo, y por eso esa pregunta la responde la profundidad estimada. El capítulo 6 desarrolla los fundamentos del módulo de percepción —flujo óptico y profundidad estimada— y el capítulo 8 la ingeniería del modelo de visión-lenguaje.

1.4 El hilo conductor de la tesis

En arquitecturas robóticas híbridas donde un modelo de lenguaje consume descripciones simbólicas o estructuradas del entorno para tomar decisiones de navegación, la integridad del contrato de interfaz entre la percepción y el modelo resulta crítica. La experiencia experimental demuestra que **el fallo más severo en estos sistemas no radica en la capacidad de razonamiento del modelo de lenguaje, sino en la fidelidad y consistencia del contrato de datos que lo alimenta**.

Cuando la capa de percepción o el estado del sistema entrega representaciones desincronizadas, degradadas o vacías, los modelos de lenguaje tienden a generar respuestas sintácticamente válidas y aparentemente razonables a partir de premisas falsas. Dado que el modelo no se interrumpe con excepciones visibles y el vehículo continúa su trayectoria, estas fallas resultan estructuralmente invisibles si solo se observa el comportamiento cinemático superficial. Zhu et al. (2024) plantean formalmente la pregunta de hasta qué punto los LLMs comprenden el contexto que reciben; este trabajo la responde empíricamente desde el ángulo opuesto: ¿qué ocurre cuando el contexto que el modelo recibe está sistemáticamente corrompido?

Este principio —la necesidad de validación formal, auditoría de contratos de datos y salvaguardas deterministas en lazos de control asistidos por SLM— constituye el hilo conductor de este trabajo. Su relevancia se extiende más allá del entorno de simulación: en aplicaciones de patrulla marítima, respuesta a incendios forestales o pulverización dirigida en agricultura de precisión —donde un fallo silencioso del contrato de datos no distingue un obstáculo real de un falso positivo, con consecuencias que en el agro se traducen directamente en aplicación errónea de fitosanitarios— y donde el UAV opera sin cobertura de red y sin supervisión humana directa, la robustez del contrato de datos entre percepción y decisión no es una propiedad académica sino un requisito operacional. El capítulo 9 documenta de manera sistemática los seis modos de falla estructurales identificados en la integración de estos componentes y las salvaguardas implementadas para mitigarlos; el capítulo 12 los retoma como conclusión central y proyecta sus implicancias hacia los dominios de aplicación descriptos.

1.5 Estructura del informe

El capítulo 2 sitúa este trabajo respecto de la literatura relevante en cuatro líneas: navegación autónoma de UAVs desde la planificación clásica hasta el control adaptativo (§2.1); percepción monocular para detección y evitación de obstáculos (§2.2); arquitecturas de control asistidas por modelos de lenguaje —incluyendo los trabajos más recientes de 2025-2026 sobre modelos VLA y SLM en UAVs— (§2.3); y cuatro dominios de aplicación de alta pertinencia local para Argentina —vigilancia del litoral marítimo, incendios forestales, agricultura de precisión y seguridad/logística urbana en Buenos Aires— donde el patrón técnico de este trabajo se transfiere directamente (§2.4), seguidos del posicionamiento explícito de esta tesis en el espacio de soluciones (§2.5).

El capítulo 3 describe la construcción del entorno de simulación —Unreal Engine 5.5 con Cosys-AirSim y la jerarquía de tres entornos de dificultad creciente (`MiniSim`, `TownSim`, `CitySim`)— y su validación estadística frente a telemetría de vuelos reales. El capítulo 4 documenta la planificación deliberativa en tierra y la estación de control WebDCS. Los capítulos 5 a 9 documentan la arquitectura del lazo táctico a bordo: el grafo de control LangGraph con un lazo rápido reactivo y un lazo lento desacoplados (cap. 5), la percepción monocular —flujo óptico en el lazo rápido y profundidad estimada en el lento— (cap. 6), la validación experimental del TTC (cap. 7), la capa de decisión del SLM con decodificación restringida (cap. 8) y los modos de falla específicos de lazos híbridos con modelos de lenguaje (cap. 9). El capítulo 10 describe la metodología experimental —diseño factorial, brazos de comparación y métricas—. Los capítulos 11 y 12 sintetizan los resultados comparativos y las conclusiones, complementados por las referencias bibliográficas estructuradas en el capítulo 13.

2. Estado del arte y trabajos relacionados

Los drones autónomos han dejado de ser una curiosidad tecnológica para convertirse simultáneamente en infraestructura civil crítica y en uno de los factores más determinantes de la guerra moderna, dos procesos que —aunque distintos en sus objetivos— comparten la misma presión técnica de fondo: la necesidad de que el vehículo tome decisiones correctas sin intervención humana en el lazo, a baja altitud, con recursos de cómputo limitados y en entornos no controlados. En el frente civil, el mercado global de UAVs comerciales alcanzó los USD 28,6 mil millones en 2025 y se proyecta que supere los USD 52,1 mil millones en 2029 (Dataintelo, 2025), impulsado por entregas de última milla, movilidad aérea urbana (UAM), inspección automatizada de infraestructura —tendidos eléctricos, oleoductos, torres de telecomunicaciones— y agricultura de precisión. Agencias reguladoras como la FAA, la EASA y la CAAC han ampliado progresivamente las autorizaciones de operaciones BVLOS (*Beyond Visual Line of Sight*) durante 2025-2026, abriendo la puerta a redes de drones autónomos operadas comercialmente a escala (Coptrz, 2026; MDPI Drones, 2026). En el frente militar, la guerra en Ucrania (2022–presente) se consolidó como el primer conflicto a gran escala en que los UAVs —drones FPV de primera persona, municiones merodeadoras y enjambres coordinados— desempeñan un papel táctico central. Ucrania fabricó cerca de dos millones de drones en 2024; sus fuerzas reentrenaron modelos de IA con datos clasificados de combate real, lo que multiplicó por tres o cuatro la tasa de impacto de sus sistemas aéreos autónomos (Breaking Defense, 2025). En respuesta, Rusia estableció a fines de 2024 su propia rama de sistemas no tripulados. El conflicto aceleró además la investigación sobre guerra de contramedidas electrónicas y navegación sin GPS, haciendo de la autonomía robusta de borde un problema de primer orden tanto técnico como estratégico (CSIS, 2024; Modern War Institute, s.f.). Esta doble presión —demanda civil de escala y urgencia militar operacional— imprime relevancia directa a la pregunta que articula este trabajo: cómo dotar a un UAV de capacidad de decisión contextual con recursos computacionales de borde reducidos.

2.1 Navegación autónoma de UAVs: de la planificación clásica al control adaptativo

Los primeros enfoques de navegación autónoma en entornos urbanos combinan planificación global sobre datos geográficos con evasión reactiva de obstáculos móviles. Castelli et al. (2016) proponen un sistema híbrido para vuelo seguro a baja altitud que combina el algoritmo A *con datos GIS y replanificación dinámica, priorizando rutas sobre techos en lugar de calles para aumentar la distancia de seguridad frente a objetos en movimiento. Hughes y Engelbrecht (2023) extienden esta idea a enjambres, con planificación de largo plazo y evasión cooperativa basada en* bounding volume hierarchies* y diagramas de Voronoi sobre un modelo 3D tipo Manhattan —el mismo tipo de trazado urbano en cuadrícula que emplea el entorno de evaluación densa de este trabajo (`CitySim` / Tier 2;ver capítulos 3 y 10)—.

Frente a estos enfoques deliberativos, las máquinas de estados finitos (FSM) siguen siendo el estándar de facto en pilotos automáticos por su predictibilidad y bajo costo computacional: Hu et al. (2024) las emplean para coordinación cooperativa de enjambres en emergencias, y Hoang et al. (2024) las utilizan como base de un sistema de recarga autónoma sobre líneas de alta tensión. Su limitación es estructural: una FSM no interpreta contexto más allá de los umbrales sobre los que fue diseñada. Ese contraste —predictibilidad y bajo costo de la FSM frente a la promesa de adaptabilidad contextual de un modelo de lenguaje— es precisamente el eje de la comparación experimental de este trabajo (cap.10), y es también donde AlMahamid y Grolinger (2022) y Bouhamed et al. (2020) sitúan al aprendizaje por refuerzo como tercera vía: políticas aprendidas en lugar de reglas explícitas o inferencia de un modelo de lenguaje.

2.2 Percepción monocular para detección y evitación de obstáculos

La adopción, en el lazo rápido, de una percepción monocular ligera sin redes neuronales profundas (cap. 6) se fundamenta en una línea de investigación consolidada sobre detección de obstáculos por visión artificial clásica. Badrloo y Varshosaz (2017) revisan el estado del arte de la detección monocular en robótica móvil, mientras que Molineros et al. (2012) y Chen et al. (2016) analizan la estimación de riesgo mediante flujo residual y análisis 3D.

Por otra parte, técnicas clásicas basadas en Mapeo de Perspectiva Inversa (IPM), como las propuestas por Kaneko et al. (2017) y retomadas por Zhou et al. (2026), asumen la existencia de un plano de suelo dominante en el campo de visión, una hipótesis adecuada para vehículos terrestres o cámaras cenitales pero inaplicable a cuadricópteros con cámara frontal a baja altitud en cañones urbanos, donde el campo visual está dominado por estructuras verticales y fachadas.

En contraste, el uso de flujo óptico denso traslacional constituye una solución físicamente rigurosa y libre de entrenamiento supervisado. Vera-Yanez et al. (2024) desarrollan un detector de obstáculos aéreos basado íntegramente en flujo óptico —morfología, foco de expansión (*focus of expansion*, FOE) y agrupamiento—, demostrando que desacoplar la rotación propia del movimiento traslacional permite aislar los vectores de aproximación generados por objetos en la trayectoria. Este principio físico —la divergencia del campo traslacional como estimador directo del Tiempo-a-Colisión (TTC)— constituye la base matemática del módulo de percepción y estimación de TTC implementado en este trabajo (cap. 6 y 7). Enfoques alternativos basados en aprendizaje profundo monocular (Rill y Faragó, 2021; Shi et al., 2024) logran estimaciones densas de profundidad pero introducen carga computacional y sensibilidad a dominios no vistos, lo que justifica un método geométrico determinista en el lazo de 5 Hz. Los modelos fundacionales de profundidad monocular redujeron ese costo: Depth Anything V2 (Yang et al., 2024; https://arxiv.org/abs/2406.09414) ofrece variantes métricas de ~25 M de parámetros, entrenadas con imágenes sintéticas y pseudo-etiquetas sobre imágenes reales. Este trabajo usa una de ellas fuera del lazo rápido, en el lazo lento, donde ~0.1 s por consulta es admisible (cap. 6, §6.14).

2.3 Arquitecturas de control asistidas por modelos de lenguaje

La idea de utilizar modelos de lenguaje como capa de razonamiento de alto nivel en un robot fue articulada con precisión por Vemprala et al. (2023) en un trabajo seminal de Microsoft Research que emplea ChatGPT para programar misiones de drones e inspección sobre una instancia de AirSim —el mismo simulador que sustenta este trabajo—. El resultado clave de Vemprala et al. no es el rendimiento en ninguna tarea específica sino la demostración de que un LLM puede generar código ejecutable de control aéreo a partir de instrucciones en lenguaje natural, separando el razonamiento semántico de la ejecución motora: una arquitectura que se volvería referencia en los años siguientes. Chahine et al. (2023) añaden una perspectiva complementaria sobre robustez fuera de distribución con redes neuronales líquidas como capa de control; Zhu et al. (2024) estudian específicamente hasta qué punto los modelos de lenguaje entienden el contexto que se les provee, una pregunta directamente relevante para el capítulo 9 de este trabajo, donde el problema no es la capacidad de razonamiento del modelo sino la integridad del contexto que efectivamente recibe.

La maduración del campo entre 2024 y 2026 fue rápida. Tian et al. (2025) publican la revisión sistemática más amplia disponible sobre la intersección de LLMs y UAVs —aceptada en *Information Fusion*—, organizando el espacio de soluciones en un marco Percepción–Cognición–Acción (P–C–A) y concluyendo que la disponibilidad de despliegue depende no solo de la capacidad del modelo sino de interfaces verificables que traduzcan salidas semánticas en representaciones accionables por los módulos de vuelo descendentes. Esta conclusión coincide con la motivación de diseño del capítulo 8 de este trabajo, que trata la decodificación restringida vía `json_schema` precisamente como esa interfaz verificable. Por su parte, la emergencia de los modelos de *Vision-Language-Action* (VLA) representa el paso siguiente: Sun et al. (2026) presentan AutoFly, un modelo VLA de extremo a extremo para navegación autónoma de UAVs en exteriores sin instrucciones detalladas —basado en un pseudo-codificador de profundidad a partir de RGB y entrenamiento en dos etapas—, logrando tasas de éxito de navegación 3,9 puntos porcentuales por encima del estado del arte anterior. Saxena et al. (2025) abordan un problema relacionado desde la perspectiva de la navegación por lenguaje: su sistema UAV-VLN integra un VLM para interpretar instrucciones en lenguaje natural y guiar la navegación de forma *end-to-end* sin entrenamiento supervisado por tarea. Estos desarrollos sitúan en perspectiva el trabajo aquí presentado: mientras la frontera investigativa de 2026 opera con VLA completos y modelos de varios miles de millones de parámetros, la pregunta que este trabajo responde es qué tan lejos puede llegar un SLM cuantizado de 1–4B parámetros ejecutado localmente —sin nube, sin entrenamiento especializado— como capa de decisión en un lazo de control real, y cuáles son sus modos de falla documentables.

Dos líneas de trabajo son especialmente pertinentes para la ingeniería de decisiones del SLM descripta en el capítulo 8. Por un lado, la generación de salida estructurada y restringida: Geng et al. (2025) sistematizan el estado del arte en generación de salidas estructuradas desde modelos de lenguaje —el problema exacto que resuelve, en este trabajo, la decodificación restringida vía `json_schema` con parser tolerante como red de contención (cap. 8)—, y Raspanti et al. (2025) muestran que la decodificación con gramática restringida mejora específicamente el desempeño en tareas de análisis lógico estructurado. Por otro lado, la simulación de alta fidelidad como banco de pruebas: Shah et al. (2017) introducen AirSim, el simulador sobre el que se desarrolla y valida este trabajo, y Jansen et al. (2023) presentan una variante extendida (Cosys-AirSim) para aplicaciones industriales complejas, evidencia de que AirSim sigue siendo, varios años después, una base activa para este tipo de investigación.

2.4 Dominios de aplicación de alta pertinencia local: vigilancia del litoral marítimo, incendios forestales, agricultura de precisión y seguridad/logística urbana en Buenos Aires

La demanda de autonomía robusta en UAVs que articula este trabajo no es solo un problema de investigación genérico: cuatro dominios aplicados de particular relevancia para Argentina muestran que los desafíos técnicos abordados —percepción monocular en tiempo real, evasión reactiva de obstáculos y decisión contextual de borde— son exactamente los requeridos en implementaciones operacionales inminentes.

Vigilancia del litoral marítimo. La Zona Económica Exclusiva argentina comprende aproximadamente 1,6 millones de km², una extensión imposible de cubrir de forma continua con medios tripulados a costo razonable. Argentina formalizó en mayo de 2026 un acuerdo con los Estados Unidos para incorporar drones V-Bat de Shield AI —sistemas VTOL que no requieren modificaciones en las embarcaciones desde las que operan— y aeronaves de patrulla marítima King Air B360 ER, bajo la *Maritime Domain Situational Awareness Initiative* (Infobae, 2026). En paralelo, una propuesta legislativa presentada en julio de 2026 plantea la creación de un Sistema Nacional de Vigilancia Marítima con drones, complementario a los despliegues navales tripulados (El Estratégico, 2026). La literatura académica específica sobre vigilancia marítima autónoma con UAVs en el litoral argentino es aún incipiente; sin embargo, el trabajo de Barišić Kulas et al. (2025) —presentado en ICUAS 2025— constituye una referencia técnica directa: los autores desarrollan un sistema completamente autónomo para detección e identificación de embarcaciones sobre grandes áreas marítimas con UAV en entornos de denegación de GNSS, empleando visión *onboard* con YOLOv8 y fusión de histogramas de matiz para identificar la embarcación objetivo bajo restricciones computacionales estrictas de borde. El problema técnico es isomorfo al de la patrulla costera: un vehículo autónomo debe procesar su entorno visual en tiempo real y tomar decisiones de seguimiento sin apoyo externo, en las mismas condiciones que este trabajo estudia en simulación.

Respuesta temprana a incendios forestales tras detección satelital. Los incendios en la Patagonia y el litoral fluvial representan uno de los riesgos ambientales y económicos más importantes del país. El Ministerio de Ambiente incorporó 17 UAVs para detección temprana de focos en parques nacionales de Neuquén y Chubut —incluyendo Lanín, Laguna Blanca, Los Alerce y Lago Puelo— con vuelo autónomo y transmisión en tiempo real a estaciones de tierra (Argentina.gob.ar, s.f.). La Provincia de Misiones avanza en un esquema más sofisticado: torres de videovigilancia con IA que, al detectar una columna de humo, programan automáticamente el vuelo de un dron autónomo hacia el foco para obtener coordenadas precisas antes de que el fuego se propague (Norte Misionero, 2026). La convergencia entre detección satelital y respuesta inmediata con UAVs es el tema de una línea de investigación en rápido crecimiento a nivel internacional. Liu y Sziranyi (2023) proponen una arquitectura de fusión que combina imágenes Sentinel-2 para localizar el foco activo con UAVs que realizan segmentación de caminos y planificación dinámica de rutas de evacuación en el área central del incendio, validada sobre el incendio de Chongqing de agosto de 2022. Danish et al. (2025), en una revisión sistemática exhaustiva publicada en *Artificial Intelligence Review*, mapean el estado del arte de los sistemas de gestión de incendios forestales con UAVs de visión artificial —desde detección hasta supresión— cubriendo más de siete años de producción científica. El trabajo más reciente en esta línea combina ambas fuentes en una arquitectura UAV–satélite: el pipeline usa datos históricos satelitales para priorizar zonas de patrulla y lanza múltiples UAVs de forma coordinada para detección temprana y predicción dinámica del perímetro, alcanzando un F1-score del 97,4 % con consumo energético minimizado para maximizar la autonomía de vuelo (*Drones*, 10(4), 263; DOI: 10.3390/drones10040263).

Agricultura de precisión. A diferencia de la vigilancia marítima y la respuesta a incendios, la agricultura de precisión con drones ya es, en Argentina, una tecnología en adopción masiva y no una promesa: la superficie tratada con UAVs pasó de 50.000 hectáreas en 2024 a un millón en julio de 2025, con una proyección de superar los dos millones al cierre de la campaña gruesa 2026 (Dataportuaria, 2026), sostenida por una flota que creció de 90 equipos importados en 2023 a 600 en agosto de 2024 (Infocampo, 2025). El uso dominante —pulverización dirigida de fitosanitarios y fertilización, con rendimientos operativos de hasta 35 ha/hora— convive con dos limitaciones estructurales bien documentadas en la prensa especializada: la autonomía de batería restringe cada vuelo a 3-4 hectáreas y el SENASA todavía no estableció un marco regulatorio específico para la aplicación aérea de agroquímicos con drones, a diferencia de la normativa de aviación no tripulada ya actualizada (Infocampo, 2025). La Red de Drones del INTA —más de 65 investigadores y 22 pilotos licenciados en unas 20 líneas de investigación distribuidas en el país— extiende el uso más allá de la pulverización hacia el fenotipado de biomasa en cultivos extensivos, la detección temprana de enfermedades en frutales y vid, y el monitoreo ganadero y de pasturas (INTA, 2025); un caso puntual reciente en el NOA combina drones con reconocimiento por IA para detectar y corregir en tiempo real fallas de plantación de caña de azúcar durante el propio proceso de siembra (Agrolatam, 2025), un patrón de "percibir y decidir en el mismo ciclo operativo" formalmente análogo al lazo táctico que este trabajo evalúa.

La literatura académica internacional sobre navegación autónoma de UAVs en agricultura converge, además, en el mismo problema técnico central de este trabajo: la evasión de obstáculos con percepción de borde. Zhang et al. (2024) proponen un algoritmo de planificación de trayectoria con prioridad de distancia mínima específicamente calibrado para agricultura de precisión, optimizando cobertura de área bajo restricciones de autonomía de batería —la misma restricción que limita la pulverización con drones en Argentina a 3-4 hectáreas por vuelo—. Wei et al. (2025) abordan el problema más cercano a la arquitectura de este trabajo: navegación de UAVs *dentro* de las hileras de un huerto usando exclusivamente una cámara frontal, con un controlador basado en aprendizaje por imitación que evita colisiones con el canopeo sin depender de mapas previos del terreno, logrando mayor distancia de vuelo autónomo con menor intervención humana. Li et al. (2025) resuelven el mismo problema —evasión de ramas y obstáculos en huertos para UAVs de cosecha— con un enfoque distinto, basado en LiDAR y un algoritmo híbrido A* con curvas B-spline; el contraste entre ambos trabajos (visión monocular pura vs. LiDAR) reproduce en el dominio agrícola la misma disyuntiva de costo-sensorial que motiva la elección de percepción monocular sin LiDAR de este trabajo (cap. 6). Castro et al. (2020), en un antecedente latinoamericano directo, aplican aprendizaje profundo sobre imágenes RGB capturadas por UAV para fenotipar biomasa en pasturas forrajeras brasileñas —el mismo tipo de sensor de bajo costo (cámara RGB, sin LiDAR ni multiespectral) que emplea este trabajo, aplicado a un problema de monitoreo en lugar de navegación—.

Seguridad urbana y logística aérea en Buenos Aires. A diferencia de los tres dominios anteriores, éste es el único que ya opera UAVs dentro del mismo tejido urbano denso que reproduce el entorno de evaluación de este trabajo (`TownSim`/`CitySim`, Tier 1 y 2, cap. 3 y 10) — no en simulación ni en proyecto piloto, sino en operación diaria. El Escuadrón de Drones de la Policía de la Ciudad, integrado a un sistema de videovigilancia con 850 puntos de captura, 3.400 cámaras y 82 % de cobertura territorial a julio de 2025, sostenido por una inversión cercana a los USD 48 millones (Perfil, 2026), vuela unidades DJI Matrice 200 con seguimiento automático de personas y vehículos y sensores de detección de obstáculos capaces de identificar cables de un milímetro de diámetro —el mismo desafío de percepción de estructuras delgadas y de bajo contraste que enfrenta el estimador de flujo óptico de este trabajo en cañón urbano (cap. 6)—, en misiones de apoyo a persecuciones, localización de focos de incendio y búsqueda de personas extraviadas sobre la Reserva Ecológica y el Riachuelo (Buenos Aires Ciudad, 2023). A esto se suma la apertura regulatoria para logística comercial: la Ciudad habilitó en 2025 la entrega comercial con drones (iProfesional, 2025) y la ANAC actualizó el marco nacional con las Resoluciones 319/2025 y 550/2025. El denominador técnico de ambos usos —seguridad y logística— es el mismo que motiva a Hughes y Engelbrecht (2023), §2.1 a elegir un trazado urbano tipo Manhattan para evaluar la evasión cooperativa de enjambres: cañones urbanos en cuadrícula regular, degradación de GPS por la altura de los edificios y proximidad constante a estructuras verticales y cableado — exactamente la morfología que el Tier 2 (`CitySim`) de este trabajo reproduce (cap. 3 y 10)—.

En los cuatro dominios, el patrón técnico subyacente —un vehículo autónomo que procesa información perceptual en tiempo real, toma decisiones sin intervención humana y actúa bajo restricciones de cómputo de borde— es exactamente el que este trabajo estudia y evalúa en entorno de simulación. La agricultura de precisión añade, respecto de la vigilancia marítima y la respuesta a incendios, una condición operativa que ambas no comparten: el UAV vuela en proximidad cercana y persistente a obstáculos semiestructurados —canopeo, ramas, alambrados, líneas de media tensión rurales— con tolerancia a error mínima, exactamente la morfología de obstáculo orgánico que el Tier 1 de este trabajo (`TownSim`) reproduce sobre vegetación urbana. La seguridad y logística urbana en Buenos Aires cierra el círculo entre escenario de evaluación y aplicación real: es el único de los cuatro dominios donde el desafío técnico no se proyecta a futuro ni se transfiere por analogía, sino que ya opera, hoy, en el mismo tipo de cañón urbano que constituye el Tier 2 de este trabajo. La ausencia de literatura académica argentina específica sobre los otros tres dominios no resta relevancia a los resultados; al contrario, señala una oportunidad de transferencia directa de los métodos y hallazgos documentados en los capítulos siguientes hacia dominios de alto impacto civil local.

2.5 Posicionamiento de este trabajo

Este trabajo se ubica en la intersección de esas tres líneas —percepción monocular sin redes de detección, control por FSM como línea de base, y un SLM con salida restringida como capa de decisión alternativa— pero con un énfasis distinto al de buena parte de la literatura revisada. Mientras la frontera de investigación de 2025-2026 (Tian et al., 2025; Sun et al., 2026) opera con modelos VLA de gran escala y múltiples miles de millones de parámetros en hardware especializado, este trabajo aborda deliberadamente el extremo opuesto del espectro: un VLM cuantizado de 1.6 B parámetros ejecutado localmente, en la misma GPU de consumo que la simulación, sin nube y sin entrenamiento por tarea. En lugar de proponer una arquitectura novedosa de percepción o de razonamiento, documenta con evidencia medida los modos de falla específicos de integrar un modelo de lenguaje pequeño en un lazo de control real —un tipo de resultado poco frecuente en la literatura publicada, que tiende a reportar arquitecturas que funcionan más que arquitecturas que fallaron de manera instructiva y cómo se detectó y corrigió esa falla—. Esta contribución es especialmente pertinente en el contexto de las aplicaciones civiles y militares descritas al inicio del capítulo: los escenarios de mayor relevancia práctica —drones de inspección en zonas sin cobertura, vehículos de rescate en áreas de desastre, plataformas tácticas en entornos de guerra electrónica, y pulverizadores agrícolas operando en campo abierto sin señal de red— son precisamente aquellos en que los modelos de nube y el hardware de alto costo son inviables, y donde entender los límites y modos de falla del SLM de borde tiene valor operacional directo.

3. Construcción y validación del entorno de simulación

3.1 Arquitectura de simulación: Unreal Engine 5.5 y Cosys-AirSim

El desarrollo y la validación de este trabajo se realizan, tal como preveía el plan de trabajo aprobado (`plan_tesis/plan-tesis.md`), sobre **AirSim**, un simulador basado en Unreal Engine que ofrece física y renderizado de alta fidelidad (Shah et al., 2017). La implementación no usa la distribución original de AirSim de Microsoft, sino **Cosys-AirSim** (específicamente el build para Windows de la versión Cosys-AirSim v3.3 for Unreal v5.5, un fork mantenido por el Cosys-Lab (Laboratorio de Co-Diseño para Sistemas Ciber-Físicos de la Universidad de Amberes, Bélgica), porque el proyecto original de Microsoft está abandonado. El plugin se compila e integra sobre proyectos de Unreal Engine 5.5 (específicamente la versión 5.5.4-40574608+++UE5+Release-5.5), que sirven como banco de pruebas y validación experimental.

Cosys-AirSim, a diferencia del AirSim clásico de Microsoft —enfocado principalmente en cámaras RGB y visión por computadora—, agrega sensores adicionales basados en GPU y CPU (LiDAR, sonar, radar), documentados en el paper oficial de la plataforma (Jansen et al., 2023). Este trabajo no usa esos sensores adicionales: la percepción descrita en el capítulo 6 depende exclusivamente de la cámara RGB monocular y de la telemetría de actitud, consistente con la restricción de hardware de bajo costo que motiva todo el proyecto (§1.2). La elección de Cosys-AirSim sobre el AirSim original responde a la actividad de mantenimiento del fork más que a una necesidad de esos sensores adicionales: el proyecto de Microsoft no recibe actualizaciones activas.

Particularidades de la API y telemetría de actitud. A diferencia del paquete clásico `airsim`, el binding `cosysairsim` no expone la función auxiliar `to_eularian_angles` para la extracción directa de ángulos de actitud; sin ella, un fallback ingenuo reporta *pitch* y *roll* en 0.0° de forma permanente. Para no introducir dependencias externas, el cliente de simulación (`AirSimClient._quaternion_to_euler`) implementa la conversión matemática directa de cuaterniones a ángulos de Euler en convención aeroespacial NED (North-East-Down), de modo que las variaciones de inclinación del cuadricóptero alimenten con fidelidad continua la telemetría del lazo táctico y el contexto del modelo deliberativo.

Casos de uso de AirSim y Cosys-AirSim en la literatura

La elección de AirSim como plataforma no es un caso aislado: es el simulador de referencia de una línea de trabajo amplia que abarca múltiples tipos de vehículo, dominios de aplicación y paradigmas de aprendizaje, lo cual sitúa a este trabajo dentro de un ecosistema de investigación activo en lugar de una herramienta de nicho.

AirSim más allá de los cuadricópteros. Aunque este trabajo usa exclusivamente el modelo de multirotor, AirSim fue diseñado desde su publicación original como una plataforma multivehículo: el mismo motor de física e interfaz de API sirve tanto para drones como para automóviles autónomos, con un *wrapper* oficial de ROS (`airsim_ros_pkgs`) que expone odometría NED, imágenes de cámara y datos de LiDAR/IMU como tópicos estándar para integrarse con pilas de navegación basadas en ROS/ROS2. Una revisión reciente de simuladores de código abierto para conducción autónoma (Li et al., 2023) ubica a AirSim junto a CARLA y LGSVL como una de las plataformas de referencia del campo, con la particularidad —relevante para este trabajo— de estar construido sobre Unreal Engine en lugar de un motor gráfico propio, lo que permite reutilizar directamente el mismo pipeline de escalabilidad gráfica y activos de entorno descripto en §3.3 para ambos tipos de vehículo.

En el dominio aéreo específicamente, AirSim ha sido la base de dos líneas de investigación directamente pertinentes a este trabajo. La primera es la competencia *Game of Drones* de NeurIPS 2019 (Madaan et al., 2020), organizada junto con Stanford y la Universidad de Zúrich sobre el framework *AirSim Drone Racing Lab*: benchmarking de percepción, planificación de trayectoria y control de multirotores compitiendo cabeza a cabeza en circuitos de carreras simulados, con el objetivo explícito de reducir la barrera de entrada de la comunidad de *machine learning* a la investigación en autonomía de drones —el mismo objetivo de accesibilidad que motiva la elección de un simulador gratuito y de código abierto en este trabajo—. La segunda línea es más cercana aún al problema central de esta tesis: Xue y Gonsalves (2021) entrenan un agente de aprendizaje por refuerzo profundo (*soft actor-critic*) para evasión de obstáculos usando exclusivamente mapas de profundidad como entrada, en un entorno construido íntegramente sobre AirSim, logrando generalización a entornos reconfigurados sin reentrenamiento. El paralelismo metodológico es directo —evasión de obstáculos a bordo de un UAV, en AirSim, a partir de un único canal de percepción visual— aunque la elección de arquitectura difiera en un punto central: donde Xue y Gonsalves aprenden una política mediante aprendizaje por refuerzo sobre profundidad, este trabajo compara una política aprendida implícitamente por un modelo de lenguaje pre-entrenado contra una heurística explícita, ambas sobre flujo óptico monocular, y usa la profundidad solo estimada desde el RGB, nunca el sensor de profundidad del simulador (cap. 6). AirSim también se ha usado para escenarios de más alto nivel de sistema: Turco et al. (2024) construyen un gemelo digital de co-simulación entre MATLAB y AirSim para movilidad aérea avanzada (*Advanced Air Mobility*), y sus mediciones de consumo de VRAM del propio AirSim sobre Unreal Engine son, de hecho, la fuente del presupuesto de memoria de video que condiciona la elección del modelo de lenguaje de este trabajo (cap. 8, §8.1).

Cosys-AirSim y la extensión a nuevos dominios de vehículo. La motivación declarada del fork de Cosys-Lab no es solamente mantenimiento activo (§3.1): es también la incorporación de modalidades de sensor y tipos de vehículo ausentes en el AirSim original, documentada en su propio artículo de referencia (Jansen et al., 2023) — un sensor LiDAR acelerado por GPU con mayor densidad de puntos, un sensor genérico de tipo *echo* que modela tanto sonar como radar mediante trazado de rayos, y un modo de simulación de dirección por deslizamiento (*skid-steering*) con vehículos terrestres de referencia (ClearPath Husky, Pioneer P3DX) ya implementados como tipos de vehículo. Ninguna de estas extensiones se usa en este trabajo, que se restringe deliberadamente a cámara RGB monocular y telemetría de actitud (§3.1) — pero su existencia es la evidencia de que Cosys-AirSim es una plataforma activamente extendida por la comunidad, no un fork estático. La extensión más reciente y más ilustrativa de esa actividad es ASVSim (Lesy et al., 2026): un framework construido directamente sobre Cosys-AirSim que añade dinámica de embarcaciones y sensores marítimos (radar, cámara) para investigación en vehículos de superficie autónomos en canales navegables y puertos —una prueba directa de que la arquitectura de Cosys-AirSim generaliza a un dominio de vehículo completamente distinto (naval en lugar de aéreo o terrestre) sin reescribir el núcleo de simulación—, publicado por el mismo grupo de investigación (Cosys-Lab, Universidad de Amberes) que mantiene el fork adoptado por este proyecto.

Este panorama sitúa la elección metodológica de este trabajo con precisión: de una plataforma que soporta multirotores, automóviles, vehículos con dirección por deslizamiento y embarcaciones, con sensores que van desde cámara RGB hasta LiDAR, sonar y radar, este trabajo usa deliberadamente el subconjunto más restringido posible —un multirotor con una única cámara RGB monocular y sin sensores de profundidad activos—, consistente con la restricción de hardware de bajo costo que motiva el proyecto (§1.2) y con el objetivo de aislar qué tan lejos puede llegar la navegación autónoma con la configuración sensorial más económica que la plataforma permite simular.

3.2 Desvío respecto del plan aprobado: de la fotogrametría de Buenos Aires a activos urbanos genéricos

El plan de trabajo aprobado describía un pipeline de digitalización de entornos urbanos específico: extraer fotogramas de videos públicos de drones en Buenos Aires (YouTube), filtrarlos para asegurar diversidad visual, anotar obstáculos y zonas de aterrizaje con LabelImg/LabelMe, y reconstruir modelos 3D hiperrealistas con RealityCapture (fotogrametría) a partir de esos videos y de OpenStreetMap, optimizados en Blender para reducir su complejidad computacional antes de importarlos a AirSim (`plan_tesis/plan-tesis.md`, §"Metodología"). El objetivo declarado era un entorno virtual 3D *personalizado* de Buenos Aires, con las particularidades geométricas de su infraestructura densa.

La implementación efectivamente construida sustituyó ese pipeline de fotogrametría por activos urbanos y suburbanos ya disponibles para Unreal Engine (marketplace de Fab/Epic Games y paquetes de terceros), configurados con el plugin de Cosys-AirSim en lugar de reconstruidos a partir de video real de Buenos Aires (disponibles en el repositorio compartido de Google Drive: https://drive.google.com/drive/folders/1roLmbGFNsHXZyT3NaNzNYMuaBQ8CulX7).

Entornos principales estructurados por Tiers de evaluación

Para responder con rigor al protocolo experimental de la tesis (capítulo 10), los entornos de simulación se organizan en una **jerarquía formal de tres niveles de complejidad (Tiers)**:

1. **Tier 0 — Base / Control (`MiniSim`):** Entorno abierto basado en el **proyecto base de Unreal Engine** https://dev.epicgames.com/documentation/unreal-engine/unreal-engine-templates-reference?application_version=5.5: un plano liso con textura de tablero, sin relieve ni objetos (la imagen `crater.png` que muestra WebDCS es solo un fondo y no corresponde al mundo). Diseñado para pruebas de calibración cinemática, sintonización de controladores de posición y determinación de la cota inferior de latencia del sistema sin interferencia de obstáculos.
2. **Tier 1 — Intermedio / Morfología orgánica y vegetación (`TownSim`):** Entorno semiurbano (mapa cenital `townsim_ortho.png`, construido con capturas del propio simulador: 4 px/m, origen NED en el centro, norte arriba; cap. 4 §4.3.2), basado en el entorno **Downtown West Modular Pack** https://www.fab.com/listings/0faf8b5d-7a5f-4fee-a297-7a8efaba8896 de PurePolygons. Presenta calles de ancho variable, fachadas intermedias, mobiliario y arboledas densas. Constituye el banco de pruebas de los escenarios de recorrido perimetral (`townsim_clear`), de travesía del corredor arbolado (`townsim_ini`) y de bloqueo frontal masivo (`townsim_calib_cruce_frontal`).
3. **Tier 2 — Complejo / Cañones urbanos de gran densidad (`CitySim`):** Proyecto de Unreal Engine 5.5 con tejido edilicio masivo (mapa cenital `citysim_ortho.png`, construido con capturas del propio simulador: 2 px/m, origen NED en el centro, norte arriba; cap. 4 §4.3.2), rascacielos, pasos restringidos y disposición en cuadrícula ortogonal, basado en el entorno **Sample City** https://www.fab.com/listings/4898e707-7855-404b-af0e-a505ee690e68 de Epic Games. El escenario de control es `citysim_clear.json`: perímetro de una manzana con 4 waypoints (~420 m) y altitud creciente en rampa (−10, −30 y −50 m) antes de volver a −10 m. Los escenarios con corredores angostos (`citysim_pilot.json`) constituyen el escenario de dificultad máxima del tier.

3.3 Configuración de rendimiento, escalabilidad gráfica e higiene de captura

La documentación técnica de Cosys-Lab sostiene que priorizar la tasa de actualización de la física por sobre la fidelidad gráfica es una práctica recomendada cuando el objetivo principal no es la fotografía cinemática de la escena. Siguiendo esas directrices y considerando que la misma estación de cómputo aloja en simultáneo el renderizado 3D y la inferencia del modelo de lenguaje local (§8.1), se establecieron los siguientes compromisos de arquitectura:

- Desactivar el ahorro de CPU en segundo plano del editor de Unreal Engine (`Use Less CPU when in Background`), evitando caídas drásticas en el refresco de la física cuando la ventana del simulador pierde el foco.
- Ajustar `ClockSpeed` en `settings.json` para sincronizar el tiempo de simulación y garantizar que el lazo físico se calcule sin pérdidas temporales cuando la carga gráfica fluctúa.

- Preferir binarios empaquetados (*standalone*) sobre la ejecución directa desde el editor, reduciendo la sobrecarga de memoria VRAM asociada a la interfaz de desarrollo.
- El modo `NoDisplay` no pudo utilizarse, dado que la captura de imágenes RGB monocular es estrictamente indispensable para la percepción del dron (capítulo 6).

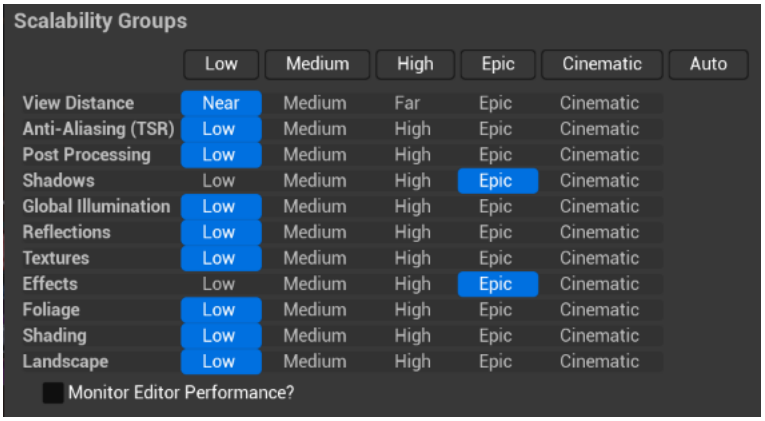
Escalabilidad gráfica en Unreal Engine 5.5

Para equilibrar la carga de GPU entre Unreal Engine y el servidor SLM/VLM local, se definió un perfil de escalabilidad mínima (*Minimal Scalability Config*) que fija sombras, postprocesamiento e iluminación global en niveles bajos (`Low`/`Medium`), reservando la VRAM necesaria para los pesos del modelo multimodal.

Sin embargo, en Unreal Engine 5.5 la reducción automática de efectos suele degradar la resolución de los render targets de captura de escena. Para evitar que la cámara frontal pierda nitidez o altere el cálculo de flujo óptico, se forzó el nivel de detalle en el archivo `Config/DefaultScalability.ini` del proyecto Unreal:

```
[EffectsQuality@0]
r.DetailMode=2
[EffectsQuality@1]
r.DetailMode=2
[EffectsQuality@2]
r.DetailMode=2
[EffectsQuality@3]
r.DetailMode=2
[EffectsQuality@Cine]
r.DetailMode=2
```

También desde el editor de Unreal Engine se puede configurar la escalabilidad de la siguiente manera:



Con esta configuración Scalability de Unreal Engine se asegura a la hora de ejecutar la simulación: - Captura de video desde AirSim (con `Effects=Epic`) - Sombras en las texturas necesarias para capturar los gradientes usados para el cálculo del TTC con el flujo visual (con `Shadows=Epic`).

(Para la especificación completa del archivo `settings.json` anotado, el perfil de escalabilidad y el protocolo de reproducibilidad del entorno de simulación, véase el Anexo 6).

Higiene y consistencia de la captura monocular

El cliente de simulación aplica dos medidas de higiene sobre el canal visual:

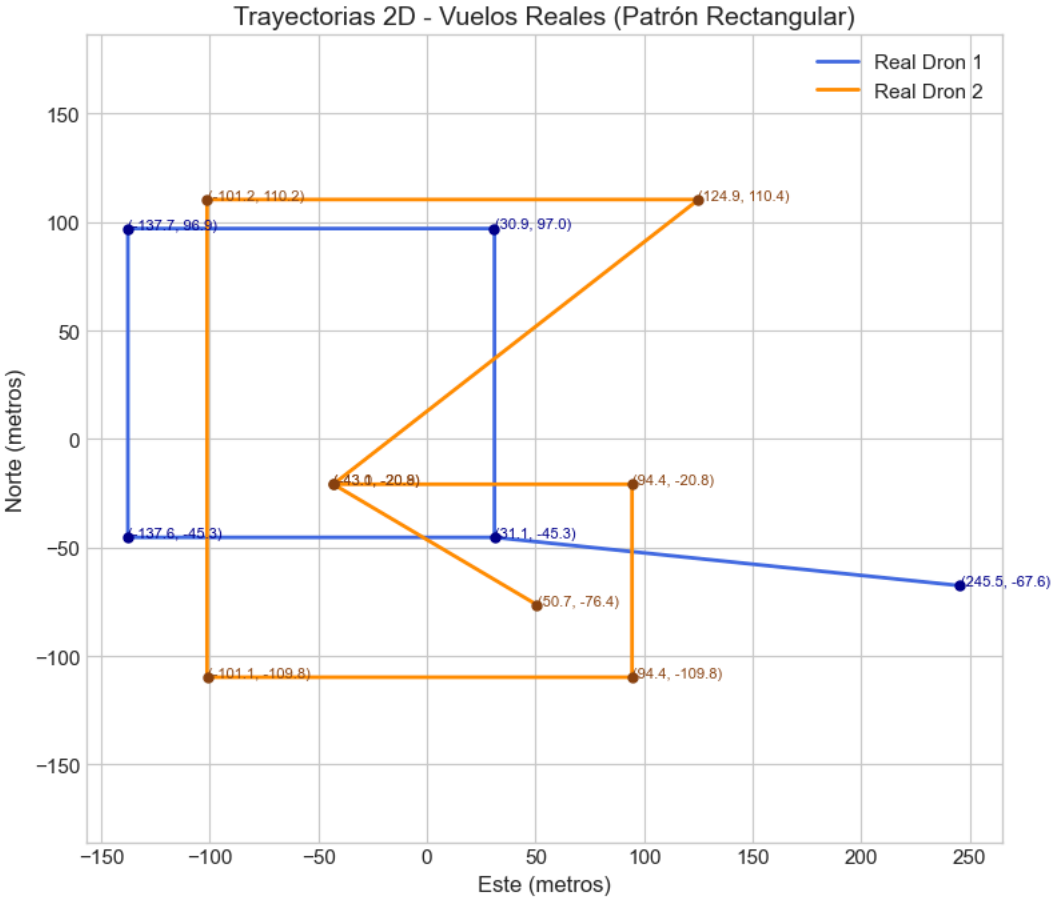
1. **Orden de canales de color (RGB vs. BGR):** El buffer crudo devuelto por `simGetImages(ImageType.Scene)` en Cosys-AirSim viene en formato RGB, mientras que OpenCV y las rutinas de compresión asumen la convención BGR; sin conversión, las capturas enviadas al VLM y guardadas en auditoría muestran rojo y azul invertidos. `AirSimClient.capture()` convierte el orden de canales, de modo que el modelo razona sobre los colores naturales de la escena (pavimento, vegetación y cielo).
2. **Supresión de marcadores de depuración persistentes:** Primitivas geométricas dibujadas en el mundo virtual por scripts de planificación (mediante `simPlotLineStrip` y `simPlotPoints` de `plot_mission_route.py`) permanecen activas en el nivel, la cámara a bordo las captura y el VLM las interpreta como obstáculos físicos. Por eso cada sesión de conexión empieza con una rutina de higiene obligatoria (`AirSimClient.clear_debug_markers()`, que invoca `simFlushPersistentMarkers()`).

3.4 Validación frente a telemetría de vuelos reales

La dinámica de vuelo simulada —no la geometría de la escena urbana, cuya fidelidad fotográfica es el desvío documentado en §3.2— se validó de forma independiente contra telemetría de drones reales, con el objetivo explícito de que la comparación experimental entre brazos (capítulo 10) se apoyara en vuelos simulados cuya cinemática fuera comparable a la de un cuadricóptero real y no solo físicamente plausible en abstracto.

Fuente de datos reales. Se descargó un conjunto de datos de telemetría real de drones cuadricópteros comerciales (DJI) desde el repositorio público de Zenodo (https://zenodo.org/records/15912415).

Protocolo de calibración. Sobre esa telemetría real se identificaron dos trayectorias de vuelo distintas — un patrón rectangular simple (Drone 1) y un patrón de cruz combinado con rectángulo (Drone 2)— y se reprodujeron en simulación con `moveOnPath` / `move` de la API de AirSim, ajustando la simulación a la misma altura y la misma velocidad que los vuelos reales correspondientes. La trayectoria completa se envía como un único comando `moveOnPath`, y no como una secuencia de tramos independientes, para que el perfil de aceleración simulado se parezca al de un vuelo real continuo; la telemetría de cada vuelo simulado se registra en un CSV individual.

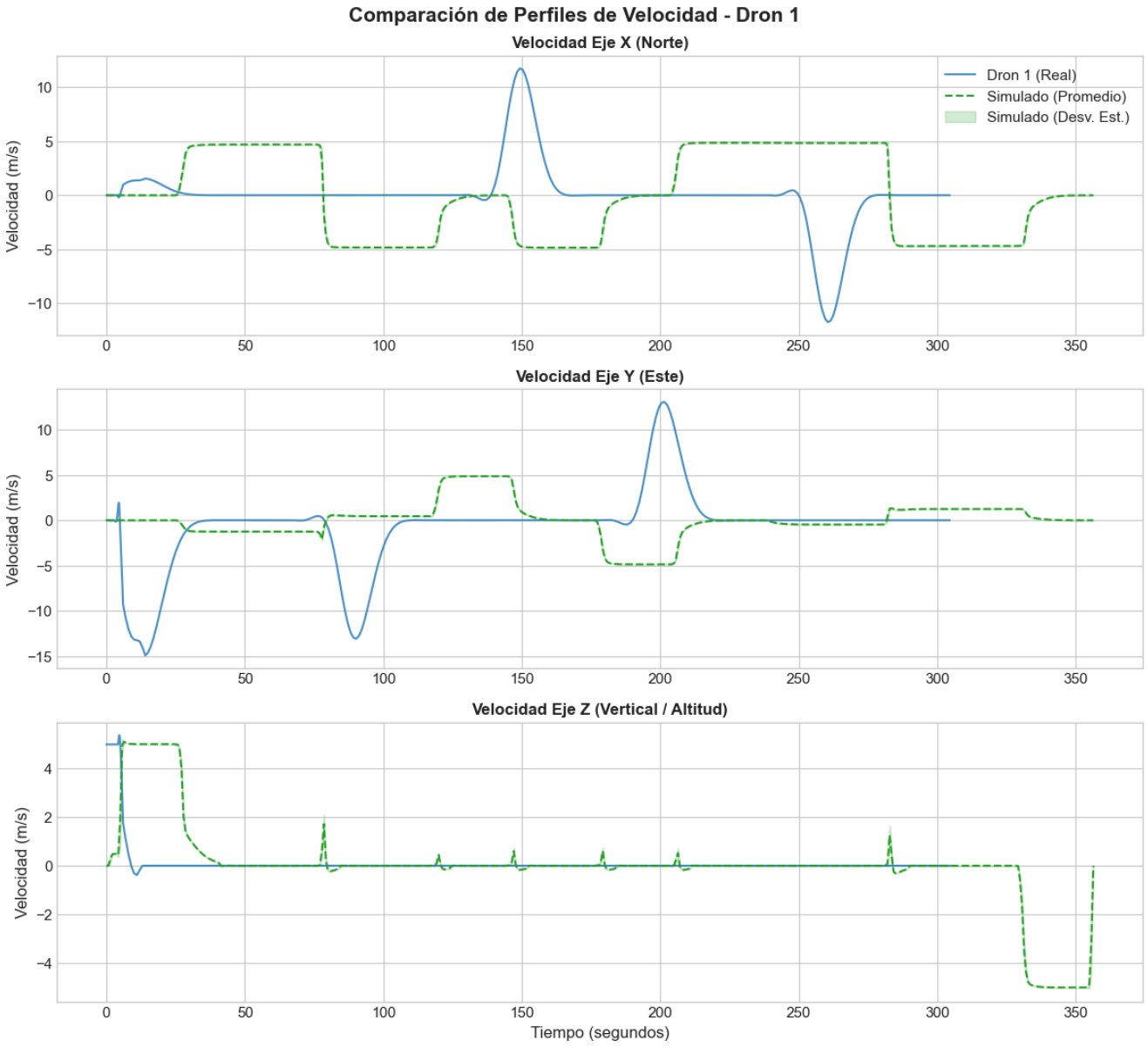


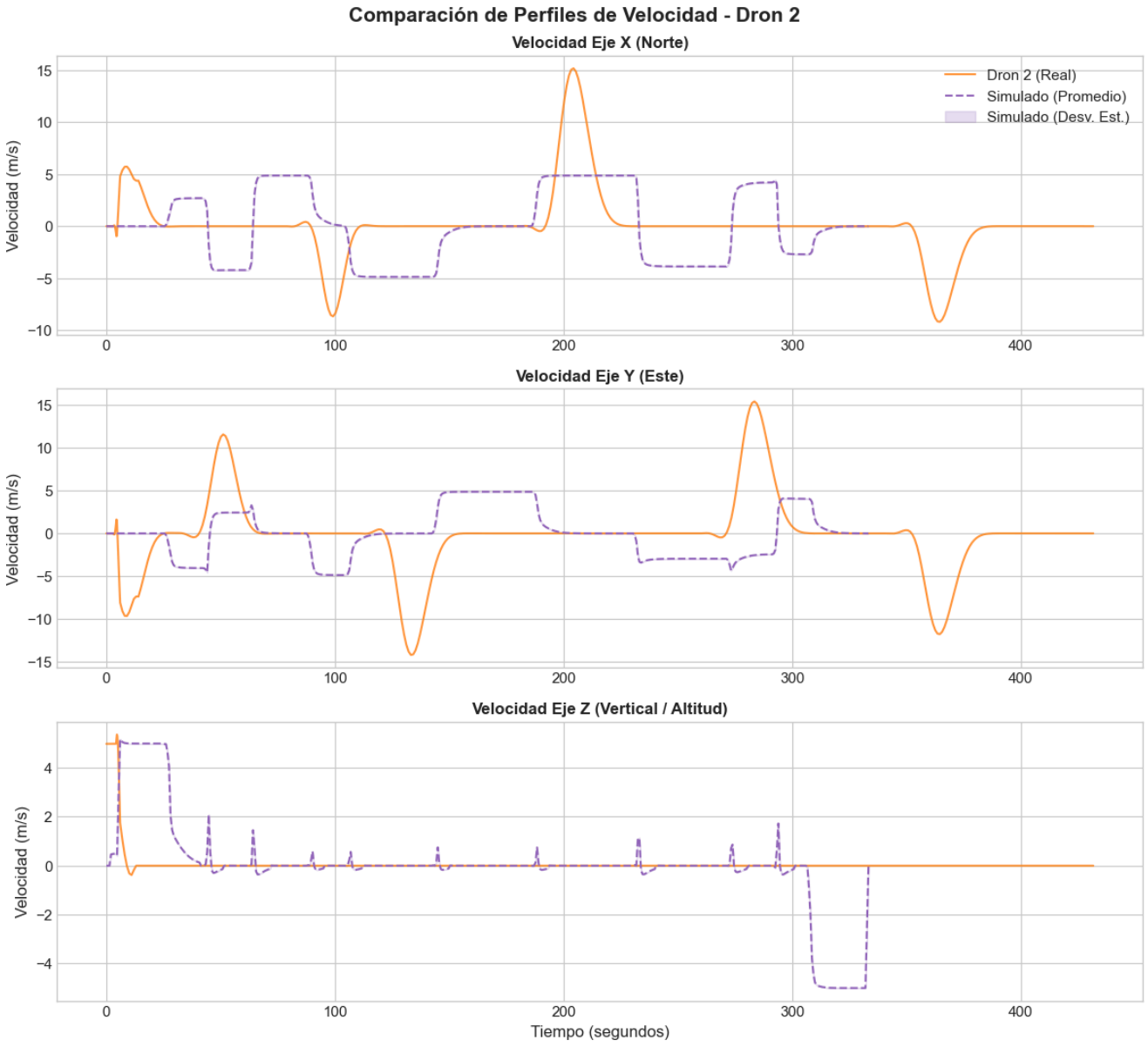
Metodología estadística. El análisis se implementó en notebooks de Jupyter del repositorio complementario del proyecto (`consolidate_telemetry.ipynb` y `telemetry_analysis_*.ipynb`, georgsmeinung/lm-drone), con estadística descriptiva y pruebas de significancia —en particular la prueba de Levene para diferencia de varianzas— aplicadas al cambio de velocidad en los tres ejes, para determinar si la distribución de la telemetría simulada difiere de forma significativa de la real.

Resultado. El análisis documenta tres hallazgos consistentes:

1. **La varianza de actitud difiere de forma significativa** (prueba de Levene, $p \ll 0{,}05$): en simulación, el dron ejecuta inclinaciones de *roll*/*pitch* extremas durante los giros rápidos para alcanzar instantáneamente el punto de trayectoria siguiente. Los drones reales, en cambio, están limitados electrónicamente por su controlador PID de estabilización (típicamente a ±30°), y muestran una varianza mucho menor y acotada durante las mismas maniobras.
2. **La segregación por trayectoria es significativa y consistente.** El Drone 1 (patrón rectangular, giros menos frecuentes) concentra la aceleración en las esquinas; el Drone 2 (patrón de cruz y rectángulo continuo) presenta una dinámica transicional más exigente y oscilaciones de *roll*/ *pitch* más ruidosas, tanto en el vuelo real como en el simulado.
3. **El vuelo simulado en tramos rectos es idealizado.** Durante las fases rectas, la telemetría simulada muestra varianza de actitud cercana a cero, sin fuerzas externas de viento ni ruido de sensores; el dron real, en cambio, mantiene una variabilidad permanente de ±2°–3° en *roll* y *pitch* incluso en

tramos rectos estables, producto del viento real y de las correcciones continuas del piloto automático.





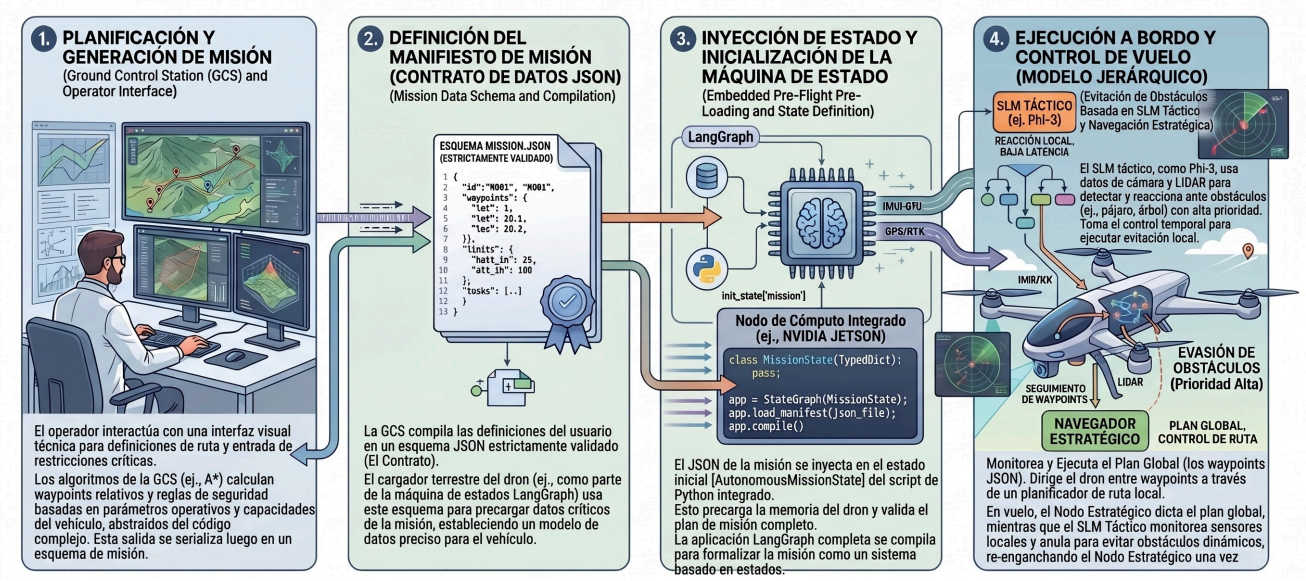
Alcance de esta validación. Confirma que la dinámica de vuelo simulada —cómo se mueve el dron dado un comando— es más ágil y menos amortiguada que la de un dron real equivalente, con una brecha de idealización conocida y cuantificada (varianza de actitud, ausencia de viento simulado). No valida, en cambio, la fidelidad fotográfica de la escena urbana frente a Buenos Aires real: esa es una validación distinta, no realizada, y coincide con el desvío documentado en §3.2. Un ejercicio de validación relacionado pero diferente —si el suavizado por histéresis y media móvil exponencial de `WaypointTracker` (capítulo 5, §5.15) acerca de forma medible la telemetría simulada a la real— da un resultado ambiguo (`std(Δvx)` prácticamente igual con y sin suavizado) y se informa como tal, no como una mejora confirmada.

4. Planificación de misiones y estación de control en tierra (WebDCS)

4.1 Arquitectura desacoplada: dos cerebros (tierra vs. vuelo)

La arquitectura general del sistema responde a un principio de diseño fundamental: la separación estricta entre la **planificación deliberativa global previa al vuelo** (ejecutada en tierra) y la **navegación táctica reactiva** (ejecutada a bordo durante el vuelo). Esta división, conceptualizada como una arquitectura de *dos cerebros*, resuelve la asimetría intrínseca de recursos computacionales y tolerancias de latencia entre ambas fases operativas:

1. **El planificador de estación terrena (`airsim-plan` / WebDCS):** opera de manera previa al despegue del vehículo. En esta etapa, el sistema puede tolerar presupuestos de tiempo del orden de 5 a 10 segundos para interpretar instrucciones en lenguaje natural, consultar información contextual del entorno, aplicar reglas de seguridad operacional y generar un plan de vuelo global validado.
2. **El lazo táctico a bordo (`airsim-loop`):** opera en tiempo real estricto una vez que el vehículo está en el aire (capítulo 5). Bajo una frecuencia objetivo de 5 a 10 Hz, su función no es diseñar la ruta global sino mantener la estabilidad cinemática, seguir los objetivos asignados y ejecutar maniobras reactivas inmediatas de evasión de obstáculos a partir de visión monocular ligera.



El desacoplamiento permite que la misión sea validada, persistida y versionada de forma determinista antes de iniciar los motores. Asimismo, garantiza que el lazo táctico a bordo no dependa de una conexión de red permanente con la estación terrena: una vez transmitido el manifiesto de vuelo, el dron opera de forma enteramente autónoma, protegiendo la seguridad del vuelo ante eventuales pérdidas de enlace de radio o telemetría.

4.2 Compilación de misiones desde lenguaje natural

El módulo de planificación en tierra integra un modelo de lenguaje (LLM) que actúa como compilador semántico. Su objetivo es transformar directivas operativas de alto nivel expresadas en lenguaje natural por un operador humano (por ejemplo, *"recorrer el perímetro norte a 10 metros de altura evitando zonas de alta densidad vehicular"*) en un artefacto estructurado y ejecutable por el piloto automático.

Para garantizar la fiabilidad del proceso de compilación sin incurrir en alucinaciones o errores de sintaxis:

- Se utiliza un cliente local compatible con la API de OpenAI (conectado a instancias locales de LM Studio u Ollama), ejecutando modelos orientados a seguimiento de instrucciones (tales como Llama-3-8B, Qwen-7B o Phi-4).
- Se define un *System Prompt* formalizado (`compiler_system.md`) que instruye al modelo sobre las dimensiones del entorno de simulación, las restricciones del espacio aéreo urbano y el formato de salida requerido.
- Se implementa una capa de coerción y extracción de JSON (`json_extract.py`) respaldada por esquemas de validación estricta en **Pydantic** (`manifest.py`). Si la salida generada por el LLM no cumple con los tipos de datos o las restricciones cinemáticas impuestas, el compilador rechaza el plan y solicita una reevaluación antes de comprometer el vuelo.

Cabe una aclaración sobre el alcance actual de la interfaz: si bien el compilador semántico descripto en esta sección está completamente implementado y expuesto como endpoint del backend, la interfaz de WebDCS en su estado presente no ofrece ningún control para invocarlo. La superficie de interacción del operador quedó acotada, por decisión de alcance del proyecto, a la edición manual de waypoints sobre la carta de navegación (§4.4.1) — priorizando el desarrollo del control de vuelo por sobre la interfaz de planificación asistida. La compilación desde lenguaje natural queda documentada aquí como una capacidad construida y validada, que puede exponerse en la interfaz sin cambios en el backend.

4.3 El contrato del Manifiesto de Misión (`MissionManifest`)

El producto final del planificador terrestre es un archivo JSON denominado **`MissionManifest`**. Este documento actúa como un contrato formal e inmutable entre la estación terrena y el lazo táctico de vuelo. Su schema canónico está definido en Pydantic (`manifest.py`) y validado contra un JSON Schema (`manifest_schema.json`).

```
{
  "mission_id": "CITYMAP_PILOT",
  "summary": "Tier 2: circuito tipo grilla urbana sobre citysim_ortho.png ...",
  "start_pose": { "x": 0.0, "y": 0.0, "z": -10.0, "yaw_deg": 90.0 },
  "waypoints": [
    {"x": 3.0,   "y": -51.0,  "z": -10.0, "label": "WP_0_SUR"},
    {"x": 26.1,  "y": -78.2,  "z": -10.0, "label": "WP_1"},
    {"x": 68.9,  "y": -78.6,  "z": -10.0, "label": "WP_2"},
    {"x": 67.7,  "y": -156.5, "z": -10.0, "label": "WP_3"},
    {"x": 26.5,  "y": -145.0, "z": -10.0, "label": "WP_4"},
    {"x": -16.3, "y": -145.0, "z": -10.0, "label": "WP_5"},
    {"x": -17.9, "y": -78.6,  "z": -10.0, "label": "WP_6"},
    {"x": 12.5,  "y": -78.6,  "z": -10.0, "label": "WP_7"}
  ],
  "map": "citysim_ortho.png"
}
```


Manifiesto vigente `citysim_pilot.json` *(cap. 10 §10.3.3 explica* `WP_0_SUR`*; el mapa se describe en §4.3.2).*

Componentes del contrato:

- **`mission_id`**: identificador único en formato `^[A-Z0-9_]{3,32}$`, validado por el schema. Identifica el escenario experimental y aparece en los nombres de carpeta de telemetría.
- **`summary`**: descripción textual de la intención operativa (campo opcional, no consumido por el lazo táctico).
- **`waypoints`**: lista ordenada de puntos de paso tridimensionales en el marco **NED** (*North-East-Down*): $z < 0$ representa altitud sobre el punto de despegue. Cada waypoint lleva una etiqueta opcional (`label`) para identificación en la traza.
- **`map`**: nombre de la imagen cenital del escenario. WebDCS la usa para ubicar y mostrar los waypoints, y el planificador de ruta (§4.3.1) para decidir por dónde va cada tramo. La escala (px/m) y el origen de cada imagen se declaran en `airsim-plan/missions/maps/map_scales.json`, con la convención: centro de la imagen = `ned_offset`, arriba = norte (+x), derecha = este (+y). Como el plan de ruta se deriva del mapa, un mapa mal registrado sí afecta al vuelo (§4.3.2).
- **`start_pose`** (opcional): pose de partida; solo se usa con `--seed-jitter`, porque el runner devuelve el dron al *spawn* de `settings.json` (cap. 10 §10.4.3). Debe coincidir con ese *spawn*.

Separación entre manifiesto y consultas en vuelo. El manifiesto entrega únicamente metas geométricas. Las consultas del lazo lento no forman parte de él porque dependen del fotograma, la pose y el waypoint activo de ese instante. Hay dos (cap. 5 §5.10, §5.12):

1. **Capa estratégica**: el recorte cuadrado central del fotograma frontal, dividido en una grilla de 3×3, con la dirección del destino. La respuesta dice, para cada uno de los nueve sectores, si se puede volar al menos 15 m en esa dirección.
2. **Barrido** (solo en un deadlock): cuatro imágenes tomadas girando en el lugar. La respuesta dice, para cada una, si se puede volar recto 15 m; el rumbo de cada imagen lo conoce el código.

En la configuración de los experimentos (cap. 10) las dos preguntas las responde la red de profundidad estimada (cap. 5 §5.10.6, cap. 6 §6.14): el p5 de la profundidad de cada sector frente a 15 m. Con `STRATEGIC_SOURCE = vlm` o `DEADLOCK_STRATEGY = deep_vlm` las responde el VLM, con prompts que se arman en vuelo y decodificación restringida (`json_schema`) a esquemas sin campo de acción (cap. 8 §8.5). En los dos casos el código convierte la respuesta en una sub-meta en coordenadas del mundo, que el `WaypointTracker` inserta delante del waypoint del manifiesto como un waypoint temporal.

4.3.1 Plan de ruta con el VLM sobre el mapa cenital

El manifiesto une los waypoints en línea recta, y en una ciudad la recta suele cruzar una manzana. En vuelo, la cámara ve solo lo que tiene delante a 10 m de altura y el sistema debe inferir por dónde rodear un edificio cuyo final no ve; desde arriba, la misma pregunta se vuelve de reconocimiento: si una línea dibujada sobre el mapa va por la calle o por encima de los edificios. Es el tipo de juicio que un VLM pequeño resuelve mejor, porque no exige estimar distancias en primera persona. Por eso la estación terrena agrega un paso de planificación con el VLM **antes del despegue** (`src/planning/route_planner.py`): corre una sola vez por misión, la misión espera su resultado y nada de esto ocurre dentro del lazo táctico.

1. **Candidatas geométricas.** Para cada tramo A → B (incluido el tramo desde el *spawn* al primer waypoint) se generan rutas sin mirar el mapa: la recta, las dos rutas en L (primero norte-sur o primero este-oeste) y desvíos paralelos a ±25 y ±50 m a cada lado.
2. **Juicio del VLM por parches** (`ROUTE_JUDGE=patches`). El modelo no juzga la ruta entera, sino lo que hay debajo de ella: cada candidata se muestrea cada `ROUTE_PATCH_STEP_M` (8 m); en cada punto se recorta un parche de `ROUTE_PATCH_M` (16 m) con `ROUTE_PATCH_CONTEXT_M` (48 m) de contexto, y el modelo lo clasifica en una de cinco clases (calle, edificio, árboles, estacionamiento, autopista elevada) con decodificación restringida (temperatura 0). La puntuación de la candidata es la probabilidad media de «calle», leída de los *logprobs*. El modelo del planificador se configura con `ROUTE_LLM_MODEL_NAME` (anexo A6) y es el mismo modelo de vuelo.

El código conserva una alternativa, `ROUTE_JUDGE=image`, que sirve de contraste para justificar la elección: dibuja cada candidata como un **pasillo** del ancho que necesita el dron (`ROUTE_RENDER=corridor`, `ROUTE_CORRIDOR_M` = 10 m) o como una línea (`ROUTE_RENDER=line`), sobre su propio recorte del mapa. El recorte se rota para que la ruta vaya del inicio (abajo, verde) al destino (arriba, azul), se acerca a la candidata (672 × 672 px) y fuera del pasillo el mapa se oscurece. Se le pregunta al modelo si todo el pasillo es calle o terreno abierto, con este prompt de sistema:

> You see a top-down city map. The bright strip outlined in red is a drone path from the green dot (bottom) to the blue dot; the rest is darkened. Answer yes only if the whole strip covers street or open ground, with no building, trees or elevated road inside it. Reply with JSON only.

y el mensaje, `Drone altitude 10 m. Is the whole bright strip street or open ground? yes or no.` La respuesta se fuerza a `yes`/`no` y la probabilidad de `yes` se lee de los *logprobs* del token de la respuesta. 3. **Elección.** Gana la candidata de mayor puntuación (con `image`, la de mayor probabilidad entre las aprobadas); a menos de 0.05 de la mejor, la más corta. Si no aprobó ninguna, queda la recta y el tramo se resuelve en vuelo como siempre. Los vértices intermedios de la elegida se insertan en el manifiesto como waypoints `VIA_<WP>_<k>` (con `planned_via: true` y la altura del waypoint de destino), delante del waypoint del tramo.

El modelo evalúa; el código solo genera las opciones y toma la mejor según el propio modelo. El plan completo —cada candidata con su imagen, respuesta y probabilidad, la elegida y el motivo— queda en `<out-dir>/<escenario>/route_plan/` junto al manifiesto planificado, y se cachea por contenido (waypoints, mapa, modelo y prompt): Con `--route-plan vlm`, `runner.py` y `batch_runner.py` planifican una vez por escenario antes de la primera corrida, de modo que todas las semillas y brazos de un lote vuelan el mismo plan, y cada `summary.json` registra el plan con el que voló (`route_plan`). `--route-plan off` vuela el manifiesto tal cual.

Por qué parches. Juzgar la ruta entera no funciona con los modelos pequeños. En el banco sobre `citysim_pilot`, ninguno de los dos modelos candidatos (cap. 8 §8.1) distinguió una candidata que va por la calle de una que cruza el techo de un edificio:

- **Con la línea sobre el mapa:** Qwen2.5-VL-3B rechazó las 37 candidatas, y LFM2.5-VL-1.6B aprobó todas.
- **Con el pasillo:** ambos aprobaron todas.
- **Con la pregunta invertida** («¿el pasillo cruza algún techo?»): ambos respondieron «sí» a todas, incluida la recta que va por la calle.

La respuesta depende de la forma de la pregunta y no de la imagen. Los parches le piden al modelo lo que sí hace bien, reconocer qué hay en una imagen, en lugar de juzgar una ruta entera.

Uso en los experimentos. El plan está activado (`ROUTE_PLAN_MODE=vlm`, `ROUTE_JUDGE=patches`) en TownSim y CitySim, sobre sus mapas registrados (§4.3.2). En el banco de prueba la puntuación separa las rutas libres de las bloqueadas con AUC 0.81, y en el único tramo de `citysim_pilot` con la recta bloqueada el plan elige un desvío libre (cap. 11 §11.0). Es la única tarea del sistema en la que el VLM aporta una señal medible. MiniSim vuela con `--route-plan off` (cap. 10 §10.5.3): su nivel es un plano sin obstáculos, de modo que la recta entre waypoints ya es la ruta óptima y no hay mapa que juzgar.

4.3.2 Mapa cenital registrado

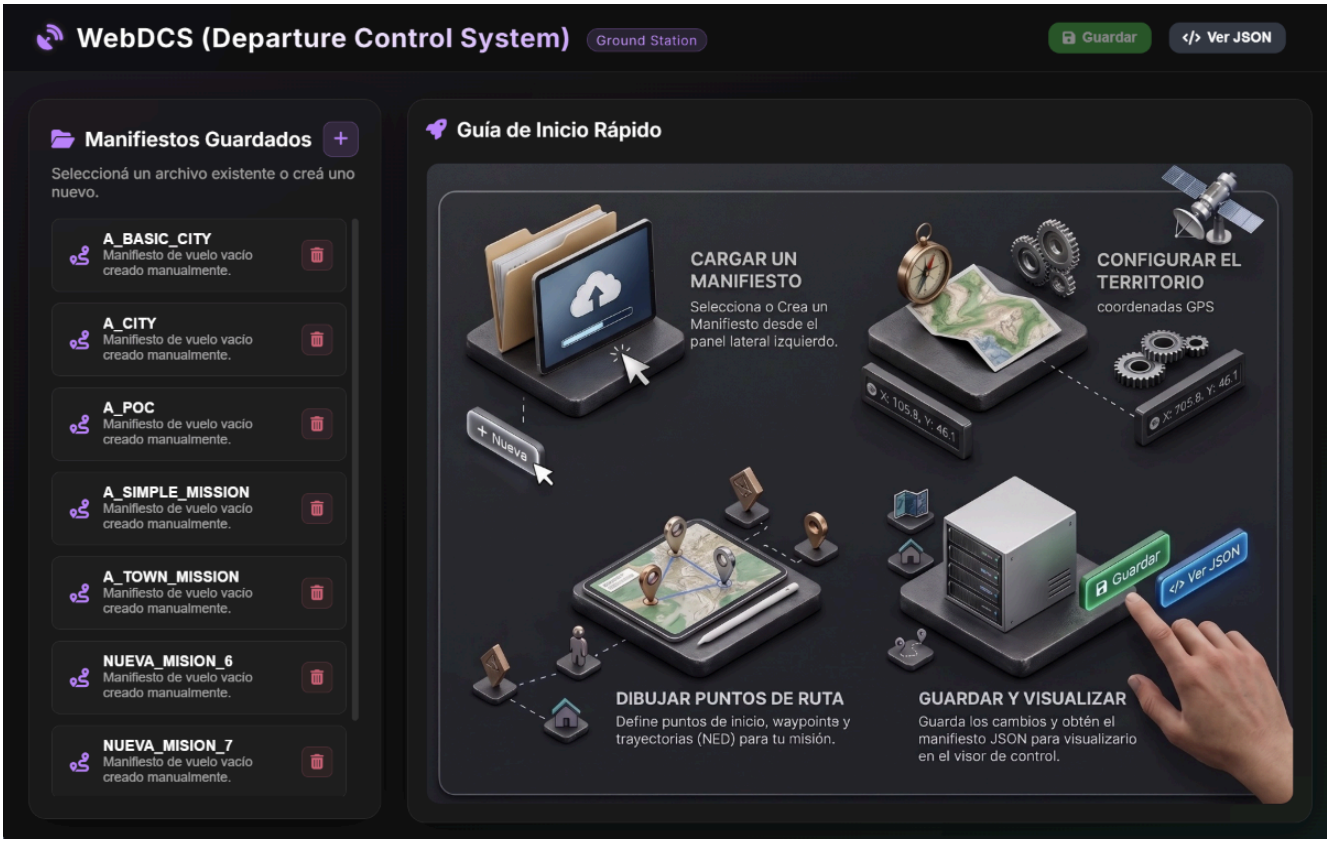
El plan de ruta solo tiene sentido si el mapa coincide con el mundo de vuelo. Los mapas de CitySim y TownSim se construyen con el propio simulador (`airsim-plan/scripts/capture_ortho_map.py`): el dron se reubica sobre una grilla de puntos a 1000 m de altura con la cámara frontal apuntando 90° hacia abajo y un campo visual de 15°, de modo que la vista es casi ortográfica. De cada toma se usa la mitad central y se la pega en el lienzo a 2 px/m en CitySim y 4 px/m en TownSim, con el origen NED en el centro de la imagen, el norte arriba y el este a la derecha. La orientación y la escala se verifican en el arranque desplazando el dron 20 m al norte y 20 m al este y midiendo el corrimiento de la imagen por correlación de fase.

Unreal Engine carga el escenario por zonas alrededor del punto de vista, y desde 1000 m no carga la zona de abajo: la imagen muestra solo calles y bases vacías. Por eso, antes de cada toma el dron baja a 40 m sobre el punto (la carga se completa en unos 3 s), vuelve a subir y captura enseguida. La toma se acepta cuando la fracción de píxeles de edificio, medida con la profundidad, deja de crecer entre dos capturas; un criterio por color no sirve, porque el agua animada nunca se estabiliza y una zona sin cargar está quieta. El resultado, `citysim_ortho.png`, cubre la ciudad completa (1920 m norte-sur por 1640 m este-oeste) y es el mapa de todas las misiones de CitySim. El de TownSim, `townsim_ortho.png` (24 tomas, 4 px/m), cubre 500 × 500 m alrededor del origen; superpuestas las trayectorias voladas, ninguna muestra de casi-contacto cae sobre pasto sin obstáculo. Las demás imágenes de esos niveles (`citymap.png`, `citysim_calib.png`, `townsim.png`, `townsim_calib.png`) no están registradas con el mundo y `airsim-plan/missions/maps/map_scales.json` las marca así.

En el marco de la metodología experimental de esta tesis (capítulo 10), el uso de manifiestos estructurados garantiza la **reproducibilidad experimental**: los escenarios de benchmark por Tiers (`minisim_clear`, `townsim_ini`, `citysim_clear`) se definen a través de manifiestos fijos e inmutables, asegurando que las comparaciones de rendimiento entre brazos de control (SLM, FSM y reactivo) se inicien exactamente con las mismas metas cinemáticas y espaciales.

4.4 Estación Terrena WebDCS: planificación y auditoría post-vuelo

WebDCS (*Web-based Drone Control Station*) es la interfaz de control en tierra, construida sobre **FastAPI** con frontend en HTML5, CSS y JavaScript. Expone su API REST en `http://localhost:8000` y sirve el panel de operación como aplicación web estática.



Funcionalidades de WebDCS:

1. **Gestión interactiva de cartas y waypoints:** El panel incorpora un visor Canvas 2D que superpone la trayectoria del manifiesto sobre la carta del entorno urbano (`CitySim`). El operador puede cargar manifiestos existentes desde `airsim-plan/missions/flightplans/`, visualizar la ruta proyectada y editar waypoints interactivamente sobre la carta — agregar, reordenar, eliminar y ajustar altitud — antes de guardar o lanzar.
2. **Lanzamiento y detención de misión:** El botón *Lanzar* invoca `POST /api/launch`, que persiste el manifiesto y arranca `airsim-loop` en un hilo de fondo sin bloquear la interfaz. `POST /api/stop` detiene todos los runners activos. `POST /api/reset` reinicia el simulador AirSim y libera el control de la API.

Dataset de corridas: estructura de `airsim-runs/`

Cada ejecución de una misión escribe una carpeta autocontenida en `airsim-runs/`. Desde `main.py` (WebDCS), el nombre de la carpeta y el prefijo común (`<stem>`) de sus archivos es `<MISSION_ID>-<YYYYMMDDTHHMMSSZ>`: el `mission_id` del manifiesto más la hora de inicio en UTC, en ISO 8601 compacto (por ejemplo, `CITYSIM_CLEAR-20260907T191226Z`). El ejecutor de experimentos (`experiments/runner.py`) ubica cada corrida en una jerarquía por celda experimental, `<escenario>/<brazo>/<celda>/seed_<N>_<timestamp>/`, con `<stem>` = `seed_<N>_<timestamp>`; la celda es la estrategia de deadlock (`depth`, `deep_vlm`, `blind`) o, si el lote varía la fuente de la capa estratégica, `strat_<fuente>__<estrategia>` (cap. 10 §10.9.1). En ningún caso una corrida sobrescribe a otra.

Archivos de una corrida

Archivo	Lo escribe	Contenido
`<stem>.jsonl`	`FlightLogger`, ciclo a ciclo	Traza completa: un objeto JSON por ciclo, con el `DroneState` serializado
`<stem>.csv`	`FlightLogger`	La misma traza en columnas, más el `DroneState` aplanado en columnas `state.<ruta>`
`<stem>.summary.json`	`FlightLogger.close()` y `finalize_run`	Indicadores de la corrida: éxito, colisiones, DistMin, recorrido, consultas, deadlocks, plan de ruta, versión del código
`<stem>.summary_by_wp.csv`	`FlightLogger.close()`	Ciclos, duración, ciclos de barrido y eventos de barrido por objetivo
`<stem>.distmin.ndjson`	hilo de auditoría (`distmin_audit.py`)	Muestras de distancia mínima a obstáculo, una por segundo
`<stem>.console.log`	runner	Copia de la salida de consola de la corrida (stdout y stderr)
`<stem>.webm`	`FlightVideoRecorder`	Video de la cámara frontal, un cuadro por ciclo, con overlay
`<stem>.follow.webm`	`follow_cam.py`	Video de la cámara externa `FollowCam` (si está definida en `settings.json`), un cuadro por ciclo
`<stem>.viewer.html`	`flight_viewer.py`	Visor de auditoría autocontenido
`photo-<ISO_UTC>.png`	`FlightLogger`	Fotogramas de cada consulta del lazo lento (capa estratégica y barrido), a resolución completa

Además, a nivel de escenario, el runner deja `route_plan/` con el plan de ruta y sus imágenes (cap. 4 §4.3.1), y `batch_runner.py` agrega en la raíz `RESULTS_SUMMARY.json` (una fila por corrida) y `RESULTS_SUMMARY.batch_version.txt`.

Traza por ciclo (`.jsonl`). Cada línea es un ciclo del lazo y tiene estos campos:

Campo	Contenido
`t`, `cycle`	Segundos desde el inicio de la corrida y número de ciclo
`arm`, `scenario`, `seed`	Celda de la corrida
`pos`, `vel`, `yaw_deg`, `pitch_deg`, `roll_deg`	Posición NED, velocidad (`vx`, `vy`, `vz`) y actitud

Campo	Contenido
`route`, `action`	Ruta que decidió el ciclo y macro-acción emitida (cap. 5 §5.3)
`obstacle_field`	`ObstacleField` completo: por sector, ocupación, TTC, confianza y bloqueo; fuente (`flow`, `holdover`, `degraded`) y confianza del FOE (cap. 6)
`latency_ms`	Duración del ciclo del grafo (`graph`) y de la lectura de telemetría (`telemetry`)
`slm`	Presente en los ciclos con una consulta del lazo lento: identificador, latencia, *fallback*, *timeout*, adherencia al esquema, prompt (vacío con la profundidad), respuesta cruda (con la profundidad, el p5 estimado por sector en metros) y rutas de los fotogramas (`frame_paths`)
`collision`	Colisión y objeto contra el que ocurrió
`wp_index`, `dist_to_wp_m`	Objetivo activo y distancia horizontal a él
`degraded`	AirSim no respondió en el ciclo (cap. 5 §5.5)
`deadlock_event`	Solo en el ciclo en que un deadlock se resuelve o cae al escape: estrategia, si lo resolvió el barrido, ciclos que llevó, si cayó al escape vertical, la macro, la respuesta cruda y la selección del rumbo de salida (`scan_selection`: candidatos, descartados con su motivo, rumbo que falló, modo y rumbo elegido; cap. 5 §5.12)
`perception`	Señales del `StallDetector` (contacto por IMU, avance sin movimiento, nivel de vibración, ciclos detenido)
`state`	El `DroneState` completo del ciclo
`min_obstacle_dist_m`	Siempre vacío: la distancia mínima la mide la auditoría aparte (`.distmin.ndjson`)

El campo `state` hace que la traza no dependa de haber previsto qué registrar: guarda todas las claves del estado (cap. 5 §5.2) con reglas fijas (`state_serializer.py`). Las imágenes y los búferes de fotogramas se reemplazan por su forma (`<ndarray shape=(720, 1080, 3)>`); `deliberations` se resume en su cantidad y la última entrada, sin prompts (que ya están en el bloque `slm`); los textos se cortan a 600 caracteres, las listas a 40 elementos y el anidamiento a 6 niveles, y los valores no finitos se escriben como texto (`"inf"`, `"nan"`) para que cada línea sea JSON válido. La serialización nunca detiene el vuelo: si falla, el ciclo registra el error en lugar del estado.

Traza plana (`.csv`). Durante el vuelo se escribe en *streaming* con un conjunto fijo de columnas, que sirve de respaldo si el proceso muere: identificación (`t`, `cycle`, `arm`, `scenario`, `seed`, `route`, `action`, `wp_index`, `dist_to_wp_m`, `degraded`), cinemática (`pos_*`, `vel_*`, actitud), colisión, latencias, la consulta del ciclo (`slm_invoked`, `slm_latency_ms`, `slm_fallback`, `slm_timeout`, `slm_adherent`, `slm_prompt`, `slm_raw_response`, `slm_delib_id`, `slm_frame_paths`), estado de control (`ctrl_*`: contador de atasco, maniobra comprometida y ciclos restantes, señales del `StallDetector`, ciclos sin progreso) y el campo de obstáculos por sector (`field_<sector>_occ`, `_ttc_s`, `_conf`, `_blocked`, `field_source`). Algunas columnas `ctrl_*` (`ctrl_depth_m`, `ctrl_traj_*`, `ctrl_stuck_invisible`) quedan siempre vacías: el grafo no las llena, pero forman parte del esquema fijo de columnas que esperan las herramientas de análisis. Al cerrar la corrida, el CSV se reescribe agregando el `DroneState` aplanado: una columna `state.<ruta>` por cada valor escalar del árbol (por ejemplo `state._vlm_strategic.outcome`). El prompt y la respuesta completos están en el CSV, y no solo en el JSONL, para poder analizar las consultas sin cruzar archivos.

Resumen de corrida (`.summary.json`). Lo consumen `analyze_lotes.py` (cap. 10 §10.7), `analyze.py`, `analyze_tesis_results.py` y `analyze_strategic_batch.py`. Campos:

- **Identificación:** `scenario`, `seed`, `arm`, `code_version` (hash del commit, con sufijo `-dirty` si había cambios sin confirmar; cap. 10 §10.9.3), `deadlock_strategy`, `strategic_source` y `route_plan` (modo y plan con el que voló).
- **Misión:** `success`, `cycles`, `duration_s`, `path_length_m` y, cuando corresponde, `termination_reason` (por ejemplo `physics_locked`) y `freeze_recoveries`.
- **Seguridad:** `collisions` y DistMin (`min_obstacle_dist_m`, con `min_obstacle_dist_t`, `min_obstacle_dist_source` y `min_obstacle_dist_samples`).
- **Lazo lento:** `slm_invocations` (consultas nuevas, contadas una vez por identificador), `deliberation_rate`, `slm_fallback_rate`, `slm_timeout_rate`.
- **Rutas:** `route_histogram`, ciclos por ruta.
- **Atascos:** `deadlock_events`, `deep_scan_events`, `deep_scan_resolution_rate`, `deep_scan_avg_cycles_to_resolve`, `deep_scan_fallback_rate`, `deep_scan_futile_rate`, `vertical_escapes`.

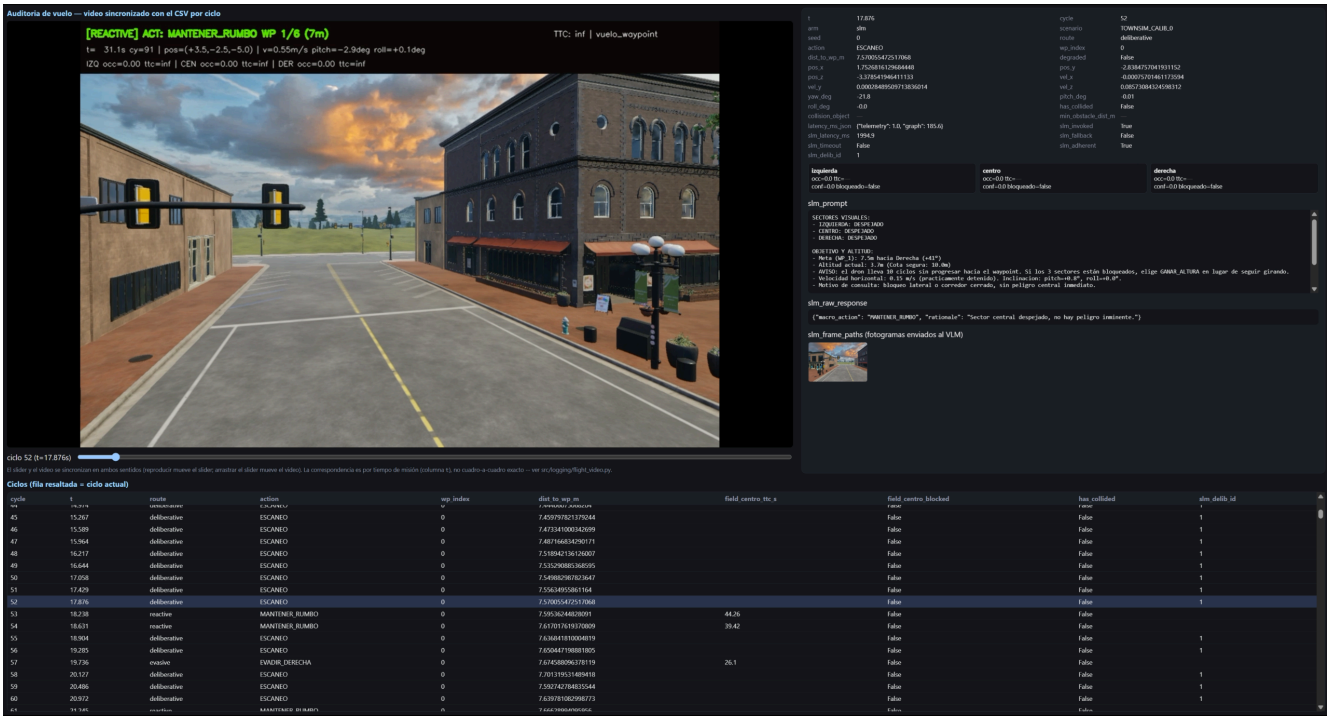
Resumen por objetivo (`.summary_by_wp.csv`). Columnas `wp_index`, `wp_label`, `cycles`, `duration_s`, `deliberative_cycles`, `deliberative_rate` y `deep_scan_events`. El índice cuenta los objetivos tal como los recorrió el tracker, incluidas las sub-metas insertadas, y la etiqueta es la del waypoint de misión al que pertenecen: varias filas pueden tener la misma etiqueta. Sirve para ver en qué tramo se concentra la dificultad sin abrir la traza.

Auditoría de DistMin (`.distmin.ndjson`). Una línea por muestra, cada `DISTMIN_TRACK_PERIOD_S` (1 s): `t` (hora epoch), `dist_m` (percentil 5 del tercio central de la profundidad de la cámara frontal) y la posición `x`, `y`, `z` en el instante de la muestra. La escribe un hilo aislado, con su propia conexión al simulador, que no comparte nada con el grafo (cap. 5 §5.16); `finalize_run` consolida el mínimo en el resumen. Como cada muestra lleva su posición, se puede calcular DistMin por tramo; su `t` es hora epoch, distinta del `t` relativo de la traza.

Fotogramas (`photo-<ISO>.png`). Cada consulta del lazo lento guarda el fotograma crudo que le dio origen, a resolución completa (no la versión reducida que viaja en el pedido), nombrado con su hora de captura en UTC con milisegundos; un barrido guarda sus cuatro imágenes. La columna `slm_frame_paths` de la traza enlaza cada consulta con sus archivos. Con la capa estratégica de profundidad hay del orden de un fotograma por segundo de vuelo.

Cierre y reparación (`finalize_run`). Es el último paso de toda corrida del runner y es idempotente: consolida DistMin en el resumen y, si el proceso murió antes de cerrar, reconstruye el CSV con las columnas `state.*` a partir del JSONL (que se vacía a disco ciclo a ciclo), repara el índice de los videos y regenera el visor. Una corrida interrumpida se cierra a mano con `python experiments/finalize_run.py <carpeta>` (o `--all <raíz>` para todas las incompletas).

Visor de auditoría (`.viewer.html`)



El visor es un único archivo HTML que se genera al cerrar cada corrida. Lleva la traza embebida como JSON dentro de la página, de modo que se abre con doble clic, sin servidor: un navegador bloquea, desde `file://`, la lectura de otro archivo local con `fetch`, pero sí reproduce un video de la misma carpeta. Tiene cuatro partes:

1. **Videos sincronizados.** El video frontal (lo que ve el grafo) y, si existe, el de la cámara externa `FollowCam`, uno al lado del otro. El externo sigue al frontal en reproducción, pausa, velocidad y saltos. Los dos tienen un cuadro por ciclo, de modo que la correspondencia entre video y traza es por número de ciclo y no por tiempo: el período real del lazo varía y el tiempo derivaría.
2. **Barra de ciclos.** Un control deslizante sobre todos los ciclos, sincronizado en ambos sentidos con el video, con el ciclo y el tiempo actuales.

3. **Panel de detalle del ciclo.** Todas las columnas del CSV del ciclo; los tres sectores del `ObstacleField` como recuadros (ocupación, TTC, confianza, resaltados si están bloqueados); el prompt y la respuesta cruda si en ese ciclo llegó una consulta; las imágenes exactas de esa consulta, y el `DroneState` completo como árbol desplegable.
4. **Tabla de ciclos.** Una fila por ciclo con ciclo, tiempo, ruta, acción, objetivo, distancia, TTC y bloqueo del sector central, colisión e identificador de consulta. La fila actual se resalta y se centra al avanzar el video; un clic en una fila lleva el video a ese ciclo.

El uso típico es partir de un evento del resumen —un deadlock, una colisión, la muestra de DistMin más baja— y llevar el visor a ese ciclo: el video muestra la escena, el panel muestra qué percibió el sistema y qué decidió, y el árbol del estado muestra por qué (contadores de atasco, sub-meta activa, resultado de la capa estratégica). Así se auditaron los modos de falla del capítulo 9.

Anotaciones del video

El video (`.webm`) registra un fotograma por ciclo del lazo, a la misma cadencia de `LOOP_HZ`. Cada fotograma lleva un overlay de cuatro líneas superpuesto **sobre una copia del fotograma**; el original, que usan la percepción y la capa estratégica y que se guarda como `.png` de auditoría, queda sin modificar para no contaminar la evidencia perceptual.

Línea 1 — ruta activa y macro-acción (texto grande, color codificado):

```
[KEEP_GOING] ACT: MANTENER_RUMBO  WP 3/7 (42m)
```

- El tag entre corchetes es la **ruta** (`route`) del comportamiento que resolvió el ciclo (cap. 5 §5.3): `REACTIVE` (crucero sin novedad, vuelo bajo el piso óptico o descenso bajo un techo), `EVASIVE` (evasión reactiva o continuación de una maniobra comprometida), `GIRAR_90` (giro ante un frente bloqueado), `DELIBERATIVE` (barrido de resolución de atasco en curso o resuelto), `TACTICAL` (escape vertical cuando el barrido no resolvió o con `blind`), `FSM` (brazo FSM) o `DEGRADED` (modo degradado, cap. 5 §5.5).
- `ACT:` es la **macro-acción** traducida a comando cinemático por el nodo `motor`.
- `WP X/N (Ym)` indica el objetivo activo y la distancia restante según `WaypointTracker` (cap. 5 §5.15). `N` cuenta también las sub-metas insertadas en vuelo, de modo que crece durante la misión.
- **Color:** verde para `MANTENER_RUMBO`, naranja para maniobras de evasión, giro o escape vertical, rosa para `FRENAR`.

Línea 2 — TTC frontal y estado de vuelo (derecha del fotograma):

```
TTC: 3.2s | vuelo
```

- `TTC` es el tiempo estimado a colisión del sector frontal según el `ObstacleField` del ciclo (cap. 6); `inf` significa sector despejado.
- El campo de texto refleja el estado general de la misión (`vuelo`, `completada`, `colisión`).

Línea 3 — telemetría del ciclo:

```
t= 18.4s cy=92 | pos=(+26.1,-78.2,-10.0) | v=2.31m/s pitch=-3.2deg roll=+0.8deg
```

- `t` y `cy` permiten cruzar el fotograma con la fila exacta del CSV por número de ciclo (la sincronización por tiempo es aproximada, la de ciclo es exacta; ver `flight_video.py`).
- `pos` es la posición 3D en el marco NED del simulador; si el guiado detectó un techo (§5.15.3), se agrega `ceil=<cota>` a continuación.
- `v`, `pitch`, `roll` son la velocidad horizontal y la actitud del vehículo en ese instante.

Línea 4 — estado del ObstacleField por sector (cap. 6):

```
C occ=0.62 ttc=3.2s! | I occ=0.11 ttc=inf | D occ=0.08 ttc=inf
```

- Los tres sectores son `C` (centro / frente), `I` (izquierda) y `D` (derecha).
- `occ` es la fracción de ocupación estimada por flujo óptico monocular; `ttc` es el tiempo a colisión estimado del sector; `!` marca el sector como bloqueado según el umbral de `is_blocked()`.
- Estos valores son los mismos que usa el lazo rápido de `navigate` (cap. 5 §5.3): el overlay permite ver, ciclo a ciclo, qué evidencia perceptual motivó la decisión visible en la línea 1. La capa estratégica no los recibe: mira el fotograma directamente (cap. 5 §5.10).

La combinación de estas cuatro líneas permite reconstruir, para cualquier instante del vuelo, exactamente qué percibió el sistema, por qué se activó el comportamiento que aparece en el tag, y qué orden cinemática emitió — sin necesidad de cruzar a mano el video con el CSV.

5. Arquitectura del lazo táctico

5.0 Escaneo espacial pre-vuelo (`spatial_scan`)

Antes de entrar al lazo, `main.py` llama de forma bloqueante a `run_initial_scan()` (`src/agents/spatial_scan.py`). No es un nodo del `StateGraph`: se ejecuta una sola vez, entre la conexión con AirSim y la compilación del grafo. Realiza un barrido de yaw en el lugar hacia `SCAN_HEADING_COUNT` (6) rumbos equiespaciados, captura un fotograma por rumbo y envía todas las imágenes en una única consulta al VLM, que devuelve un contexto cualitativo de la escena y un rumbo recomendado.

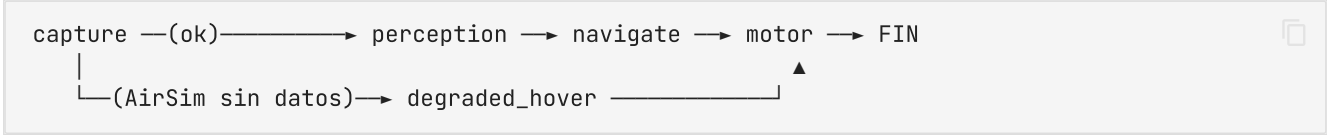
Con la configuración por defecto es la única consulta al VLM que hace `main.py`, y el resultado es **solo un chequeo de cordura del plan de misión**: si el rumbo recomendado difiere más de 90° del rumbo al primer waypoint, `main.py` emite un aviso por consola. No se transfiere al grafo ni modifica el vuelo. El ejecutor de experimentos (`experiments/runner.py`) **no** lo ejecuta, de modo que no interviene en ninguna corrida experimental. Si el VLM no responde, la misión arranca igual.

5.1 Visión general del lazo

El lazo se ejecuta sobre el entorno de simulación del capítulo 3 (Unreal Engine 5.5 con Cosys-AirSim), siguiendo los manifiestos compilados por la estación terrena (capítulo 4). Es un **grafo de decisión por ciclo** (`StateGraph` de LangGraph, LangChain, 2024) que corre a `LOOP_HZ` = 5 Hz (ciclo de ≈ 0.2 s).

El grafo se compila **una vez** y el bucle externo (`main.py` o `experiments/runner.py`) lo invoca con `graph.invoke()` en cada ciclo. El grafo es acíclico: la naturaleza cíclica de la navegación la aporta el bucle externo. Esta distinción importa porque LangGraph construye los canales del estado a partir del `TypedDict` declarado (`DroneState`, §5.2): una clave que un nodo escriba sin estar declarada se **descarta en silencio** al terminar `graph.invoke()`. Esa propiedad es la causa raíz de varios modos de falla del capítulo 9.

5.1.1 Topología: cuatro nodos y una arista condicional



`navigate` es un **único nodo** que contiene la política. Los comportamientos concretos (guiado, evasión lateral, giro de 90°, escape vertical, barrido panorámico) son funciones que `navigate` invoca, no nodos del grafo.

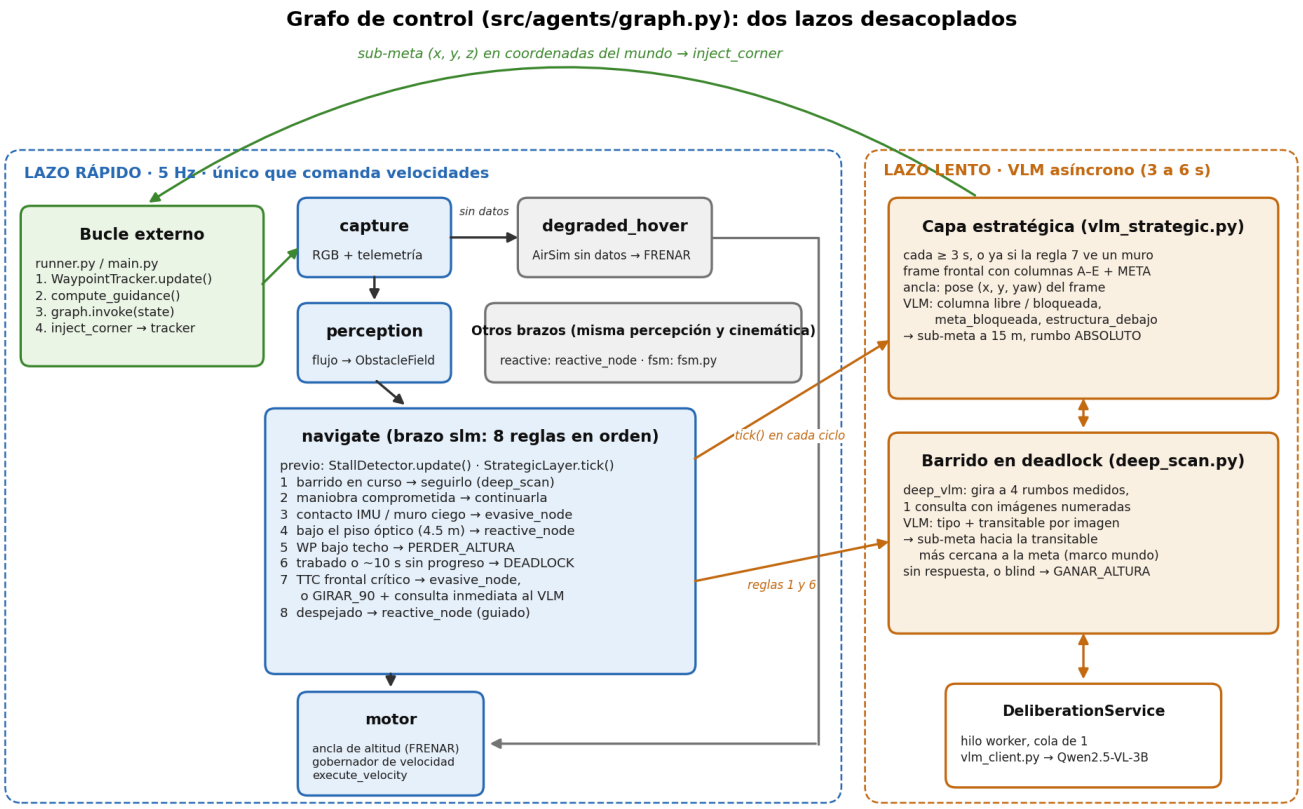


Figura 5.1. Grafo de control. El lazo rápido (izquierda) es el único que comanda velocidades; el lazo lento (derecha) consulta de forma asíncrona a su fuente —por defecto, la red de profundidad estimada (§5.10.6); la figura muestra la alternativa con el VLM— y devuelve sub-metas en coordenadas del mundo, que el bucle externo inserta en el `WaypointTracker`.

5.1.2 Dos lazos desacoplados

La organización responde a un hecho medido: la consulta al VLM local tarda ~0.8 s (§8.6), cuatro ciclos del lazo, y la de la red de profundidad ~0.1 s, medio ciclo. Cualquier respuesta expresada **en el marco del cuerpo** del dron («el frente está libre», «evadir a la izquierda») describe un fotograma que ya no existe cuando la respuesta llega: el dron giró y avanzó varios metros. El capítulo 9 (§9.8) documenta las fallas que produce usar una respuesta así: maniobras hacia el lado equivocado y respuestas que llegan tarde y se descartan.

El sistema separa entonces dos escalas de tiempo:

1. **Lazo rápido (5 Hz).** Guiado al waypoint, evasión por flujo óptico, reacción a contacto y detección de atasco. Es el **único** que comanda velocidades y el responsable de todo lo inminente.
2. **Lazo lento (asíncrono).** Una fuente de percepción mira un fotograma y responde una pregunta cuya respuesta es **válida en el mundo**, no en el cuerpo: por cuál dirección se puede volar hacia la meta. Cada pedido guarda la pose del fotograma (ancla); la respuesta se traduce a una **sub-meta en coordenadas del mundo** con esa pose, de modo que sigue siendo correcta aunque el dron haya girado mientras la fuente procesaba (§5.10). La fuente es configurable (`STRATEGIC_SOURCE`): por

defecto, una red de profundidad monocular estimada desde el mismo RGB (§5.10.6); la alternativa es el VLM, que en el banco de prueba no separa sectores libres de bloqueados (cap. 11, §11.0).

El patrón es el de las arquitecturas de capas para robots móviles, con un controlador reactivo de respuesta inmediata y una capa deliberativa que opera a otra escala temporal (Gat, 1998), en la que las capas inferiores conservan siempre la capacidad de actuar (Brooks, 1986). La particularidad es la interfaz entre ambas: la capa lenta no emite acciones sino **metas geométricas ancladas**, que la capa rápida persigue con su guiado normal.

Tres principios de diseño acompañan esta separación:

- **La fuente decide; las reglas no la reescriben.** No hay reglas deterministas que corrijan la respuesta de la fuente: una capa de reglas sobre su decisión haría que la comparación entre brazos midiera esas reglas y no a la fuente (cap. 9, §9.8.6). El fallback determinista existe solo ante **falla**: timeout, formato inválido o rotación imposible durante un barrido.
- **El dron nunca espera al lazo lento en vuelo normal.** La capa estratégica no frena al vehículo; solo el barrido de resolución de atasco (§5.12) mantiene al dron en el lugar, porque ese es justamente el momento en que avanzar ya no es posible.
- **Sin profundidad del simulador.** Ni el grafo, ni `main.py`, ni `experiments/runner.py` leen el canal de profundidad de AirSim, ni para control ni como métrica (§5.16). La profundidad **estimada** desde el RGB de la cámara (§5.10.6) sí se usa: equivale a una red a bordo de un dron sin sensor de profundidad.

5.2 El estado compartido: `DroneState`

`DroneState` (`graph.py`) es el único canal de comunicación entre nodos y entre el grafo y el bucle externo.

Datos sensoriales y percepción:

Campo	Tipo	Descripción
`rgb_image` / `prev_image`	`ndarray`	Fotograma del ciclo actual y del anterior (el par que usa el flujo óptico)
`frame_history` / `frame_history_ts`	`List`	Ring buffer de `VLM_FRAME_HISTORY_SIZE` fotogramas (1 en producción) con sus timestamps de captura
`telemetry` / `prev_telemetry`	`Dict`	Posición NED, velocidad, orientación, colisión, `landed_state`, timestamp y `source`
`degraded`	`bool`	`True` si AirSim no respondió este ciclo
`obstacle_field`	`Obstacle-Field`	Contrato único de percepción (cap. 6)
`estimated_ttc` / `scene_summary`	`float` / `str`	Derivados del campo, para logging y overlay

Misión y guiado (los escribe el bucle externo): `waypoints`, `current_wp_index`, `target_waypoint`, `waypoint_guidance` (salida de `WaypointTracker.compute_guidance()`), `mission_completed`, `evasion_stuck_cycles` (contador de progreso del tracker; lo usa el brazo `fsm`).

Decisión y actuación:

Campo	Tipo	Descripción
`next_action`	`str`	Macro-acción del ciclo
`velocity_command`	`Dict`	`{macro_action, vx, vy, vz, yaw_rate, target_yaw, rationale}` para `motor_node`
`route`	`str`	Origen del comando: `reactive`, `evasive` (incluye continuación de maniobra), `girar_90`, `tactical` (escape de deadlock sin VLM), `deliberative` (barrido), `fsm`, `degraded`
`flight_status`	`str`	Estado textual para logging y overlay
`active_maneuver` / `maneuver_cycles_left` / `maneuver_command`	—	Maniobra comprometida (anti flip-flop)
`_hover_alt_anchor` / `_speed_cap`	`float|None`	Ancla de altitud en `FRENAR` y tope del gobernador de velocidad (§5.13)

Lazo lento y canales hacia el bucle externo:

Campo	Tipo	Descripción
`deliberations`	`List[Dict]`	Registro de solo agregado de todas las consultas del lazo lento, a la profundidad o al VLM (estratégicas, de barrido y fallidas)
`last_deliberation`	`Dict|None`	Última entrada, actualizada por `motor_node`
`_vlm_strategic`	`Dict|None`	Último resultado de la capa estratégica: `outcome`, motivo, latencia, sector de la meta, ancla y sub-meta (§5.10)
`_clear_subgoals`	`bool`	La capa estratégica vio libre el camino directo al waypoint real: el bucle externo descarta la sub-meta pendiente (§5.15.4)
`inject_corner`	`Dict|None`	Sub-meta `{x, y, z, label}` en coordenadas del mundo que el bucle externo inserta en el tracker (§5.15.4)
`_deliberation_pending`	`bool`	`True` durante un barrido: el bucle externo no cuenta esos ciclos como falta de progreso

Campo	Tipo	Descripción
`_pending_delib_prompt` / `_pending_delib_frames` / `_last_delib_frames`	—	Auditoría: prompt y fotogramas enviados; `_last_delib_frames` es un canal de una sola pasada hacia `FlightLogger`

Deadlock y barrido panorámico:

Campo	Tipo	Descripción
`_escape_reset`	`bool`	Señal de flanco: el bucle externo llama a `WaypointTracker.reset_progress()` en el ciclo en que un deadlock se resuelve
`_deadlock_cycles`	`int`	Ciclos dentro del deadlock en curso
`_deadlock_event`	`Dict|None`	Métricas de resolución (estrategia, resuelto por el barrido o no, respuesta cruda, sub-meta), consumidas por `FlightLogger`
`_scan_phase`	`str|None`	`rotando` | `asentando` | `capturado` | `None`
`_scan_heading_index` / `_scan_start_yaw_deg` / `_scan_settle_left` / `_scan_rot_stall`	—	Progreso del barrido
`_scan_frames`	`List`	`[(rumbo_medido_deg, frame, capture_ts)]`
`_deep_scan_request_id` / `_deep_scan_request_ts` / `_scan_started_ts`	—	Pedido del barrido en vuelo

Señales del `StallDetector` (publicadas solo para logging, §5.3.1): `imu_jitter_level`, `imu_contact_event`, `blind_wall_event`, `_stopped_cycles`, `_wp_no_progress_cycles`. El bucle externo agrega `_freeze_cycles` (vigilante de congelamiento físico, §5.19).

Regla de diseño crítica. Todo campo que cruce la frontera entre invocaciones de `graph.invoke()` debe estar declarado en este `TypedDict`: LangGraph no lanza error si un nodo escribe una clave no declarada, la descarta en silencio (cap. 9, §9.5). Dos decisiones reducen esa superficie: los contadores de atasco y el estado de la capa estratégica viven en objetos de proceso (`StallDetector`, `StrategicLayer`) y no en el estado, y los tests de integración corren el **grafo compilado** y verifican que las claves de control (`inject_corner`, `_escape_reset`, `_deadlock_event`, `_vlm_strategic`) cruzan la frontera (`tests/test_graph_integration.py`, `tests/test_vlm_strategic_graph.py`).

5.3 Nodo `navigate`

`navigate_node` (`graph.py`) recibe el estado enriquecido por `perception` y produce un `velocity_command`. Durante el despegue vertical (§5.15), los tres brazos delegan en `reactive_node`: el guiado sube en el lugar y el flujo óptico de un ascenso puro no es válido para evadir. Fuera del despegue, para los brazos de comparación delega de inmediato: `AGENT_ARM = reactive` → `reactive_node`; `AGENT_ARM = fsm` → `fsm_node` (§5.11). Para el brazo `slm`, ejecuta dos pasos previos en cada ciclo y luego una cascada de ocho reglas en orden estricto: **la primera que se cumple decide el ciclo**.

5.3.1 `StallDetector`: señales de atasco

Los contadores de atasco viven en un objeto `StallDetector` (`src/agents/stall_detector.py`) que persiste entre invocaciones. En cada ciclo, `update(state)` actualiza cuatro señales y `publish(state)` las escribe en el estado solo para logging.

Señal	Condición	Umbral	Uso en `navigate`
`imu_contact`	RMS de la aceleración lateral ≥ `IMU_CONTACT_THRESHOLD_MPS2` (5 m/s²) con velocidad comandada > 0.3 m/s	≥ 2 ciclos	Regla 4: evasión inmediata
`blind_wall`	Avance comandado (`vx ≥ 0.45 m/s`) con velocidad medida < 0.30 m/s, campo óptico con corredor libre (`blocked_fraction < 0.25`) y el dron moviéndose en el ciclo previo	≥ 2 ciclos	Regla 4: evasión inmediata
`stopped_prolonged`	**Se ordenó avanzar** (velocidad horizontal comandada ≥ `STOPPED_MIN_CMD_MPS` = 0.30 m/s) y la velocidad medida es < 0.10 m/s; no cuenta bajo el piso óptico ni durante el despegue	≥ `STOPPED_DELIBERATIVE_CYCLES` = 10 ciclos	Regla 2: deadlock (trabado)
`wp_no_progress`	La distancia al objetivo actual no mejora `WP_NO_PROGRESS_MIN_M` (2 m); se reinicia al cambiar el objetivo, durante un barrido, con `_escape_reset`, bajo el piso óptico y durante el despegue	≥ `WP_NO_PROGRESS_THRESHOLD` = 50 ciclos (10 s)	Regla 2: deadlock (sin progreso)

Tres decisiones de diseño:

- **«Detenido» significa «se le ordenó moverse y no se movió».** Durante el despegue, el guiado ordena subir y girar en el lugar hacia el primer waypoint con `vx = 0` durante ~3 s; un contador que sumara cualquier ciclo sin velocidad horizontal declararía un deadlock al cruzar el piso óptico (cap. 9, §9.8.4). Girar, subir o mantener un hover a propósito no es estar trabado.
- **La velocidad se mide por desplazamiento, no se lee de la telemetría.** Con el dron apoyado contra una estructura, AirSim reporta una velocidad cercana a la comandada aunque la posición no cambie (cap. 9, §9.8.8). `stopped_prolonged` y `blind_wall` usan la velocidad horizontal calculada con la diferencia de posición entre ciclos consecutivos.

- **El objetivo se identifica por su etiqueta y posición, no por su índice.** Una sub-meta del VLM se inserta en el mismo índice que el waypoint al que precede (§5.15.4); identificar el objetivo por índice heredaría la «mejor distancia» del objetivo anterior y dispararía un deadlock falso 10 s después de cada sub-meta.

5.3.2 Pasos previos y orden de evaluación

Pasos previos (cada ciclo): `StallDetector.update()` y `StrategicLayer.tick()` (capa estratégica, §5.10; no se ejecuta con `STRATEGIC_SOURCE = off`). El segundo nunca comanda velocidades: envía un pedido si corresponde, o procesa una respuesta y deposita una sub-meta en `inject_corner`.

#	Condición	Acción	Por qué
1	Barrido panorámico en curso (`_scan_phase` o pedido de barrido pendiente)	Continuar el barrido (`_deadlock_resolve` → `deep_scan_cycle`)	Un barrido es dueño del dron hasta resolver o caer por sus watchdogs (§5.12); si otra regla tomara el ciclo a mitad del barrido, éste quedaría sin completar.
2	`stopped_prolonged` o `wp_no_progress`	`_deadlock_resolve` (§5.3.4)	Dron trabado o sin acercarse al objetivo.
3	Maniobra comprometida (`active_maneuver`, `maneuver_cycles_left > 0`)	Continuarla; ruta `evasive`	Anti flip-flop: una maniobra se completa antes de aceptar otra decisión.
4	`imu_contact` o `blind_wall`	`evasive_node`	Contacto que el flujo óptico no ve.
5	Altitud < `OPTICAL_MIN_ALT_M` (4.5 m) y sin techo detectado	`reactive_node`	Aterrizaje y vuelo bajo: el flujo ve el suelo en movimiento y produce TTC falsos.
6	`z_path_blocked` y `dz > 1 m` (waypoint dentro de la banda de seguridad de un techo, §5.15.3)	`PERDER_ALTURA`; ruta `reactive`	Obstáculo vertical → desplazarse en z, simétrico a la evasión en xy.
—	Altitud < 4.5 m bajo un techo detectado	`reactive_node`	Vuelo bajo deliberado: sin evasión basada en flujo.
7	TTC central ≤ `TTC_EVASION_THRESHOLD` (3.2 s), o centro bloqueado con TTC ≤ `TTC_SAFE_THRESHOLD` (4.6 s)	Si `blocked_fraction > FOV_BLOCKED_THRESHOLD` (0.6): `GIRAR_90` hacia el lado del waypoint **y** consulta inmediata a la capa estratégica (§5.9); si no: `evasive_node`	Peligro frontal: el lazo rápido esquiva; el rodeo lo decide la capa estratégica.

#	Condición	Acción	Por qué
7b	Centro bloqueado o TTC mínimo ≤ 4.6 s	`evasive_node`	Ventana de advertencia.
8	—	`reactive_node`	Camino despejado: guiado nominal.

Los umbrales de TTC provienen de la calibración del capítulo 7. El orden tiene tres justificaciones: el barrido (1) precede a todo porque, una vez iniciado, es dueño del dron hasta resolver; el deadlock (2) precede a toda otra regla porque cualquier acción que se repita sin mover al dron —una maniobra comprometida, una evasión por contacto, un descenso hacia un techo— taparía al detector de atasco indefinidamente (cap. 9, §9.8.8), y una maniobra que no mueve al dron en 2 s no es una decisión en curso sino un atasco; y el contacto físico (4) precede al piso óptico y al TTC porque no depende del sensado visual.

5.3.3 Lazo rápido: reacción sin el lazo lento

Las reglas 3, 4, 5, 7 y 8 forman el lazo rápido: actúan en el mismo ciclo, sin esperar al lazo lento. Son compartidas en espíritu con los otros brazos —misma percepción, mismo `action_to_command()` (§5.14) — de modo que la comparación entre brazos mide quién decide los rodeos y los deadlocks, no quién reacciona mejor a un muro inminente.

5.3.4 Resolución de deadlock (`_deadlock_resolve`)

Al entrar, `_deadlock_resolve` cancela cualquier consulta estratégica en vuelo (el barrido es más reciente y su sub-meta no debe ser pisada por una respuesta más vieja) e incrementa `_deadlock_cycles`. Luego:

1. **`DEADLOCK_STRATEGY = depth` (default) o `deep_vlm`:** ejecuta un paso de `deep_scan_cycle()` (§5.12). Mientras el barrido gira, se asienta o espera el juicio de los rumbos (red de profundidad o VLM), retorna sin más. Cuando el barrido se resuelve, reinicia el `StallDetector` y el bucle externo reinicia el contador del tracker (`_escape_reset`).
2. **Sin resolución del barrido** (estrategia `blind`, el barrido no encontró un rumbo transitable, o falló por watchdog, formato inválido o rotación imposible): escape determinista `GANAR_ALTURA` durante `ESCAPE_MANEUVER_DURATION_S` (1.6 s); si la altitud ya supera `MAX_ESCAPE_ALT_M` (30 m), `GIRAR_90` hacia el waypoint. Se registra un `_deadlock_event` con `fell_back_to_blind = True`.

`GANAR_ALTURA` es el escape determinista porque es la maniobra que libera al dron de una estructura en la que quedó apoyado (una cornisa, en `citysim_pilot`); un retroceso (`RETROCEDER`), en cambio, tiene un avance mediano negativo hacia el waypoint (cap. 9, §9.8.6). Por eso el lazo no emite `RETROCEDER`.

5.4 Nodo `capture`

Salidas: `rgb_image`, `telemetry`, `degraded`, `frame_history`, `frame_history_ts`, `prev_image`, `prev_telemetry`.

Copia el par imagen–telemetría del ciclo anterior a `prev_*`, llama a `airsim_client.capture()` y, si la imagen es `None` o `telemetry["source"] != "airsim"`, marca `degraded = True` sin tocar el ring buffer. En modo normal agrega el fotograma y su timestamp a `frame_history` (tamaño `VLM_FRAME_HISTORY_SIZE` = 1). Las consultas del lazo lento no usan historial temporal: la capa estratégica toma el fotograma del ciclo y el barrido, uno por rumbo.

5.5 Nodo `degraded_hover`

Activado por `degraded_router` cuando `degraded = True`. Emite `FRENAR` sin percepción ni decisión. El diseño no sustituye el dato faltante por uno sintético plausible, por la misma razón que motiva el capítulo 9: un dato sintético es indistinguible de uno real y esconde el tipo de fallo silencioso que este trabajo documenta.

5.6 Nodo `perception`

Salidas: `obstacle_field`, `estimated_ttc`, `scene_summary`.

Invoca `FlowTTCEstimator.estimate()` con el par de fotogramas y la telemetría de ambos ciclos y produce el `ObstacleField` de tres sectores con ocupación, TTC y bloqueo por sector (cap. 6). Es el único objeto de percepción que consumen el lazo rápido, la FSM y el logger.

5.7 Comportamiento reactivo (`reactive_node`)

Función: `reactive_node` en `src/agents/reactive.py`. **Activación:** camino despejado, altitud bajo el piso óptico, o brazo `reactive`.

1. **Misión completada:** `FRENAR`, `flight_status = "mision_completada"`.
2. **Waypoint activo:** toma `vx`, `vy`, `vz`, `yaw_rate` de `waypoint_guidance` (§5.15) y emite `MANTENER_RUMBO`. Con el centro a TTC < `TTC_BLOCKED_THRESHOLD_S` limita la tasa de guiñada para no superar el rango de derotación del estimador de flujo.
3. **Sin waypoints:** avance a `REACTIVE_FORWARD_SPEED` (3 m/s en producción) con amortiguación de la deriva vertical.

5.8 Comportamiento de evasión (`evasive_node`)

Función: `evasive_node` en `src/agents/evasive.py`. **Activación:** contacto (`imu_contact` / `blind_wall`) o peligro frontal sin FOV mayormente bloqueado (reglas 3, 7 y 7b).

Compara la ocupación de los sectores laterales del `ObstacleField` y elige `EVADIR_IZQUIERDA` o `EVADIR_DERECHA` hacia el menos ocupado (a igualdad, el de mayor TTC), con velocidad de evasión agresiva (`vx` = 1.2 m/s). `action_to_command()` fija un `target_yaw` redondeado al múltiplo de 90° más cercano (`_manhattan_snap_yaw`), adecuado a la cuadrícula urbana. La maniobra se compromete en `active_maneuver` y `navigate` la continúa (regla 3).

5.9 Giro de 90° (`GIRAR_90`)

Activación: regla 7, cuando `blocked_fraction > FOV_BLOCKED_THRESHOLD` (0.6): el campo de visión está tan obstruido que una corrección lateral no tiene sentido.

Gira 90° sin traslación (`yaw_rate` ±20°/s, `target_yaw` redondeado a la cuadrícula) **hacia el lado del waypoint**, durante `GIRAR90_DURATION_S` (1 s). En el mismo ciclo, `navigate` llama a `StrategicLayer.expedite()` y vuelve a ejecutar `tick()`: si la capa estratégica no tiene un pedido en vuelo y la meta está dentro del campo visual, el fotograma que ve el muro se envía de inmediato a la fuente, sin esperar el período. Así el rodeo del edificio se decide con la imagen que justamente lo motiva.

5.10 Capa estratégica (`vlm_strategic.py`)

Archivos: `src/agents/vlm_strategic.py` (`StrategicLayer`), `src/agents/vlm_client.py` (consulta al VLM), `src/agents/depth_client.py` y `src/perception/depth_estimator.py` (consulta a la red de profundidad), `src/agents/deliberation_service.py` (hilo worker). La capa no es un nodo del grafo: `navigate` llama a `StrategicLayer.tick(state)` una vez por ciclo.

Fuente (`STRATEGIC_SOURCE`). La pregunta de la capa —qué sectores de una grilla 3×3 están libres— la puede responder el VLM (`vlm`) o la red de profundidad estimada (`depth`, default); con `off` la capa no se crea. Las dos fuentes devuelven la misma estructura (`{"sectores": {celda: libre|bloqueado}}`), de modo que el ancla, la traducción a sub-metas y las validaciones de §5.10.3 son las mismas. Las subsecciones §5.10.1–§5.10.5 describen la capa con el VLM como fuente, porque esa fuente fija la forma de la pregunta y de la respuesta; §5.10.6 describe la fuente por defecto, la profundidad estimada, y §5.10.7 justifica esa elección.

5.10.1 Cuándo se consulta

`tick()` envía un pedido solo si se cumplen todas las condiciones:

- pasó el período desde el último pedido — `VLM_STRATEGIC_PERIOD_S` (3 s) con el VLM, `DEPTH_STRATEGIC_PERIOD_S` (1 s) con la profundidad, que tarda ~0.1 s—, o la regla 7 llamó a `expedite()`;
- no hay barrido en curso ni pedido de barrido pendiente (comparten la cola del servicio);
- altitud ≥ `VLM_STRATEGIC_MIN_ALT_M` (8 m);

- el waypoint real está a más de `VLM_NEAR_WP_M` (8 m) en horizontal: más cerca, el guiado directo alcanza, y una sub-meta lo saltearía;
- la meta cae dentro del recorte cuadrado que analiza el modelo (±33.7° en un fotograma de 3:2 con 90° de campo horizontal; `VLM_STRATEGIC_MAX_GOAL_OFF_DEG` puede acotarlo más): si no, el guiado todavía está girando hacia ella y el fotograma no muestra lo que importa. La excepción es un dron que lleva `VLM_STUCK_QUERY_CYCLES` (15) ciclos sin acercarse a su objetivo —por ejemplo, deslizándose a lo largo de una fachada, de costado a la meta—: entonces se consulta igual, porque conviene mirar hacia donde está el dron. En ese caso no hay sector de la meta, la marca «GOAL» queda en el borde del lado del destino, el prompt dice que el destino está fuera de la imagen y la respuesta nunca se interpreta como «camino directo libre».

La consulta se hace **también con una sub-meta activa**. La pregunta es siempre sobre el waypoint real, y una respuesta nueva reemplaza a la sub-meta pendiente: así una sub-meta mala se corrige en la consulta siguiente, en lugar de persistir hasta que el detector de atasco la saque (cap. 9, §9.8.9).

5.10.2 Qué ve el modelo y qué responde

El pedido lleva **un fotograma**: un **recorte cuadrado** del centro del fotograma del ciclo —se cortan los dos costados del ancho—, reescalado a 384 × 384 px y dividido en una **grilla de 3×3 por tercios**, la grilla de composición fotográfica. El original, que usa el flujo óptico, no se modifica. El recorte hace que los sectores laterales y los verticales queden a la misma distancia angular del centro; sobre el fotograma completo de 3:2, un sector lateral quedaría a 33.7° y uno vertical a 24°, y un rodeo por arriba o por abajo ganaría siempre a uno por el costado por la forma del cuadro y no por la escena. Las columnas A, B y C van de izquierda a derecha; las filas 1, 2 y 3, de arriba abajo. La fila 2 es la altura del dron; la fila 1, pasar por arriba; la fila 3, por abajo. Cada sector lleva su etiqueta (`A1` … `C3`) y una marca roja «GOAL» señala la dirección del destino en azimut **y** elevación.

Una partición en columnas solo describe el eje horizontal: no puede decir que un obstáculo se pasa por arriba, ni distinguir un tablero de autopista a la altura del dron de la calle que se ve debajo. La grilla separa las dos dimensiones.

El modelo responde, con decodificación restringida por esquema JSON (§8.2):

```
{"cells": {"A1": "free",    "B1": "free",    "C1": "free",
           "A2": "blocked", "B2": "blocked", "C2": "free",
           "A3": "blocked", "B3": "blocked", "C3": "blocked"}}
```

`free` significa «se puede volar al menos 15 m en la dirección de ese sector»; `blocked`, que un edificio, muro, árbol, puente o techo está a menos de 15 m. El prompt y los valores están en inglés (cap. 8, §8.5); internamente se registran como `libre` y `bloqueado`.

5.10.3 Ancla y sincronización

Al enviar, la capa guarda un **ancla**: posición (x, y, z) y yaw del fotograma, timestamp, etiqueta del waypoint real y su distancia horizontal, la dirección de la meta (azimut y elevación respecto del eje óptico) y el sector en el que cae. Cada sector tiene así una **dirección absoluta**: rumbo = yaw del ancla + azimut del centro del sector, y elevación del centro del sector, ambos calculados con proyección *pinhole* sobre los tercios del recorte, con la focal del fotograma completo (para un fotograma de 3:2 y 90° de campo horizontal, el recorte abarca ±33.7° y los centros de los sectores quedan a ±24.0° tanto en azimut como en elevación). Cuando la respuesta llega (`get_result(request_id)`), `decide_subgoal()` la traduce y valida contra el presente:

1. Descarta la respuesta si pasaron más de `VLM_STRATEGIC_MAX_AGE_S` (10 s) o si el waypoint real cambió.
2. Si el sector de la meta está libre: **no agrega nada** (`directo_libre`). Si había una sub-meta pendiente, la marca para descartar (`_clear_subgoals`): el camino directo está libre y el desvío sobra.
3. Si no: elige el sector libre más cercano en ángulo a la dirección de la meta —entre los que quedan a menos de `VLM_SUBGOAL_TIE_DEG` (10°) del más cercano, la fila del medio, después la de arriba, después la de abajo, de modo que con la meta de frente un rodeo por el costado precede a uno vertical— y fija una sub-meta **desde el ancla** en la dirección de ese sector (`subgoal.build_subgoal`, la misma función que usa el barrido, §5.12), a una distancia horizontal `d = min(VLM_SUBGOAL_DIST_M, distancia al waypoint real)` —15 m como máximo, y **nunca más lejos que el waypoint**—. La altitud cambia `d · tan(elevación del sector)`, acotada a ± `VLM_MAX_DZ_M` (4 m) y sin bajar de `VLM_SUBGOAL_MIN_ALT_M` (6 m). Sin sectores libres: nada (`sin_sector_libre`).
4. Descarta la sub-meta si el dron ya la superó mientras el modelo pensaba (quedó a menos de `VLM_SUBGOAL_MIN_AHEAD_M` = 4 m).

La sub-meta se deposita en `inject_corner`; el bucle externo la inserta en el tracker (§5.15.4) y el guiado normal la persigue. **El dron nunca se detiene a esperar.** Como la geometría está anclada al fotograma, girar mientras el modelo piensa no invalida la respuesta; solo la invalida el paso del tiempo o haberla superado, y ambos casos se verifican.

Si el servicio no tiene el resultado y no hay pedido pendiente durante más de `VLM_STRATEGIC_LOST_GRACE_S` (3 s) —porque un pedido de barrido lo reemplazó en la cola—, la capa lo da por perdido.

5.10.4 Auditoría

Cada respuesta deja una entrada en `deliberations[]` con `arm = "{fuente}_strategic"` (`vlm_strategic` o `depth_strategic`), el prompt (vacío con la profundidad), la respuesta cruda (con la profundidad, el p5 estimado por sector, en metros), el motivo de la decisión (`directo_libre`, `sector C2 (+34 deg, +0 deg, 15 m)`, `vencida`, `wp_cambio`, `superada`, `sin_sector_libre`, `no_parseable`) y la latencia; el fotograma exacto que vio el modelo se guarda como `photo-*.png` vía `_last_delib_frames`. El último resultado queda en `_vlm_strategic`. Esto permite medir, por corrida, con qué frecuencia la capa propone un rodeo, hacia qué fila, y con qué frecuencia el modelo declara «todo libre».

5.10.5 `DeliberationService`

Un hilo daemon con cola de entrada de tamaño 1: `request(payload)` vacía la cola antes de encolar, de modo que el worker procese siempre el pedido más reciente. Los resultados se guardan por identificador (últimos 16), y cada consumidor recupera el suyo con `get_result(request_id)`, de modo que un pedido posterior no pisa un resultado ya producido. Hay a lo sumo dos servicios por misión, cada uno con su hilo, y cada uno se crea solo si alguna capa lo usa: el del VLM (capa estratégica `vlm` o barrido `deep_vlm`) y el de la profundidad (capa `depth` o barrido `depth`). Con la configuración por defecto (`depth` / `depth`) el VLM no corre durante el vuelo: queda en la planificación de ruta previa (cap. 4), y el ejecutor de experimentos solo verifica que el servidor de inferencia responda si alguna capa lo va a usar. El barrido y el brazo FSM usan el servicio de su estrategia.

`vlm_client._query_slm_impl()` atiende dos modos, cada uno con su system prompt, esquema JSON y parser: `strategic` (una imagen, tope `VLM_MAX_TOKENS_STRATEGIC` = 384 tokens, timeout HTTP `SLM_HTTP_TIMEOUT_S` = 15 s) y `deep_scan` (N imágenes, tope `VLM_MAX_TOKENS_DEEP` = 512, timeout `SLM_DEEP_HTTP_TIMEOUT_S` = 20 s). Con decodificación restringida la generación termina sola al cerrar el esquema; el tope solo protege de una generación desbocada en modo libre, y tiene que sobrar. Si el servidor informa que la respuesta se cortó por el tope (`finish_reason = length`), la consulta devuelve el error `respuesta_truncada` y no se interpreta (cap. 9, §9.8.10). Intenta primero con `response_format = json_schema` y, si el servidor no lo soporta, reintenta en modo libre; el parser extrae el primer objeto JSON del texto y lo valida contra el formato del modo.

5.10.6 Fuente por defecto: profundidad monocular estimada (`depth_client.py`)

El capítulo 6 (§6.14) describe la red, su evaluación y sus limitaciones; aquí se resume su integración en la capa. La red es Depth Anything V2 Metric Outdoor Small (Yang et al., 2024), de ~25 M de parámetros, que estima la profundidad **en metros** a partir del fotograma RGB del ciclo. No es el sensor de profundidad de AirSim, que sigue fuera del lazo (§5.16): es lo que haría una red a bordo de un dron que solo tiene la cámara.

Del mapa de profundidad a la grilla. Sobre el mismo recorte cuadrado central y los mismos tercios que la grilla del VLM (§5.10.2), se toma el **percentil 5** de la profundidad estimada de cada sector —lo más cercano, sin píxeles sueltos— y el sector es `libre` si ese valor es ≥ `DEPTH_FREE_M` (15 m, la misma distancia que la sub-meta). Es el mismo estadístico con el que se construyeron las etiquetas de referencia del banco de prueba (cap. 11, §11.0), de modo que la exactitud medida allí es la del juicio que usa el vuelo.

Servicio. `make_depth_service()` envuelve la red en un `DeliberationService` con la misma interfaz que el del VLM (§5.10.5); los pedidos llevan el fotograma crudo, no JPEG en base64. La red corre en el hilo del servicio, nunca en el lazo: ~110 ms por consulta con la GPU compartida con Unreal, frente a ~790 ms del VLM local, y ~440 MiB de VRAM adicionales. Al crearse, el servicio carga la red con un fotograma vacío, para que la primera consulta en vuelo no pague la carga. Una falla de la red se reporta como error y el llamador cae a su camino de falla, como ante un timeout del VLM.

5.10.7 Por qué la profundidad y no el VLM

En el banco de prueba a bordo (cap. 11, §11.0) la grilla del VLM separa sectores libres de bloqueados con AUC 0.53 —azar— y el p5 de la profundidad estimada, con AUC 0.83–0.87 y exactitud balanceada del 75 %; sus errores son conservadores (2 % de sectores marcados libres con un obstáculo a menos de 5 m). Por eso la fuente por defecto es `STRATEGIC_SOURCE = depth`. Si la capa con profundidad mejora el vuelo frente a no tenerla es una de las comparaciones del diseño experimental (`slm:depth` frente a `fsm:depth`, cap. 10 §10.12).

5.11 Nodo `fsm`

Función: `fsm_node(state, service)` en `src/agents/fsm.py`. **Activación:** `AGENT_ARM = "fsm"`.

La FSM usa el mismo `ObstacleField`, el mismo vocabulario de macro-acciones y el mismo `action_to_command()` que el brazo `slm`; lo que cambia es quién elige la acción: umbrales deterministas.

Estado FSM	Macro-acción	Condición
`CRUISE`	`MANTENER_RUMBO`	Centro despejado (default)
`AVOID_LEFT` / `AVOID_RIGHT`	`EVADIR_*`	Centro bloqueado o TTC ≤ 3.5 s; lado menos ocupado
`CLIMB` / `DESCEND`	`GANAR_ALTURA` / `PERDER_ALTURA`	Atasco con todos los sectores bloqueados (alternados)
`BRAKE`	`FRENAR`	TTC central ≤ 1.5 s con un lateral libre

Su detección de atasco usa el contador de progreso del tracker (`evasion_stuck_cycles`) con los umbrales efectivo y duro (§5.15.2). Ante un atasco, si `DEADLOCK_STRATEGY` es `depth` o `deep_vlm`, delega en el mismo barrido panorámico del brazo `slm` (§5.12); si no, alterna `CLIMB`/`DESCEND` hasta `MAX_CONSECUTIVE_ESCAPES` (2) y luego cambia de estrategia con `GIRAR_90` y una esquina de desvío propia.

Dos asimetrías con el brazo `slm` que deben tenerse en cuenta al comparar. Primero, el brazo `fsm` resuelve los deadlocks con el **mismo barrido** que `slm` (con `depth`, juzgado por la red de profundidad; con `deep_vlm`, por el VLM): la comparación `slm:depth` vs. `fsm:depth` aísla la capa estratégica y la política del lazo rápido, no el barrido; el brazo sin barrido es `fsm` con `--deadlock-strategies blind` (`fsm:blind`, cap. 10 §10.12). Segundo, la FSM tiene su propia lógica de atasco y escape (con esquinas de desvío deterministas), distinta de la del brazo `slm`.

5.12 Barrido panorámico en deadlock (`deep_scan`)

Archivo: `src/agents/deep_scan.py`. **Activación:** regla 2 (o el atasco de la FSM) con `DEADLOCK_STRATEGY` en `depth` o `deep_vlm`. La estrategia `blind` no ejecuta barrido (§5.3.4). Valores admitidos: `depth` (default en `config/.env` y en el runner: los rumbos los juzga la red de profundidad), `deep_vlm` (los juzga el VLM; ablación) y `blind`; cualquier otro valor aborta el arranque. La máquina de estados es la misma para las dos fuentes; solo cambia quién juzga cada imagen en la fase `capturado`.

Máquina de estados sostenida entre ciclos por los campos `_scan_*`:

Inicio. Si el waypoint real está a menos de `VLM_NEAR_WP_M` (8 m) en horizontal, no barre: desde tan cerca el rumbo a la meta es casi ruido y una sub-meta la saltearía; devuelve `False` (escape de §5.3.4). Si no, `_scan_phase = "rotando"`, `_scan_start_yaw_deg = yaw actual`, cancela cualquier maniobra activa. Durante todo el barrido, `_deliberation_pending = True` evita que el bucle externo cuente el giro en el lugar como falta de progreso.

`rotando`. Comanda `target_yaw = yaw_inicial + i × 360° / SCAN_HEADING_COUNT_DEEP` (4 rumbos: 0°, +90°, +180°, +270°), sin traslación. Al llegar a `SCAN_YAW_TOLERANCE_DEG` (5°) pasa a `asentando`. Si en `SCAN_ROT_TIMEOUT_CYCLES` (10) ciclos no alcanza el rumbo —dron trabado en una malla—, abandona el barrido y devuelve `False` (escape de §5.3.4).

`asentando`. Hover durante `SCAN_SETTLE_CYCLES_DEEP` (2) ciclos; luego guarda `(rumbo_medido, frame, capture_ts)` con el **yaw medido** en ese ciclo (no el objetivo del giro) y el fotograma que `capture_node` ya produjo (nunca una captura adicional).

`capturado`. Envía un único pedido con hasta `MAX_DEEP_SCAN_IMAGES` (5) imágenes a 256 px, rotuladas solo `[Image 1]` … `[Image N]`. El prompt (`_build_panorama_prompt`) da únicamente la cantidad de imágenes y la altura; no da ángulos, meta ni sugerencias de acción, porque el modelo solo describe y el rumbo de cada imagen lo conoce el código. El modelo responde una entrada por imagen:

```
{"views": [{"img": 1, "view": "facade", "free": false},
           {"img": 2, "view": "open",   "free": true}, ...]}
```

Con la profundidad (`depth`). El pedido lleva los fotogramas crudos de los rumbos; la red estima la profundidad de cada uno y un rumbo es transitable si el p5 del sector central (`B2`, hacia donde volaría el dron en línea recta) es ≥ `DEPTH_FREE_M` (15 m). La respuesta tiene la misma forma que la del VLM (`view` = `open` o `wall`), de modo que la interpretación y la selección de salida que siguen son las mismas. En el banco, el VLM marca transitables todos los rumbos de un barrido; la profundidad separa rumbos abiertos de cerrados con AUC 0.87 (cap. 11, §11.0).

Interpretación en marco mundo (`panorama_to_subgoal`). Las entradas se asignan a las imágenes por el campo `img` (o por orden si falta); las entradas sobrantes que el modelo inventa se descartan. Una imagen es transitable si el modelo la marcó `free: true` **y** con vista `open`: una entrada `free: true` con otra vista (`inside`, `facade`) se contradice a sí misma y no se usa. Entre las imágenes transitables se elige la de **rumbo absoluto** más cercano al rumbo absoluto hacia el waypoint real (calculado con posiciones, no con el yaw), y se fija una sub-meta en ese rumbo con la misma función y las mismas reglas que la capa estratégica (`subgoal.build_subgoal`, §5.10.3): `VLM_SUBGOAL_DIST_M` (15 m) como máximo, nunca más lejos que el waypoint real, a la altitud actual del dron —las imágenes del barrido no describen filas, de modo que la elevación es cero—, con macro `MANTENER_RUMBO`: el guiado la persigue. Sin imágenes transitables: `GANAR_ALTURA` (con el VLM, `PERDER_ALTURA` si todo es vegetación). La resolución se registra en `deliberations[]` (`arm = "{brazo}_deep_scan"`), escribe `_deadlock_event` con la sub-meta y pide `_escape_reset`.

Selección de salida. Un deadlock es evidencia de que el rumbo que el dron intentaba no funciona, y elegir el transitable más cercano a la meta suele devolverlo contra el mismo obstáculo. Por eso, entre los rumbos que el modelo marcó transitables, el código descarta: (1) los que caen a menos de `SCAN_FAILED_SECTOR_DEG` (45°) del **rumbo que falló** —la dirección hacia el objetivo activo al iniciar el barrido—; y (2) los que caen a menos de 45° de un rumbo **ya probado** —fallado o elegido— en un deadlock anterior a menos de `SCAN_REPEAT_RADIUS_M` (10 m), para el mismo waypoint. Entre los que quedan, sin historia en la zona se elige el más cercano a la meta (`hacia_meta`); con historia, el más distinto de los ya probados (`exploracion`). Si todos los transitables estaban descartados: `GANAR_ALTURA` (`sin_rumbo_nuevo`). El modelo sigue decidiendo qué está libre: el código solo elige entre sus opciones usando lo que el dron ya comprobó al quedar trabado. La historia (`_deadlock_history`, últimos 12 deadlocks, incluidos los barridos que fallaron) y la selección completa —candidatos, descartados con su motivo, rumbo que falló, rumbos probados, modo y rumbo elegido— quedan en `_deadlock_event["scan_selection"]` para medir si la selección ayuda.

Este diseño evita dos errores de marco de referencia que convierten una respuesta correcta en una maniobra en sentido contrario (cap. 9, §9.8.1): etiquetar las imágenes con el rumbo absoluto y pedir al modelo un ángulo relativo, y medir el error de rumbo a la meta desde el yaw final del barrido en lugar del de cada imagen. Por eso el modelo nunca reporta ángulos: identifica imágenes por número y el código usa el yaw medido.

Fallas. Watchdog `SLM_DEEP_WATCHDOG_MS` (18 s en producción) vencido, pedido perdido (`SCAN_LOST_GRACE_MS`, 3 s), respuesta sin el formato del modo o respuesta cortada por el tope de tokens: el barrido deja una entrada `arm = "deep_scan_failed"` con sus fotogramas y devuelve `False`.

5.13 Nodo `motor`

Único nodo que interactúa con el actuador. Lee `velocity_command` y aplica, en orden:

Ancla de altitud en `FRENAR`. En el primer ciclo de `FRENAR` ancla `_hover_alt_anchor = z`; en los siguientes, si `|anchor - z| > 0.3 m`, inyecta `vz = clamp(0.35 × dz, ±0.8 m/s)`. Corrige la deriva vertical de reemitir `moveByVelocityBodyFrameAsync(vz = 0)` (medido: hasta 9 m en 120 s sin corrección).

Gobernador de velocidad (`src/navigation/speed_governor.py`). Limita solo el avance de `MANTENER_RUMBO`, según cuánta evidencia perceptual hay y qué pasó recientemente: sin flujo válido durante 3 ciclos seguidos, tope de 1.5 m/s, y durante 8, de 1.0 m/s; tras un frente bloqueado (GIRAR_90 o `blocked_fraction` ≈ 1), tope de 1.0 m/s durante 40 ciclos; al terminar una maniobra, rampa lineal del tope de 1.0 a 6.0 m/s en 15 ciclos (por encima de la velocidad de crucero, el tope deja de actuar). Su función es que la ausencia de evidencia no se trate como espacio libre: sin él, con el campo sin evidencia y una pared a 4 m, el guiado puede acelerar al dron hacia ella. Se desactiva con `GOV_ENABLED = false`.

Actuación no bloqueante. `execute_velocity(vx, vy, vz, yaw_rate, target_yaw)`: con `target_yaw` usa orientación absoluta (`YawMode(is_rate=False)`); sin él, tasa angular.

5.14 Mapa de macro-acciones (`action_to_command`)

Todos los comportamientos comparten `action_to_command()` (`src/agents/action_map.py`) como fuente única de la cinemática de cada macro-acción.

Macro-acción	`vx` (m/s)	`vy`	`vz` (m/s)	`yaw_rate` (°/s)	`target_yaw`
`MANTENER_RUMBO`	guiado	0	guiado	guiado	—
`EVADIR_DERECHA` / `EVADIR_IZQUIERDA`	1.2 (agresiva) / 0.3–0.8	0	guiado	±15	snap 90°
`GANAR_ALTURA`	0	0	−1.5	guiado	—
`PERDER_ALTURA`	1.0	0	+0.8	0	—
`GIRAR_90`	0	0	0	±20	snap 90° hacia el WP
`FRENAR`	0	0	0 (+ ancla)	0	—
`RETROCEDER`	−1.2	0	guiado	0 (rumbo congelado)	—

`RETROCEDER` forma parte del vocabulario, pero **ningún comportamiento del brazo `slm` lo emite** (§5.3.4). `GANAR_ALTURA` asciende en el lugar y sin deriva lateral: una deriva alejaría el waypoint en el plano XY, la métrica que decide si el atasco se resolvió.

Las duraciones de las maniobras comprometidas son: `MANEUVER_DURATION_S` (2.0 s en producción) para `EVADIR_*` y para la macro resuelta por el barrido; `ESCAPE_MANEUVER_DURATION_S` (1.6 s) para el escape vertical; `GIRAR90_DURATION_S` (1.0 s) para el giro.

5.15 Guiado a waypoint: `WaypointTracker`

`WaypointTracker` (`src/navigation/waypoint_tracker.py`) corre en el bucle externo, **fuera del grafo**, una vez por ciclo antes de `graph.invoke()`. `update(pos)` avanza el waypoint activo al entrar en el radio de aceptación (3.5 m; bajo un techo detectado, por distancia horizontal) y `compute_guidance()` produce `vx`, `vy`, `vz`, `yaw_rate` en marco del cuerpo, con corrección de *cross-track*, zona muerta angular, tope de guiñada diferenciado (15°/s, 45°/s con desvío > 60°), histéresis entre regímenes, suavizado EMA y el modo `near_vertical` (`vx = 0` si `dist_xy < 1 m` y `|dz| > 0.3 m`, necesario para el patrón *climb-first* de Tier 2).

Despegue vertical. Hasta que el dron llega a `TAKEOFF_ALT_TOL_M` (1 m) de la altitud del primer objetivo, `compute_guidance()` no avanza en horizontal (`vx = 0`): sube en el lugar mientras alinea el rumbo, y exporta `takeoff = True`. La fase ocurre una vez por misión y termina antes si se detecta un techo. Avanzar mientras se sube puede llevar al dron contra una estructura más baja que la altitud de crucero pero más alta que la altitud de ese momento (cap. 9, §9.8.8). Durante la fase, los tres brazos usan `reactive_node` y el `StallDetector` no cuenta atasco.

5.15.1 Contador de progreso

`record_progress(dist_xy, bearing_err_deg)` mantiene `evasion_stuck_cycles`: se reinicia cuando la distancia mejora `WAYPOINT_PROGRESS_EPS_M` (0.5 m) sobre la mínima vista; se exime al dron que gira activamente (error de rumbo > 30°) hasta 15 ciclos consecutivos; en otro caso, se incrementa. Este contador lo consume el brazo `fsm`; el brazo `slm` usa las señales del `StallDetector` (§5.3.1).

5.15.2 Umbrales de atasco

`effective_stall_threshold()` es el mínimo de ciclos sin progreso físicamente demostrable: con `eps = 0.5 m`, velocidad mínima 0.25 m/s y 5 Hz, 10 ciclos. `hard_stall_threshold()` lo multiplica po `STUCK_HARD_FACTOR` (1.5 en producción: 15 ciclos). Ambos los usa la FSM.

5.15.3 Detección de techo

Con un waypoint bajo una estructura horizontal, la corrección de altitud empuja al dron contra la losa. El tracker lo detecta: si durante 10 ciclos, con el dron libre en horizontal, se **ordena** ascenso (`vz < -0.3 m/s` en el comando efectivamente ejecutado, que el bucle externo le informa con `note_executed_command()`; altitud ≥ 3 m) sin que la cota varíe más de 0.15 m, declara un techo (`ceiling_z`), liberado al alejarse 15 m o al cambiar de waypoint. Con techo, `compute_guidance()` limita la altitud objetivo a `ceiling_z + CEILING_MARGIN_M` (0.8 m) y, si el waypoint cae dentro de `CEILING_SAFE_GAP_M` (3 m) del techo, lo baja a esa banda y exporta `z_path_blocked = True`, que `navigate` resuelve con `PERDER_ALTURA` (regla 6). Usar el comando ejecutado y no la demanda del guiado importa: durante un `GIRAR_90` o un `FRENAR` el comando vertical es cero aunque el guiado pida subir, y esos ciclos no prueban que haya una losa encima. Por la misma razón no cuentan los ciclos en que el dron está trabado de costado —se ordenó avanzar y no se desplazó más de `CEILING_SIDEWAYS_STUCK_M` (3 cm por ciclo)—: el roce contra una pared también impide subir, y ese caso lo resuelve el deadlock.

5.15.4 Sub-metas del lazo lento

`inject_corner_waypoint(x, y, z, label)` inserta una sub-meta temporal delante del waypoint activo. Todas las sub-metas del brazo `slm` provienen del lazo lento (capa estratégica o barrido, con la red de profundidad o con el VLM) y se insertan **tal cual**: una sub-meta nueva reemplaza a las temporales pendientes y reinicia el contador de progreso. La única condición es la deduplicación: si la sub-meta pendiente está a menos de `SUBGOAL_DEDUP_M` (10 m) de la nueva, la nueva se ignora; reemplazarla por otra casi igual reiniciaría el contador de progreso y un dron atascado nunca declararía el deadlock.

Una sub-meta es un medio, no una condición. `update()` mide en cada ciclo la distancia al waypoint **real** aunque el objetivo activo sea una sub-meta: si el dron entra en el radio de aceptación del waypoint real, descarta las sub-metas pendientes (`drop_temporary()`) y lo acepta. El bucle externo también las descarta cuando la capa estratégica marca `_clear_subgoals` (§5.10.3).

5.16 Salvaguardas y contratos de diseño

Exclusión total de profundidad. Ningún componente del lazo de vuelo —nodos del grafo, `main.py`, `experiments/runner.py`, módulos de navegación— lee el canal de profundidad del simulador. El test estático `tests/test_no_depth_in_flight_path.py` lo verifica sobre todos los módulos de `src/agents`, `src/perception` y `src/navigation` y sobre ambos puntos de entrada, buscando también canales indirectos, como un tope de velocidad armado desde el runner con la cámara de profundidad (`_depth_brake_left`), que daría al brazo evaluado acceso a la profundidad del simulador, de la que salen las etiquetas de referencia (cap. 9, §9.8.5). La red de profundidad **estimada** desde el RGB (§5.10.6) no lee ese canal: recibe el mismo fotograma que el flujo óptico, y sus dos módulos están en la lista de la guardia como consumidores legítimos del RGB. Solo los scripts de calibración offline (cap. 7) y la auditoría de la distancia mínima a obstáculo (`src/logging/distmin_audit.py`, cap. 10 §10.6.1) usan el canal de profundidad. La auditoría es un hilo aislado con su propia conexión al simulador, que solo escribe un archivo de muestras; la guardia verifica que no importe ningún módulo del grafo, que el runner se limite a arrancarlo y detenerlo, y que ningún otro módulo de vuelo lo referencie.

La fuente sin overrides. La respuesta del VLM o de la red de profundidad se traduce a geometría y se aplica; no pasa por reglas deterministas que la reemplacen. Las validaciones que sí existen (edad, cambio de waypoint, sub-meta superada, deduplicación) descartan respuestas **obsoletas**, no respuestas que una regla considere equivocadas.

Contrato de `deliberations[]`. Solo agregado: se suman entradas, nunca se borran campos. Los análisis de los capítulos 9 a 11 se derivan de este registro.

Cliente AirSim como singleton. `compile_workflow(airsim_client)` recibe un cliente ya conectado; un segundo cliente tendría su propio `takeoffAsync`, desconectado del grafo.

5.17 Modo degradado

Cuando `degraded = True`, el ciclo va directo a `degraded_hover`: hover explícito, sin percepción ni decisión, sin sustituir el dato faltante por uno sintético.

5.18 Registro de vuelo y auditoría post-vuelo

Cada ejecución —de `main.py` o de cada celda del runner— genera una carpeta autocontenida con traza JSONL y CSV, resumen de la corrida y por objetivo, muestras de DistMin, copia de la consola, fotogramas de cada consulta del lazo lento, video anotado de la cámara frontal (`.webm`), video de la cámara de seguimiento externa (`.follow.webm`, solo auditoría) y visor HTML. El capítulo 4 (§4.4) describe cada archivo, cada campo y el uso del visor. El runner graba video por defecto (`--no-video` lo desactiva).

Traza completa del `DroneState`. El JSONL conserva por ciclo el estado completo sin imágenes (`src/logging/state_serializer.py`); el CSV lo aplana en columnas `state.<ruta>`. La telemetría incluye `landed_state`.

Auditoría del lazo lento. Cada consulta resuelta —estratégica o de barrido, a la profundidad o al VLM— deja una entrada en `deliberations[]` y sus fotogramas RAW se asocian a la fila del ciclo en que la respuesta llegó (`_last_delib_frames`). Cada deliberación nueva se cuenta en `slm_invocations` en el ciclo en que aparece, cualquiera sea la ruta del ciclo (la capa estratégica resuelve mientras el dron vuela por la ruta reactiva o evasiva). Los barridos fallidos también se registran.

Video en un hilo aparte. La codificación VP8 (Bankoski et al., 2011) corre en un hilo con cola acotada y el cuadro se reduce a 0.6: codificar a 1080×720 dentro del lazo cuesta ~275 ms por cuadro, lleva el período de 0.21 a 0.45 s y degrada el flujo óptico en el 64–82 % de los ciclos.

Visor HTML. Sincroniza los dos videos y la traza por índice de ciclo y muestra, para cada ciclo, el campo de obstáculos, la consulta del lazo lento con sus fotogramas y el árbol del `DroneState` (cap. 4 §4.4).

5.19 Cierre de misión y vigilancia de congelamiento

Aterrizaje suave (`land_smooth`). Hover de 1.5 s, descenso con `moveToZAsync` a 0.5 m/s hasta z = −0.3 m y desarmado; evita la caída libre de `landAsync()` desde la altura de crucero.

Vigilante de congelamiento físico (`src/navigation/freeze_watchdog.py`, en el bucle externo). Si el estado físico del dron (posición, actitud y velocidad) queda idéntico durante `FREEZE_CYCLES` (25) ciclos, el dron está incrustado en la malla: con `FREEZE_RECOVERY = abort` (producción) la corrida termina con `termination_reason = "physics_locked"`; con `teleport`, vuelve a la última pose libre hasta `FREEZE_MAX_RECOVERIES` veces y lo registra como evento.

Ningún criterio de terminación usa la distancia al obstáculo: medirla en vuelo requeriría el canal de profundidad del simulador (§5.16). La métrica `min_obstacle_dist_m` la mide el hilo de auditoría y `finalize_run` la consolida en el `summary.json` al cerrar la corrida (cap. 10 §10.6.1).

6. Percepción monocular: flujo óptico, TTC, campo de obstáculos y profundidad estimada

La percepción usa una sola cámara RGB y tiene dos canales, uno por lazo de control (cap. 5, §5.1.2): el **flujo óptico** (§6.1–§6.13), que alimenta al lazo rápido cada 0.2 s, y la **profundidad monocular estimada** con Depth Anything V2 (§6.14), que alimenta al lazo lento y al barrido de resolución de atascos. §6.15 explica cómo se complementan.

6.1 Fundamentos de diseño: percepción geométrica sin redes neuronales en el lazo rápido

La percepción del lazo rápido prescinde deliberadamente de redes neuronales profundas de detección (detectores de cajas delimitadoras tipo YOLO — Redmon et al., 2015 — o segmentadores semánticos densos) y fundamenta la estimación de obstáculos en visión por computadora clásica: **flujo óptico denso derotado y divergencia del campo traslacional**. Esta decisión no es de conveniencia sino de principio, y se sustenta en tres argumentos de ingeniería robótica:

1. Determinismo y presupuesto de cómputo compartido. En una plataforma donde el cómputo se comparte con la simulación y con las redes del lazo lento, un estimador de flujo clásico (DIS en OpenCV) garantiza una latencia determinista y acotada por ciclo (5–10 Hz), sin los picos de inferencia asociados a redes convolucionales o transformadores visuales densos. Shi et al. (2024) y Goel et al. (2021) documentan este problema de presupuesto en sistemas embebidos de visión con requisitos de tiempo real; en este sistema el presupuesto disponible para percepción es del orden de 20–50 ms por ciclo.

2. Generalización universal y robustez fuera de distribución. Los detectores supervisados están limitados a las clases presentes en su conjunto de entrenamiento (automóviles, personas, señales). En un entorno urbano tridimensional, los obstáculos potenciales incluyen cables, salientes arquitectónicas, andamios o geometrías irregulares sin etiqueta semántica previa. El flujo óptico modela directamente el fenómeno **físico** de aproximación espacial mediante la expansión de textura en el plano focal: reacciona ante cualquier objeto que genere paralaje, sin importar su clase ni su apariencia (Badrloo & Varshosaz, 2017; Vera-Yanez et al., 2024). Esta propiedad es especialmente valiosa en entornos de simulación que aún no reproducen fielmente la distribución fotométrica del mundo real (cap. 9).

3. Modelado físico del Tiempo-a-Colisión (TTC). Una caja delimitadora 2D no provee información cinemática de cierre ni distancia métrica directa. La divergencia del campo de flujo traslacional permite derivar matemáticamente el TTC ($TTC \approx d/v_{\text{cierre}}$) de forma continua a lo largo del plano de la imagen, entregando una señal temporal interpretable y calibrable para las decisiones de maniobra reactiva. El principio no es una invención de la robótica aérea: es la teoría del *tau* (τ) de Lee (1976), que demostró que el tiempo-a-colisión puede leerse directamente de la tasa de expansión óptica de una superficie en el plano de la imagen, sin necesidad de recuperar distancia ni velocidad absolutas por separado. Esta derivación es una consecuencia directa de la ecuación del flujo óptico bajo traslación pura, instanciada en visión artificial aplicada a UAVs por Al-Kaff et al. (2017) y Vera-Yanez et al. (2024).

Limitación estructural y rol del lazo lento. La elección de percepción clásica tiene un costo explícito: el flujo óptico requiere **traslación entre frames** para generar evidencia válida. En hover puro, giro puro o crucero muy lento, la señal traslacional colapsa a cero y el campo de obstáculos pierde toda confianza; lo mismo ocurre con un obstáculo centrado en el foco de expansión. El lazo lento mira un único fotograma y no necesita traslación. Por defecto lo resuelve una red de profundidad monocular (§6.14); la alternativa, el VLM, no distingue sectores libres de bloqueados en el banco de prueba (cap. 11, §11.0). La arquitectura de dos canales —flujo rápido para lo inminente, profundidad estimada de un fotograma para decidir el rodeo— es el mecanismo con el que este trabajo aborda esa limitación.

(Para el desarrollo matemático riguroso de la cámara pin-hole, las ecuaciones continuas de Longuet-Higgins, la deducción del FOE por mínimos cuadrados y la teoría de τ de Lee, véase el Anexo 5).

6.2 Pipeline de percepción: cinco etapas

`FlowTTCEstimator.estimate()` (`src/perception/flow_ttc.py`) ejecuta cada ciclo cinco etapas en secuencia, produciendo un `ObstacleField` (`src/perception/obstacle_field.py`) listo para el router de política:

```
Frame(t-1) + Frame(t) + Telemetría(t-1, t)
    ↓
[1] Escala y conversión a escala de grises
    ↓
[2] Flujo óptico denso (DIS / Farneback)
    ↓
[3] Derotación por telemetría de actitud (IMU)
    ↓
[4] Estimación del FOE + RANSAC-lite de outliers
    ↓
[5] TTC por píxel + divergencia → agregación por celdas → ObstacleField
```

Si en cualquier etapa no hay evidencia suficiente (primer ciclo, modo degradado, rotación grande entre frames, flujo bajo el piso de ruido), la función retorna un `empty_field()` con `source="degraded"` o `"flow"` y `foe_confidence=0.0`, marcando explícitamente la ausencia de evidencia sin ningún clamp cosmético que la enmascare.

6.3 Etapa 1: escala y parámetros intrínsecos

Los fotogramas BGR de AirSim se convierten a escala de grises y se escalan a un ancho normalizado `FLOW_DOWNSCALE_WIDTH`: 640 px en `config/.env` (320 px es el valor por defecto del código y el que se usó en la validación de TTC del capítulo 7). El vuelo usa 640 px porque a 320 px los objetos finos, como las ramas, quedan bajo el piso de ruido y el estimador devuelve confianza cercana a cero frente a un acercamiento real. Esta reducción sirve dos propósitos: reducir el costo de cómputo del estimador de flujo y suavizar el ruido de alta frecuencia que genera falsos vectores de flujo. La distancia focal se escala proporcionalmente:

$$f'_x = f_x \cdot \frac{W'}{W}, \quad f'_y = f_y \cdot \frac{W'}{W}$$

con `CAMERA_FX = CAMERA_FY = 554.0` px (cámara simétrica a la resolución de captura de AirSim) y el centro óptico $(c_x, c_y) = (W'/2, H'/2)$.

Guard de rotación. Antes de computar el flujo, el estimador verifica que la rotación máxima entre frames (pitch, yaw, roll) no exceda `FLOW_MAX_ROTATION_DEG` (default 2°). Durante maniobras activas (GIRAR_90, EVADIR) el yaw cambia 4–9°/ciclo a `LOOP_HZ=5` Hz. En esas condiciones, el modelo de derotación lineal de primer orden que se describe en §6.4 introduce errores de linealización que el estimador no puede compensar, y regiones de baja textura (cielo, fachadas uniformes) generan flujo esencialmente aleatorio. Sin este guard, cada giro produce «nubes» de falsos obstáculos. La solución conservadora es descartar el ciclo y retornar `empty_field(source="degraded")`: es mejor declarar incertidumbre que producir un falso positivo que cancele una maniobra en ejecución.

6.4 Etapa 2: flujo óptico denso (DIS)

El backend de flujo óptico es **DIS** (*Dense Inverse Search*, `cv2.DISOpticalFlow_create(PRESET_MEDIUM)`), con fallback a Farneback si DIS no está disponible. DIS produce un campo vectorial $\mathbf{v}(x, y) = (u, v)$ de desplazamientos en píxeles para cada posición del frame actual respecto al anterior.

Por qué DIS y no Farneback. DIS (*Dense Inverse Search*, Kroeger et al., 2016) ofrece una relación velocidad/calidad superior para flujo denso en imágenes de tamaño pequeño (≤ 320 px): su esquema de búsqueda inversa por parches es entre 5 y 10 veces más rápido que Farneback para resoluciones comparables, manteniendo la suavidad necesaria para la estimación del FOE. La propia implementación de OpenCV (`cv2.DISOpticalFlow`) usada en este trabajo es directamente la del algoritmo original de Kroeger et al. Vera-Yanez et al. (2024) y Molineros et al. (2012) usan variantes de flujo óptico denso con esquemas de pirámide similares —Farneback en el primer caso—; la elección de DIS en este trabajo está motivada por el mismo compromiso latencia/densidad que documentan esos trabajos, resuelto a favor de la variante más rápida disponible en OpenCV.

El campo resultante $\mathbf{v}(x, y)$ mezcla el componente traslacional (paralaje de objetos en escena) con el componente rotacional inducido por los cambios de actitud del vehículo. Separar ambos componentes es el objetivo de la etapa siguiente.

6.5 Etapa 3: derotación analítica por telemetría IMU

La rotación propia del vehículo entre frames consecutivos genera un flujo óptico angular parásito $\mathbf{v}_{\text{rot}}$ que enmascara la expansión traslacional real. Para un modelo de cámara *pinhole* con rotación pequeña entre frames, la contribución rotacional al flujo óptico en el plano normalizado es:

$$u_{\text{rot}}(x,y) = \frac{\theta_x \cdot x \cdot y}{f_x} - \theta_y \left(f_x + \frac{x^2}{f_x} \right) + \theta_z \cdot y$$

$$v_{\text{rot}}(x,y) = \theta_x \left(f_y + \frac{y^2}{f_y} \right) - \frac{\theta_y \cdot x \cdot y}{f_y} - \theta_z \cdot x$$

donde (x, y) son coordenadas en el plano de la imagen centradas en (c_x, c_y), y $(\theta_x, \theta_y, \theta_z)$ son los incrementos de actitud (pitch, yaw, roll) entre frames, extraídos de la telemetría del IMU del simulador. El mapeo cámara-cuerpo asume alineación frontal: $\theta_x \approx \Delta\text{pitch}$, $\theta_y \approx \Delta\text{yaw}$, $\theta_z \approx \Delta\text{roll}$.

El campo traslacional derotado es:

$$\mathbf{v}_{\text{trans}} = \mathbf{v}_{\text{medido}} - \mathbf{v}_{\text{rot}}$$

La implementación (`_derotate()`) vectoriza esta operación sobre el array completo usando `np.mgrid` para construir los mapas de coordenadas (x, y) en una sola operación, sin bucles por píxel. El salto de wrap-around del yaw ($\pm\pi$) se corrige con el módulo estándar antes de usar Δyaw como θ_y.

Pre-integración de la velocidad angular del IMU (P1). La telemetría incluye `imu_angular_velocity = {wx, wy, wz}` (rad/s, `getImuData()`), y cuando está presente el estimador usa `wz · Δt` como Δyaw en lugar de la diferencia entre los ángulos de orientación de dos ciclos. La diferencia importa si la tasa de guiñada varía dentro del intervalo entre frames. En AirSim, cuyo IMU es ideal, ambos métodos son numéricamente equivalentes; el cambio deja el diseño preparado para hardware real, donde la velocidad angular medida por el giroscopio es la señal disponible.

Esta corrección es crítica para el sistema: sin ella, cada corrección de guiado (giro de unos pocos grados hacia el waypoint) genera un flujo rotacional en el sector central que se interpreta como un obstáculo frontal, disparando deliberaciones espurias en cada ciclo de crucero. El mismo principio de derotación por IMU se usa en sistemas de visión activa para vehículos terrestres (Dickmanns, 2024) y SLAM monocular (Chen et al., 2022).

6.6 Etapa 4: estimación del Foco de Expansión (FOE) con RANSAC-lite

Bajo traslación pura (sin rotación residual), todos los vectores del campo traslacional apuntan radialmente desde un único punto llamado **Foco de Expansión** (FOE, también llamado punto de fuga traslacional). La posición del FOE en el plano de la imagen codifica la dirección de traslación del vehículo, y la magnitud del flujo en cada píxel es inversamente proporcional a la distancia al obstáculo que generó ese paralaje.

Descarte de ruido de baja magnitud. Los píxeles con $\|\mathbf{v}_{\text{trans}}\| \leq \text{FLOW\_NOISE\_FLOOR\_PX}$ (default 0.35 px) se descartan antes de cualquier ajuste. Por debajo de este umbral, los vectores de flujo son artefactos de cuantización del estimador, no señal real. Si la fracción de píxeles válidos resultante es menor que `MIN_VALID_FRACTION_FOR_FOE` (default 1%), el campo entero se declara sin evidencia.

Ajuste por mínimos cuadrados ponderados (primera pasada). Para los píxeles válidos, la condición geométrica de consistencia con el FOE es que cada vector de flujo $\mathbf{v}(p)$ esté **alineado** con la línea que une p al FOE. La componente normal al vector $\mathbf{v}$ debe ser cero en el FOE: para el píxel (p_x, p_y) con vector (v_x, v_y), la normal normalizada es $\mathbf{n} = (-v_y, v_x)/|\mathbf{v}|$, y la condición es $\mathbf{n}^T(\text{FOE} - p) = 0$. El sistema lineal resultante es:

$$\left(\sum_i w_i \mathbf{n}_i \mathbf{n}_i^T\right) \text{FOE} = \sum_i w_i \mathbf{n}_i (\mathbf{n}_i^T p_i)$$

donde $w_i = |\mathbf{v}_i|$ pondera cada ecuación por la magnitud del flujo (vectores más largos son más confiables). Este sistema 2×2 se resuelve con `np.linalg.solve()`.

Recorte de outliers tipo RANSAC-lite (segunda pasada). El FOE de primera pasada puede estar contaminado por píxeles de fondo, regiones de baja textura, o residuos de la derotación. El refinamiento evalúa, para cada píxel, el ángulo entre su vector de flujo y la línea $p \to \text{FOE}$: si el ángulo supera `FOE_OUTLIER_ANGLE_RAD` (default 0.35 rad ≈ 20°), el píxel se descarta como outlier. Con los inliers restantes se repite el ajuste por mínimos cuadrados. Este esquema es una versión simplificada del RANSAC estocástico de Kaneko et al. (2017) y de la estimación robusta de FOE en flujo de egomotion documentada en la literatura de detección de obstáculos por flujo residual (Molineros et al., 2012).

Confianza del FOE. Se distinguen dos casos: - Pocos inliers (< 30): FOE poco confiable; `foe_confidence` se clipea a 0.3 como señal de evidencia degradada. - Suficientes inliers (≥ 30): `foe_confidence = min(n_inliers / (H × W) × 3.0, 1.0)`. El factor 3.0 compensa que la fracción de inliers esperada en una escena real es aproximadamente 1/3 del total de píxeles válidos (fondo vs. objetos en aproximación); el boost normaliza la confianza a un rango comparable al de la fracción bruta.

Sanity check de límites. Un FOE fuera del rectángulo de la imagen (`0 ≤ foe_x ≤ W`, `0 ≤ foe_y ≤ H`) no es un punto de fuga físicamente plausible para esa imagen; indica que el ajuste convergió a una solución artefactual (ruido dominante o residuo de derotación mal compensada). En ese caso, `foe_confidence` se fuerza a 0.0 y el campo se retorna sin celdas bloqueadas.

6.7 Etapa 5: TTC por píxel y divergencia

Con el FOE estimado y el intervalo temporal real Δt (diferencia de timestamps del simulador entre frames consecutivos — nunca el período nominal del lazo), se calcula para cada píxel válido:

$$TTC(x, y) = \frac{\|p - \text{FOE}\| \cdot \Delta t}{\|\mathbf{v}_{\text{trans}}(x, y)\|}$$

Esta fórmula es la forma discreta de la ecuación continua $TTC = Z/v_{\text{cierre}}$, válida bajo la aproximación de cámara *pinhole* con traslación dominante. Para píxeles cerca del FOE o con flujo muy pequeño, el denominador puede producir TTC extremadamente alto (sin evidencia de aproximación), lo que se traduce en $TTC \to \infty$.

Canal de divergencia. En paralelo al TTC, el estimador computa la divergencia del campo traslacional como verificación independiente:

$$\text{div}(x, y) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

implementada con `np.gradient()` (diferencias de segundo orden centradas, normalizadas por el intervalo temporal Δt). La divergencia positiva indica expansión local del flujo — el objeto en ese punto del plano focal está acercándose al vehículo. Es un indicador complementario al TTC que no requiere estimar el FOE: incluso si el ajuste del FOE falla, una celda con divergencia positiva elevada sigue siendo evidencia de aproximación.

6.8 Grilla de celdas 3×3 y agregación por sector

El campo traslacional se divide en una grilla de 3 columnas (sectores: *izquierda*, *centro*, *derecha*) × 3 filas (bandas: *superior*, *medio*, *inferior*), totalizando 9 celdas. Las columnas dividen la imagen en tercios iguales de ancho; las filas, en tercios iguales de alto. Para cada celda se calcula:

Confianza (`confidence`): fracción de píxeles válidos (magnitud > piso de ruido) sobre el total de la celda, multiplicada por `foe_confidence`. Esta confianza combinada refleja tanto la densidad local de flujo válido como la calidad global del FOE estimado.

TTC de celda (`ttc_s`): percentil `TTC_AGGREGATION_PERCENTILE` (default 20) de la distribución de TTC finitos de la celda. El percentil 20 es conservador: estima el TTC del 20% más rápido de los objetos que se aproximan en la celda, resistente al ruido espurio que produce TTC extremadamente bajos en píxeles aislados. Celdas con menos de 5 píxeles válidos reciben `ttc_s = ∞` (sin evidencia).

Ocupación (`occupancy`): media de la divergencia positiva en la celda, escalada por el factor de calibración:

$$\text{occupancy} = \text{clip}\left(\bar{\text{div}}^{+} \times \frac{0.450}{8.0},\ 0,\ 1\right)$$

El factor $0.450/8.0$ fue determinado empíricamente comparando las salidas del estimador con profundidad de referencia del simulador; no es un valor físicamente derivado sino el mejor punto de operación encontrado en las condiciones de vuelo de los escenarios de esta tesis. El protocolo de validación completo —conjunto de datos, curva ROC y la interpretación correcta de las métricas resultantes— se describe en el capítulo 7.

6.9 El predicado de bloqueo: fusión de dos canales

La decisión de si una celda está **bloqueada** fusiona los dos canales (ocupación y TTC) con una compuerta disyuntiva, condicionada a la confianza:

```
def is_blocked(self) -> bool:
    if self.confidence < MIN_CONFIDENCE_FOR_BLOCKED:   # 0.15
        return False
    if self.occupancy >= OCCUPANCY_BLOCKED_THRESHOLD:  # 0.011 (calibrado, D2)
        return True
    return (
        self.confidence >= MIN_CONFIDENCE_FOR_TTC_BLOCKED  # 0.35
        and self.ttc_s <= TTC_BLOCKED_THRESHOLD_S          # 2.5 s
    )
```

La lógica es: una celda está bloqueada si tiene evidencia mínima de percepción **y** (la ocupación es alta, **o** el TTC es bajo con suficiente confianza).

Calibración del umbral de ocupación (D2). Un umbral fijado a ojo (por ejemplo, 0.35) supondría que la ocupación toma valores comparables a los del canal de TTC, y no es así. Un dataset de calibración (196 frames capturados en TownSim frente a una pared sólida, a cuatro velocidades de aproximación de 0.5 a 3 m/s; etiqueta de referencia binaria `gt_depth_centro < 5 m`, cap. 7 §7.1) muestra que la ocupación central nunca supera 0.082 en ese conjunto, de modo que con 0.35 la tasa de verdaderos positivos sería 0: el canal de ocupación quedaría, en la práctica, **desactivado** y el TTC cargaría solo con la detección. La curva ROC sobre `occ_centro` dio AUC = 0.87 y un umbral óptimo por índice de Youden de **0.011** (TPR = 0.93, FPR = 0.22), valor que adoptan `config/.env` y el default del código. El detalle metodológico y sus límites se discuten en §7.5. Para atenuar la dependencia de la escena, `main.py` incorpora un `OccupancyCalibrator` que mide el ruido de ocupación del entorno actual en los primeros 25 ciclos válidos (`mean + 3σ`, acotado a [0.005, 0.05]) y reemplaza el umbral global; **el ejecutor de experimentos no lo usa**, de modo que las corridas en lote operan con el valor fijo de `config/.env`.

Umbrales diferenciados de confianza. La confianza mínima para que la ocupación vote bloqueo es `MIN_CONFIDENCE_FOR_BLOCKED = 0.15`, pero para que el TTC vote bloqueo por sí solo (sin apoyo de ocupación) se requiere `MIN_CONFIDENCE_FOR_TTC_BLOCKED = 0.35`. La razón es que el camino de "pocos inliers" en la estimación del FOE (§6.6) produce `foe_confidence = 0.3` como señal de evidencia degradada, que supera el piso general (0.15). Con un único umbral, el TTC degradado votaría bloqueo con la misma autoridad que un FOE robusto y produciría falsos positivos.

6.10 La API pública de `ObstacleField`

Todos los consumidores del sistema de percepción — `navigate_node` (lazo rápido), `evasive_node`, `fsm_node`, el gobernador de velocidad de `motor_node` y `FlightLogger` — acceden al campo de obstáculos **únicamente** a través de esta interfaz. La capa estratégica **no** recibe el `ObstacleField`: mira el fotograma directamente (cap. 5, §5.10). Ningún módulo de control lee campos crudos de flujo ni coordenadas del FOE.

Método	Descripción
`is_blocked(sector)`	`True` si alguna celda del sector está bloqueada según §6.9
`sector_ttc(sector)`	Mínimo TTC del sector, sobre celdas con confianza ≥ 0.15
`sector_occupancy(sector)`	Máxima ocupación del sector, sobre celdas con confianza ≥ 0.15
`sector_confidence(sector)`	Máxima confianza del sector
`blocked_fraction()`	Fracción de celdas bloqueadas sobre el total de la grilla 3×3
`min_ttc()`	TTC mínimo global, sobre todas las celdas con confianza suficiente
`has_evidence()`	`True` si `source == "flow"` y `foe_confidence > 0.0`
`summary_text()`	Texto compacto por sector para el overlay y el logging (`scene_summary`), p. ej. "CENTRO: BLOQUEADO (TTC=3.2s, fuente: flujo óptico)"
`to_dict()`	Representación serializable para el JSONL de auditoría

El campo `source` del `ObstacleField` indica el origen de la evidencia: `"flow"` (solo flujo óptico), `"degraded"` (ciclo con rotación alta o sin flujo), `"holdover"` (campo del ciclo anterior por degradación temporal), `"none"` (sin datos). El gobernador de velocidad usa `source` para limitar el avance cuando no hay evidencia (cap. 5, §5.13).

El diseño como objeto inmutable (`@dataclass(frozen=True)`) garantiza que ningún consumidor pueda modificar el estado de percepción: los nodos solo pueden leer el campo, no escribirlo.

6.11 Consultas de nivel superior: `has_open_corridor` y `sector_towards_waypoint`

Dos funciones de módulo sirven como interfaz de alto nivel. Solo la FSM las usa (`has_open_corridor`, para decidir si un atasco tiene un corredor visible); el brazo `slm` detecta el atasco con el `StallDetector` (cap. 5, §5.3.1).

`sector_towards_waypoint(bearing_err_deg)` mapea el error de rumbo al waypoint activo a uno de los tres sectores visuales: - Si `bearing_err_deg < -BEARING_SECTOR_DEG` (default 15°) → `"izquierda"` - Si `bearing_err_deg > +15°` → `"derecha"` - Caso contrario → `"centro"`

Esta función conecta el espacio de guiado de `WaypointTracker` con el espacio de percepción del `ObstacleField`, permitiendo que el escape de atasco priorice el sector hacia el waypoint.

`has_open_corridor(field, guidance)` determina si la percepción tiene evidencia real de al menos un sector transitable, evaluado en el contexto del waypoint activo:

```
if not field.has_evidence():
    return False          # sin flujo ≠ "despejado"
for sector in (target, otros_sectores):
    if confidence(sector) >= MIN_CONFIDENCE y not is_blocked(sector):
        return True
return False
```

La condición `has_evidence()` es crítica: un hover puro produce un `ObstacleField` con `foe_confidence = 0`, y tratar esa situación como "camino despejado" desactivaría el escape de atasco precisamente cuando el dron está parado y atrapado. Tratar la falta de evidencia como corredor libre lleva a decisiones de escape sin fundamento perceptual: un dron en hover puede subir 12 m seguidos mientras el `ObstacleField` reporta `DERECHA: DESPEJADO` ciclo tras ciclo.

6.12 Comportamiento bajo condiciones extremas

Hover / velocidad < 0.25 m/s. Sin traslación entre frames, el flujo traslacional es indistinguible del ruido del estimador. La fracción de píxeles válidos cae por debajo de `MIN_VALID_FRACTION_FOR_FOE = 1%` y el campo retorna vacío (`foe_confidence = 0`). `has_evidence()` devuelve `False`, marcando la situación como "sin información" — no como "despejado". La consecuencia es doble: el gobernador de velocidad limita el avance tras varios ciclos sin evidencia (cap. 5, §5.13), y la capa estratégica, que mira el fotograma con la profundidad estimada y no el campo, sigue juzgando la escena (§6.14).

Giro puro (yaw_rate alto). La rotación excede `FLOW_MAX_ROTATION_DEG = 2°` y el ciclo retorna `empty_field(source="degraded")`. El lazo continúa con la maniobra comprometida (continuación de la maniobra comprometida en `navigate`) sin recurrir a nueva evidencia perceptual. La inhibición se verificó empíricamente (D3, §7.6): con guiñadas comandadas de 0.3 a 1.0 rad/s (18.7–63.9 °/s reales) el 100 % de los 200 frames ensayados quedó en `source = "degraded"` con `foe_confidence = 0`.

Holdover temporal del TTC (P2). Cuando un frame aislado devuelve `foe_confidence = 0` (guiñada, textura baja), `FlowTTCEstimator` no vacía el campo de inmediato: devuelve el último campo válido con el TTC decrementado en el tiempo transcurrido (`ObstacleField.decay_ttc()`, copia inmutable con `source = "holdover"`) durante hasta `FLOW_HOLDOVER_MAX_FRAMES = 3` frames (0.6 s a 5 Hz). Superado ese límite el estimador vuelve al campo vacío, de modo que la memoria del obstáculo nunca se prolonga indefinidamente y el origen queda auditable en los logs.

Supresión activa de guiñada cerca de obstáculos (P3). En lugar de solo inhibir el estimador cuando el giro ya ocurrió, el nodo reactivo limita la guiñada del guiado a `FLOW_MAX_YAW_DPS_NEAR_OBSTACLE = 5 °/s` mientras el TTC del sector central esté por debajo del umbral de bloqueo (`safe_yaw_rate()`), un valor dentro del rango en que la derotación opera (≈ 10 °/s a 5 Hz). No aplica a `GIRAR_90` ni a `EVADIR_*`, cuya guiñada es intencional.

Textura baja o uniformidad fotométrica. En regiones sin gradiente (cielo, fachadas, superficies sin textura), `np.gradient` produce valores de flujo cercanos a cero que no contribuyen a la estimación del FOE ni al cómputo de TTC. La confianza de esas celdas es cercana a cero y el predicado `is_blocked()` las descarta.

Transiciones de iluminación brusca. Cambios rápidos de exposición entre frames (que pueden ocurrir en el simulador al cruzar una sombra) producen flujo sistemático en toda la imagen que puede contaminar el FOE. El recorte de outliers (§6.6) mitiga esto parcialmente; la calibración de `FLOW_MAX_ROTATION_DEG` y el piso de ruido `FLOW_NOISE_FLOOR_PX` son los otros mecanismos de contención.

6.13 Variables de entorno calibrables

El estimador expone todos sus parámetros clave a través de variables de entorno (versionadas en `config/.env`):

Variable	Default	Descripción
`FLOW_ALGORITHM`	`"dis"`	Backend de flujo: `"dis"` o `"farneback"`
`FLOW_DOWNSCALE_WIDTH`	320 (código) · **640** (`config/.env`)	Ancho de trabajo para flujo óptico (px)
`CAMERA_FX` / `CAMERA_FY`	554.0	Distancia focal a resolución de captura (px)
`FLOW_NOISE_FLOOR_PX`	0.35	Piso de ruido: vectores más pequeños se descartan
`MIN_VALID_FRACTION_FOR_FOE`	0.01	Fracción mínima de píxeles válidos para estimar FOE
`FOE_OUTLIER_ANGLE_RAD`	0.35	Umbral de ángulo para RANSAC-lite (rad, ≈ 20°)
`TTC_AGGREGATION_PERCENTILE`	20	Percentil de TTC usado como estimado por celda
`FLOW_MAX_ROTATION_DEG`	2.0	Rotación máxima (°) para la que la derotación es confiable
`OBSTACLE_OCCUPANCY_BLOCKED`	0.011	Umbral de ocupación para declarar celda bloqueada (calibrado por ROC/Youden, §6.9 y §7.5)
`FLOW_HOLDOVER_MAX_FRAMES`	3	Frames máximos de holdover del TTC ante un frame sin evidencia (P2)
`FLOW_MAX_YAW_DPS_NEAR_OBSTACLE`	5.0	Tope de guiñada (°/s) del guiado cerca de un obstáculo (P3)
`OBSTACLE_TTC_BLOCKED_S`	2.5	Umbral de TTC para declarar celda bloqueada (s)
`OBSTACLE_MIN_CONFIDENCE`	0.15	Confianza mínima para que ocupación vote bloqueo

Variable	Default	Descripción
`OBSTACLE_MIN_CONFIDENCE_TTC`	0.35	Confianza mínima para que TTC vote bloqueo solo

6.14 Profundidad monocular estimada: Depth Anything V2

6.14.1 Qué es y por qué se usa

Depth Anything V2 (Yang et al., 2024; artículo: https://arxiv.org/abs/2406.09414; código y modelos: https://github.com/DepthAnything/Depth-Anything-V2) es una familia de modelos fundacionales de estimación de profundidad monocular: a partir de **una sola imagen RGB** predicen la profundidad de cada píxel. Su receta de entrenamiento tiene tres pasos: un modelo maestro grande se entrena solo con imágenes sintéticas, cuya profundidad es exacta; ese maestro etiqueta un conjunto grande de imágenes reales sin anotar (pseudo-etiquetas), y los modelos de producción, más chicos, se entrenan sobre esas pseudo-etiquetas. El codificador de todas las variantes es un Vision Transformer de DINOv2 con una cabeza de predicción densa (DPT). Las variantes **métricas** se ajustan sobre datos sintéticos con escala conocida y devuelven la profundidad **en metros**, no en una escala relativa. Este trabajo usa `Depth-Anything-V2-Metric-Outdoor-Small` (`DEPTH_MODEL`): la variante para exteriores con el codificador más chico, de ~25 M de parámetros, distribuida en Hugging Face y cargada con la biblioteca `transformers`.

La elección responde a una medición, no a una preferencia. El lazo lento hace una pregunta de **distancia**: si se puede volar 15 m en la dirección de cada sector. En el banco de prueba a bordo (cap. 11, §11.0) se midieron, contra la misma etiqueta de referencia, tres formas de responderla con el mismo fotograma (`experiments/vlm_bench/seg_depth_probe.py`; 625 muestras × 9 sectores, 61 % de sectores abiertos):

Señal	Exactitud balanceada [IC 95 %]	AUC	Latencia
Depth Anything V2 Metric Outdoor Small: p5 estimado por sector	**75 % [72–77 %]**	**0.83**	82 ms
Segmentación semántica (`yolo26n-sem`): 1 – fracción de obstáculo	45 % [42–47 %]	0.50	32 ms
VLM de vuelo (LFM2.5-VL-1.6B), grilla 3×3	53 % [50–55 %]	0.53	880 ms

La profundidad estimada es la única que supera el criterio del banco (exactitud ≥ 65 % con el intervalo por encima del azar). Separa sin error los casos obvios (AUC 1.00) y, por fila de la grilla, logra AUC 0.97 arriba, 0.80 a la altura del dron y 0.78 abajo. La segmentación no sirve para esta pregunta porque una clase no dice la distancia: «edificio» incluye uno a 100 m. En los barridos reales (290 imágenes, sector central) la profundidad separa rumbos abiertos de cerrados con AUC 0.87 y exactitud balanceada del 75 % [68–82 %], mientras que el VLM declaró transitables todos los rumbos de los 14 barridos del banco. Sus errores son conservadores: solo el 2 % de los sectores que marca libres tienen un obstáculo a menos de 5 m.

La lectura es que el espacio libre es una pregunta de geometría, que resuelve una red chica entrenada para estimar distancia y no un modelo de lenguaje generalista (cap. 12, §12.2).

6.14.2 Del fotograma a la decisión

`DepthEstimator` (`src/perception/depth_estimator.py`) procesa el fotograma crudo del ciclo, el mismo que usa el flujo óptico:

1. **Estimación.** Convierte el fotograma de BGR a RGB, lo pasa por el procesador de la red (que lo reescala a su resolución de trabajo), corre la red en la GPU sin gradientes e interpola la salida al tamaño original: una matriz de profundidades en metros, una por píxel.
2. **Recorte y grilla.** Toma el recorte cuadrado central —el mismo que recibe el VLM con `STRATEGIC_SOURCE = vlm`, ±33.7° en un fotograma de 3:2 con 90° de campo horizontal— y lo divide en una grilla de 3×3 por tercios: columnas A, B, C y filas 1 (arriba), 2 (altura del dron) y 3 (abajo).
3. **Resumen por sector.** Calcula el **percentil 5** de la profundidad de cada sector (`sector_p5`): lo más cercano del sector sin que lo decidan píxeles sueltos. Es exactamente el estadístico con el que se construyen las etiquetas de referencia del banco (cap. 7, §7.1), de modo que la exactitud medida allí es la del juicio que usa el vuelo.
4. **Umbral.** Un sector es `libre` si su p5 es ≥ `DEPTH_FREE_M` (15 m, la misma distancia a la que se fija la sub-meta) y `bloqueado` si no.

El servicio (`src/agents/depth_client.py`) devuelve **la misma estructura que el VLM**, de modo que el resto del sistema no distingue la fuente:

- **Capa estratégica:** `{"sectores": {"A1": "libre", …, "C3": "bloqueado"}}`. La traducción a una sub-meta en coordenadas del mundo, anclada a la pose del fotograma, es la de cap. 5 §5.10.3.
- **Barrido de resolución de atascos:** por cada uno de los cuatro rumbos, `{"img": i, "view": "open" | "wall", "free": …}`, transitable si el p5 del sector central `B2` —hacia donde volaría el dron en línea recta— es ≥ 15 m (cap. 5 §5.12).

La respuesta cruda que queda en la auditoría es el p5 de cada sector en metros (`{"p5_m": [...]}`), de modo que cada decisión se puede revisar contra la distancia que la produjo (cap. 4 §4.4).

6.14.3 Dónde corre y cuánto cuesta

La red **nunca corre dentro del lazo de 5 Hz**: corre en el hilo de un `DeliberationService` (cap. 5, §5.10.5) y el lazo solo deposita pedidos y recoge resultados. Costo medido con la GPU compartida con Unreal Engine:

Recurso	Valor
Latencia por fotograma	82–94 ms en el banco; ~110 ms en vuelo, con la simulación renderizando
VRAM adicional	~440 MiB
Carga inicial	~19 s al compilar el grafo; el servicio precarga la red con un fotograma vacío para que la primera consulta en vuelo no la pague
Período de consulta	`DEPTH_STRATEGIC_PERIOD_S` = 1 s en la capa estratégica (con el VLM, `VLM_STRATEGIC_PERIOD_S` = 3 s); un pedido con cuatro imágenes por barrido

Una consulta de profundidad tarda la mitad de un ciclo del lazo, y el VLM local, ~0.8 s. Por eso la profundidad no reemplaza al flujo en el lazo rápido: el flujo detecta un cierre en milisegundos y en CPU, y la profundidad decide, cada segundo, hacia dónde está el espacio libre.

6.14.4 Qué no es: la exclusión del sensor de profundidad

La red **no lee el canal de profundidad del simulador**: recibe el mismo fotograma RGB que el flujo óptico, como lo haría una red a bordo de un dron que solo tiene una cámara. El canal de profundidad de AirSim sigue fuera del lazo (cap. 5, §5.16): de él salen las etiquetas de referencia, y si alimentara al sistema evaluado, las mediciones dejarían de ser independientes. La guardia `tests/test_no_depth_in_flight_path.py` declara los dos módulos de la red como consumidores legítimos del RGB y verifica que ningún módulo de vuelo acceda al canal del simulador.

6.14.5 Limitaciones

- **Dominio.** Depth Anything V2 se entrenó en parte con imágenes sintéticas, y la etiqueta del banco sale del mismo render que la red ve. En una cámara real el desempeño puede ser menor; la validación en imágenes reales está pendiente.
- **Escala métrica.** La variante para exteriores se ajustó con datos de conducción, a la altura de un auto. Las distancias absolutas a 10–30 m de altura no están calibradas: el umbral de 15 m funciona como corte de clasificación validado en el banco, no como una medición en metros.
- **Objetos finos y follaje.** Ramas, cables y copas semitransparentes son el caso más difícil para cualquier estimador denso; el lote del corredor arbolado de TownSim (cap. 10 §10.3.2) lo pone a prueba.
- **Sin memoria.** Cada consulta juzga un fotograma aislado, sin fusión temporal ni mapa.

- **Campo de visión.** Solo se analiza el recorte central; lo que queda a los costados del fotograma no participa del juicio.

6.15 Complementariedad entre flujo óptico y profundidad estimada

El diseño asume que ningún canal de percepción es completo por sí solo. Hay dos canales, uno por lazo de control (cap. 5, §5.1.2):

Flujo óptico (lazo rápido). Fuerte cuando el dron se traslada a velocidad moderada (> 0.5 m/s), con textura suficiente e iluminación estable: produce bloqueos en < 5 ms, a tiempo para evadir. Es ciego en hover y giro puro, ante texturas uniformes y, por construcción, ante obstáculos centrados en el FOE y superficies horizontales que no se expanden en el sector frontal.

Profundidad estimada (lazo lento). Mira un único fotograma y no necesita traslación: ve el obstáculo centrado en el FOE, la escena en hover y, gracias a las filas de la grilla, una estructura horizontal a la altura del dron. Tarda ~0.1 s, demasiado para lo inminente pero suficiente para decidir el rodeo de un edificio cada segundo, que es lo que el flujo no puede decidir. Con `STRATEGIC_SOURCE = vlm`, el mismo lugar lo ocupa el VLM, con una pregunta igual y la misma salida, pero sin capacidad medida de responderla (cap. 11, §11.0).

La interfaz entre canales es geométrica, no textual. Cada canal produce su salida en su propio dominio: el flujo, bloqueos por sector para el lazo rápido; la profundidad, sub-metas en coordenadas del mundo que el guiado persigue. Ninguno recibe la salida del otro, de modo que no hay que reconciliar dos descripciones de la misma escena con latencias distintas.

Las referencias bibliográficas citadas en este capítulo corresponden a las entradas del §13 del presente informe. La evaluación cuantitativa del estimador de flujo (curvas ROC, AUC, comparación con profundidad de referencia) se desarrolla en el capítulo 7; la de la profundidad estimada, en el capítulo 11 (§11.0). El análisis de los modos de falla de la percepción y su impacto en el comportamiento del lazo se presenta en el capítulo 9.

7. Validación del estimador de TTC y calibración del campo de obstáculos

El capítulo 6 describe el modelo algorítmico del estimador de flujo óptico y su salida, el `ObstacleField`. Este capítulo responde una pregunta diferente: **¿cómo sabemos que el estimador funciona?** El enfoque adoptado es experimental — se recolectó un conjunto de datos etiquetado con la profundidad del simulador y se aplicó análisis de curva ROC para calibrar y evaluar ambos canales (TTC y ocupación) con datos reales de vuelo.

7.1 Etiquetas de referencia a partir de la profundidad de AirSim: ventaja metodológica

Etiqueta de referencia. En este trabajo se llama *etiqueta de referencia* (en la bibliografía, *ground truth*) al valor correcto contra el que se compara, en el mismo instante, lo que estimó o respondió el sistema: por ejemplo, si el sector central está bloqueado a menos de 5 m, cuál es el TTC real o si un sector de la imagen está libre a 15 m. La etiqueta no la produce el sistema evaluado sino un instrumento independiente y más preciso, la profundidad planar del simulador, que el lazo de vuelo nunca lee (cap. 10, §10.3.0). Se calcula siempre en tres pasos: (1) se toma la profundidad del mismo cuadro que vio el sistema, con la misma cámara y el mismo campo de visión; (2) se resume la profundidad de cada región de interés con un percentil bajo, que representa lo más cercano de la región sin que lo arrastren píxeles sueltos; (3) una regla fija y explícita (un umbral de distancia o una fórmula, como la del TTC más abajo) convierte ese resumen en la etiqueta. Como la regla es fija y se puede auditar, la etiqueta no depende del juicio de una persona ni del sistema que se evalúa.

AirSim expone un canal de profundidad planar (`DepthPlanar`, en metros por píxel) sin costo computacional adicional: es un buffer de renderizado que el motor ya produce como subproducto de la rasterización. Esto elimina la necesidad de instrumentación externa (LiDAR, cámaras estéreo calibradas, marcadores en escena) y permite construir etiquetas de referencia densas y de alta frecuencia en condiciones de vuelo idénticas a las de la misión real — la misma geometría, los mismos patrones de movimiento, el mismo ciclo de control (Shah et al., 2017; Jansen et al., 2023).

El TTC de referencia por celda se define como:

$$TTC_{\text{gt}}(\text{celda}) = \frac{\text{percentil}_{20}(Z_{\text{celda}})}{\max(v_{\text{cierre}},\, \varepsilon)}$$

donde Z_{celda} es la distribución de profundidades de los píxeles de la celda (el percentil 20 actúa como estimado conservador resistente a píxeles de fondo), y v_{cierre} es la componente de la velocidad del vehículo proyectada sobre el eje óptico (velocidad de cierre frontal positiva = aproximación), extraída de la telemetría del simulador. El término $\varepsilon = 0.01$ m/s evita la división por cero durante hover. Esta definición produce un TTC de referencia que es una función *observable y auditable* del estado físico del sistema en cada ciclo.

Una limitación de diseño de la etiqueta de referencia: la profundidad planar mide la distancia perpendicular al plano focal, no la distancia euclidiana al obstáculo más cercano en cada dirección de la cámara. Para obstáculos cercanos y cámaras con ángulo de campo amplio, esta diferencia no es negligible, pero dentro del rango de interés práctico (< 15 m) y con el campo visual horizontal de 90° de la cámara de AirSim, la discrepancia es inferior al 6 % dentro del tercio central (±15° del eje óptico, $1/\cos 15^\circ \approx 1.035$ en el eje horizontal y algo más en las esquinas). Se acepta como buena aproximación para la validación.

(Para el desarrollo formal del TTC como observable óptico directo τ, la formulación de divergencia y el cálculo por percentil 20 en la grilla 3×3, véase el Anexo 5, §A5.5–A5.6).

7.2 Diseño del conjunto de datos de validación

`experiments/collect_ttc_dataset.py` recolecta, por ciclo y por celda, el conjunto de variables listado en la tabla siguiente, generando registros JSONL en `runs/ttc/`:

Campo	Descripción
`ttc_est`	TTC estimado por `FlowTTCEstimator` (segundos)
`ttc_gt`	TTC de referencia por profundidad (segundos)
`occupancy`	Ocupación estimada por celda
`divergence_raw`	Divergencia bruta antes de escalar
`confidence`	Confianza de celda (fracción de píxeles válidos × `foe_confidence`)
`sector`, `band`	Identificación de celda en la grilla 3×3
`speed_mps`	Velocidad de traslación horizontal del vehículo (m/s)
`yaw_rate_rps`	Tasa de guiñada instantánea (rad/s)
`flow_algorithm`	Backend usado (`dis` o `farneback`)
`foe_confidence`	Confianza del FOE estimado (antes de multiplicar por fracción válida)

El conjunto se recolectó con `FLOW_DOWNSCALE_WIDTH` = 320 px, el valor por defecto del código; el vuelo usa 640 px (cap. 6, §6.3) y los umbrales de TTC no están recalibrados a ese ancho. Totaliza **3735 registros** de 9 celdas cada uno, cubriendo tres escenarios de vuelo scripteados diseñados para ejercitar aspectos específicos del estimador:

1. **Aproximación frontal a un edificio**: vuelo recto hacia una fachada a distintas velocidades (1–6 m/s); maximiza la señal de expansión en el sector central y verifica la correlación TTC estimado vs. TTC de referencia en condiciones ideales.
2. **Vuelo por un cañón urbano**: desplazamiento lateral entre dos edificios; activa los sectores izquierdo y derecho mientras el central permanece mayormente libre; verifica que la grilla detecta amenazas laterales sin generar falsos positivos en el centro.
3. **Giros de guiñada sin traslación**: rotación pura sobre el eje vertical, diseñada para ejercitar el guard de rotación (`FLOW_MAX_ROTATION_DEG`) y la derotación de guiñada. La expectativa correcta es que, durante el giro, el estimador declare incertidumbre (`foe_confidence = 0`) en lugar de reportar obstáculos espurios.

Limitación documentada del conjunto de datos: el escenario de giros de guiñada quedó concentrado en el rango $|\dot{\psi}| \in [0.0,\ 0.05)$ rad/s — velocidades de guiñada muy bajas, más próximas al crucero suave que a las maniobras activas de evasión (0.3–0.5 rad/s). Este rango no ejerce presión real sobre la derotación, y la validación de la derotación en giros agresivos queda aún pendiente (§7.5).

7.3 Resultados del canal de TTC: dos métricas, una interpretación correcta

El análisis en `experiments/analyze_ttc.py` produce dos números que, leídos de forma independiente, parecen contradictorios:

Métrica 1 — Correlación puntual: $r \approx -0.034$, error relativo mediano del 66%.

Métrica 2 — AUC ROC para detección binaria: 0.96–0.97 para el evento "colisión dentro de τ segundos" con $\tau \in \{1, 2, 3\}$.

La reconciliación entre estos dos números es el resultado estadístico central de este capítulo. Procede así:

¿Por qué la correlación puntual es baja? La correlación de Pearson mide si existe una relación *lineal* entre dos variables. Un TTC estimado de 4.8 s cuando el TTC real es 3.1 s (error puntual de 55%) contribuye igual a la correlación que si ese par estuviera perfectamente alineado *o no*. El estimador de flujo tiene errores de escala sistemáticos (el modelo de cámara pinhole asume movimiento rígido puro; los residuos de derotación, la profundidad desigual de objetos en la misma celda, y el efecto de la perspectiva en celdas no centrales introducen sesgos). Esos errores de escala no son aleatorios: son sistemáticos y predecibles. En consecuencia, la correlación puntual es baja, pero el *ordenamiento relativo* de los TTC estimados sigue siendo correcto: una celda con TTC estimado de 2.0 s es consistentemente más peligrosa que una con 5.0 s.

¿Por qué el AUC es alto? El AUC ROC mide exactamente la propiedad que la correlación no mide: la capacidad del estimador de *ordenar* los casos por riesgo. Un AUC de 0.96 significa que, si se escoge al azar una celda "peligrosa" (TTC real < τ) y una celda "segura" (TTC real ≥ τ), el estimador le asigna un TTC menor a la peligrosa con probabilidad 0.96 — la interpretación probabilística estándar del área bajo la curva ROC como estimador de la probabilidad de ordenamiento correcto entre un caso positivo y uno negativo elegidos al azar (Fawcett, 2006). Ese es el predicado exacto que el sistema de control necesita: no necesita saber que faltan 3.2 s para la colisión; necesita saber que *esta celda* es más peligrosa que *aquella*.

Formulación defendible del estimador: "El TTC estimado es un *puntaje de riesgo de cierre calibrado por curva ROC*, cuyo valor numérico tiene unidades de segundos por construcción matemática del modelo (§6.7) pero no interpretación métrica directa. Los umbrales operativos (`TTC_EVASION_THRESHOLD`, `TTC_SAFE_THRESHOLD`) son umbrales en esa escala de puntaje, derivados por criterio de Youden, no parámetros físicos independientes."

Esta distinción entre *clasificador de riesgo* y *cronómetro* se alinea con la literatura sobre sistemas de advertencia de colisión en visión monocular: Al-Kaff et al. (2017) y Vera-Yanez et al. (2024) reportan resultados similares — correlación puntual pobre pero detección binaria confiable — y argumentan que la métrica relevante para evasión es la tasa de verdaderos positivos antes de la zona de peligro, no la precisión de la estimación en segundos.

7.4 Derivación de umbrales operativos por índice de Youden

El índice de Youden ($J = \text{sensibilidad} + \text{especificidad} - 1$; Youden, 1950) es el criterio de selección de umbral óptimo que maximiza simultáneamente la tasa de detección de peligro real y la de descarte de falsos peligros — geométricamente, el punto de la curva ROC más alejado de la diagonal de clasificación aleatoria. Para cada τ se barrió el espacio de umbrales sobre la curva ROC y se encontró:

Tiempo de horizonte τ	Umbral de TTC (Youden)	Sensibilidad	Especificidad
1 s	~2.1 s	0.91	0.88
2 s	~3.18 s	0.93	0.89

Tiempo de horizonte τ	Umbral de TTC (Youden)	Sensibilidad	Especificidad
3 s	~4.58 s	0.90	0.91

El umbral para $\tau = 2$ s (3.18 s) es `TTC_EVASION_THRESHOLD` (redondeado a 3.2 s). El umbral para $\tau = 3$ s (4.58 s) es `TTC_SAFE_THRESHOLD` (redondeado a 4.6 s). Ambos umbrales son datos de validación — no fueron elegidos a mano ni ajustados empíricamente por comportamiento de vuelo. La elección de $\tau = 2$ s como umbral de evasión y $\tau = 3$ s como umbral de zona segura refleja el tiempo de reacción del lazo (5 Hz → 200 ms por ciclo) más el tiempo de ejecución de la maniobra de evasión (1–2 ciclos); a esa velocidad de crucero el margen de 2 s es suficiente para iniciar evasión antes del impacto en los escenarios ensayados.

Interpretación de la banda entre umbrales. La zona $TTC \in (3.2,\ 4.6)$ s corresponde al estado "advertencia" de la capa reactiva de `navigate` (cap. 5, §5.3): no hay bloqueo activo pero el riesgo es elevado; el sistema corrige lateralmente con `evasive_node`, sin esperar al lazo lento. La banda de histéresis entre evasión (3.2 s) y zona segura (4.6 s) previene el *chattering* — oscilación rápida entre maniobra de evasión y crucero normal cuando el TTC estimado flota alrededor del umbral de evasión.

7.5 Calibración del canal de ocupación (D2)

El canal de ocupación (divergencia escalada → `occupancy`) se valida contra profundidad con un experimento propio, D2, sobre un conjunto más chico que el del TTC.

Protocolo. Dataset capturado con `callibration-flight/d2_occupancy_capture.py`: 196 frames en TownSim frente a una pared sólida en (x = 61, y = −42), a cuatro velocidades de aproximación (0.5, 1, 2 y 3 m/s). Etiqueta de referencia binaria: `gt_depth_centro < 5.0 m` (29 frames positivos, 167 negativos). Curva ROC del predicado `occ_centro ≥ t` barriendo 500 umbrales (`d2_roc_analysis.py`) y umbral óptimo por índice de Youden, análogo al de §7.4.

Resultado. AUC = 0.87; umbral óptimo t = 0.011 con TPR = 0.931 y FPR = 0.222. El máximo de ocupación central observado en todo el dataset es 0.082, de modo que un umbral fijado a ojo como 0.35 tendría TPR = 0: **el canal de ocupación no se dispararía en ningún frame** y la detección recaería íntegramente en el TTC. Ambas cifras (AUC y umbral) se reproducen recalculando la curva sobre el archivo del dataset. `OBSTACLE_OCCUPANCY_BLOCKED` = 0.011 en `config/.env` y en el código.

Límites de la calibración. (i) Una sola escena y una sola geometría (una pared lisa), no la diversidad de vegetación y fachadas de los tiers 1 y 2; (ii) 29 positivos: el intervalo de confianza del umbral es amplio y no se estimó; (iii) FPR = 0.22 significa que aproximadamente uno de cada cinco frames sin obstáculo cercano vota bloqueo por ocupación, un costo de falsos positivos aceptado a cambio de la sensibilidad; (iv) los valores de ocupación dependen de la textura de la escena, por lo que un umbral fijo calibrado en TownSim puede ser inadecuado en CitySim. Para (iv), `main.py` incorpora un `OccupancyCalibrator` (§6.9) que mide el ruido de ocupación durante los primeros ciclos de cada misión (diseñado para ejecutarse con el dron quieto en zona despejada); su efecto no se validó en ningún experimento, y el runner de lotes no lo emplea. La validación cruzada del umbral en escenas distintas queda pendiente.

Métrica de validación	Canal TTC	Canal Ocupación
Protocolo de validación ROC	✓ Completo	✓ Realizado (dataset reducido)
Umbral derivado por Youden	✓ (`TTC_EVASION_THRESHOLD = 3.2 s`)	✓ (`OBSTACLE_OCCUPANCY_BLOCKED = 0.011`)
AUC reportada	0.96–0.97	0.87
Conjunto de datos	3735 registros	196 frames, una escena

7.6 Validación de la derotación en giros agresivos (D3)

El guard `FLOW_MAX_ROTATION_DEG = 2°` descarta todos los ciclos donde la rotación entre frames supera ese umbral (cap. 6). El conjunto de §7.2 no lo ejercita: el 97 % de los registros del escenario de guiñada cae en $|\dot{\psi}| < 0.05$ rad/s. El experimento D3 (`d3_derotation_capture.py`) lo ejercita con 250 frames frente a una pared sólida a 5 m de distancia constante, a cinco tasas de guiñada comandadas (0.0, 0.3, 0.5, 0.8 y 1.0 rad/s; 50 frames por tasa tras descartar 5 de estabilización).

Resultado. En las cuatro tasas no nulas (guiñadas reales medias de 18.7, 32.9, 51.4 y 63.9 °/s) el 100 % de los frames quedó con `source = "degraded"` y `foe_confidence = 0`: el estimador **se inhibe por sí mismo** y no entrega TTC contaminado con confianza alta. Coincide con el umbral teórico: 2°/frame a 5 Hz equivale a ~10 °/s (0.175 rad/s), y la primera tasa ensayada por encima de él (0.3 rad/s) ya produce confianza cero.

Alcance de la validación. Se validó la *inhibición*, no la *exactitud* de la derotación. Dos preguntas siguen abiertas: (i) no se ensayaron tasas entre 0 y 0.3 rad/s, de modo que no se midió el error de TTC en el tramo previo al guard ni se confirmó que 0.175 rad/s sea el punto de corte; (ii) la condición de referencia (0.0 rad/s) tampoco tiene traslación, por lo que su `foe_confidence` es también casi nula (media 0.005) y no permite contrastar con un caso de traslación sin giro; la reanudación correcta de la estimación una vez que la rotación decrece no se ejercitó. La estratificación del error relativo de TTC por bins de tasa de guiñada con los datos existentes sigue siendo informativa. Estas dos preguntas, junto con la validación en hardware real, quedan como trabajo pendiente.

7.7 Implicaciones para el diseño del sistema

Los resultados de validación tienen consecuencias directas sobre el diseño del lazo de control:

El AUC de 0.96–0.97 justifica usar TTC como señal primaria de evasión. Una tasa de verdaderos positivos del 93% con especificidad del 89% (a $\tau = 2$ s) es suficiente para evasión reactiva en entorno simulado; los falsos negativos (11%) son la fracción de situaciones peligrosas que el estimador no detecta a tiempo; la cubren en parte la detección de contacto (IMU y avance sin movimiento) y la capa estratégica con profundidad estimada, que no depende de la traslación (cap. 6, §6.15).

La baja correlación puntual justifica la banda de histéresis. Si el TTC fuera un cronómetro confiable, bastaría un solo umbral. La incertidumbre de escala hace necesaria la zona de advertencia (3.2–4.6 s), en la que el lazo rápido corrige sin comprometer una maniobra definitiva. La distancia en metros, que el TTC no da, la aporta en el lazo lento la profundidad estimada (cap. 6, §6.14).

La validación del canal de ocupación es parcial, y esa deuda es controlada pero no nula. Con el umbral de D2 el canal de ocupación actúa como detector temprano independiente del TTC, pero está calibrado con una escena y 29 positivos (§7.5). Antes de desplegarlo en entornos más exigentes (o en hardware real), la validación cruzada del umbral en escenas distintas debería completarse.

8. Ingeniería del VLM: percepción semántica y decisiones de implementación

El capítulo 5 documenta la arquitectura de la consulta asíncrona del lazo lento (cuándo se solicita, cómo se integra al grafo, qué hace el sistema con la respuesta). El capítulo 6 documenta por qué hace falta un lazo lento (los puntos ciegos del estimador de flujo). Este capítulo documenta las decisiones de ingeniería del VLM, que el lazo lento admite como fuente alternativa y que usa el planificador de ruta: selección del modelo bajo restricciones de hardware, salida estructurada, rol del modelo como perceptor semántico, ingeniería del prompt y gestión de la latencia.

> **Rol del VLM.** Con la configuración por defecto el VLM no interviene en el vuelo. Medido contra etiquetas de referencia, no distingue sectores volables de bloqueados ni rumbos abiertos de cerrados (cap. 11, §11.0). La capa estratégica y el barrido usan la profundidad estimada (cap. 6, §6.14), y el VLM trabaja en la planificación de ruta previa al vuelo (cap. 4, §4.3.1), la única tarea en la que aporta una señal medible. Lo que sigue describe la ingeniería del VLM, compartida por el planificador de ruta y por las fuentes alternativas del lazo lento, `STRATEGIC_SOURCE = vlm` y `DEADLOCK_STRATEGY = deep_vlm`.

8.1 Selección del modelo: restricciones de hardware como parámetro de diseño

La selección del modelo de lenguaje no es un proceso de *benchmark* abstracto sino un proceso de diseño con restricciones duras. El sistema corre en una RTX 5060 con 8 GB de VRAM compartidos con una simulación de Unreal Engine 5.5. AirSim sobre UE5.5 consume, en condiciones de vuelo activo, entre 4.5 y 6 GB de VRAM (Turco et al., 2024). El presupuesto efectivo disponible para la inferencia del modelo es de **2 a 3.5 GB de VRAM**, incluyendo la KV cache del historial de prompt.

Esta restricción excluye inmediatamente a los modelos de referencia del estado del arte (GPT-4o, Claude 3.5, Gemini 1.5 — todos en la nube o con hardware dedicado de 24–80 GB) y orienta la búsqueda hacia modelos multimodales compactos cuantizados (*Small Vision-Language Models*, VLM), del orden de 1.5 a 7 mil millones de parámetros, ejecutados localmente (LM Studio, Ollama o vLLM).

8.1.1 El requisito de visión directa

Una distinción de diseño previa a la selección del modelo es si la consulta al modelo debe usar un modelo textual puro (recibe solo telemetría y resúmenes de percepción serializados) o un VLM multimodal (recibe además el fotograma de la cámara). El sistema adopta un VLM multimodal por razones que no son de conveniencia sino de cobertura funcional:

- **Puntos ciegos del flujo óptico** (§6.1, §6.12): cuando el estimador de flujo carece de evidencia (hover, giro puro, baja textura), el `ObstacleField` entregado al modelo es esencialmente vacío. Un modelo textual no puede razonar sobre la escena sin ese resumen; un VLM puede observar directamente el fotograma y evaluar el contexto visual.

- **Semántica que el flujo no codifica**: la distinción entre "árbol vs. edificio", "sombra vs. obstáculo sólido", o "calle con objetos vs. calle vacía" no está presente en el campo de divergencia. El VLM puede discriminar escenas semánticamente similares en percepción pero distintas en consecuencias de navegación.
- **Geometría que el flujo no representa**: las superficies horizontales a la altura de vuelo (tablero de una autopista elevada, cornisas, balcones) no producen expansión en el sector frontal; el fotograma sí las muestra. Es el obstáculo sobre el que el dron puede quedar apoyado e inmovilizado (cap. 9, §9.8.7).

8.1.2 Modelo adoptado y candidatos evaluados

El modelo adoptado es **LFM2.5-VL-1.6B** (Liquid AI; cuantización Q4_K_M, formato GGUF, 731 MB más el codificador visual), servido por Ollama en la **misma estación y la misma GPU** que la simulación y el grafo de control, con un contexto limitado a 8192 tokens. En Ollama es el tag local `lfm2.5-vl-1.6b:q4km-ctx8k`, creado desde `hf.co/LiquidAI/LFM2.5-VL-1.6B-GGUF:Q4_K_M` con `num_ctx = 8192`, y es el valor de `LOCAL_LLM_MODEL_NAME` y `ROUTE_LLM_MODEL_NAME` en `config/.env` (Anexo 6). Corre en la GPU de la simulación porque un VLM a bordo comparte el cómputo con el resto del sistema de vuelo, y esa es la condición representativa del dron real. Con 8192 tokens de contexto el modelo ocupa ~1.6 GB de VRAM medidos (con el contexto por defecto de 128 k, 3.1 GB, casi todo caché de atención); la consulta más pesada del sistema, un barrido de cuatro imágenes, usa ~800 tokens de entrada. La variante *Extract* del mismo modelo, ajustada para extracción, no se usa: completa el esquema JSON sin mirar la imagen (la misma grilla en las 43 consultas de un piloto). Combina un núcleo de lenguaje de 1.2 B de parámetros con arquitectura híbrida de convoluciones (decodificación rápida) y un codificador visual SigLIP2 de 400 M que representa cada imagen con 64–256 tokens. Frente a **Qwen2.5-VL-3B-Instruct**, el otro candidato con visión que cabe en la VRAM disponible, gana por latencia: Qwen tarda ~10–12 s por consulta con imagen (§8.6) y no discrimina mejor (cap. 9, §9.8.3). En los *benchmarks* públicos de comprensión visual ambos están en el mismo rango; ninguno mide la tarea de vuelo, que la decide el banco de prueba (cap. 11, §11.0). Se usa la versión GGUF y no la conversión MLX porque esta última devuelve una respuesta vacía cuando la consulta pide a la vez salida JSON restringida y *logprobs*. LFM2.5-VL-1.6B también es el modelo del planificador de ruta previo al vuelo (cap. 4, §4.3.1); Qwen2.5-VL-3B es el segundo modelo del banco de prueba (§10.13.4). Los modelos candidatos (documentados en `informe/anexos/A1-EXPLORACION-SLM-GGUF.md`) son:

Modelo	Tamaño (cuant.)	VRAM aprox.	Descartado por
Phi-4-mini-Instruct	~2.5 GB	~2–3 GB	Sin capacidad de visión nativa
Qwen3-4B-Instruct	~3 GB	~2.5–3.5 GB	Sin soporte visual en la rama 3B
SmolLM3-3B	~2 GB	~1.8 GB	Sin soporte visual; razonamiento débil
LFM2.5-VL-1.6B (Q4_K_M, GGUF)	0.73 GB + codificador visual	**~1.6 GB** (contexto 8 k)	**Adoptado**

Modelo	Tamaño (cuant.)	VRAM aprox.	Descartado por
Qwen2.5-VL-3B-Instruct (Q4_K_M)	~2 GB	~2.0 GB	~10–12 s por consulta con imagen, sin mejor discriminación (§8.1.2)
Qwen2.5-VL-7B-Instruct	~4.5 GB	~4–5 GB	VRAM insuficiente con UE5.5 activo

El criterio de selección es: **(a)** soporte de visión nativa, **(b)** memoria < 2.5 GB cuantizado, **(c)** soporte de `response_format: json_schema` en el backend de inferencia, **(d)** latencia de inferencia < 2 s en el hardware disponible para prompts de ~500 tokens con dos imágenes. Es un criterio de diseño; las latencias medidas se documentan en §8.6 (el barrido de cuatro imágenes lo supera).

8.1.3 Posicionamiento respecto al estado del arte

La arquitectura de lazo cerrado estado→VLM→macro-acción→ejecución no es, hacia 2025–2026, una contribución conceptualmente novedosa: es un patrón estandarizado en investigación y prototipos de UAVs controlados por modelos de lenguaje (véase la discusión en `informe/anexos/A1-EXPLORACION-SLM-GGUF.md`). Hwang et al. (2024) en EMMA y Saxena et al. (2025) en UAV-VLN son trabajos representativos del patrón. El aporte de esta tesis no está en la novedad de la arquitectura general, sino en la ingeniería de su interfaz con la percepción monocular (cap. 6) y en la caracterización sistemática de sus modos de falla (cap. 9).

8.2 Salida estructurada: decodificación restringida y parser

Sin restricciones sobre la salida, la causa más frecuente de fallo de la consulta al VLM no es una decisión incorrecta, sino una respuesta JSON malformada que rompe el parser. La salida se asegura en dos capas.

8.2.1 Decodificación restringida (`json_schema`)

La respuesta se solicita con `response_format={"type": "json_schema", "json_schema": {...}}`, disponible en LM Studio y en el backend OpenAI-compatible de Ollama. Durante la generación, el motor enmascara los logits de los tokens que producirían JSON inválido para el esquema, de modo que el modelo solo muestrea tokens válidos. El mecanismo —reformular la gramática del esquema como una máquina de estados finitos que restringe el conjunto de tokens admisibles en cada paso— es el que formalizan Willard y Louf (2023), base de la decodificación guiada que exponen hoy backends como llama.cpp; Raspanti et al. (2025) y Geng et al. (2025) evalúan empíricamente su impacto sobre la fiabilidad de la salida (desarrollo formal en el Anexo 4). Con la decodificación activa, el `adherence_rate` medido en escenarios de validación pasa de ~73 % (solo prompt) a ~98 %.

Hay **dos esquemas**, uno por modo de consulta (§5.10.5), y ninguno contiene un campo de acción: el modelo describe, el código traduce.

Esquema estratégico (`RESPONSE_JSON_SCHEMA_STRATEGIC`, `vlm_strategic.py`): una entrada por sector de la grilla de 3×3 del fotograma con valor `free` o `blocked` (enum cerrado); los nueve sectores son obligatorios. El parser los traduce a los valores internos `libre` y `bloqueado`, que son los que usan la sub-meta, los registros y el banco de prueba.

```
{
  "type": "object",
  "properties": {
    "cells": {
      "type": "object",
      "properties": {"A1": {"enum": ["free", "blocked"]}, "B1": {...}, ..., "C3": {...}},
      "required": ["A1", "B1", "C1", "A2", "B2", "C2", "A3", "B3", "C3"],
      "additionalProperties": false
    }
  },
  "required": ["cells"],
  "additionalProperties": false
}
```

Esquema panorámico (`RESPONSE_JSON_SCHEMA_PANORAMA`, `deep_scan.py`): un array `views` de objetos `{img, view, free}`: número de imagen, lo que hay delante a la altura del dron (enum de seis valores: `open`, `facade`, `wall`, `vegetation`, `inside`, `unclear`) y si se puede volar recto 15 m. El parser los traduce a los valores internos (`libre`, `fachada`, `muro`, `vegetacion`, `interior`, `indeterminado`; `transitable`). El esquema no pide certeza ni un indicador de imagen degradada: el modelo los rellena con valores fijos (el mismo `0.95, 0.92, 0.89, 0.86` en tres de cuatro barridos de un piloto; `degradada: true` en todas las respuestas) y ninguno decide nada. Una entrada `free: true` cuya vista no es `open` se contradice a sí misma y no cuenta como transitable (cap. 5, §5.12). Las imágenes se identifican por número y no por ángulo: el rumbo real de cada imagen lo conoce el código, no el modelo (§8.3.3).

8.2.2 Parser como red de seguridad

La garantía de `json_schema` depende de que el backend la soporte. Si la llamada con esquema falla, `vlm_client._query_slm_impl()` reintenta en modo libre; en ambos casos `_extract_json_object()` elimina cercos de markdown y extrae el primer objeto `{...}` del texto, y el parser del modo (`parse_strategic` o `parse_panorama_description`) lo valida: una respuesta estratégica con algún sector ausente o con un valor fuera del enum se rechaza completa, y una respuesta panorámica sin array `views` también. Una respuesta rechazada **no se sustituye por una decisión determinista**: la capa estratégica simplemente no propone sub-meta en ese ciclo (motivo `no_parseable`), y un barrido sin respuesta válida cae al escape vertical de §5.3.4 como cualquier otra falla del modelo.

8.3 Rol del VLM: percepción semántica anclada al mundo

El VLM **no elige macro-acciones**: describe lo que ve, y la capa de navegación convierte esa descripción en geometría. Lo que se le pregunta está elegido para que la respuesta siga siendo válida cuando llega.

8.3.1 Por qué esta pregunta

Una interfaz aparentemente más simple —pedir al modelo que describa tres sectores relativos al cuerpo (`frente`, `izquierda`, `derecha`)— tiene dos problemas, medidos en corridas de diagnóstico con esa interfaz (cap. 9, §9.8):

1. **Latencia.** Una respuesta con imagen tarda varios ciclos del lazo (cuatro con el modelo local; 5.3 s de mediana, ~26 ciclos, en un servidor compartido). El «frente» de la respuesta ya no es el frente del dron.
2. **Contenido informativo.** En 52 de 53 respuestas sectoriales (excluida una corrida en vegetación) el modelo declaró el frente transitable, incluso cuando la profundidad de referencia del simulador, consultada a posteriori, marcaba una superficie a 0.1–1 m. La respuesta es casi constante.

El diseño responde al primero anclando la respuesta a la pose del fotograma (§8.3.2) y, al segundo, con más granularidad (una grilla de 3×3, con la meta marcada en la imagen) y un prompt sin ejemplos de valores concretos que un modelo pequeño pueda copiar (§8.5.1). El banco de prueba responde si eso alcanza: no alcanza. Con LFM2.5-VL-1.6B la grilla queda en 53 % de exactitud balanceada (AUC 0.53), con las filas de arriba y de abajo «libres» en todas las muestras y la del medio dependiente de la posición del sector más que de la escena (cap. 11, §11.0).

8.3.2 Capa estratégica: grilla de 3×3 con dirección absoluta

Un recorte cuadrado del centro del fotograma frontal —se cortan los dos costados del ancho— se divide en una grilla de 3×3 por tercios —la grilla de composición fotográfica—: columnas A, B, C (izquierda a derecha) y filas 1, 2, 3 (arriba, a la altura del dron, abajo). Se marca en la imagen la dirección del destino, en azimut y elevación. Para cada sector el modelo dice si se puede volar en esa dirección al menos 15 m.

Una partición en columnas solo describe el eje horizontal. La grilla agrega el vertical: permite que la respuesta diga que un obstáculo a la altura del dron se pasa por arriba (fila 1 libre, fila 2 bloqueada) y que una superficie horizontal se ve en la fila del medio, sin pedir al modelo un concepto abstracto como «estructura debajo». Una pregunta de ese tipo («¿hay una estructura horizontal cercana?») recibe `true` en 5 de 6 consultas de diagnóstico, con imágenes muy distintas (cap. 9, §9.8.9).

La traducción (`decide_subgoal`, detalle en §5.10.3) usa la pose **del fotograma**: la dirección absoluta del sector es el yaw del ancla más el azimut del centro del sector, con la elevación del centro del sector. La sub-meta se fija desde la posición del ancla, a no más de 15 m y nunca más lejos que el waypoint real, con la misma función que usa el barrido. El recorte cuadrado deja los sectores laterales y los verticales a la misma distancia angular del centro (±24°), de modo que la preferencia entre un rodeo por el costado y uno por arriba no la decide la forma del cuadro: a igual distancia gana la fila del medio. La igualdad se toma con un margen de 10° (`VLM_SUBGOAL_TIE_DEG`): con empate exacto, una meta medio grado por debajo del dron haría ganar siempre a la fila de abajo, y el dron podría descender hasta apoyarse en el tablero de una autopista elevada (cap. 11, §11.0). Así, la geometría que el modelo describió se transporta al mundo sin depender de hacia dónde mira el dron cuando la respuesta llega. Si el sector de la meta está libre, la capa no agrega nada y descarta cualquier desvío pendiente: la respuesta más frecuente esperable («el camino directo está libre») no genera maniobras.

8.3.3 Barrido: imágenes numeradas, rumbos medidos

En el barrido de resolución de atasco (§5.12), el modelo recibe cuatro imágenes numeradas y describe cada una (`img`, `view`, `free`). El código asigna a cada imagen el yaw **medido** en el ciclo en que se capturó y elige, entre las transitables, la de rumbo absoluto más cercano al rumbo absoluto hacia el waypoint real. Las entradas se asignan por número de imagen (o por orden si falta), y las entradas que el modelo inventa de más se descartan.

El diseño evita un error de marco de referencia documentado en el capítulo 9 (§9.8.1): si las imágenes se etiquetan con su rumbo absoluto y se pide al modelo un ángulo relativo, el modelo copia la etiqueta y el código la interpreta como relativa, de modo que «la dirección hacia la meta está libre» termina ejecutándose como una evasión hacia el lado contrario.

8.3.4 Vocabulario de macro-acciones

El vocabulario del sistema es fijo (`VALID_ACTIONS`, `action_map.py`): `MANTENER_RUMBO`, `EVADIR_IZQUIERDA`, `EVADIR_DERECHA`, `GANAR_ALTURA`, `PERDER_ALTURA`, `FRENAR`, `GIRAR_90` y `RETROCEDER`. El modelo no elige ninguna. Las descripciones del VLM producen, a través del código, `MANTENER_RUMBO` hacia una sub-meta, `GANAR_ALTURA` o `PERDER_ALTURA` (barrido sin rumbos transitables); el resto las emite el lazo rápido (evasión, `GIRAR_90`) o la FSM. `RETROCEDER` está definido pero ningún comportamiento del brazo `slm` lo emite (cap. 5, §5.3.4).

8.4 `action_to_command`: la frontera entre decisión y cinemática

Cada macro-acción se traduce a un comando cinemático en `action_to_command()` (`src/agents/action_map.py`), función compartida por el lazo rápido, el barrido y la FSM (tabla completa en cap. 5, §5.14). Valores relevantes: evasión lateral con `yaw_rate` ±15°/s y `target_yaw` redondeado a la cuadrícula de 90°, `vx` de 1.2 m/s en la evasión agresiva del lazo rápido; `GANAR_ALTURA` a −1.5 m/s en el lugar; `PERDER_ALTURA` a +0.8 m/s con avance de 1.0 m/s; `GIRAR_90` a ±20°/s sin traslación; `RETROCEDER` a −1.2 m/s con rumbo congelado.

Invariante de diseño. Centralizar la traducción impide que una misma macro-acción tenga cinemáticas distintas según quién la active. Si cada brazo tuviera su propia tabla de velocidades, divergirían sutilmente (por ejemplo, distinta velocidad vertical para `GANAR_ALTURA`) y producirían diferencias de trayectoria entre brazos que no provienen de la decisión.

8.5 Ingeniería del prompt

8.5.1 Principios

Los prompts siguen cinco reglas:

1. **Solo hechos, ninguna sugerencia de acción.** Un prompt que pide al modelo «describir sin decidir» y a la vez le sugiere qué maniobra priorizar es contradictorio (cap. 9, §9.8.3). Los prompts dan solo lo que el modelo no puede ver en la imagen (distancia al destino, sector de la meta, altura) y piden una descripción.
2. **Sin ejemplos.** El esquema JSON ya fija la forma de la respuesta, y un modelo pequeño copia los valores de cualquier ejemplo (`"ok": true, "conf": 0.9`) en sus respuestas (cap. 9, §9.8.3).
3. **La geometría la aporta el código.** El modelo nunca reporta ángulos ni posiciones: identifica sectores o imágenes por su etiqueta, y el código conoce su dirección.
4. **Lo visual en la imagen, no en el texto.** La meta se marca **sobre** el fotograma (marca roja «GOAL» en su dirección, dentro de la grilla) en lugar de describirse como «X grados a la derecha».
5. **En inglés y breves.** Los modelos de este tamaño se ajustan a instrucciones mayoritariamente en inglés, y cada token del prompt se procesa en cada consulta. Cada prompt de sistema tiene tres o cuatro oraciones: qué ve el modelo, qué debe responder con una definición operativa de una línea y «solo JSON». Los valores del esquema también están en inglés; el parser los traduce a los valores internos. Ningún prompt nombra objetos de un escenario en particular (cap. 10, §10.12.2).

8.5.2 Prompt de la capa estratégica

System prompt (`SYSTEM_PROMPT_STRATEGIC`), completo:

> You see the front camera of a drone flying low in a city, split into a 3×3 grid: columns A B C (left to right), rows 1 2 3 (top, drone height, bottom). The red GOAL mark is the destination. For each cell: "free" if the drone can fly 15 m that way, "blocked" if a building, wall, tree, bridge or roof is closer than 15 m. Reply with JSON only.

User prompt (`strategic_prompt`): `Goal WP_3: 34 m, in cell B2. Altitude 10 m.` Si la meta está fuera del recorte: `Goal WP_3: 34 m, outside the image, 97 deg to the left (GOAL mark on the edge). Altitude 10 m.` Le sigue una única imagen de 384 px con la grilla y la marca de la meta dibujadas, rotulada `[Front camera, 3x3 grid]`.

8.5.3 Prompt del barrido

System prompt (`SYSTEM_PROMPT_DEEP_SCAN`), completo:

> You see numbered images from a drone flying low in a city; it turned in place and each image faces a different direction. For each image give: - view: what is ahead at drone height within 15 m: open (street, square, sky), facade, wall, vegetation, inside (dark or uniform, inside a building) or unclear. - free: true only if the drone can fly straight 15 m without hitting anything. Reply with JSON only.

User prompt (`_build_panorama_prompt`): `4 images. Altitude 10 m.`, seguido de hasta cinco imágenes de 256 px rotuladas `[Image 1]` … `[Image N]`. El prompt no da ángulos, meta ni duración del atasco: el modelo solo describe cada imagen, y el rumbo de cada una y la elección los resuelve el código (cap. 5, §5.12).

8.5.4 Parámetros de inferencia

`temperature = 0.2`; tope de 384 tokens en el modo estratégico (nueve valores de enum) y 512 en el barrido. El tope no fija la longitud de la respuesta —con decodificación restringida, la generación termina al cerrar el esquema—: solo corta una generación desbocada, y una respuesta cortada se reporta como falla (`respuesta_truncada`), nunca se interpreta. Las respuestas no tienen campos de texto libre: no hay razonamiento que generar, lo que acota la latencia de decodificación.

Los system prompts están en el código (`vlm_strategic.py`, `deep_scan.py`).

8.6 Gestión de latencia

El tiempo de inferencia del VLM es un orden de magnitud mayor que el período del lazo (200 ms). La consulta corre en el hilo de `DeliberationService` (§5.10.5) y nunca bloquea al grafo. Esta separación entre una capa de ejecución rápida que siempre responde y una capa de razonamiento lento desacoplada es la instancia concreta, con un VLM como capa deliberativa, del patrón de arquitecturas de tres capas (Gat, 1998).

Latencia observada. LFM2.5-VL-1.6B (Q4_K_M), en la misma GPU que la simulación:

Fuente	Condición	Latencia
Piloto con `STRATEGIC_SOURCE = vlm` (65 consultas, sin *timeouts*)	Capa estratégica, un fotograma de 384 px	mediana 0.77 s · p90 0.80 s · máx 2.8 s
Banco de prueba (cap. 11, §11.0)	Grilla 3×3, un fotograma	0.88 s
Banco de prueba	Barrido de cuatro imágenes de 256 px	2.6–3.1 s

No provienen de una medición controlada que aísle carga de GPU, tamaño de imagen y arranque en frío. Como referencia, el candidato Qwen2.5-VL-3B (§8.1.2) tarda 5.3 s de mediana con un fotograma (p95 12.9 s, 73 consultas) y 5.6 s con un barrido de dos imágenes en las corridas de diagnóstico del capítulo 9. La consecuencia de diseño es la misma para ambos: ninguna latencia cabe en el ciclo de 200 ms; en 0.8 s el dron avanza unos metros y gira, y en 5 s recorre ~10 m y puede girar más de 90°.

Consecuencia para el comportamiento del dron.

- **Capa estratégica.** El dron **no se frena**: el lazo rápido sigue volando y la respuesta, cuando llega, se traduce con la pose del fotograma (§5.10.3). Mantener el avance preserva además la traslación que el flujo óptico necesita.
- **Barrido de resolución de atasco.** El dron permanece en el lugar mientras espera, bajo el watchdog `SLM_DEEP_WATCHDOG_MS` (18 s en producción, muy por encima de la latencia medida del barrido y por debajo del corte HTTP de 20 s). Al expirar, cae al escape vertical.

Validez de las respuestas. La validez se controla geométricamente, no solo por edad: la respuesta se descarta si tiene más de 10 s, si cambió el waypoint o si el dron ya superó la sub-meta que produciría; dentro de esos límites, girar mientras el modelo piensa no invalida la respuesta, porque está anclada al fotograma.

8.7 Nombres de campo compactos

Cada token de la respuesta cuesta tiempo de decodificación: en un servidor que genera ~15 tokens/s, ~67 ms por token. El esquema panorámico usa tres campos cortos por imagen (`img`, `view`, `free`), sin certeza, sin indicador de imagen degradada y sin campo de razonamiento; con nombres descriptivos (`relativo_deg`, `transitable`, `confianza`) una respuesta de cuatro imágenes llega a ~80–95 tokens. El esquema estratégico lleva la idea al extremo: nueve valores de enum y ningún texto libre. Los parsers aceptan también los nombres largos y normalizan a los nombres internos.

8.8 Nota: LoRA como alternativa explorada y no adoptada

El fine-tuning ligero vía LoRA (*Low-Rank Adaptation*) es una vía posible de especialización del modelo para la tarea de este sistema (Anexo 3). El argumento a favor es que un conjunto de pares (imagen, respuesta correcta) derivados de las corridas del sistema podría reducir la frecuencia de respuestas erróneas sin reentrenar el modelo base.

No se implementa por dos razones:

1. **Cobertura de datos insuficiente.** Las corridas del sistema contienen situaciones de evasión relativamente homogéneas. Un LoRA entrenado sobre ese conjunto podría sobreajustar a los escenarios vistos y degradar la generalización en configuraciones urbanas no ensayadas.

2. **Complejidad de mantenimiento.** Un adaptador LoRA es un artefacto de entrenamiento que requiere ser versionado, evaluado y actualizado junto con el modelo base. Para el alcance de esta tesis, el beneficio esperado no justifica el costo de infraestructura. Se deja constancia explícita de que es una alternativa explorada y no adoptada por decisión de alcance, no por omisión.

9. Modos de falla en lazos de control híbridos con LLM

9.1 El argumento central: contratos de interfaz como vector de falla primario

En sistemas robóticos que combinan percepción clásica, orquestación de grafo de control y un modelo de lenguaje, la intuición natural sobre modos de falla apunta al modelo: respuestas alucinadas, razonamiento incorrecto, formato inválido. La auditoría de este sistema señala en una dirección diferente: los fallos más costosos —en términos de horas de diagnóstico y de consecuencias en vuelo— ocurren en los **contratos de interfaz entre componentes**, no en el razonamiento del modelo.

Un modelo de lenguaje tiene una propiedad que lo hace especialmente peligroso en este contexto: raciocina sobre lo que recibe, no sobre la realidad. Si recibe un resumen de percepción que dice "camino despejado", produce una respuesta consistente con esa premisa. No tiene acceso privilegiado a la verdad del mundo — solo al texto que se le entrega. El corolario es que cualquier corrupción, omisión o malinterpretación en los datos que llegan al modelo produce decisiones estructuralmente correctas pero factualmente equivocadas, sin que el modelo (ni el sistema) lo detecte como error. El sistema "funciona" — no arroja excepciones, no se cae, produce salida JSON válida — mientras toma decisiones sistemáticamente incorrectas. Esta propiedad no es una idiosincrasia de este sistema: es la manifestación, en un lazo de control robótico, del problema general de las respuestas fluidas y plausibles pero no fundamentadas en la entrada real (*hallucination*) documentado extensamente en la literatura de generación de lenguaje natural (Ji et al., 2023), agravado aquí por el hecho de que la salida del modelo no es texto para un lector humano crítico, sino un comando que se ejecuta directamente sobre un vehículo físico.

Este capítulo documenta seis familias de fallas estructurales, observadas en corridas de diagnóstico de la arquitectura y organizadas por la capa del sistema donde se originan. Para cada una se describe el mecanismo, la evidencia que lo muestra y el mecanismo de contención que tiene el sistema. La tabla siguiente resume la taxonomía:

#	Categoría	Capa de origen	¿Genera excepción?	Consecuencia en vuelo
1	Schema drift / campo ciego	Contrato percepción→control	No	Navegación a ciegas hacia obstáculos
2	Desalineación temporal de frames	Buffer de historial visual	No	FOE/movimiento inferido incorrectamente
3	Descalibración de escala en divergencia	Operador diferencial en flujo	No	Saturación del canal de ocupación

#	Categoría	Capa de origen	¿Genera excepción?	Consecuencia en vuelo
4	State clipping + ciclos límite en LangGraph	Orquestación de grafo	No	Ascenso acumulado de >350 m
5	Degradación sensorial silenciosa	Captura de imagen y telemetría	No	Deliberación sobre datos corruptos
6	Integración del VLM: marco de referencia, defaults silenciosos y reglas apiladas	Interfaz VLM → control	No	Respuestas correctas del modelo ejecutadas al revés o ignoradas

El denominador común en los seis casos es la ausencia de excepción en tiempo de ejecución. Este resultado metodológico se discute en §9.7.

9.2 Falla 1 — Schema drift: el campo nulo leído como ausencia de peligro

9.2.1 Descripción del modo de falla

Si el estado del campo de obstáculos se comunica como una lista de objetos detectados (`detected_obstacles: List[dict]`), con una lista vacía como valor por defecto, la lista vacía es ambigua: puede significar "la percepción corrió y no detectó nada" (espacio despejado) o "la percepción no corrió, no tiene confianza, o falló silenciosamente" (ausencia de información).

Los consumidores —la capa de decisión, el constructor del prompt del VLM y la FSM— tienden a resolver esa ambigüedad en favor de la primera interpretación. En una configuración de diagnóstico con ese contrato, la cadena fue:

1. El generador de contexto construía un prompt que afirmaba al modelo "trayectoria frontal sin obstáculos detectados".
2. El VLM, con perfecta coherencia lógica respecto de la información provista, respondía que se podía seguir con alta confianza.
3. El lazo ejecutaba la maniobra de crucero normal.
4. No se generaba ninguna excepción ni alerta de log, porque desde la perspectiva del código la lista vacía es un valor válido.

Este patrón es una instancia de un anti-patrón de diseño general — tratar el valor neutro de un tipo de dato (lista vacía, cero, `None`) como evidencia de ausencia de fenómeno, en lugar de como ausencia de información — que colapsa dos estados epistémicos distintos en una única representación. En señales de seguridad la distinción es crítica y la convención correcta es la inversa: la ausencia de evidencia no es evidencia de ausencia de peligro, principio formulado explícitamente para la interpretación de resultados nulos en Altman y Bland (1995) y que aquí se aplica a un contrato de datos entre percepción y control en lugar de a la interpretación estadística de un experimento.

9.2.2 Mecanismo de contención

El contrato de percepción es el tipo `ObstacleField` (cap. 6): un objeto inmutable que distingue explícitamente tres estados epistémicos:

- `source = "flow"`, `foe_confidence > 0`: la percepción corrió con evidencia real. Los valores de TTC y ocupación son interpretables.
- `source = "flow"`, `foe_confidence = 0`: la percepción corrió pero no produjo evidencia confiable (FOE fuera de imagen, menos de 30 inliers). Interpretable como "sin información".
- `source = "degraded"`: la percepción fue inhibida activamente (guard de rotación, modo degradado, primer ciclo). El campo no contiene ninguna información sobre el mundo.

La invariante que impone `ObstacleField` es que **ningún consumidor puede confundir "sin obstáculos detectados" con "sin información"** sin leer explícitamente `has_evidence()`. La falta de evidencia tiene consecuencias explícitas: el gobernador de velocidad limita el avance tras varios ciclos sin evidencia (cap. 5, §5.13) y `has_open_corridor()` no declara corredor libre sin evidencia (cap. 6, §6.11). La capa estratégica no recibe el campo: mira el fotograma directamente (cap. 5, §5.10).

9.3 Falla 2 — Desalineación temporal del historial de frames

9.3.1 Descripción del modo de falla

Una consulta multimodal que envía al VLM un historial de dos frames (t y $t-1$), para que el modelo estime la dirección de movimiento, depende de que los dos frames sean genuinamente distintos en el tiempo. Un buffer implementado como lista simple, actualizado sin verificar timestamps, produce frames duplicados o fuera de contexto en dos situaciones:

1. **Cambio de waypoint:** si al saltar al siguiente waypoint el nodo de captura inyecta el último frame del tramo anterior como "t-1", el VLM recibe el frame de un punto de la misión completamente distinto como contexto temporal inmediato.
2. **Ciclos sin frame nuevo:** si AirSim no responde a tiempo (picos de carga de GPU), el buffer retiene el frame anterior, y bajo las etiquetas `[Fotograma t-1]` y `[Fotograma t]` aparece la misma imagen.

Con dos frames idénticos, el modelo infiere que el dron está estático o que nada se acerca. En ambos casos, la deliberación sobre movimiento y peligro es incorrecta de forma sistemática.

9.3.2 Mecanismo de contención

El sistema no envía historial temporal al VLM: `VLM_FRAME_HISTORY_SIZE` vale 1, y la capa estratégica usa el fotograma del ciclo y el barrido un fotograma por rumbo, cada uno con su timestamp y su pose de captura (cap. 5, §5.4, §5.10, §5.12). El riesgo equivalente pasa a ser espacial: enviar al modelo una etiqueta de orientación que no corresponde al fotograma (§9.8.1), y se contiene anclando cada respuesta a la pose medida del fotograma que la originó.

9.4 Falla 3 — Descalibración de escala en el canal de divergencia

9.4.1 Descripción del modo de falla

El canal de divergencia del `ObstacleField` (cap. 6, §6.7) estima $\partial u/\partial x + \partial v/\partial y$ sobre el campo de flujo. Un cálculo con gradientes de Sobel 3×3 sin el factor de normalización $1/8$, que el kernel estándar requiere para ser un estimador de derivada de primer orden en unidades de píxel-por-píxel,

$$K_{\text{Sobel}} = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

produce divergencias aproximadamente 8 veces mayores que las correctas. Con un factor de ocupación ajustado sobre esa escala inflada, el canal de ocupación satura a `1.0` en condiciones de textura moderada — es decir, prácticamente siempre que haya cualquier gradiente de flujo en la imagen, incluso en vuelo en espacio abierto sin obstáculos cercanos.

La consecuencia operativa es que el predicado disyuntivo `is_blocked()` (§6.9) queda dominado completamente por el canal de ocupación: aunque el TTC indique una zona segura (> 4 s), la ocupación saturada vota bloqueo, y el canal de TTC calibrado queda efectivamente anulado.

9.4.2 Interacción con la lógica disyuntiva

Este modo de falla ilustra un problema de diseño de predicados disyuntivos en sistemas con múltiples canales de diferente confianza. La lógica $\text{is\_blocked} = (\text{occ} \geq t_{occ}) \vee (\text{conf} \geq 0.35 \wedge \text{ttc} \leq 2.5)$ asume que ambos canales están calibrados en escalas comparables. Si un canal está sistemáticamente sobre-estimado, la disyunción lo convierte en el canal dominante y anula la información del otro. El TTC, que tiene validación empírica (cap. 7), pierde toda influencia frente a una ocupación descalibrada. El caso inverso también ocurre: un umbral de ocupación demasiado alto para la escala real del canal lo deja mudo y la detección recae solo en el TTC (cap. 7, §7.5).

9.4.3 Mecanismo de contención

La contención tiene tres partes: 1. La divergencia se calcula con `np.gradient()` (diferencias centradas de segundo orden, normalizadas intrínsecamente). 2. El umbral de ocupación está calibrado contra profundidad de referencia por curva ROC (cap. 7, §7.5: `OBSTACLE_OCCUPANCY_BLOCKED` = 0.011). 3. Los dos canales tienen umbrales de confianza diferenciados (`MIN_CONFIDENCE_FOR_BLOCKED = 0.15` vs. `MIN_CONFIDENCE_FOR_TTC_BLOCKED = 0.35`), de modo que el canal de TTC requiere mayor confianza para votar solo que el canal de ocupación acompañado de confianza mínima.

La tercera medida es especialmente importante: la normalización corrige la escala, pero el sistema necesita ser robusto a recalibraciones futuras. Los umbrales diferenciados hacen que ningún canal pueda dominar completamente al otro sin confianza suficiente.

9.5 Falla 4 — State clipping y ciclos límite en LangGraph

Esta familia es la más costosa en consecuencias observadas (un ascenso acumulado de ~356 m en una sola corrida de diagnóstico) y la más ilustrativa de los riesgos de orquestadores de grafos que descartan silenciosamente estado no declarado.

9.5.1 El mecanismo de descarte silencioso de LangGraph

LangGraph construye los canales de estado de un `StateGraph` a partir de un `TypedDict` declarado en tiempo de compilación (`build_workflow()`). En cada invocación del grafo (`graph.invoke(state)`), el runtime fusiona el estado de salida de cada nodo con el estado entrante usando únicamente las claves presentes en el `TypedDict`. Cualquier clave que un nodo escriba en su diccionario de retorno pero que no esté declarada es **descartada silenciosamente** — sin excepción, sin advertencia, sin registro en el log. El nodo siguiente recibe un estado donde esa clave no existe o tiene el valor por defecto del último ciclo en que sí estaba.

El mecanismo específico importa: el `TypedDict` de Python es una anotación de tipos, no una estructura de datos con validación en tiempo de ejecución. `state["clave_nueva"] = valor` no falla aunque `clave_nueva` no esté en el `TypedDict`; simplemente el runtime de LangGraph no la persiste en el canal de estado entre invocaciones (cap. 5, §5.2).

9.5.2 Cuatro vulnerabilidades

Vulnerabilidad A — Claves de control no declaradas.

En corridas de diagnóstico, tres claves de control cruzaban la frontera entre nodo y lazo sin estar declaradas en `DroneState`:

- `_reset_stall_counter`: debía reiniciar el contador de progreso estancado al llegar a un waypoint. No declarada, el contador crece monótonamente desde el inicio de la misión, independientemente del progreso real.

- `_delib_memory`: debía acumular un historial corto de resultados de la consulta al VLM. No declarada, el historial está siempre vacío y el VLM se consulta sin contexto de lo que decidió un ciclo antes.
- `_corner_wp`: debía inyectar un sub-waypoint de esquina para la maniobra `GIRAR_90`. No declarada, el decisor la lee en el ciclo siguiente como el valor inicial (`None`), nunca como el punto calculado.

Ningún log de nivel de aplicación muestra el descarte; diagnosticarlo exige comparar el estado completo antes y después de cada invocación.

Vulnerabilidad B — Ciclo límite de período 3 en la red de seguridad.

Un mecanismo de escape que se resetea a sí mismo,

```
si stall_count > hard_threshold:
    activar FRENAR (ciclo 1)
    si stall_count > hard_threshold + N:
        resetear stall_count = 0  ← la red de seguridad se resetea a sí misma
```

convierte el estado "atasco severo → acción de escape" en un estado transitorio: al ciclo siguiente, `stall_count = 0 < hard_threshold` y el sistema vuelve a crucero; tres ciclos después, el atasco se re-detecta y el ciclo se repite. El período 3 (crucero → pre-stall → escape → reseteo → crucero) no es observable como un error: en el visor se ve como "el dron frena ocasionalmente sin razón aparente".

Vulnerabilidad C — Métrica de progreso auto-inconsistente.

Un predicado de atasco `ciclos_sin_progreso > effective_stall_threshold`, con `effective_stall_threshold = max(10, eps_m / (min_speed / loop_hz))`, combina `eps_m` (distancia mínima de avance por ciclo) y `min_speed` (velocidad mínima de crucero), definidos en unidades y archivos distintos. Con `eps_m = 0.5 m`, `min_speed = 2.0 m/s` y `loop_hz = 5 Hz`, la velocidad de avance implícita es 2.5 m/s, mientras que en un giro de 90° la velocidad de aproximación al waypoint oscila entre 0 y `vx · cos(θ)`, con media ≈ 1.25 m/s. El resultado es un detector de atasco que se dispara prácticamente al inicio de cada waypoint que requiera maniobra.

Vulnerabilidad D — Escape ciego a la percepción.

Si el decisor evalúa las condiciones de escape de atasco *antes* de leer el `ObstacleField`,

```
if state["stall_count"] > hard_threshold:
    return "deadlock_escape"   # ← sin verificar percepción
if field.is_blocked("centro"):
    return "evasive"
```

el dron asciende aunque el estimador de flujo tenga evidencia válida de un sector lateral despejado.

9.5.3 El ascenso acumulado como síntoma compuesto

La combinación de las cuatro vulnerabilidades produjo, en una corrida de diagnóstico, un ascenso acumulado de ~356 m en una sola misión:

1. `_reset_stall_counter` no declarada → `stall_count` creció desde el ciclo 1 sin resetearse al llegar a cada waypoint.
2. A los ~8 ciclos de misión, `stall_count > hard_threshold` a pesar de que el dron avanzaba normalmente.
3. El decisor activó el escape (ciego a la percepción), que ejecutaba `GANAR_ALTURA` con deriva lateral.
4. `stall_count` se reseteó (ciclo límite B) → al ciclo siguiente, el decisor volvió a crucero.
5. `stall_count` volvió a crecer rápidamente (umbral demasiado bajo por la métrica inconsistente C).
6. El ciclo se repitió centenares de veces, con un ascenso neto acumulado.

Desde el exterior, la corrida registraba una misión "sin errores" (no hay excepciones en el log) que simplemente "no llegó a destino". Sin el análisis del visor cuadro a cuadro y la lectura del JSONL de auditoría, el patrón es opaco.

9.5.4 Instancias adicionales: estado descartado y precedencia entre rutas

Otras tres situaciones comparten la propiedad que organiza el capítulo: el sistema produce respuestas válidas y plausibles, sin excepciones en el log, y aun así no las ejecuta.

(a) Un flag descartado por LangGraph. Un indicador que difería la inyección de un desvío hasta el escaneo siguiente se escribía correctamente pero no estaba declarado en `DroneState`; desaparecía entre invocaciones y el desvío nunca se inyectaba, de modo que el dron volvía a la misma fachada. Es otra instancia de la Vulnerabilidad A.

(b) Resultados válidos del VLM que nunca se ejecutan. Dos errores de *precedencia* entre rutas, no de estado descartado, con el mismo síntoma:

- *Maniobra cancelada por un disparador de atasco.* Si al despachar `EVADIR_DERECHA` no se reinicia el contador de atasco, en el ciclo siguiente un disparador basado en ese contador vuelve a enrutar hacia una nueva consulta, cuyo resultado sobrescribe la maniobra comprometida antes de que se ejecute. En una corrida de diagnóstico, el VLM devolvió `EVADIR_DERECHA` durante 200 ciclos consecutivos sin que la maniobra se ejecutara una sola vez.
- *Resultado huérfano por el camino de deadlock.* Si la rama de escape por deadlock se evalúa antes que la lectura del pedido pendiente, y el contador de progreso queda congelado por encima del umbral, la rama intercepta todos los ciclos y el resultado nunca se lee: durante 80 ciclos, el log del VLM contenía la respuesta correcta y la acción ejecutada era `MANTENER_RUMBO` o `FRENAR`.

Ninguno de los dos se ve en las métricas agregadas; los muestra la comparación, ciclo a ciclo, del campo `slm.raw_response` de la traza JSONL con la acción realmente ejecutada (técnica 2 de §9.7).

(c) Un pedido pendiente que nunca se cierra. Si la consulta estratégica y el barrido comparten la cola del servicio (tamaño 1), un barrido que se encola mientras hay una consulta estratégica en vuelo la invalida. Si el sistema solo cerrara el pedido al llegar un resultado con *ese* identificador, el pedido quedaría activo indefinidamente, y un detector de atasco que interpretara la espera como intencional detendría sus contadores.

9.5.5 Mecanismo de contención

- **Estado declarado y verificado sobre el grafo compilado.** Todo campo que cruza la frontera entre invocaciones está declarado en `DroneState`, los contadores de atasco y el estado de la capa estratégica viven en objetos de proceso (`StallDetector`, `StrategicLayer`) y no en el estado, y los tests de integración corren el **grafo compilado** y verifican que las claves de control (`inject_corner`, `_escape_reset`, `_deadlock_event`, `_vlm_strategic`) cruzan la frontera (cap. 5, §5.2; `tests/test_graph_integration.py`, `tests/test_vlm_strategic_graph.py`).
- **Escape absorbente.** Un barrido en curso es dueño del dron hasta resolver o caer por sus watchdogs, y el deadlock se reinicia solo cuando se resuelve (`_escape_reset`), nunca dentro de su propia rama (cap. 5, §5.3.2, §5.3.4).
- **Progreso físicamente alcanzable.** El progreso se mide como distancia horizontal al objetivo, y los umbrales de atasco se derivan del avance mínimo demostrable a la velocidad mínima (cap. 5, §5.3.1, §5.15.2).
- **Precedencia explícita.** El orden de las reglas de `navigate` es fijo y está justificado (cap. 5, §5.3.2): barrido en curso, deadlock, maniobra comprometida, contacto, piso óptico y TTC. El resultado de cada consulta se recupera por su identificador (`get_result`) al inicio de cada ciclo, de modo que no depende de qué ruta tome el ciclo. En la FSM, el escape exige además que la percepción no muestre un corredor libre (`has_open_corridor()`, cap. 6, §6.11).
- **Pedidos perdidos.** La capa estratégica da por perdido un pedido reemplazado si el servicio no tiene resultado ni pedido pendiente durante 3 s, o al cumplir 10 s, y el deadlock cancela explícitamente la consulta estratégica en vuelo (cap. 5, §5.10.3, §5.3.4).

9.6 Falla 5 — Degradaciones sensoriales silenciosas

9.6.1 Inversión de canales de color RGB/BGR

`simGetImages` entrega el buffer de imagen en el orden RGB del motor, mientras que OpenCV y el resto del pipeline asumen la convención BGR, en la que el canal azul ocupa el índice 0. Sin conversión, cada píxel tiene los canales rojo y azul invertidos: fachadas de ladrillo rojo en tonos azulados, cielo amarillento, vegetación de colores no naturales. Un VLM entrenado sobre datos fotométricamente correctos delibera sobre esas imágenes sin emitir ninguna queja sobre su calidad ni reducir su confianza: no tiene cómo detectar que la imagen recibida está fuera de distribución por inversión de canal.

Detectarlo exige guardar en disco las imágenes exactas enviadas al VLM y abrirlas en un visor externo — el análogo de "mirar los datos crudos", primer paso de depuración en visión por computadora pero fácil de omitir en un sistema integrado cuando el modelo "parece razonar correctamente".

Mecanismo de contención. `AirSimClient.capture()` invierte el orden de canales en el punto de captura (cap. 3, §3.3), y cada consulta del lazo lento guarda sus fotogramas exactos como `photo-*.png` de auditoría (cap. 4, §4.4).

9.6.2 Contaminación por primitivas de depuración persistentes

El simulador AirSim/cosysairsim permite dibujar primitivas geométricas en el mundo virtual (`simPlotLineStrip`, `simPlotPoints`, `simPlotArrows`) para visualización durante el desarrollo: trayectorias planificadas, sectores del `ObstacleField`, vectores de flujo. Estas primitivas son *persistentes* en la sesión del simulador: no se borran al reiniciar el script de control, solo al llamar explícitamente `simFlushPersistentMarkers()`.

Al relanzar un vuelo sin reiniciar AirSim, las primitivas de la sesión anterior permanecen en el mundo virtual y la cámara a bordo las captura como geometría física: franjas de colores brillantes que el VLM interpreta como cables, estructuras metálicas o señales de tráfico. El comportamiento depende entonces del historial de la sesión: un vuelo iniciado tras reiniciar AirSim se comporta normalmente, y el mismo vuelo sin reiniciar produce evasiones espurias repetidas. Jansen et al. (2023) documentan comportamientos similares de estado persistente en CosysAirSim como fuente de irreproducibilidad entre corridas.

Mecanismo de contención. Cada sesión de conexión empieza con `AirSimClient.clear_debug_markers()`, que invoca `simFlushPersistentMarkers()` (cap. 3, §3.3).

9.6.3 Supresión de ángulos de actitud en la telemetría

El binding `cosysairsim` (fork de AirSim con soporte UE5 — Jansen et al., 2023) no implementa el método `to_eularian_angles()` del binding oficial de Microsoft AirSim. Un fallback que no lo note reporta pitch y roll en `0.0°` en cada ciclo, independientemente de la inclinación física del vehículo. Durante maniobras de aceleración (pitch de −8° a −12°), descenso agresivo o giros, el sistema recibe la información de que el dron está perfectamente nivelado: la derotación del flujo óptico pierde dos de sus tres ejes y cualquier consumidor de la actitud queda desalineado con la imagen.

Mecanismo de contención. El cliente convierte directamente el cuaternión de orientación, disponible en la telemetría de CosysAirSim, a ángulos de Euler (roll, pitch, yaw) con las fórmulas estándar de Wahba (1965), sin depender del método del binding:

$$\phi = \arctan 2(2(q_w q_x + q_y q_z),\ 1 - 2(q_x^2 + q_y^2))\$\$\$\$\theta = \arcsin(2(q_w q_y - q_z q_x))\$\$\$\$\psi = \arctan 2$$

Un test unitario (`tests/test_airsim_client_telemetry.py`) verifica la conversión contra cuaterniones conocidos.

9.7 Por qué el sistema sigue funcionando: el problema de la observabilidad

Ninguno de los seis modos de falla descritos se manifiesta como un error visible en tiempo de ejecución. En todos los casos:

- El modelo de lenguaje sigue produciendo respuestas JSON válidas y plausibles.
- El lazo de control ejecuta el ciclo sin excepciones.
- Las métricas agregadas de la corrida (distancia recorrida, waypoints completados, velocidad media) no distinguen estas corridas de corridas fallidas por otras causas.

Esta propiedad tiene una implicación metodológica directa: las técnicas de depuración habituales para sistemas de software — observar el output, monitorear las métricas, leer el log de errores — son ciegas a esta clase de error. La observación del comportamiento es precisamente la herramienta menos útil porque el sistema exhibe comportamiento *consistente con sus premisas internas*, aunque esas premisas estén corrompidas. Esta dificultad no es exclusiva de este proyecto: el estudio de campo de Amershi et al. (2019) sobre equipos de ingeniería de software que incorporan componentes de aprendizaje automático en Microsoft identifica, entre las diferencias estructurales frente al desarrollo de software tradicional, precisamente que los componentes de IA son más difíciles de aislar, probar y depurar como módulos independientes que el código determinista convencional — la ausencia de una frontera de excepción clara entre "el componente falló" y "el componente decidió mal" es un síntoma de esa misma dificultad, no una particularidad de los modos de falla de este capítulo.

Las técnicas de diagnóstico que sí funcionan son:

1. Auditoría de contratos en la frontera de cada componente. Para cada par (productor, consumidor) de datos de estado, verificar explícitamente: ¿qué valor devuelve el productor cuando no tiene información? ¿Cómo interpreta el consumidor ese valor? La pregunta "¿qué pasa cuando este campo es None/vacío/cero?" tiene que tener una respuesta documentada y verificable.

2. Guardar los datos crudos enviados al modelo. El registro de vuelo guarda el fotograma exacto de cada consulta, el prompt, la respuesta cruda y el estado completo de cada ciclo, y el visor (`*.viewer.html`) los muestra sincronizados con el video (cap. 4, §4.4), precisamente para hacer auditable lo que el modelo recibió. Sin esa capacidad de inspección retroactiva, la inversión RGB/BGR sería indetectable.

3. Tests de invariantes de contrato, no de comportamiento. Un test que verifique "el dron evadió el obstáculo" es frágil y dependiente de la escena. Un test que verifique "si `has_evidence()` es False, ningún consumidor reporta el campo como 'despejado'" es un test de contrato que captura la Falla 1 sin necesitar una corrida de vuelo.

4. Tests sobre el grafo compilado. Las claves no declaradas (Falla 4-A) no aparecen en tests que llaman a los nodos como funciones sueltas, porque el descarte ocurre en el runtime de LangGraph. Los tests de integración del sistema compilan el grafo y verifican qué claves sobreviven a `graph.invoke()` (§9.5.5), y cada secuencia de falla observada en una corrida piloto tiene un test de regresión que la reproduce sobre el grafo compilado (§9.8).

9.8 Falla 6 — Integración del VLM: respuestas correctas ejecutadas al revés, ignoradas o casi constantes

Esta familia se identifica auditando corridas de diagnóstico sobre `citysim_pilot` con el VLM como fuente del lazo lento (doce corridas, ninguna completó la misión; tres terminaron por `physics_locked`) con la técnica 2 de §9.7: comparar, ciclo a ciclo, la respuesta cruda del modelo con la acción ejecutada y con la profundidad de referencia del simulador, consultada a posteriori solo para la auditoría. Como en las fallas anteriores, ninguno de los mecanismos produce una excepción: el modelo responde JSON válido, el lazo ejecuta una acción en cada ciclo y las métricas agregadas registran deadlocks «resueltos» en el 75–80 % de los casos.

9.8.1 Error de marco de referencia: «libre hacia la meta» ejecutado como «evadir a la izquierda»

En las tres corridas analizadas con una interfaz de barrido basada en ángulos, el primer barrido respondió correctamente: `{"deg": -53, "tipo": "libre", "ok": true}` para la imagen tomada en el rumbo −53°, a ~16° del rumbo al waypoint. La acción ejecutada fue `EVADIR_IZQUIERDA`, y el dron giró alejándose de la meta. Dos inconsistencias de marco se combinaban:

1. Las imágenes se etiquetaban con su **rumbo absoluto** («[Rumbo −53°]») y el prompt pedía un ángulo **relativo**; el modelo copiaba la etiqueta y el parser interpretaba siempre el valor como relativo.
2. El error de rumbo hacia la meta se medía respecto del yaw **al final** del barrido (153°), no del yaw de cada imagen.

El dron se alejó de 82.6 m a 94–95 m del primer waypoint y tardó ~37 s en volver a la distancia de partida, en las tres corridas.

Mecanismo de contención. El modelo nunca reporta ángulos: identifica las imágenes por número, y el código asigna a cada una el yaw **medido** al capturarla y compara rumbos absolutos, calculando el rumbo a la meta desde las posiciones (cap. 5, §5.12). Un test reproduce el caso (`tests/test_deep_vlm_world_frame.py`).

9.8.2 Default silencioso: la respuesta del modelo nunca llega a la decisión

Si el productor de la respuesta devuelve una descripción de escena por sectores y el consumidor lee `decision.get("macro_action", "EVADIR_DERECHA")`, la clave no existe y **toda respuesta se ejecuta como `EVADIR_DERECHA`**, con razonamiento vacío: 11 de 15 despachos en las tres corridas, 5 de ellos contra un lado que el propio modelo había descrito como bloqueado. Es un caso de *schema drift* (§9.2) en sentido inverso: el productor y el consumidor no comparten esquema y el consumidor, en lugar de fallar, funciona con un valor por defecto plausible. Ningún test recorría el camino completo desde la respuesta hasta la acción.

Mecanismo de contención. Cada modo de consulta tiene un esquema estricto y un parser que lo valida; una respuesta que no coincide no produce ninguna acción (motivo `no_parseable`, registrado), nunca un valor por defecto (cap. 8, §8.2.2). Los tests de integración corren el grafo compilado de punta a punta, desde la respuesta del modelo hasta la sub-meta que llega al bucle externo (§9.9, riesgo A).

9.8.3 Salida casi constante del modelo

Las 185 respuestas del VLM de esas corridas se compararon con la profundidad de referencia registrada. En las consultas por sectores, excluida una corrida en vegetación, **52 de 53 respuestas declararon el frente transitable**; con la superficie más cercana a ≤ 1.2 m en los 8 s previos, 18 de 19 lo declararon igual. En los barridos, la primera imagen salió marcada como transitable en la gran mayoría de los casos, con valores (`"ok": true, "conf": 0.9`) idénticos al ejemplo del prompt. La respuesta, en la práctica, no dependía de la escena. Ninguno de los 8 deadlocks «resueltos» por el modelo en las tres corridas analizadas produjo avance neto hacia el waypoint (máximo 2.2 m en 10 s).

El hallazgo no se puede atribuir solo al modelo: el prompt contenía un ejemplo con valores concretos y una frase que pedía priorizar evasiones, contradictoria con la instrucción de solo describir, y la pregunta («¿se puede avanzar en este sector?») se hacía sobre imágenes de 256–384 px.

Mecanismo de contención. La pregunta, la granularidad y el prompt están diseñados contra estas causas (cap. 8, §8.3–§8.5), y el banco de prueba contra etiquetas de referencia mide si el modelo discrimina (cap. 10, §10.13). No lo hace (§9.9, riesgo D).

9.8.4 Deadlock falso en el despegue

En todas las corridas de diagnóstico, el primer deadlock se declaró en los ciclos 17–18, con el dron todavía subiendo a la altitud de crucero. Durante el despegue el guiado ordena subir y girar en el lugar hacia el primer waypoint con `vx = 0`; un contador de «detenido» que suma esos ciclos dispara la resolución de atasco al cruzar el piso óptico de 4.5 m.

Mecanismo de contención. «Detenido» se define como «se ordenó avanzar y el dron no se movió», y no se cuenta bajo el piso óptico ni durante el despegue (cap. 5, §5.3.1). Aplicada sobre la telemetría registrada, esa definición no declara el deadlock en ninguna de las tres corridas.

9.8.5 Una señal de referencia dentro del brazo evaluado

Un runner que captura el canal de profundidad del simulador «solo como métrica» puede terminar alimentando al brazo evaluado: en una configuración de diagnóstico, esa captura armaba un freno de proximidad que el grafo consumía, cancelaba maniobras, inyectaba retrocesos y registraba puntos de contacto que el tracker usaba para validar desvíos. El brazo evaluado tenía así acceso indirecto a la profundidad exacta del simulador, lo que invalida su comparación con un sistema monocular, y una guardia estática limitada a `src/` no lo detecta porque el runner está fuera de su alcance.

Mecanismo de contención. Ningún componente del lazo de vuelo lee el canal de profundidad, tampoco como métrica, y la guardia (`test_no_depth_in_flight_path.py`) cubre el runner, `main.py` y todos los módulos de vuelo, incluido el canal indirecto; la distancia mínima a obstáculo la mide un hilo de auditoría aislado (cap. 5, §5.16).

9.8.6 Reglas apiladas sobre la decisión del modelo

Compensar cada falla observada en una corrida piloto con una regla determinista acumula capas de reescritura sobre la decisión del modelo. En una configuración de diagnóstico había retrocesos antes del barrido, escalamiento a un retroceso si el modelo alternaba de lado, un desvío perpendicular con el lado invertido respecto del modelo, cinco reglas sobre la historia de la trayectoria, un escape vertical tras escaneos «fútiles» y, en el tracker, filtros que reflejaban o descartaban los desvíos: la decisión del VLM pasaba por hasta seis capas de reescritura, y el brazo `slm` medía esas reglas tanto como al modelo.

Medidas contra el avance hacia el waypoint real en los 10–20 s siguientes, ninguna mostró beneficio. Los retrocesos tuvieron avance mediano negativo (en el 31–43 % de los casos alejaron al dron más de 2 m); 74 desvíos deterministas dieron una mediana de 0.35 m de avance en 20 s, 16 de ellos alejaron al dron más de 2 m y lo llevaron sobre la autopista elevada; y un disparador de deadlock basado en la historia de la trayectoria produjo 45 deadlocks sin avance posterior. Solo el lazo reactivo (guiado, evasión por flujo, giro de 90°) mostró avance mediano positivo.

La lección metodológica es la de §9.7 aplicada al diseño: una regla que corrige un síntoma observado en una corrida, sin medir su efecto agregado, no se puede distinguir —desde las métricas de éxito— de una regla que empeora el sistema.

Mecanismo de contención. La respuesta de la fuente se traduce a geometría y se aplica sin reglas que la reemplacen; solo se descartan respuestas obsoletas (cap. 5, §5.16). El criterio para incluir un componente en el lazo es haber mostrado avance hacia el waypoint real, o ser parte del lazo rápido común a los tres brazos.

9.8.7 La causa física de los `physics_locked`

En las tres corridas el dron no chocó de frente: quedó apoyado sobre una **superficie horizontal a la altura de crucero** —el tablero de una autopista elevada (dos corridas) y la cornisa de un edificio (una)—, con los sectores frontales del flujo libres (`occ = 0`, TTC infinito). En una corrida, un `GANAR_ALTURA` liberó al dron hasta −12.2 m y el guiado de altitud lo devolvió a −10 m, sobre la cornisa. Ni el flujo óptico ni una descripción por columnas representan ese tipo de obstáculo.

Mecanismo de contención. El manifiesto evita la autopista (`WP_0_SUR`, cap. 10 §10.3.3); la capa estratégica juzga el fotograma en una grilla de 3×3, de modo que un obstáculo a la altura del dron puede rodearse por la fila de arriba; y el escape determinista ante falla de la fuente es `GANAR_ALTURA` (cap. 5, §5.3.4, §5.10).

9.8.8 Señales que se tapan entre sí: un techo inventado y una velocidad que no existe

En una corrida piloto sobre `citysim_pilot`, el dron quedó 220 ciclos apoyado sobre el parapeto de la autopista elevada, pegada al punto de partida, sin que se declarara un solo deadlock. Tres mecanismos, cada uno razonable por separado, se encadenaron:

1. **Avance durante el ascenso.** El guiado avanzaba hacia el primer waypoint mientras subía; a 7.4 m, todavía por debajo del borde del parapeto, el TTC frontal cayó a 0.23 s y el lazo ejecutó `GIRAR_90`.
2. **Un techo inventado.** La detección de techo contaba los ciclos en que el *guiado* pedía subir sin que la cota cambiara. Durante el giro el comando vertical era cero, de modo que diez ciclos de giro se leyeron como diez ciclos empujando contra una losa: techo en z = −7.44. Con ese techo, el waypoint a −10 m quedó «bloqueado verticalmente» y el lazo ordenó `PERDER_ALTURA` en cada ciclo, contra el parapeto.
3. **Una velocidad que no existe.** Con el dron apoyado en la estructura, AirSim reportaba 0.31 m/s horizontales y 0.75 m/s verticales con la posición inmóvil. El contador de «detenido» leía esa velocidad y nunca avanzó; el de «sin progreso» sí llegó a 200 ciclos, pero la regla de descenso por techo se evaluaba antes que la de deadlock y tomaba el ciclo.

Ninguna de las tres produjo un error: cada ciclo tenía una acción válida y una justificación registrada. Es la misma clase de falla que §9.5 y §9.7 —el sistema «funciona» mientras repite indefinidamente una acción inútil—, en este caso entre señales del lazo rápido y no entre el modelo y el lazo.

Mecanismo de contención. El despegue es vertical hasta la altitud del primer objetivo (cap. 5, §5.15); la detección de techo usa el comando efectivamente ejecutado, no la demanda del guiado (§5.15.3); el `StallDetector` mide la velocidad por desplazamiento entre ciclos (§5.3.1); y el deadlock se evalúa antes que cualquier otra regla salvo el barrido en curso (§5.3.2), de modo que ninguna acción que se repita sin mover al dron puede taparlo. Un test reproduce la secuencia sobre el grafo compilado (`tests/test_pilot_seed99_regressions.py`).

9.8.9 Sub-metas que ocultan al waypoint real

En otra corrida piloto, el dron llegó a la zona del primer waypoint a los 40 s y pasó el resto de la corrida —más de cuatro minutos— pegado a la fachada vidriada del edificio contiguo. Estuvo **573 ciclos a menos de 3.5 m del waypoint**, el radio de aceptación, sin aceptarlo nunca. Cinco mecanismos se encadenaron:

1. **La sub-meta ocultaba al waypoint.** El tracker solo medía la llegada al objetivo activo. En los 573 ciclos el objetivo activo era una sub-meta del VLM (332 de la capa estratégica, 241 del barrido), de modo que pasar por encima del waypoint no contaba.
2. **Sub-metas más lejos que el waypoint.** La sub-meta se fijaba siempre a 15 m. Con el waypoint a 6 m, una sub-meta a 15 m queda 9 m más allá: en este caso, dentro del edificio. A 1 m del waypoint, el rumbo hacia la meta que usa el barrido es casi ruido, y la sub-meta lo amplifica.
3. **Sin re-planificación.** La capa estratégica solo consultaba con un waypoint real como objetivo. Con una sub-meta mala activa quedaba muda, y el lazo rápido entraba en un ciclo límite: el guiado apuntaba a la sub-meta dentro del edificio, el TTC caía a 0–0.2 s, `GIRAR_90` desviaba un segundo y el guiado volvía a apuntar a la fachada (cinco giros en 55 ciclos, seis en 50). Solo un deadlock, 10 s después, reemplazaba la sub-meta.
4. **Respuestas casi constantes, incluso sobre imágenes del interior del edificio.** La capa estratégica, con una partición en columnas, respondió «la primera columna bloqueada, las demás libres» en 5 de 6 consultas, incluida una con la fachada ocupando casi todo el cuadro, y marcó «estructura debajo» en 5 de 6, lo que subía todas las sub-metas 4 m. En los 8 barridos, las imágenes 2 y 4 salieron libres y la 1 y la 3 bloqueadas, sin importar hacia dónde miraba el dron. Con el dron pegado a la fachada, la cámara quedó dentro de la malla del edificio en dos barridos: el modelo marcó el interior de una oficina como `libre, ok: true` y, como apuntaba hacia el waypoint, el barrido lo eligió.
5. **Un techo inventado por una pared.** Con el dron trabado contra la fachada, el roce impedía subir y la detección de techo declaró una losa a −11.8 m; los escapes `GANAR_ALTURA` subieron después hasta −16 m sin obstáculo, y la regla de techo los devolvía hacia abajo.

Además, el waypoint mismo estaba mal ubicado: sus coordenadas se habían tomado del mapa y caían sobre la fachada, en contra del principio de derivar coordenadas solo de posiciones ya voladas (cap. 10, §10.3.0).

Mecanismo de contención. El tracker acepta el waypoint real aunque haya una sub-meta activa y descarta los desvíos pendientes (cap. 5, §5.15.4); ninguna sub-meta queda más lejos que el waypoint real, y a menos de 8 m del waypoint no se consulta a la capa estratégica ni se barre (§5.10.1, §5.12); la capa estratégica re-planifica con una sub-meta activa, y si ve libre el camino directo descarta el desvío (§5.10.3); el barrido no usa entradas que se contradicen (`free: true` con una vista distinta de `open`), lo que no reemplaza la decisión de la fuente sino que descarta respuestas internamente inconsistentes, igual que una respuesta mal formada (§5.12); la capa estratégica juzga una grilla de 3×3 en lugar de columnas, de modo que el eje vertical sale de la descripción y no de una pregunta abstracta (cap. 8, §8.3.2); la detección de techo no cuenta los ciclos en que el dron está trabado de costado (§5.15.3); y los waypoints de los manifiestos están en puntos ya volados. Los tests `tests/test_pilot_190842z_regressions.py` reproducen cada mecanismo.

Estas medidas no corrigen el punto 4: que el modelo casi no distinga escenas. Con ellas, una respuesta mala deja de atrapar al dron; que la fuente aporte información es la pregunta de §9.9, riesgo D.

9.8.10 Una respuesta cortada y una capa sin voz

En una tercera corrida piloto, el dron llegó a los tres primeros waypoints —incluido el que está detrás de una manzana— y quedó trabado en el tramo hacia el cuarto, contra la fachada de un edificio que la recta atraviesa. Tres mecanismos lo dejaron sin salida:

1. **El modelo marcó libres los costados de una fachada.** En las cinco consultas del tramo marcó bloqueado el sector de frente —lo correcto— y libres los demás, incluso en fotogramas en los que la fachada ocupaba todo el cuadro. En las 28 consultas de la corrida, el sector de la izquierda a la altura del dron salió libre las 28 veces y el de arriba a la izquierda, bloqueado 24: un sesgo por posición en la grilla, no una descripción de la escena. La sub-meta elegida quedó del otro lado de la fachada, y el dron se deslizó a lo largo de ella unos 180 ciclos.
2. **La capa estratégica no consultó en todo ese tiempo.** El dron miraba a lo largo de la pared, con la meta fuera de los ±40° del eje óptico, la condición para consultar: justo cuando hacía falta replanificar, la capa no tenía cómo hacerlo.
3. **Los barridos se perdían por el tope de tokens.** Un tope de 160 tokens cortaba la respuesta del barrido —un JSON con indentación— antes de la llave final, en 5 de 12 barridos; esos barridos caían al escape vertical sin usar la respuesta del modelo. La falla no se veía: el registro la clasificaba como «sin formato válido», igual que una respuesta mal formada.

La corrida terminó por `physics_locked`, con el dron apoyado en la baranda de un balcón (§9.8.7). Además, se ejecutó con cambios sin commitear y quedó registrada con el mismo `code_version` que otra corrida con distinto código: un criterio de descarte por versión basado solo en el hash no la distingue.

Mecanismo de contención. Los topes de tokens son 384 (estratégico) y 512 (barrido), y una respuesta cortada se reporta como `respuesta_truncada`, separada de las respuestas mal formadas (cap. 5, §5.10.5); la capa estratégica consulta con la meta fuera de cuadro cuando el dron lleva `VLM_STUCK_QUERY_CYCLES` (15 ciclos, 3 s) sin acercarse a su objetivo, sin interpretar nunca esa respuesta como «camino directo libre» (§5.10.1); y `code_version` lleva el sufijo `-dirty` si hay cambios sin commitear, lo que excluye la corrida de un lote (cap. 10, §10.12). Los tests `tests/test_pilot_193631z_regressions.py` reproducen cada caso. El punto 1 —que el modelo describa la grilla por posición y no por contenido— no tiene mecanismo de contención dentro del VLM: es la pregunta de §9.9, riesgo D, y la medición que la responde es la comparación sector por sector contra una etiqueta de referencia (cap. 10, §10.13).

9.8.11 Un barrido que no describe la escena

Con las respuestas del barrido completas, tres corridas piloto suman 19 respuestas utilizables. En las 19 el patrón de transitabilidad fue el mismo: las imágenes 2 y 4 —a 90° a cada lado del rumbo inicial— transitables, y la 1 y la 3 —el frente y la dirección de la que venía el dron— bloqueadas, sin importar dónde estaba el dron ni qué mostraban las imágenes. La dirección de la que venía el dron, casi siempre libre, salió bloqueada las 19 veces. Además, el modelo marcó todas las respuestas como degradadas, incluidas imágenes nítidas de calle abierta, y en varias asignó `ok: true` a imágenes de tipo `fachada` o `interior`. Medido contra el avance hacia el waypoint real, las sub-metas del barrido no mostraron efecto: en la mayoría de los casos el avance a los 20 s fue nulo, y en dos el dron se alejó más de 5 m.

La consecuencia de diseño es doble. Primero, una respuesta constante convierte al barrido, en la práctica, en una regla fija («girar 90° hacia el lado más cercano a la meta»), y una regla de falla basada en el indicador de imagen degradada lo convertiría en un escape vertical ciego (en una de esas corridas habría descartado los 8 barridos). Segundo, elegir el rumbo transitable más cercano a la meta devuelve al dron, deadlock tras deadlock, contra el mismo obstáculo.

Mecanismo de contención. El esquema no pide indicador de imagen degradada ni certeza (cap. 8, §8.2.1); y la salida del deadlock descarta el rumbo que falló y los rumbos ya probados en la zona, y con deadlocks repetidos explora el rumbo libre más distinto de lo ya probado (cap. 5, §5.12). Esto no corrige que el modelo responda lo mismo siempre: con una respuesta constante, la salida queda determinada por la historia del dron y no por la imagen. El registro de la selección permite separar ambos efectos.

9.9 Riesgos residuales y trabajo pendiente

Los modos de falla documentados tienen su mecanismo de contención en el sistema y los de §9.8 están cubiertos por tests de regresión sobre el grafo compilado. La arquitectura contiene puntos donde pueden aparecer instancias análogas:

Riesgo A — Nuevas claves de DroneState. Cada funcionalidad que requiere persistir estado entre ciclos corre el riesgo de que la clave correspondiente no se declare en `DroneState`. Las mitigaciones son mantener el estado de control en objetos de proceso y los tests de integración sobre el grafo compilado, que verifican que las claves de control (`inject_corner`, `_escape_reset`, `_deadlock_event`, `_vlm_strategic`) cruzan la frontera de `graph.invoke()` (`tests/test_graph_integration.py`, `tests/test_vlm_strategic_graph.py`). Esos tests cubren las claves conocidas, no las nuevas: una defensa más general sería un test de contrato que recorra el código fuente de `src/agents/`, extraiga todas las claves `state["_..."]` y `state.get("...")` y falle si alguna no pertenece a `DroneState`; no existe.

Riesgo B — Cambios de API del binding de AirSim. El binding `cosysairsim` es un fork activamente mantenido (Jansen et al., 2023). Sus actualizaciones pueden modificar el comportamiento de métodos (como la conversión de actitud de §9.6.3). Los tests unitarios de conversión de telemetría son la mitigación; deben ejecutarse tras cada actualización del binding.

Riesgo C — Distribución de imágenes fuera del dominio de entrenamiento. La conversión RGB/BGR es una transformación fija. La brecha de dominio entre imágenes de AirSim/UE5 y el dominio de entrenamiento del VLM y de la red de profundidad (predominantemente imágenes reales) es una fuente estructural de error que no se elimina con ninguna corrección puntual (cap. 6, §6.14.5).

Riesgo D — Que el VLM no discrimine (materializado). El diseño corrige cómo se pregunta y cómo se usa la respuesta, pero no puede garantizar que el modelo distinga sectores volables de bloqueados (§9.8.3). El banco de prueba con el modelo de vuelo (LFM2.5-VL-1.6B, `lfm2.5-vl-1.6b:q4km-ctx8k`) confirma el riesgo: la grilla queda en AUC 0.53 y el barrido declara transitables todos los rumbos (cap. 11, §11.0). La respuesta del diseño no es una regla que compense al modelo, sino una fuente que sí responde la pregunta, la profundidad estimada (cap. 6, §6.14), con el VLM donde aporta, el plan de ruta (cap. 4, §4.3.1).

10. Metodología experimental

Este capítulo especifica el protocolo experimental con el que se evalúa la hipótesis central de la tesis: si una capa deliberativa sobre percepción monocular aporta un beneficio medible sobre una heurística rígida en la capa táctica de un dron autónomo, y a qué costo. La pregunta se responde con dos instrumentos. El **banco de prueba a bordo** (§10.13) mide, fuera de vuelo y contra etiquetas de referencia, si una fuente —el VLM o la red de profundidad estimada— sabe responder la pregunta que el lazo lento le hace. Los **lotes de vuelo** (§10.5) miden si el sistema completo, con la fuente que pasa el banco, vuela mejor que una máquina de estados y que un controlador reactivo.

La estructura del capítulo responde a un orden deliberado: primero *qué se quiere probar* (§10.1), luego *con qué factores* (§10.2), *sobre qué escenarios* (§10.3), *con cuántas repeticiones y por qué* (§10.4), *organizado en qué lotes* (§10.5), *midiendo qué* (§10.6), *analizado cómo* (§10.7–§10.8), *dejando qué evidencia* (§10.9) y *qué queda fuera* (§10.10–§10.11). §10.12 detalla lo propio de los lotes con obstrucción y §10.13 describe el banco de prueba. El capítulo 11 consume directamente esta especificación: las métricas primarias y las comparaciones están fijadas aquí antes de existir los datos, para evitar el sesgo de elegirlas después de verlos.

10.1 Preguntas experimentales e hipótesis operativas

El diseño no persigue una única comparación global "SLM vs. FSM", sino tres preguntas separables, cada una con su propia condición de falsación:

H1 — Beneficio de la capa deliberativa. En escenarios donde la trayectoria directa está obstruida por geometría que exige decidir un rodeo (elegir entre dos corredores, anticipar que una fachada continua no tiene salida lateral), el brazo `slm` alcanza un SPL superior al de una FSM con significancia estadística. *Condición de falsación:* si en los escenarios con obstrucción la FSM iguala o supera al `slm` en SPL, la capa deliberativa no aporta, y el resultado negativo es igualmente publicable.

H2 — Costo de la arquitectura. El sobrecosto temporal del brazo `slm` no escala con la distancia recorrida sino con la **frecuencia de consulta del lazo lento** (`deliberation_rate`): es aproximadamente un término aditivo $N_{\text{inv}} \times \bar{\ell}$ (número de consultas por su costo medio en la misión), no un factor multiplicativo sobre el tiempo de misión. *Condición de falsación:* si el ratio $t_{\text{slm}}/t_{\text{reactive}}$ es aproximadamente constante entre escenarios de control en lugar de decrecer con la longitud de la ruta, la hipótesis es incorrecta y el costo sí es multiplicativo.

H3 — Estratificación del beneficio por dificultad ambiental. El signo del efecto de H1 depende del escenario: en ambientes despejados la capa deliberativa solo agrega costo; el beneficio, de existir, aparece únicamente donde la geometría exige una decisión que la heurística no puede tomar. La hipótesis operativa es que **existe una interacción entre el factor "brazo" y el factor "escenario"**, no un efecto principal uniforme del brazo.

Esta última hipótesis es la que dicta la arquitectura de escenarios en tres niveles, cada uno con un escenario de control despejado: sin controles no hay forma de separar el costo fijo del brazo `slm` de su beneficio condicional, y sin escenarios con obstrucción genuina no hay forma de que el beneficio se manifieste. El diseño es, en este sentido, deudor directo de la práctica establecida en los *benchmarks* de conducción autónoma y navegación encarnada, donde la dificultad se estratifica explícitamente en niveles y cada nivel se corre un número fijo de episodios: CARLA organiza sus tareas en cuatro niveles crecientes (recta, un giro, navegación, navegación con tráfico dinámico) sobre el mismo simulador (Dosovitskiy et al., 2017), y su sucesor *NoCrash* define tres niveles de densidad de agentes dinámicos (`Empty`, `Regular`, `Dense`) con 25 episodios por tarea y condición (Codevilla et al., 2019). La estratificación por densidad de obstáculos es también la forma canónica de reportar navegación aérea de alta velocidad en bosque simulado (Loquercio et al., 2021), donde la métrica se informa en función de la densidad de árboles por metro cuadrado en lugar de agregarse en un único número.

10.2 Diseño factorial

10.2.1 Factor principal: brazo de decisión (`AGENT_ARM`)

La evaluación compara tres brazos de decisión sobre **el mismo** `ObstacleField` (cap. 6), **el mismo espacio de macro-acciones** y **el mismo traductor a comandos cinemáticos** (`action_to_command`, cap. 8). Esta invariancia es lo que hace legítima la comparación: la diferencia entre brazos es *quién elige la macro-acción y quién decide los rodeos*, no qué percibe el sistema ni cómo ejecuta la decisión.

- **`slm`** — el grafo de control con sus dos lazos (cap. 5). El lazo lento, cada ≥ 1 s o de inmediato cuando el lazo rápido ve un muro de frente, juzga qué sectores de una grilla de 3×3 del fotograma frontal son volables; la respuesta se traduce a una sub-meta en coordenadas del mundo anclada a la pose del fotograma, y el dron nunca se detiene a esperarla. En un deadlock, un barrido panorámico de 4 rumbos produce la sub-meta de salida; ante una falla, escape vertical. La fuente del juicio es `STRATEGIC_SOURCE` (§10.2.2): en los lotes, la red de profundidad estimada (cap. 6, §6.14). El nombre del brazo designa la arquitectura con capa deliberativa; con la configuración de los lotes el VLM no interviene en vuelo y solo planifica la ruta antes del despegue en TownSim y CitySim (cap. 4, §4.3.1). No hay reglas deterministas que reescriban la respuesta de la fuente.
- **`fsm`** — una máquina de estados finitos explícita (`CRUISE → AVOID_LEFT | AVOID_RIGHT | CLIMB | BRAKE → CRUISE`), con transiciones gobernadas por umbrales fijos sobre el mismo `ObstacleField`, y su propia lógica de atasco y escape (cap. 5, §5.11). Es la línea de base directa: alta predictibilidad y costo computacional mínimo, sin capa deliberativa.
- **`reactive`** — navegación guiada al waypoint con flujo óptico y TTC, sin deliberación ni resolución de atascos: **cota inferior de rendimiento** que permite desacoplar qué porción del desempeño proviene de la capa táctica y cuánto del seguimiento cinemático de base. Su varianza entre semillas es además el control negativo del diseño (§10.4.1).

El brazo se selecciona por variable de entorno `AGENT_ARM`, leída **a nivel de módulo** en `src/agents/graph.py`. Los tres brazos comparten el mismo bucle externo, el mismo `WaypointTracker` y el mismo nodo `motor` (ancla de altitud y gobernador de velocidad). Esta decisión de implementación tiene una consecuencia metodológica directa: **cada corrida debe correr en un intérprete propio**, porque reutilizar el proceso arrastraría el primer valor leído. El runner lanza por eso un subproceso por corrida, lo que además aísla el fallo de una corrida del resto del lote.

10.2.2 Factores secundarios: resolución de atascos (`DEADLOCK_STRATEGY`) y fuente de la capa estratégica (`STRATEGIC_SOURCE`)

Resolución de atascos. Ante un deadlock declarado, barrido panorámico **dentro del lazo de control** (`SCAN_HEADING_COUNT_DEEP = 4` rumbos, `SCAN_SETTLE_CYCLES_DEEP = 2` ciclos de asentamiento por rumbo, hasta `MAX_DEEP_SCAN_IMAGES = 5` imágenes) y una única consulta que juzga cada imagen; el código fija una sub-meta en el rumbo transitable elegido (cap. 5, §5.12). Sin respuesta válida, escape vertical. Valores:

- **`depth`** (default): los rumbos los juzga la profundidad estimada (p5 del sector central ≥ 15 m).
- **`deep_vlm`**: los juzga el VLM (alternativa; no se usa en los lotes).
- **`blind`**: sin barrido; escape vertical determinista (`GANAR_ALTURA`) en el brazo `slm` y alternancia `CLIMB` / `DESCEND` propia en el brazo `fsm`.

El brazo `reactive` nunca declara atasco.

Fuente de la capa estratégica (solo brazo `slm`): `depth` (default), `vlm` u `off` (sin capa). Se selecciona con `--strategic-sources`; con el argumento, las celdas se nombran `strat_<fuente>__<estrategia>`. Los lotes la fijan en `depth`.

Advertencia de diseño: el barrido es común a `slm` y `fsm`. Con `depth`, la FSM delega sus deadlocks en el mismo barrido que el brazo `slm`. La comparación `slm:depth` vs. `fsm:depth` aísla, por lo tanto, la capa estratégica y la política del lazo rápido, no la resolución de atascos. El brazo sin barrido es `fsm:blind`, que solo se corre donde hay atascos que resolver (§10.12.2).

10.2.3 Estructura de celdas

La celda elemental es la tupla (escenario × brazo × estrategia de atasco), que el runner recibe como `--cells brazo:estrategia`. Cada celda se replica con $K = 10$ semillas (§10.4.4):

Celda	Qué aísla	Escenarios despejados	Escenarios con obstrucción
`slm:depth`	El sistema completo: capa estratégica + política del lazo rápido del brazo `slm` + barrido	✓	✓
`fsm:depth`	La misma resolución de atascos con profundidad, sin capa estratégica y con política FSM	✓	✓

Celda	Qué aísla	Escenarios despejados	Escenarios con obstrucción
`fsm:blind`	La FSM sin barrido (escape determinista)	—	✓
`reactive`	Flujo óptico + TTC, sin deliberación	✓	✓
Corridas por escenario		**30**	**40**

10.3 Escenarios: contenido, particularidad y dificultad

El protocolo define una jerarquía en tres niveles crecientes de dificultad ambiental, cada uno sobre un **proyecto de Unreal Engine distinto** (cap. 3), con un escenario de control despejado por nivel y uno o dos escenarios con obstrucción en los niveles 1 y 2.

10.3.0 Principios de diseño comunes a todos los escenarios

Cuatro principios metodológicos gobiernan la construcción de cualquier escenario, y explican por qué los manifiestos tienen la forma que tienen:

1. **Prohibición de teletransporte cinemático.** El campo `start_pose` está inhabilitado en la ruta de producción: todas las misiones despegan desde el *PlayerStart* nativo del nivel de Unreal Engine y ejecutan un ascenso vertical controlado previo a la navegación horizontal. La razón es que `simSetVehiclePose(ignore_collision=True)` es un reposicionamiento instantáneo que **no verifica colisión**, con lo que un escenario podría materializar el dron dentro de un edificio o a centímetros de un tronco, generando variación entre corridas que nada tiene que ver con la política de control evaluada. La consecuencia de diseño es que un escenario que necesita empezar lejos del *spawn* se arma como **misión de varios tramos**: un tramo de tránsito a altitud validada hasta el punto de interés, y recién allí el tramo de prueba a nivel de calle.
2. **Ninguna coordenada se inventa sobre el PNG del mapa.** Los mapas registrados de TownSim y CitySim (cap. 4, §4.3.2) sirven para planificar la ruta y ubicar waypoints candidatos, pero no reemplazan la verificación en vuelo. Toda coordenada nueva se deriva **por interpolación de coordenadas ya voladas y confirmadas**, o se verifica ubicando el dron en ella; la imagen se usa sólo cualitativamente (disposición relativa de plaza, bloques, corredor, avenida). Las coordenadas obtenidas por subdivisión geométrica —no por vuelo— se marcan explícitamente como **PROVISORIAS** y no entran a un lote sin pasar el protocolo de validación del punto 4.
3. **Exclusión total de profundidad del lazo de vuelo.** Ni el grafo, ni `main.py`, ni el runner leen el canal de profundidad del simulador: ni como entrada de control, ni como freno, ni como criterio de aborto. Cualquier uso de esa señal dentro del lazo, aun «solo para observabilidad», abre un canal por el que la profundidad del simulador —de la que salen las etiquetas de referencia— puede terminar alimentando al brazo evaluado (cap. 9, §9.8.5). Una guardia automatizada

(`tests/test_no_depth_in_flight_path.py`) cubre el runner, `main.py` y todos los módulos de vuelo. La única lectura de profundidad de una corrida es la **auditoría de la distancia mínima a obstáculo** (§10.6.1): un hilo aislado, con su propia conexión al simulador, que solo escribe un archivo de muestras; la misma guardia verifica que no importe ningún módulo del grafo y que el runner se limite a arrancarlo y detenerlo.

4. **Protocolo de validación previo obligatorio.** Antes de que un escenario entre a un lote: (a) se redacta el manifiesto en `airsim-plan/missions/flightplans/`; (b) se dibuja la ruta con `scripts/plot_mission_route.py` y se confirma en el *viewport* de UE que la línea cruza donde el escenario pretende (fachada para los de obstrucción, calle abierta para los de control); (c) se corre una **misión piloto** (`slm:depth`, una semilla) y se exige el criterio de pase del escenario (§10.12.2) —o, si no completa, que el motivo sea instructivo (atasco genuino) y no un error de geometría; (d) recién entonces se lanza el lote completo. El criterio es explícito: *si ninguna celda puede tener éxito en el escenario, no tiene sentido gastar el presupuesto del lote*.

Una consecuencia práctica del punto 4 es la limpieza obligatoria de estado antes de cada corrida: el runner invoca `client.reset()` (que devuelve el vehículo a la pose de *spawn* de `settings.json` y limpia colisión/velocidad del controlador interno) y `client.clear_debug_markers()` (que descarta las primitivas persistentes dibujadas por `plot_mission_route.py`, las cuales de otro modo aparecen en el fotograma que recibe la capa estratégica, cap. 9 §9.6.2).

10.3.1 Tier 0 — MiniSim: línea de base y control

Entorno. Proyecto base de Unreal Engine 5.5: un plano liso con textura de tablero, sin relieve, edificaciones ni vegetación. Iluminación uniforme y una textura de suelo con contraste suficiente para que el estimador de flujo óptico produzca campo denso, sin ninguna estructura vertical en la trayectoria. Sin obstáculos, la recta entre waypoints es la ruta óptima, de modo que el escenario vuela sin plan de ruta (`--route-plan off`). La distancia mínima a obstáculo que se mide ahí (§10.6.1) es la distancia al suelo.

Escenario `minisim_clear` (`MINISIM_CLEAR`, lote 1).

WP	x (m)	y (m)	z (m, NED)	Rol del tramo
WP_1	60.0	0.0	−10.0	Crucero recto de 60 m
WP_2	60.0	60.0	−10.0	Quiebre de rumbo de 90°
WP_3	0.0	60.0	−10.0	Segundo tramo recto de 60 m

Recorrido nominal en "L": 120 m entre waypoints declarados, ~180 m contando el tramo desde el *spawn* hasta WP_1. Altitud constante de −10 m.

Qué se quiere probar. Este escenario **no busca ejercitar la evasión** — busca medir lo que un escenario con obstáculos no permite medir limpiamente:

1. **Cota inferior de latencia de ciclo** y consumo cinemático ideal del sistema, sin ninguna contribución del lazo de evasión.
2. **Costo puro del brazo `slm` sin beneficio compensatorio.** Es el escenario donde H2 se mide en su forma más cruda: cualquier segundo de diferencia entre `slm` y `reactive` es sobrecosto neto de la arquitectura, porque no hay nada que decidir.
3. **Tasa de falsos positivos de percepción.** En un campo sin obstáculos, *todo* evento de evasión y *todo* deadlock son, por construcción, falsos positivos. Es una medida directa de la especificidad del predicado `is_blocked()` (cap. 6, §6.9) en condiciones benignas — una métrica que ningún escenario con obstáculos puede aislar.
4. **Fidelidad del quiebre de rumbo de 90°.** El giro entre WP_1 y WP_2 ejercita la guardia de rotación del estimador de flujo (`FLOW_MAX_ROTATION_DEG = 2°`) y la derotación analítica por IMU (cap. 6, §6.5). Un pico espurio de evasiones concentrado en ese tramo —visible en `summary_by_wp.csv`— indicaría que la derotación no está compensando el giro.

Particularidad y dificultad. Es el único escenario donde el resultado esperado es *desfavorable* a la hipótesis principal: se espera que `reactive` gane por amplio margen y que `slm` sea el más lento. Su dificultad no es de navegación sino de **interpretación**: sirve para calibrar cuánto del desempeño observado en Tier 1 y 2 es atribuible al escenario y cuánto al costo estructural del brazo.

10.3.2 Tier 1 — TownSim (`townsim_ortho.png`): vegetación y morfología orgánica

Entorno. Entorno semiurbano construido sobre el *Downtown West Modular Pack* (PurePolygons). De norte a sur: una plaza con rotonda y jardín circular, y luego **tres bloques rectangulares de edificios en fila**, cada uno separado del siguiente por una calle transversal. Cada bloque tiene dos filas de edificios (oeste y este) con un **pasaje peatonal central** entre ambas. Una avenida ancha sale hacia el este, aproximadamente a la altura del límite entre la plaza y el primer bloque. Todo el complejo está rodeado de bosque denso; hay un cuerpo de agua al sur, fuera del complejo.

La geometría métrica del complejo se deriva exclusivamente de una misión volada y confirmada (`TOWNSIM_CALIB_0`, 656 m, `success = True`, 0 colisiones):

- El complejo mide **≈156 m de norte a sur** ($y \in [-71, +84]$) y **≈146 m de ancho** desde el eje del corredor central ($x \approx 0$) hasta el borde oeste exterior ($x \approx -146$).
- Centroide del complejo: $x \approx -73$, $y \approx 6.5$.
- Con tres bloques repartidos en tercios sobre el rango norte-sur (≈52 m por bloque), los límites entre bloques quedan aproximadamente en $y \approx -19$ e $y \approx 32$ — **subdivisión PROVISORIA**, sujeta a confirmación visual.

La particularidad que define a este tier es que sus obstáculos son de **morfología orgánica y semitransparente**: follaje, arboledas densas, mobiliario urbano. A diferencia de una fachada plana, una copa de árbol produce un campo de flujo óptico de alta frecuencia y baja coherencia — el estimador de FOE pierde inliers, la confianza cae, y el sistema entra en el régimen donde la evidencia geométrica es ambigua. Es el régimen donde un juicio sobre el fotograma completo (la capa estratégica) debería aportar sobre la reacción al flujo. Un riesgo específico es el ascenso: si el guiado combina ascenso y traslación lateral, el dron cruza la copa de los árboles todavía bajo y se engancha en el follaje; por eso el despegue es vertical (cap. 5, §5.15).

Escenario 1.A — `townsim_clear` (control de crucero largo, lote 2)

Perímetro completo del complejo, 6 waypoints, ~640 m, altitud de tránsito −30 m (por encima de la línea de tejados). Derivado directamente de `TOWNSIM_CALIB_0`: mismos 6 waypoints, misma geometría verificada por vuelo.

Qué se quiere probar. Rol análogo al de `minisim_clear` en Tier 0, pero con una diferencia crítica: **la ruta es 3.3× más larga**. Es el escenario que permite testear H2 de forma directa — si el costo del lazo lento es aditivo en el número de consultas y no multiplicativo en la distancia, el ratio $t_{\text{slm}}/t_{\text{reactive}}$ debe caer sustancialmente respecto de Tier 0.

Dificultad. Baja en términos de navegación (a −30 m el dron sobrevuela los tres bloques en lugar de negociarlos), pero **alta en términos de duración**: expone al sistema a la deriva acumulada del estimador y a la degradación por GPU compartida entre UE y las redes del lazo lento.

Escenario 1.B — `townsim_ini` (recorrido con vegetación en ruta, lote 3)

Travesía norte–sur del corredor peatonal arbolado central de TownSim, arrancando en el *PlayerStart* real (0,0), con dos particularidades estructurales. Primera: **WP_0_ASCENSO sube en el lugar** (mismo x, y del *spawn*, hasta −30 m) antes de trasladarse, para no cruzar la copa de los árboles durante el ascenso. Segunda: **WP_0b_SOBRE_PLAZA** (−60, 5, −22) cruza la plaza central a 22 m de altitud, porque la ruta directa a −10 m atraviesa la copa de los árboles del patio (12–15 m de altura) y el dron entra en la malla de colisión antes de llegar al corredor. Recién entonces desciende a nivel de calle (−10 m): WP_1_ENTRADA_NORTE (−75, 10) entra por el extremo norte, entre los edificios norte y centro; **WP_1b_PASO_MOLDURA** (−70, 0) lo desplaza 8 m al este del saliente de cornisa del edificio oeste (x ≈ −78, y ≈ −14), donde, sin ese desplazamiento, las maniobras de evasión y el sesgo del controlador reactivo lo dejan trabado (960 ciclos sin salir de y ≈ −14 en una corrida piloto); WP_2_CENTRO_CORREDOR (−75, −35) cruza la zona de mayor densidad de árboles y WP_3_SALIDA_SUR (−75, −70) alcanza la avenida sur. WP_4 sale al este subiendo a −30 m y WP_5 regresa al punto de partida. La ruta total es de ~340 m (8 waypoints).

Qué se quiere probar. Es el escenario de **interés real** de Tier 1: cruza el interior del complejo por un corredor vegetado e incorpora los tramos de mayor dificultad del tier —el corredor bajo la copa de los árboles y el paso junto a la moldura del edificio oeste—. El desglose `summary_by_wp.csv` es aquí la herramienta central: permite aislar el comportamiento de cada tramo (ascenso, cruce de la plaza, entrada al corredor, paso de la moldura, cruce central) de la fase de crucero.

Dificultad. Media-alta. La obstrucción no es un muro sino un dosel: el sistema debe decidir con evidencia geométrica degradada.

Escenario 1.C — `townsim_calib_cruce_frontal` (bloqueo frontal masivo, lote 4)

WP	x (m)	y (m)	z (m)	Rol
WP_1	−150.0	0.4	−30.0	Tránsito de ida a altitud validada
WP_2	−150.0	0.4	−10.0	Descenso **fuera** del complejo, en bosque abierto
WP_3	0.0	0.4	−10.0	**Cruce real a nivel de calle** contra la fila de edificios oeste

Qué se quiere probar. Este es el escenario diseñado explícitamente para **forzar la activación de la capa estratégica y de la resolución de atascos**. El tramo WP_2→WP_3 pone al dron a nivel de calle frente a una fachada continua, obligándolo a elegir entre `GANAR_ALTURA` (sobrevolar) y un corredor lateral. La estructura de tres tramos no es arbitraria: los dos primeros son **tránsito a geometría ya verificada**, necesarios sólo porque el teletransporte está prohibido; el único tramo sin verificar —y el único que efectivamente prueba algo— es el tercero.

Particularidad. Es el escenario donde H1 se manifiesta más claramente: si la FSM resuelve el bloqueo escalando por encima del edificio y el `slm` lo resuelve encontrando la calle transversal, ambos tendrán `success = True` pero con SPL muy distintos. **Por eso la métrica primaria de los escenarios con obstrucción no es el éxito sino el SPL** — la diferencia de calidad entre soluciones aparece en la longitud de la trayectoria, no en el binario de llegada.

Dificultad. Alta y deliberada: se espera una tasa de atascos apreciable en `slm` y `fsm`, lo que ejercita el barrido de resolución de bloqueos.

10.3.3 Tier 2 — CitySim (`citysim_ortho.png`): cañones urbanos

Entorno. Proyecto de Unreal Engine 5.5 basado en el *Sample City* de Epic Games: tejido edilicio masivo en cuadrícula ortogonal regular, con edificios de **50 a 100 m de altura**, pasos restringidos y cañones urbanos angostos.

La particularidad que define a este tier es doble. Primero, la **altura de los obstáculos convierte a la altitud en variable de diseño de primer orden**: a diferencia de TownSim, donde −30 m basta para sobrevolar todo, en CitySim la maniobra `GANAR_ALTURA` deja de ser una salida universal — un escape por altura desde −20 m no despeja un edificio de 80 m. Segundo, la **regularidad y uniformidad de la textura urbana** es adversa para la percepción monocular: dos calles paralelas de ancho idéntico producen campos de flujo casi indistinguibles, y las fachadas repetitivas degradan el emparejamiento denso de DIS. Como en TownSim, el VLM planifica la ruta sobre el mapa cenital registrado antes del vuelo (`--route-plan vlm`, cap. 4, §4.3.1).

Escenario 2.A — `citysim_clear` (control: perímetro de manzana en rampa, lote 5)

Perímetro de una manzana del grid regular, **4 waypoints**, ~420 m. La altitud crece en rampa a lo largo del perímetro —WP_1 a −10 m, WP_2 a −30 m, WP_3 a −50 m— y el último tramo vuelve a −10 m en WP_4, sobre el punto de partida. Los cambios de altitud se hacen en diagonal, combinados con el avance horizontal.

Qué se quiere probar. Rol de control del tier, análogo a `minisim_clear` y `townsim_clear`: el costo de la arquitectura en tránsito libre, ahora entre edificios altos y con cambios de altitud en cada tramo. Si los tres brazos lo completan sin colisiones, la ruta no presenta obstrucciones a esas altitudes y cualquier atasco en el escenario de corredores de este tier es atribuible a la geometría del corredor.

Escenario 2.B — `citysim_pilot` (circuito de corredores, lote 6)

Circuito en grilla urbana a −10 m de altitud constante, atravesando corredores entre edificios en lugar de sobrevolarlos (manifiesto `citysim_pilot.json`, cap. 4 §4.3). Tres elementos del manifiesto merecen explicación:

- **Mapa** `citysim_ortho.png` (cap. 4 §4.3.2): mosaico cenital construido con el propio simulador, registrado con el marco NED por construcción. Sobre él se planifica la ruta antes del vuelo (cap. 4 §4.3.1).
- `start_pose` en el *spawn* real (0, 0, yaw 90°), que solo se usa con `--seed-jitter` (§10.4.3).
- **Waypoints verificados en el simulador.** Cada waypoint se comprueba ubicando el dron en él a −10 m y midiendo la profundidad en las cuatro direcciones y hacia abajo. `WP_0_SUR` (3, −51, −10) está en el centro de la calle este-oeste que sale del *spawn*, con la fachada del edificio sur a 13 m, y evita la autopista elevada; `WP_3` (67.7, −156.5, −10) está en el centro de la calle oeste de la manzana que separa WP_2 de WP_3, con fachadas a unos 7 y 10 m. WP_1, WP_2 y WP_5 quedan a 1.6–2.6 m por encima del tablero de la autopista elevada que cruza la zona a la altura de crucero (cap. 9, §9.8.7). El tramo WP_2 → WP_3 en línea recta atraviesa esa manzana (x ≈ 50–92 m, y ≈ −146 a −112 m): es el rodeo que la misión obliga a resolver.

Qué se quiere probar. Es el escenario terminal de la batería: **rodear edificios y elegir corredor**. Ni el `reactive` ni la FSM tienen un juicio sobre el fotograma para elegir entre dos calles de ancho similar; su decisión se reduce a distancia pura o a la asimetría accidental del flujo óptico. La hipótesis es que el brazo `slm`, al juzgar con la profundidad estimada qué sectores del fotograma están libres a 15 m, elegirá el rodeo con mayor consistencia.

El resultado negativo es igualmente válido y está previsto por diseño: si la capa estratégica no aporta en un entorno de textura urbana uniforme, ése es un hallazgo relevante sobre los límites de la percepción monocular en entornos repetitivos.

Dificultad. Máxima de la batería. A −10 m entre edificios de 50–100 m, el escape por altura está efectivamente vetado (`MAX_ESCAPE_ALT_M = 30 m` no despeja la línea de tejados), lo que **elimina la salida universal de la FSM** y obliga a una decisión lateral genuina. Es, en consecuencia, el escenario donde H1 tiene la mayor potencia discriminativa y donde se espera la mayor tasa de fallo absoluto en todas las celdas.

10.3.4 Síntesis de la batería

Lote	Tier	Entorno (proyecto UE)	Escenario	Rol	Long. óptima	Altitud	Dificultad esperada	Hipótesis que ejercita
1	0	MiniSim (plano sin obstáculos)	`minisim_clear`	Control / cota inferior	~180 m	−10 m	Nula	H2, H3 (costo puro)
2	1	TownSim (`townsim_ortho.png`)	`townsim_clear`	Control de crucero largo	~640 m	−30 m	Baja	H2 (dilución del costo)
3	1	TownSim	`townsim_ini`	Vegetación en ruta	~340 m	−30/−22/−10 m	Media-alta	H1
4	1	TownSim	`townsim_calib_cruce_frontal`	Bloqueo frontal masivo	~320 m	−30/−10 m	Alta	**H1**
5	2	CitySim (`citysim_ortho.png`)	`citysim_clear`	Control en rampa sobre el perímetro de una manzana	~420 m	−10/−30/−50 m	Baja	H2, H3
6	2	CitySim	`citysim_pilot`	Rodeo de manzanas y elección de corredor	~336 m	−10 m	**Máxima**	**H1, H3**

10.4 Semillas: por qué varias y qué cambia cada una

10.4.1 Por qué se requieren múltiples semillas

Una corrida única no permite ninguna inferencia. La razón no es formal sino sustantiva: **el sistema bajo prueba es estocástico en varios puntos independientes** (§10.4.2), y una diferencia observada entre dos corridas únicas es indistinguible de la varianza intrínseca del entorno. La práctica de reportar resultados de agentes secuenciales a partir de pocas corridas está bien documentada como fuente sistemática de conclusiones no reproducibles (Henderson et al., 2018; Colas et al., 2018), y el problema se agrava exactamente en el régimen de este trabajo: **pocas corridas por celda, porque cada corrida cuesta minutos de tiempo real de simulación** (Agarwal et al., 2021).

Hay tres funciones distintas que cumplen las semillas en este diseño, y conviene no confundirlas:

1. **Función inferencial.** Sin réplicas no hay distribución muestral, y sin distribución muestral no hay prueba de hipótesis. Las pruebas no paramétricas del §10.7 operan sobre las distribuciones de las K semillas de cada celda.
2. **Función de robustez.** Un brazo que tiene éxito en 10 de 10 semillas es cualitativamente distinto de uno que tiene éxito en 6 de 10, aun con el mismo tiempo medio. La **tasa de éxito sobre semillas** es en sí misma la métrica primaria de fiabilidad, análoga al `success rate` sobre episodios de los *benchmarks* de conducción (Codevilla et al., 2019).
3. **Función de detección de acoplamiento.** Si dos semillas del brazo `reactive`, sin deliberación, producen trayectorias idénticas al milímetro, el entorno no está aportando la variación que se supone que aporta, y las semillas no son réplicas independientes. La varianza del brazo `reactive` funciona así como **control negativo del propio diseño experimental**: es la medida de cuánta variabilidad introduce el entorno sin intervención de la capa deliberativa.

10.4.2 Qué cambia realmente una semilla

Este punto exige precisión, porque la implementación tiene un matiz que es fácil de reportar mal. En la configuración de los lotes (**sin** `--seed-jitter`), la variable `AIRSIM_SEED` **no perturba deliberadamente ningún parámetro del sistema**: se propaga al `FlightLogger` como etiqueta de identificación de la corrida. El runner llama `client.reset()`, que devuelve el vehículo a su pose de *spawn* original definida en `settings.json`, y no vuelve a reposicionarlo.

En consecuencia, **una semilla identifica una réplica independiente, no una condición distinta**. La variación entre semillas proviene íntegramente de la estocasticidad intrínseca del sistema, que tiene cinco fuentes identificadas:

1. **Jitter temporal del lazo de control.** El lazo corre a `LOOP_HZ = 5.0` con temporización de reloj de pared (`time.sleep(max(0, 1/f - t_ciclo))`). El tiempo de cada ciclo depende de la latencia RPC a AirSim, de la carga de GPU y del planificador del sistema operativo. Un ciclo que se pasa del presupuesto desplaza todos los siguientes: dos corridas idénticas divergen en **qué instante exacto**

de la trayectoria se toma cada decisión. Con `REACTIVE_FORWARD_SPEED = 3.0 m/s`, 20 ms de desfase equivalen a 6 cm de desplazamiento por ciclo, que se acumulan.

2. **Renderizado y física no reproducibles cuadro a cuadro.** Unreal Engine no garantiza determinismo cuadro a cuadro bajo carga variable de GPU; la física de AirSim avanza en tiempo real, no en pasos fijos sincronizados con el cliente. El fotograma capturado en el ciclo n no es bit a bit el mismo entre corridas, y el estimador de flujo óptico —que opera sobre diferencias de intensidad a nivel de píxel— amplifica esas diferencias.
3. **Sensibilidad del estimador de flujo a condiciones iniciales.** El cálculo de FOE por mínimos cuadrados ponderados con rechazo de *outliers* (RANSAC-lite, `FOE_OUTLIER_ANGLE_RAD = 0.35`) es una función no lineal del campo de flujo. Pequeñas diferencias en el conteo de inliers cruzan el umbral de confianza diferenciada (< 30 vs. ≥ 30 inliers), lo que cambia el estado epistémico reportado por `ObstacleField` y, con él, qué regla de `navigate_node` decide el ciclo. Es un sistema con **realimentación**: una decisión distinta cambia la pose, que cambia el fotograma siguiente, que cambia la decisión siguiente.
4. **Inferencia de las redes del lazo lento.** La red de profundidad es determinista salvo diferencias numéricas mínimas de la GPU, pero recibe fotogramas distintos (fuente 2) en instantes distintos (fuente 1). El VLM no interviene en vuelo; en TownSim y CitySim planifica la ruta una vez por escenario, con temperatura 0 y caché por contenido, de modo que todas las semillas y celdas de un lote vuelan el mismo plan (cap. 4, §4.3.1). Si se usara el VLM en vuelo (`STRATEGIC_SOURCE = vlm`), su muestreo (`temperature = 0.2`, sin semilla fijada en el servidor) sería una fuente adicional exclusiva del brazo `slm`; la determinación de la inferencia en servidores de LLM no está garantizada aun con temperatura cero, porque los kernels no son invariantes al tamaño de lote y este varía con la carga del servidor (He, 2025).
5. **Deriva temporal de las respuestas asíncronas.** Que una respuesta estratégica llegue antes o después de su edad máxima (10 s), o de que el dron supere la sub-meta que produciría, y que un barrido se resuelva antes o después de su watchdog (`SLM_DEEP_WATCHDOG_MS`), depende de la latencia de la red en ese instante; el mismo fotograma puede producir un rodeo en una corrida y nada en otra. Cambia cualitativamente qué decide el ciclo.

Estas cinco fuentes se acumulan multiplicativamente sobre una misión de miles de ciclos. La divergencia entre dos semillas de la misma celda es, por tanto, real y sustancial — pero es **estocasticidad no controlada**, no un factor manipulado.

10.4.3 El modo de perturbación controlada y por qué no es el modo por defecto

El runner ofrece un modo alternativo (`--seed-jitter`) en el que la semilla **sí** determina una condición distinta y reproducible: siembra un `random.Random(seed)` que genera un desplazamiento de la pose inicial de ± 1.5 m en x e y y $\pm 10^\circ$ en guiñada (`SEED_JITTER_XY_M`, `SEED_JITTER_YAW_DEG`). Es el análogo directo de las *sticky actions* introducidas en la revisión del Arcade Learning Environment para inyectar estocasticidad controlada y evaluar la robustez de una política en lugar de su capacidad de memorizar una trayectoria (Machado et al., 2018).

Este modo está deshabilitado por defecto, y la razón es una decisión metodológica explícita. El jitter se aplica vía `simSetVehiclePose(ignore_collision=True)`, un teletransporte que no verifica colisión: con ± 1.5 m de desplazamiento, el dron puede materializarse más cerca de un poste o un tronco que en el *spawn* limpio de AirSim, introduciendo variación entre corridas que **no proviene de la política de control sino de la posición del obstáculo más cercano al punto de partida**. Eso viola el principio de no teletransporte del §10.3.0 y contamina la comparación entre brazos: la corrida "difícil" le tocaría a un brazo por sorteo, no por diseño.

La consecuencia se asume y se declara: **en los lotes, la variación entre semillas es la variación intrínseca del entorno, y la comparación entre brazos es una comparación de réplicas independientes bajo condiciones iniciales nominalmente idénticas.** El modo con jitter se reserva para el plan extendido (§10.10), donde se lo trata como un factor experimental propio y se lo valida antes de usarlo — verificando en el *viewport* que ninguna de las poses sorteadas cae dentro de un obstáculo.

10.4.4 Número y asignación de semillas

Se ejecutan $K = 10$ **semillas por celda**. El número lo impone la potencia estadística. Con K semillas por grupo, el p-valor mínimo alcanzable de la prueba de rangos (bilateral, sin empates) es $2/\binom{2K}{K}$: con $K = 5$ sería 0.0079, de modo que ninguna comparación podría sobrevivir la corrección de Bonferroni del §10.7 ($\alpha_{\text{ajustado}} \approx 0.0028$) ni siquiera con separación perfecta; con $K = 10$ baja a 1.1×10^{-5}, holgadamente por debajo del umbral corregido. El límite superior lo impone el presupuesto de tiempo real (§10.5.4).

Asignación. Las semillas son los primeros 60 números primos, sin repetir entre lotes (diez por lote, en orden). Dentro de cada semilla, las celdas se alternan, con el orden rotado de una semilla a la siguiente (`runner.py --cells`): la deriva del simulador a lo largo de horas (temperatura de la GPU, fragmentación de memoria de UE) se reparte así entre celdas en lugar de concentrarse en la que corre al final.

El reporte no se apoya sólo en el p-valor: se informan tamaños de efecto con intervalo de confianza *bootstrap* y las distribuciones completas, que es la práctica recomendada precisamente para el régimen de pocas corridas (Agarwal et al., 2021). Un tamaño de efecto grande con un intervalo de confianza ancho es información útil y honestamente reportable; un p-valor no significativo **no es evidencia de ausencia de efecto**, y así se lo declara en el capítulo 11.

10.5 Organización en lotes

10.5.1 Por qué un lote por escenario, agrupados por nivel de Unreal Engine

Los tres tiers corresponden a **tres proyectos de Unreal Engine distintos** (MiniSim, TownSim y CitySim, §10.3), cada uno con su propio nivel, su propio *PlayerStart* y su propio `settings.json` de AirSim. El cliente de AirSim se conecta por RPC a la instancia de UE que esté corriendo en ese momento; **no existe forma programática de cambiar de proyecto desde el runner**. Un escenario de Tier 1 lanzado contra el nivel de Tier 0 no falla con un error legible: vuela sobre el terreno equivocado y produce una corrida silenciosamente inválida.

La unidad de ejecución es el **lote**: todas las corridas de un escenario (sus celdas × 10 semillas). Seis escenarios dan seis lotes, y solo se corren juntos los lotes del mismo nivel de UE: 1 | 2, 3, 4 | 5, 6. La organización tiene dos consecuencias metodológicas favorables:

1. **Aísla la fuente de contaminación cruzada más peligrosa.** Todas las corridas de un lote comparten el mismo nivel, la misma configuración de escalabilidad gráfica y la misma sesión de UE. Comparar entre celdas *dentro* de un lote es por tanto una comparación limpia.
2. **Obliga a declarar el análisis entre lotes como comparación de segundo orden** (§10.8.3): los tiers no comparten sesión de simulador, de modo que las diferencias absolutas entre tiers arrastran un efecto de sesión no controlado, y sólo las **razones normalizadas dentro de cada lote** (p. ej. $t_{\text{slm}}/t_{\text{reactive}}$) son comparables entre lotes.

10.5.2 Procedimiento por lote

Cada lote se ejecuta según la siguiente secuencia, en este orden estricto:

1. **Árbol confirmado.** El runner no arranca con cambios sin commitear en `airsim-loop/`, `config/`, `airsim-plan/` o `experiments/` (termina con código 2), de modo que todas las corridas de un lote llevan el mismo `code_version` (§10.9.3). Tampoco arranca con otro intérprete de Python que el de `AIRSIM_ENV_DIR` en `config/.env` (Anexo 6).
2. **Preparación del entorno.** Abrir en Unreal el nivel del escenario y aplicar el perfil de escalabilidad documentado en el capítulo 3 (`Config/DefaultScalability.ini`, con las sombras necesarias para que la estimación de TTC disponga de los gradientes que usa el flujo). En los lotes 2 a 6, Ollama debe responder en `127.0.0.1:11434` para el plan de ruta (`startOllama.ps1`; el script del lote lo verifica): el runner calcula el plan una vez por escenario antes de la primera corrida y todas las corridas del lote vuelan ese plan. En el lote 1 (`--route-plan off`) el servidor puede quedar cerrado y su VRAM queda para Unreal (Anexo 6). La red de profundidad la carga el propio grafo al compilarse (~19 s).
3. **Higiene de estado.** El runner ejecuta `client.reset()` y `client.clear_debug_markers()` al inicio de cada corrida. No se depende de que el operador recuerde limpiar los marcadores de `plot_mission_route.py`.

4. **Piloto de validación.** En los lotes con obstrucción (3, 4 y 6), una corrida `slm:depth` con la semilla 99, revisada contra el criterio de pase del escenario antes de lanzar el lote (§10.12.2). Los pilotos se guardan aparte y no entran al análisis.
5. **Lote completo.** Un subproceso por corrida (escenario × celda × semilla), con las celdas alternadas dentro de cada semilla (§10.4.4). Un lote cortado se reanuda con `-Resume`, que saltea las corridas válidas ya hechas.
6. **Análisis.** `airsim-loop/experiments/analyze_lotes.py` sobre la raíz de los seis lotes (§10.7).

10.5.3 Celdas, semillas y presupuesto de cada lote

Los scripts de los lotes están versionados en `experiments/` (`run_lotes.ps1` define el diseño; cada `run_loteN_*.ps1` lo llama con su número). El diseño completo es:

Lote	Escenario	Nivel UE	Celdas	Corridas	Semillas	Plan de ruta	Tiempo estimado
1	`minisim_clear`	MiniSim	`slm:depth`, `fsm:depth`, `reactive`	30	2 3 5 7 11 13 17 19 23 29	off	~1.5 h
2	`townsim_clear`	TownSim	ídem	30	31 37 41 43 47 53 59 61 67 71	vlm	~3 h
3	`townsim_ini`	TownSim	ídem + `fsm:blind`	40	73 79 83 89 97 101 103 107 109 113	vlm	≤ 11.5 h
4	`townsim_calib_cruce_frontal`	TownSim	ídem + `fsm:blind`	40	127 131 137 139 149 151 157 163 167 173	vlm	≤ 14 h
5	`citysim_clear`	CitySim	`slm:depth`, `fsm:depth`, `reactive`	30	179 181 191 193 197 199 211 223 227 229	vlm	~2 h
6	`citysim_pilot`	CitySim	ídem + `fsm:blind`	40	233 239 241 251 257 263 269 271 277 281	vlm	≤ 12 h
Total				**210**			**~44 h** (peor caso)

En los lotes despejados el tiempo estimado es el típico (las corridas terminan en pocos minutos); en los de obstrucción, el peor caso, con las celdas que se traban agotando el presupuesto.

10.5.4 Presupuesto por corrida y ejecución

Todas las corridas tienen el mismo techo: **1200 s / 6000 ciclos**. Es el mismo en todos los escenarios y holgado, para que una celda lenta falle por el obstáculo y no por el tiempo; las corridas exitosas terminan antes, y el techo solo alarga las que se traban. A `LOOP_HZ = 5`, con el lazo cumpliendo su presupuesto, se agota antes el tiempo que los ciclos; el límite de ciclos protege el caso patológico opuesto —un lazo que se acelera porque las capturas fallan silenciosamente—.

Los lotes se lanzan desde `experiments/`, en PowerShell:

```
.\run_lote3_townsim_ini.ps1 -Pilot    # piloto -> airsim-
runs\produccion\v4\pilotos\lote3_townsim_ini
.\run_lote3_townsim_ini.ps1           # lote completo
.\run_lote3_townsim_ini.ps1 -Resume   # reanudar un lote cortado
.\run_lotes.ps1 -Lotes 2,3,4          # varios lotes del mismo nivel de UE
```

Cada script llama a `runner.py` con `--scenarios`, `--cells`, `--seeds`, `--out-dir`, `--max-seconds 1200`, `--max-cycles 6000` y `--route-plan`; las opciones `-SeedJitter` y `-NoVideo` agregan `--seed-jitter` y `--no-video`.

10.6 Métricas reportadas

Todas las métricas se derivan del registro por ciclo del `FlightLogger` y se consolidan en el `summary.json` de cada corrida (cap. 4, §4.4).

10.6.1 Eficacia y seguridad de vuelo

- **Tasa de éxito de misión** (`success`): arribo a todos los waypoints dentro del presupuesto de ciclos y segundos. Es un binario por corrida; sobre K semillas se convierte en una proporción, que se informa con su intervalo de Wilson al 95 %.
- **Colisiones** por misión y **normalizadas por kilómetro** recorrido. La normalización es necesaria porque los escenarios difieren en longitud por un factor de 3.3×.
- **Motivo de terminación** (`termination_reason`): éxito, tope de tiempo o de ciclos, colisión, o `physics_locked` (el vigilante de congelamiento detectó el estado físico idéntico durante 25 ciclos: dron incrustado en la malla, cap. 5 §5.19). `physics_locked` cuenta como falla de misión, igual que una colisión.
- **Distancia mínima a obstáculo** (`min_obstacle_dist_m`, DistMin): la mide un **hilo de auditoría aislado** durante el vuelo. Con su propia conexión al simulador, cada `DISTMIN_TRACK_PERIOD_S` (1 s, una muestra cada 5 ciclos) captura la profundidad `DepthPlanar` de la cámara frontal y toma el percentil 5 del tercio central de la imagen; DistMin es el mínimo sobre la corrida. Las muestras se escriben en `<corrida>.distmin.ndjson` y `finalize_run` vuelca el mínimo en el `summary.json` al cerrar la corrida.

El hilo no comparte ningún objeto con el grafo, de modo que la medición no puede influir en el control (§10.3.0, punto 3).

- **SPL** (*Success weighted by Path Length*), en su definición estándar para agentes de navegación (Anderson et al., 2018): $\mathrm{SPL} = S \cdot \ell_{\mathrm{opt}} / \max(\ell_{\mathrm{real}}, \ell_{\mathrm{opt}})$, con S el indicador de éxito. La longitud óptima ℓ_{opt} se computa como la suma de distancias euclídeas entre waypoints consecutivos del manifiesto: no es un camino volable en presencia de obstáculos, pero es la referencia estándar y, sobre todo, es **idéntica para todas las celdas**, que es lo que la comparación requiere. El SPL es la única métrica que penaliza simultáneamente no llegar y llegar dando vueltas, y es por eso la métrica primaria de los escenarios con obstrucción.

> **Sesgo conocido del cálculo.** `analyze.py` (que usa también `analyze_lotes.py`) computa ℓ_{opt} sumando únicamente las distancias entre waypoints **declarados en el manifiesto**, sin incluir el tramo desde el *spawn* hasta el primer waypoint, que sí queda contabilizado en ℓ_{real}. En `minisim_clear` ese tramo son 60 m sobre 120 m declarados: el SPL queda subestimado en aproximadamente un tercio. El sesgo es **constante por escenario e idéntico para todas las celdas**, de modo que no invalida ninguna comparación dentro de un lote, pero sí impide interpretar el valor absoluto del SPL y compararlo entre escenarios. Se reporta en consecuencia el SPL como métrica **relativa dentro de cada lote**. - **Tiempo y ciclos totales** a destino.

10.6.2 Dinámica del lazo táctico y latencias

- **Latencia total por ciclo** (p_{50} y p_{95}), desagregada por brazo y por **ruta activa del grafo de control** (`reactive`, `evasive`, `girar_90`, `deliberative` —barrido—, `tactical` —escape vertical—, `fsm`, `degraded`). El p_{95} es la métrica relevante para la viabilidad en tiempo real: un p_{50} aceptable con un p_{95} que excede el presupuesto de ciclo describe un sistema que cumple "en promedio" y falla justo cuando importa.
- **Histograma de rutas** por corrida: la distribución de ciclos entre nodos del grafo. Es la caracterización más informativa del comportamiento cualitativo de un brazo — una corrida mayormente `reactive` describe un vuelo que sobrevoló los obstáculos en lugar de negociarlos, independientemente de lo que digan sus métricas de éxito.
- `deliberation_rate`: fracción de ciclos con una consulta nueva del lazo lento. Se contabiliza por **transición de identificador único de deliberación**, cualquiera sea la ruta del ciclo; sin esa distinción, una consulta que abarca varios ciclos de espera se contaría múltiples veces. Incluye las consultas de la capa estratégica, que se resuelven mientras el dron vuela por la ruta reactiva o evasiva.
- El **escaneo espacial pre-vuelo** (`spatial_scan`, cap. 5 §5.0) solo lo ejecuta `main.py`; el runner no lo corre, de modo que no interviene en ninguna corrida experimental.

10.6.3 Comportamiento del lazo lento

- **Consultas efectivas por misión** (`slm_invocations`).
- **Tasa de *fallback*** (`slm_fallback_rate`) ante respuestas inválidas o fuera de esquema, y **tasa de *timeout*** (`slm_timeout_rate`).

- **Distribución de resultados de la capa estratégica**, a partir de las entradas `arm = "depth_strategic"` de `deliberations[]`: fracción de respuestas que proponen sub-meta, que declaran el camino directo libre (`directo_libre`), sin sector libre, vencidas, superadas o no parseables. Complementa al banco de prueba (§10.13), que mide si la fuente **discrimina** contra etiquetas de referencia: una capa que declara `directo_libre` en casi todas las consultas, incluso cuando el dron termina en deadlock, no está aportando información.
- **Avance hacia el waypoint real tras cada sub-meta**: distancia al waypoint de misión (no a la sub-meta) en el momento de la inyección y 20 s después. Es la medida directa de si las sub-metas ayudan a completar la misión: una sub-meta que no acerca al waypoint real en ese plazo es un desvío sin beneficio.
- **Avance en waypoints del plan a tiempos de referencia** y **segundos a menos de 1 m de un obstáculo** por tramo (con la posición de cada muestra de DistMin), calculados por `analyze_strategic_batch.py`.

10.6.4 Resolución de atascos

- `deadlock_events`: número de atascos declarados.
- `deep_scan_resolution_rate`: fracción de atascos resueltos por el barrido.
- `deep_scan_avg_cycles_to_resolve`: costo medio en ciclos de una resolución exitosa.
- `deep_scan_fallback_rate`: fracción de atascos donde el barrido no resolvió y se cayó al escape vertical.

Estas cuatro métricas sólo tienen valor conjunto: una tasa de resolución alta con un costo de ciclos desmesurado no describe un mecanismo exitoso sino uno caro. «Resuelto» significa que la fuente del barrido encontró un rumbo transitable y se fijó una sub-meta, **no** que el dron haya salido del atasco; por eso se reporta además el avance hacia el waypoint real tras cada resolución.

10.6.5 Desglose espacial por waypoint

Cada corrida emite `<stem>.summary_by_wp.csv` con ciclos consumidos, duración, ciclos deliberativos, tasa de deliberación y eventos de barrido **por tramo de misión**. Es la herramienta que permite aislar los tramos de ascenso vertical de las fases de crucero y maniobra horizontal: sin este desglose, una métrica agregada de misión atribuye a "el escenario" una dificultad que puede pertenecer a un solo tramo de 20 segundos, como el ascenso inicial entre follaje de `townsim_ini`.

10.7 Protocolo estadístico

El análisis lo implementa `airsim-loop/experiments/analyze_lotes.py`, que escribe `analisis_lotes.md` y dos CSV en la raíz de los lotes:

```
D:\TesisMCD\airsimenv\python.exe ..\airsim-loop\experiments\analyze_lotes.py ..\airsim-runs\produccion\v4 --family 18
```

10.7.1 Métricas primarias y comparaciones planificadas

La métrica primaria se declara por tipo de escenario antes de ver los datos:

- **Lotes despejados (1, 2 y 5):** tiempo de las misiones exitosas (H2). Comparaciones: `slm` – `fsm`, `slm` – `reactive` y `fsm` – `reactive`.
- **Lotes con obstrucción (3, 4 y 6):** SPL (H1). Comparaciones: `slm:depth` vs. `fsm:blind`, `fsm:depth` vs. `fsm:blind` y `slm:depth` vs. `fsm:depth` (§10.12.2).

Tres comparaciones por lote y seis lotes forman la familia de 18 comparaciones primarias.

10.7.2 Pruebas de hipótesis

Cada comparación primaria usa una **prueba de permutación exacta sobre rangos** (bilateral), equivalente a la U de Mann-Whitney exacta (Mann & Whitney, 1947) pero válida con empates, frecuentes en SPL porque todas las corridas fallidas tienen SPL = 0. Hipótesis nula: la distribución de la métrica no difiere entre las dos celdas.

La elección de pruebas no paramétricas no es una precaución de rutina: es una necesidad del tipo de dato. Las distribuciones de tiempo de misión están **truncadas por la derecha** por el presupuesto (una corrida que se agota registra el presupuesto, no su tiempo real), y las de SPL tienen una masa en cero; ninguna prueba que asuma normalidad es defendible sobre esos datos.

Secundarias, sin corrección y rotuladas como tales: el éxito (prueba exacta de Fisher y diferencia de tasas con intervalo de Newcombe al 95 %) y la distancia mínima al obstáculo (delta de Cliff y la misma prueba de rangos).

10.7.3 Tamaños de efecto

Se reporta **Cliff's Delta** (δ) para toda comparación (Cliff, 1993), definido como la proporción de pares en que una muestra domina a la otra: $\delta = 2U/(n_A n_B) - 1$, con un intervalo de confianza *bootstrap* percentil (5000 remuestreos de corridas dentro de cada celda). Se adopta la escala de interpretación de Romano et al. (2006): $|\delta| < 0.147$ despreciable, 0.147–0.330 pequeño, 0.330–0.474 mediano, ≥ 0.474 grande.

El tamaño de efecto no es un complemento decorativo del p-valor: **es el resultado principal cuando K es pequeño**. Un p significativo con δ despreciable describe una diferencia real de magnitud práctica nula; un δ grande con p no significativo describe un efecto que el diseño no tuvo potencia para detectar, no un efecto ausente. Ambos casos se reportan explícitamente como tales.

10.7.4 Comparaciones múltiples

Se aplica **corrección de Bonferroni** sobre la familia de 18 comparaciones primarias: $\alpha = 0.05/18 \approx 0.0028$. Con $K = 10$, una separación perfecta da $p = 1.1 \times 10^{-5}$ y pasa la corrección.

La corrección se aplica **sólo a las comparaciones declaradas antes de ver los datos**. Cualquier comparación adicional sugerida por la inspección de los resultados se reporta como **exploratoria**, sin pretensión de significancia — la alternativa (corregir sobre una familia que crece a medida que se exploran los datos) sería a la vez estadísticamente incorrecta y prácticamente inútil.

10.7.5 Análisis desagregado

El análisis **no se limita a valores agregados globales**. Se evalúa el rendimiento desagregado por escenario y morfología de obstáculo, porque es la única forma de contrastar H3: la pregunta no es "¿es mejor el `slm`?" sino "¿en qué condiciones lo es, y a qué costo?". Un promedio sobre los seis escenarios mezclaría escenarios donde el brazo `slm` sólo puede perder (los de control) con escenarios donde puede ganar, produciendo un número sin interpretación posible. Es la misma razón por la que los *benchmarks* de conducción reportan por nivel de dificultad en lugar de agregarlos (Codevilla et al., 2019).

10.8 Análisis esperado y análisis comparativo conjunto

Esta sección declara, **antes de tener los datos**, qué resultado se espera de cada lote y cómo se leerá. Un resultado que contradiga estas expectativas es informativo precisamente porque las expectativas estaban escritas de antemano.

10.8.1 Lotes despejados (1, 2 y 5) — qué se espera

El resultado esperado es **desfavorable al brazo `slm` y favorable a la hipótesis H2**:

- Orden esperado de tiempo de misión: `reactive` < `fsm` < `slm`, con δ grande en la comparación `reactive` vs. `slm` en MiniSim.
- Ratio $t_{\text{slm}}/t_{\text{reactive}}$ decreciente con la longitud de la ruta: mayor en `minisim_clear` (~180 m) que en `townsim_clear` (~640 m). **Ésta es la prueba directa de H2**, y su forma es una comparación de razones, no de valores absolutos.
- Sin diferencias de seguridad: en un campo despejado, ningún brazo puede ser más seguro que otro.
- En `citysim_clear`, cero atascos en las tres celdas confirma que la ruta en rampa (−10 a −50 m) no presenta obstrucciones y que las altitudes son adecuadas.

Qué invalida la lectura: cualquier fallo de misión en el lote 1. Sin obstáculos, un fallo indica un problema del sistema (percepción, guiado o infraestructura), no del escenario, y obliga a diagnosticarlo antes de continuar.

10.8.2 Lotes con obstrucción (3, 4 y 6) — qué se espera

- `townsim_ini`: se espera la primera aparición de una ventaja del `slm` en SPL, concentrada según `summary_by_wp.csv` en los tramos con follaje (corredor y paso de la moldura). Si la ventaja aparece pero está distribuida uniformemente entre tramos, la explicación no es la capa estratégica sino un sesgo sistemático del guiado, y debe investigarse como tal.
- `townsim_calib_cruce_frontal`: se espera una tasa de atascos apreciable en `slm` y `fsm`. La lectura esperada es que el `slm` produzca el mejor SPL al resolver el bloqueo lateralmente, mientras la FSM recurre al escape por altura; `fsm:depth` frente a `fsm:blind` mide cuánto aporta el barrido por sí solo.
- `citysim_pilot`: es el escenario terminal y el de mayor incertidumbre. Se esperan **tasas de éxito inferiores a 1.0 en todas las celdas** —es el único escenario donde el fracaso es un resultado esperado y no un síntoma— y una mayor dispersión entre semillas. Si la capa estratégica no discrimina entre corredores de textura uniforme, `slm:depth` y `fsm:depth` empatarán, y ese empate es un resultado sustantivo.

Qué invalida la lectura: cero atascos en `townsim_calib_cruce_frontal`. Significaría que el tramo de cruce no está encontrando la fachada —error de geometría, no de política— y obligaría a re-validar el manifiesto con `plot_mission_route.py`.

10.8.3 Análisis comparativo conjunto

El cruce entre lotes es donde se contrasta **H3**, y exige una precaución declarada en §10.5.1: los lotes de distintos tiers corren en sesiones de simulador distintas y sobre proyectos de UE distintos, de modo que **las diferencias absolutas entre tiers arrastran un efecto de sesión no controlado**. La comparación entre lotes se realiza en consecuencia sobre tres tipos de cantidad, todas normalizadas dentro de su propio lote:

1. **Razones normalizadas por brazo.** $t_{\text{brazo}}/t_{\text{reactive}}$ dentro de cada escenario. Al dividir por el brazo de referencia del mismo escenario y la misma sesión, el efecto de sesión se cancela en primer orden. La progresión de $t_{\text{slm}}/t_{\text{reactive}}$ entre los tres controles es la evidencia central de H2.
2. **Perfil de consulta por dificultad.** `deliberation_rate` y deadlocks en función del escenario. La predicción de H3 es un perfil **no monótono**: bajo en los controles y alto en los escenarios con obstrucción real, con consultas *justificadas*. La distinción entre "alto por ruido" y "alto por necesidad" se hace cruzando `deliberation_rate` con el SPL resultante y con `summary_by_wp.csv`.
3. **Interacción brazo × escenario.** El resultado que sostiene H3 es un **cambio de signo** del efecto `slm` − `fsm` entre los escenarios de control y los de obstrucción. Con la potencia disponible ($K = 10$), esta interacción se reporta como patrón de tamaños de efecto con sus intervalos de confianza, no

como una prueba formal de interacción: un ANOVA de dos factores sobre datos truncados y con 10 réplicas no sería defendible.

El producto final del análisis conjunto es una tabla de doble entrada **celda × escenario** con éxito, SPL, tiempo normalizado, seguridad y tasa de consulta, más el conjunto de comparaciones con su δ e intervalo de confianza. La conclusión de la tesis no es un único número sino ese mapa: **dónde la capa deliberativa paga y dónde no**.

10.9 Organización de los resultados, auditoría y reproducibilidad

10.9.1 Disposición física de los artefactos

Los resultados quedan fuera de git (`airsim-runs/` está en `.gitignore`: los logs, CSV y videos son demasiado pesados), bajo una estructura derivada de los propios factores del diseño:

```
airsim-runs/produccion/v4/
├── loteN_<escenario>/
│   ├── batch_<ts>.log                  # registro del lote
│   └── <escenario>/                    # stem del manifiesto: townsim_ini, citysim_pilot, …
│       ├── route_plan/                 # plan de ruta del escenario (TownSim y CitySim)
│       └── <brazo>/                    # slm | fsm | reactive
│           └── <estrategia>/           # depth | blind
│               └── seed_<N>_<timestamp>/    # una carpeta autocontenida por corrida
│                   ├── seed_<N>_<ts>.jsonl            # DroneState completo ciclo a ciclo
│                   ├── seed_<N>_<ts>.csv              # misma corrida, tabular (columnas
state.*)
│                   ├── seed_<N>_<ts>.summary.json     # indicadores agregados
│                   ├── seed_<N>_<ts>.summary_by_wp.csv
│                   ├── seed_<N>_<ts>.distmin.ndjson   # muestras de la auditoría de DistMin
(1 Hz)
│                   ├── seed_<N>_<ts>.console.log      # copia de la consola de la corrida
│                   ├── seed_<N>_<ts>.webm / .follow.webm / .viewer.html
│                   └── photo-<timestamp_ISO>.png      # fotogramas de cada consulta del lazo
lento
└── pilotos/loteN_<escenario>/          # pilotos de validación, fuera del análisis
```

El contenido de cada archivo —campos de la traza, columnas del CSV, indicadores del resumen— y el uso del visor se describen en el capítulo 4 (§4.4).

La ruta **es** el registro del punto del diseño al que pertenece la corrida: escenario, brazo, estrategia y semilla son directorios, no metadatos que haya que parsear, y ninguna corrida sobrescribe a otra. Los fotogramas de auditoría se nombran por *timestamp* de captura con precisión de milisegundos y se cruzan con la telemetría por el campo `slm_frame_paths` del CSV.

El CSV plano por corrida contiene, además de la cinemática y las lecturas sectoriales del `ObstacleField`, la respuesta cruda de cada consulta del lazo lento (con la profundidad, el p5 por sector), junto al identificador de la entrada correspondiente del JSONL. La decisión de duplicar esa información en el CSV es deliberada: permite analizar el comportamiento de la fuente sin cruzar archivos.

10.9.2 Video y visor de auditoría

Tanto las corridas interactivas (`main.py`) como las del runner generan, por defecto, el video anotado de la cámara frontal (`.webm`, VP8, un cuadro por ciclo con superposición de telemetría, TTC por sector y ruta activa), el video de la cámara de seguimiento externa (`.follow.webm`, solo auditoría) y un visor HTML autocontenido que sincroniza ambos videos con la traza por índice de ciclo y muestra el `DroneState` y la consulta del lazo lento de cada ciclo (cap. 4, §4.4). La codificación corre en un hilo aparte para no alterar el período del lazo (cap. 5, §5.18). En lotes largos puede desactivarse con `-NoVideo`.

10.9.3 Trazabilidad

Cada corrida incorpora el hash de commit de Git (`code_version`) en su `summary.json`. Si al correr había cambios sin commitear en el código, la configuración, los manifiestos o los scripts de los lotes (`airsim-loop/`, `config/`, `airsim-plan/`, `experiments/`), el valor lleva el sufijo `-dirty`: el hash solo identifica el código si el árbol está limpio. El runner no arranca un lote con el árbol sucio (§10.5.2), y `analyze_lotes.py` toma como referencia el `code_version` del lote y descarta las corridas con otro hash o con sufijo `-dirty` (§10.12.2). Agrupar corridas con software de decisión distinto mezclaría el efecto del software con el de los brazos; por eso solo entran al análisis estadístico las corridas de los seis lotes con el mismo hash. Las corridas piloto de validación y las corridas de diagnóstico del capítulo 9 no entran: son evidencia de diagnóstico.

10.9.4 Publicación

El conjunto de validación de TTC (cap. 7, 3735 registros), los archivos de telemetría de los seis lotes, los manifiestos de misión y la configuración completa (`config/.env`) serán archivados y publicados con un identificador digital persistente (DOI vía Zenodo), acompañados del hash de commit correspondiente, garantizando la auditabilidad y reproducibilidad de los resultados.

(Para la especificación formal del archivo `settings.json`, el perfil de escalabilidad, la estructura detallada de los manifiestos de misión JSON y el protocolo de ejecución paso a paso, véase el Anexo 6).

10.10 Plan extendido de pruebas

Los seis lotes son el mínimo necesario para responder H1–H3. Esta sección esboza los experimentos adicionales que el diseño admite sin cambios estructurales, ordenados por relación aporte/costo.

10.10.1 Extensiones de Tier 0 (MiniSim)

Diseño	Qué aporta
`minisim_clear` × `REACTIVE_FORWARD_SPEED ∈ {2, 5, 7}` m/s × 10 semillas	Establece la **frontera de viabilidad temporal** del brazo `slm`: a qué velocidad la latencia del lazo lento deja de ser absorbible por el avance. Los umbrales de TTC son de tiempo, no de distancia, por lo que en teoría el margen de reacción no se degrada con la velocidad; este experimento verifica o refuta esa afirmación empíricamente. Es el experimento de mayor valor por costo de toda la lista.
`minisim_clear` × `--seed-jitter` × 10 semillas, tres brazos	Convierte la semilla en factor manipulado (§10.4.3). En un entorno **sin obstáculos**, la objeción del teletransporte no aplica, de modo que Tier 0 es el único tier donde el jitter es metodológicamente seguro. Permite estimar la **varianza atribuible a condiciones iniciales** y separarla de la varianza intrínseca del §10.4.2.

10.10.2 Extensiones de Tier 1 (TownSim)

Diseño	Qué aporta
Patrón de tránsito de `townsim_calib_cruce_frontal` alineado con el límite entre bloques ($y \approx 32$, PROVISORIO)	Escenario donde **la solución correcta es lateral** en lugar de escalar. Permite medir si el `slm` elige la lateral correcta con mayor frecuencia que el azar — la forma más directa de contrastar H1 sin confundirla con la ganancia de altura. Requiere validación previa de la coordenada y.
`townsim_calib_cruce_frontal` × $z \in \{-10, -15, -20, -25\}$ m	Convierte la altitud en factor experimental. Traza la **curva de transición** entre el régimen donde el escape por altura resuelve el bloqueo y el régimen donde no, e identifica la altitud a la que la diferencia entre brazos es máxima.
`townsim_ini` × {mediodía, atardecer, nublado} vía `simSetTimeOfDay`	Ataca la vulnerabilidad de la percepción monocular a cambios de iluminación (cap. 6): el flujo óptico degrada con iluminación baja, y la profundidad estimada puede degradar de otro modo.
Banco de prueba (§10.13) en TownSim	Extiende la validación fuera de vuelo de la profundidad estimada a un segundo entorno (§10.13.5).

10.10.3 Extensiones de Tier 2 (CitySim)

Diseño	Qué aporta
Escenario con tramos de ancho de corredor decreciente sobre el grid regular	Traza la **curva de degradación de la tasa de éxito en función del ancho del corredor** (Loquercio et al., 2021) — la única forma de responder "¿hasta qué angostura funciona el sistema?" en lugar de "¿funciona?". Es el experimento de mayor valor científico de la lista.
`citysim_pilot` con tráfico vehicular y agentes de IA activados	Introduce **obstáculos móviles**, que el diseño actual no cubre. Es la extensión que más acerca el protocolo a la práctica de la literatura de conducción (Codevilla et al., 2019) y la que mayor costo computacional implica por el aumento de varianza ambiental.
`citysim_clear` con ruido inyectado en la estimación de posición del guiado	Aproxima el cañón urbano real, donde la degradación de GPS es el modo de falla dominante (cap. 1). Mide cuánto de la fiabilidad observada depende de un guiado con posición exacta.

10.10.4 Extensiones transversales

Diseño	Qué aporta
Banco de prueba con otros VLM (Qwen2.5-VL-3B y modelos más grandes, fuera del presupuesto de VRAM a bordo)	Separa lo que es propiedad de **la pregunta y la arquitectura** de lo que es propiedad del modelo concreto. Sin este experimento, la conclusión de que el VLM no responde la pregunta del lazo lento está condicionada a un único modelo.
Lotes con obstrucción × `STRATEGIC_SOURCE = off`	Aísla la capa estratégica dentro del brazo `slm`, con la misma política de lazo rápido (`slm:depth` con y sin capa).
Repetición completa del lote 1 en una segunda sesión de simulador, mismo `code_version`	Mide directamente el **efecto de sesión** declarado en §10.5.1. Valida la decisión de comparar tiers sólo mediante razones normalizadas. Costo bajo y pertinencia alta para la validez interna del diseño.

10.10.5 Priorización

Si el presupuesto sólo permite un subconjunto, el orden de prioridad es: (1) curva de degradación por ancho de corredor —mayor valor científico—, (2) barrido de velocidad en Tier 0 —frontera de viabilidad temporal, costo mínimo—, (3) reproducibilidad entre sesiones —validez interna—, (4) banco de prueba en TownSim y con otros VLM —validez externa de las conclusiones sobre las fuentes—. Las restantes son deseables pero no condicionan las conclusiones de la tesis.

10.11 Amenazas a la validez

El protocolo tiene cinco limitaciones conocidas que se declaran explícitamente y se arrastran a las conclusiones del capítulo 12:

1. **Potencia estadística.** Con $K = 10$, una separación perfecta supera el umbral de Bonferroni, pero una separación parcial puede no hacerlo. Todo resultado no significativo debe leerse como "no detectado con esta potencia", nunca como "ausente", y se informa con su tamaño de efecto e intervalo.
2. **Validez externa respecto de los modelos.** Las conclusiones sobre el brazo `slm` son conclusiones sobre una red de profundidad concreta (Depth Anything V2 Metric Outdoor Small), y las conclusiones sobre el VLM, sobre LFM2.5-VL-1.6B cuantizado (`lfm2.5-vl-1.6b:q4km-ctx8k`), no sobre "modelos de lenguaje pequeños" en general. La mitigación es la comparación de modelos descrita en §10.10.4.
3. **Efecto de sesión entre tiers.** Los lotes de distintos tiers corren en sesiones y proyectos distintos, de modo que las comparaciones absolutas entre tiers no son limpias (§10.5.1). La mitigación parcial es el uso de razones normalizadas; la mitigación completa es la repetición de un lote en segunda sesión descrita en §10.10.4.
4. **Ausencia de obstáculos dinámicos.** Todo el diseño asume geometría estática. Las conclusiones no se extienden a entornos con agentes móviles, que es precisamente el escalón de dificultad que la literatura de conducción autónoma identifica como decisivo (Codevilla et al., 2019). La mitigación es la extensión con agentes de IA descrita en §10.10.3.
5. **Un escenario por lote.** Cada nivel de dificultad está representado por una o dos rutas concretas. Un efecto observado en un escenario es una propiedad de esa geometría hasta que se replique en otra ruta del mismo tipo; por eso los resultados se informan por escenario (§10.7.5) y la generalización entre escenarios es descriptiva.

Ninguna de estas limitaciones invalida el diseño para las preguntas que sí responde; todas acotan el alcance de lo que puede afirmarse a partir de él.

10.12 Lotes con obstrucción

10.12.1 Componentes del sistema que ejercen los escenarios con obstrucción

Los escenarios despejados apenas ejercen la capa táctica: el TTC rara vez baja del umbral y los detectores de atasco no disparan. Los escenarios con obstrucción (lotes 3, 4 y 6) sí, y son los que permiten evaluar los componentes del sistema:

Componente	Comportamiento que se evalúa	Referencia
Capa estratégica con profundidad estimada	Sub-metas ancladas a la pose del fotograma; si las sub-metas acercan al waypoint real y si alejan al dron de los obstáculos	§5.10, §6.14
Consulta inmediata ante un muro (`GIRAR_90` + `expedite`)	Que el rodeo se decida con el fotograma que ve el muro	§5.9
Barrido en marco mundo (`depth`)	Que la sub-meta de salida apunte a un rumbo transitable y que el dron salga del atasco	§5.12

Componente	Comportamiento que se evalúa	Referencia
Detección de atasco de dos señales	Ausencia de deadlocks falsos (despegue, sub-metas nuevas) y presencia de los reales	§5.3.1
Escape vertical cuando el barrido no resuelve	Que libere al dron de estructuras horizontales	§5.3.4
Detección de techo	Altitud limitada bajo estructuras horizontales	§5.15.3
Gobernador de velocidad	Avance limitado sin evidencia perceptual	§5.13
Vigilante de congelamiento	Terminación `physics_locked`	§5.19
Plan de ruta con el VLM sobre el mapa cenital (juicio por parches)	Si el plan elige desvíos libres cuando la recta está bloqueada, en TownSim y CitySim, sobre sus mapas registrados	§4.3.1, §11.0
Manifiestos `townsim_ini` (WP_0b, WP_1b) y `citysim_pilot` (WP_0_SUR)	Evitan la copa de la plaza, la moldura del edificio oeste y la autopista elevada	§10.3.2, §10.3.3

Los umbrales de la capa estratégica (período de 1 s con la profundidad, edad máxima de 10 s, sub-meta a 15 m, cambio de altura de 4 m, deduplicación de 10 m) no están calibrados: son decisiones de diseño.

10.12.2 Celdas, pilotos, comparaciones y descartes

Calibración previa. La fuente del lazo lento pasa por el banco de prueba a bordo (§10.13) antes de llevarse al vuelo: una pregunta se vuela solo si la fuente la responde, contra la etiqueta de referencia, por encima del criterio de pase. Con el resultado del banco (cap. 11, §11.0), las preguntas de la capa estratégica y del barrido las responde la profundidad estimada, y el VLM solo planifica la ruta antes del vuelo (`--route-plan vlm` en TownSim y CitySim).

Celdas. Los tres lotes con obstrucción —3 (`townsim_ini`), 4 (`townsim_calib_cruce_frontal`) y 6 (`citysim_pilot`), también llamados D, E y F— agregan a las tres celdas comunes (§10.2.3) la celda `fsm:blind`: la FSM sin barrido, con su escape determinista. Solo se corre donde hay atascos que resolver; en los controles sería idéntica a `fsm:depth`.

Pilotos y criterios de pase. Cada lote va precedido por un piloto de validación (`slm:depth`, semilla 99, guardado en `airsim-runs/produccion/v4/pilotos/`). Criterios: en el lote 3, el dron cruza el corredor arbolado; en el 4, rodea el edificio o al menos lo bordea; en el 6, alcanza WP_0_SUR y WP_1 sin quedar trabado sobre la autopista. Un piloto que no cumple su criterio indica una regresión que se diagnostica antes de lanzar el lote.

Comparaciones planificadas por lote (métrica primaria: SPL, §10.7.1). (i) `slm:depth` vs. `fsm:blind`: ¿aporta el sistema completo frente a una FSM sin barrido? (ii) `fsm:depth` vs. `fsm:blind`: ¿aporta el barrido con profundidad en la resolución de atascos, con la misma política de lazo rápido? (iii) `slm:depth` vs. `fsm:depth`: ¿aporta la capa estratégica junto con la política del lazo rápido del brazo `slm`? La pregunta «¿aporta el VLM en vuelo?» no es una comparación de estos lotes: la responde el banco (§10.13).

Métricas adicionales. Además de las de §10.6: la distribución de resultados de la capa estratégica y el avance hacia el waypoint real tras cada sub-meta (§10.6.3); las señales del `StallDetector` (`_stopped_cycles`, `_wp_no_progress_cycles`, `imu_contact_event`, `blind_wall_event`) y qué señal disparó cada deadlock; los deadlocks en los primeros 30 ciclos (deben ser cero: el despegue no es un atasco, cap. 5 §5.3.1); y la fracción de ciclos con el gobernador de velocidad activo (`_speed_cap`).

Condiciones de descarte y de invalidación (para los seis lotes). Se descarta una corrida individual si su `code_version` difiere del de referencia del lote, si termina en `-dirty`, o si tiene menos de 50 ciclos (falla de infraestructura, no del dron); los pilotos no entran al análisis. Se invalida un lote si contiene corridas con hashes distintos que no puedan reemplazarse. Una corrida terminada por `physics_locked` termina con éxito falso y se cuenta como falla de misión, al igual que una colisión.

10.13 Banco de prueba a bordo (`vlm_bench`)

10.13.1 Qué es y para qué sirve

El banco de prueba es una herramienta de medición **fuera de vuelo**. Responde una pregunta previa a cualquier lote: **¿la fuente del lazo lento sabe responder la pregunta que el lazo le hace?** Por ejemplo, si un sector de la imagen está libre a 15 m, si un rumbo del barrido es transitable o si un tramo de ruta sobre el mapa va por la calle. Para responderla compara, muestra por muestra, la respuesta de la fuente con una **etiqueta de referencia** independiente, calculada con la profundidad del simulador en la misma pose (cap. 7, §7.1).

Un lote de vuelo no puede responder esa pregunta. El resultado de una misión mezcla la percepción con el guiado, el lazo rápido, los atascos y el azar del simulador; con pocas semillas, una fuente que responde al azar puede empatar con una que acierta. El banco aísla el juicio. Si una fuente no lo supera, volarla no puede aportar, y no se lleva al vuelo. Si lo supera, los lotes miden lo que el banco no puede medir: si ese juicio mejora el vuelo (§10.12).

Tres reglas lo mantienen como medición **generalista** y no como ajuste del sistema a un escenario:

1. **Prompts y preguntas fijos**, iguales a los de vuelo e independientes del escenario: no nombran objetos de un entorno.
2. **La respuesta se lee tal como sale.** El criterio de pase usa la etiqueta que la fuente devuelve, sin umbrales calibrados sobre los datos. El AUC, calculado con la probabilidad de cada opción, se informa solo como diagnóstico: indica si hay información que la respuesta no expresa, pero no configura el vuelo.

3. **Pasar en todos los entornos medidos con el mismo prompt.** Por entorno, el criterio es exactitud balanceada ≥ 0.65 y límite inferior del intervalo de confianza del 95 % por encima del azar. El intervalo se calcula por *bootstrap* por corrida: se remuestrean corridas enteras, porque las vistas de una misma corrida no son independientes. Si una pregunta no pasa, se descarta: el modelo no tiene esa capacidad sin entrenamiento, y adaptarla al escenario sería un entrenamiento manual por entorno, lo que el enfoque generalista evita.

10.13.2 Cómo está construido

El código está en `airsim-loop/experiments/vlm_bench/` y la tubería completa se ejecuta con `run_all.sh`. Cada etapa es reanudable: si se corta, al relanzarla saltea lo ya hecho.

Etapa	Módulo	Qué hace
1. Muestras	`dataset.py`	Toma las corridas que tienen telemetría y los dos videos (frontal y `FollowCam`) y extrae poses registradas en vuelo (posición, actitud y meta activa). Hay tres clases de muestra: **estratégicas** (cada consulta de la capa estratégica, con la respuesta que dio en vuelo y la foto enviada), **de barrido** (cada rumbo de cada barrido) y **de trayectoria** (ciclos sin consulta, a ≥ 8 m de altura, sin duplicados en celdas de 2 m y 20°: un dron trabado cientos de ciclos cuenta una vez).
2. Etiquetas	`ground_truth.py`, `labels.py`	Ver §10.13.3. Es el único módulo del banco que lee la profundidad del simulador.
3. Preguntas	`questions.py`	Catálogo: para cada pregunta, a qué muestras aplica, cómo arma la imagen y el prompt, su esquema JSON (igual que en vuelo), cuál es la etiqueta verdadera y cómo se lee la respuesta. Las opciones tienen iniciales distintas para leer la probabilidad de cada una en los *logprobs*.
4. Evaluación	`evaluate.py`	Selecciona hasta 60 muestras válidas por clase verdadera (los barridos, todos) y hace cada pregunta con decodificación restringida y temperatura 0. Guarda la respuesta, la probabilidad de cada opción y la latencia.
5. Reporte	`report.py`	Calcula exactitud balanceada con su intervalo, AUC, respuestas constantes, sesgo por posición y acuerdo bajo permutación, y aplica el criterio de pase. Con bancos de varios entornos produce `cross_report.md`.

Las preguntas del catálogo son:

Pregunta	Qué pregunta	Para qué está
`grid_prod`	La pregunta de vuelo de la capa estratégica tal cual: grilla de 3×3, marca de la meta, «libre» o «bloqueado» por sector	Medir la capacidad que usa el vuelo
`grid_perm`	La misma, con las etiquetas de los sectores permutadas	Control: si la respuesta no cambia con la permutación, el modelo responde a la posición en la imagen y no a la escena

Pregunta	Qué pregunta	Para qué está
`centro_libre`	¿Se puede avanzar recto 15 m? (sin grilla)	La forma más simple de la pregunta
`borde_lateral`	Con un obstáculo de frente, ¿por qué lado termina?	Información para elegir el rodeo
`borde_superior`	Con un obstáculo de frente, ¿hay espacio libre por encima?	Ídem, en vertical
`direccion_abierta`	¿Qué tercio tiene más espacio libre a la altura del dron?	Ídem, como elección
`scan_prod` / `scan_perm`	La pregunta de vuelo del barrido con las imágenes de un barrido real, en el orden de vuelo y en otro orden	Medir el barrido y su sesgo por orden

Cuatro pruebas derivadas usan las mismas muestras o la misma etiqueta para preguntas puntuales:

Prueba	Módulo	Pregunta
Mapa	`map_probe.py`	¿El VLM reconoce qué hay en un parche del mapa cenital (calle o techo)? La etiqueta se obtiene mirando el parche desde 40 m con la profundidad.
Rutas	`route_bench.py`	¿El juicio del planificador de ruta (cap. 4, §4.3.1) separa rutas libres de bloqueadas? La etiqueta recorre la ruta cada 4 m a la altura de vuelo: la ruta es libre si en ningún punto hay un obstáculo a menos de 6 m en la ventana central.
Regiones	`region_probe.py`	Si el VLM no juzga distancias en la cámara frontal, ¿reconoce al menos qué hay en cada sector?
Profundidad y segmentación	`seg_depth_probe.py`	¿Dos redes chicas que solo ven el RGB juzgan el espacio libre por sector mejor que el VLM, con la misma geometría y el mismo estadístico que la etiqueta?

10.13.3 Etiquetas de referencia

Cada muestra es una foto o un cuadro de video de una corrida, con la pose del dron registrada en ese instante. Su etiqueta se calcula fuera de vuelo con `ground_truth.py`, una herramienta de laboratorio aislada del lazo, de la misma categoría que la calibración del TTC (cap. 7). El procedimiento tiene cinco pasos:

1. **Reubicar el dron.** Se lleva el dron a la pose exacta (posición y actitud) con la cámara en el campo de visión de vuelo (90°; el cliente rechaza conectarse si el campo de visión fue alterado).
2. **Esperar la carga de la escena.** Unreal carga la escena por zonas y, tras un salto largo, la primera captura puede salir sin edificios. Se repite la captura hasta que la profundidad de cada sector cambia menos de 0.5 m entre dos capturas seguidas.

3. **Capturar y filtrar.** Se capturan la imagen y la profundidad planar. Se descartan las muestras con la cámara dentro de la geometría, con captura inestable o cuya imagen reproducida no coincide con la registrada en vuelo (correlación normalizada < 0.5: la pose no se reprodujo bien, por ejemplo con el dron atravesando una fachada).
4. **Resumir por sector.** Sobre el recorte cuadrado central —el mismo que recibe la fuente en vuelo—, dividido en una grilla de 3×3, se toma el percentil 5 de la profundidad de cada sector: lo más cercano, sin píxeles sueltos.
5. **Etiquetar.** Un sector es **libre** si ese percentil es ≥ 15 m, la misma distancia que se le pide juzgar a la fuente y a la que se fija la sub-meta.

De ahí salen las etiquetas de cada pregunta: - *Centro libre*: el estado del sector central. - *Lado por donde termina el obstáculo*: el lado del hueco libre más cercano al centro en la franja a la altura del dron. - *Espacio libre por encima*: si al menos la mitad del sector superior central está libre. - *Dirección más abierta*: el tercio de la franja con mayor profundidad mediana, o «ninguna» si ninguno llega a 15 m.

Estas etiquetas solo sirven para medir: nunca entran al lazo de vuelo (§10.3.0, punto 3).

10.13.4 Qué decide el banco en la configuración del sistema

Tres componentes del lazo de control salen de lo que el banco mide: la fuente de la capa estratégica, el juicio del barrido y el plan de ruta.

Medición	Resultado	Consecuencia de diseño
Juicio de rutas enteras dibujadas sobre el mapa (línea, pasillo, pregunta invertida)	Ningún modelo distingue una ruta por la calle de una sobre un edificio: la respuesta depende de la forma de la pregunta	El planificador no juzga rutas enteras (cap. 4, §4.3.1)
`map_probe`: reconocimiento de parches del mapa con cinco clases	LFM2.5-VL-1.6B: AUC 0.80 para «calle»; Qwen2.5-VL-3B: 0.63	Juicio **por parches** y LFM2.5-VL-1.6B también para el planificador
`route_bench`: rutas juzgadas por parches	AUC 0.81	Plan de ruta activado sobre mapas registrados (`ROUTE_PLAN_MODE=vlm`)
`grid_prod` y `scan_prod` con el VLM de vuelo	Ninguna de las ocho preguntas pasa; las filas de arriba y de abajo salen «libres» siempre; el barrido declara todo transitable	El VLM no responde la pregunta del lazo lento; desempate por fila con margen si se usa (`VLM_SUBGOAL_TIE_DEG`, cap. 5 §5.10.3)
`region_probe`: reconocimiento por sector en la cámara frontal	No pasa (exactitud 48–49 %; AUC 0.53–0.62)	No se pregunta «qué hay» en lugar de «a qué distancia»

Medición	Resultado	Consecuencia de diseño
`seg_depth_probe`: profundidad monocular y segmentación	Profundidad: 75 % y AUC 0.83 por sector, AUC 0.87 en los barridos; segmentación: AUC 0.50	La capa estratégica y el barrido usan la profundidad estimada (`STRATEGIC_SOURCE=depth`, `DEADLOCK_STRATEGY=depth`)

La tabla muestra el uso previsto del instrumento: cada pregunta que no pasa se descarta en lugar de compensarse con reglas, y cada fuente que pasa se lleva al vuelo y se juzga con los lotes. Las cifras completas están en el capítulo 11 (§11.0), y la descripción de la red de profundidad, en el capítulo 6 (§6.14).

10.13.5 Alcance y límites

- **Un solo entorno medido.** Las 6926 poses del banco provienen de corridas de `citysim_pilot`; TownSim no tiene banco. El criterio exige pasar en todos los entornos medidos. Para el VLM la conclusión no cambia, porque no pasa ni siquiera en CitySim. Para la profundidad y para el plan de ruta por parches, en TownSim solo los evalúan los lotes de vuelo (lotes 2 a 4). Un banco en TownSim es la extensión pendiente más directa (§10.10.2).
- **Muestras sesgadas hacia los lugares difíciles.** Las muestras estratégicas y de barrido se toman donde el sistema consultó, que suelen ser los tramos con obstáculos; la evaluación equilibra las clases verdaderas para que la exactitud balanceada no dependa de esa proporción.
- **Render sintético.** La etiqueta sale del mismo render que ve la fuente. Una red entrenada en parte con imágenes sintéticas, como la de profundidad, puede rendir menos con una cámara real.
- **Pocos barridos.** 14 y 22 barridos alcanzan para ver que el VLM responde siempre lo mismo, pero no para interpretar su AUC.

11. Resultados comparativos SLM vs. FSM

> **Alcance de este capítulo.** Salvo el banco de prueba del VLM (§11.0) y los lotes de la capa estratégica (§11.6), que usan la configuración vigente, los resultados corresponden al **lote base** (75 corridas ejecutadas el 8 y 9 de septiembre). El lote base se ejecutó con una configuración del brazo `slm` distinta de la descrita en los capítulos 5–8: el VLM elegía macro-acciones tácticas y el barrido de resolución de atascos le pedía una acción, en lugar de proponer sub-metas ancladas a la pose del fotograma. Tres advertencias para leerlo:
>
> 1. **Escenarios de control** (§11.1, §11.4.1, §11.4.2, §11.4.3): no ejercen la capa táctica de atasco y son válidos como medida del costo del brazo `slm` (H2).
> 2. **Escenarios con obstrucción** (`townsim_ini`, `townsim_calib_cruce_frontal`, `citymap_pilot`; §11.4.2b, §11.4.2c, §11.4.3b y §11.5.2–§11.5.4): se ejecutaron sin los waypoints WP_0b y WP_1b de `townsim_ini` ni `WP_0_SUR` de `citymap_pilot` (cap. 10 §10.3.3), con más de un `code_version` por escenario y con el umbral de ocupación en 0.35 (canal efectivamente inactivo, cap. 7 §7.5). **No describen el desempeño del sistema de los capítulos 5–8**, que se evaluará con los lotes D, E y F (§10.12).
> 3. **El VLM intervino en los brazos `slm` y `fsm`.** Ambos resolvieron sus deadlocks con `deep_vlm` (barrido + consulta al VLM) y el lote no tiene una celda sin VLM en la resolución de atascos, por lo que **ninguna diferencia entre los brazos deliberativos y el `reactive` puede atribuirse al contenido de las respuestas del modelo** (cap. 10 §10.2.2). La auditoría de corridas de diagnóstico (cap. 9 §9.8) encontró además respuestas del barrido casi constantes, lo que refuerza esta cautela.
>
> La distancia mínima al obstáculo (`DistMin`) se midió en este lote con el canal de profundidad del simulador, como auditoría externa al lazo de control; el sistema de los capítulos 5–8 no usa ese canal (cap. 10 §10.3.0).

11.0 Banco de prueba del VLM: línea base a bordo

El banco (`airsim-loop/experiments/vlm_bench/`; qué es, cómo se construyó y cómo se usó para decidir, en cap. 10 §10.13) mide, fuera de vuelo, qué preguntas sabe responder el modelo, contra una etiqueta de referencia tomada de la profundidad del simulador en la misma pose. La **línea base** de todas las comparaciones es la corrida del 7–8 de octubre de 2026 (`airsim-runs/vlm_bench/v2/`) con la configuración de vuelo vigente: LFM2.5-VL-1.6B Q4_K_M en Ollama, contexto de 8192 tokens, **en la misma GPU que Unreal Engine y el grafo de control** (cap. 8 §8.1.2). Se toma esa condición, y no la del servidor dedicado anterior, porque un VLM a bordo comparte el cómputo con el resto del sistema de vuelo.

Datos. Muestras de las 21 corridas que tienen telemetría y los dos videos (frontal y FollowCam): pilotos `citysim_pilot` v2 y v3 y los tres pilotos con el modelo local. De 6926 poses se reprodujeron en AirSim 1908 (todas las consultas estratégicas, 429, y los barridos, 579, más 900 de trayectoria al azar). Válidas para evaluar: 915; se excluyen 190 con la cámara dentro de la geometría, 384 con captura inestable y 589 con la pose mal reproducida (correlación con lo registrado en vuelo < 0.5). El centro está libre en el 51 % de las válidas. Cada pregunta se hace con decodificación restringida y temperatura 0, a 384 px (los barridos a 256 px, como en vuelo), sobre hasta 60 muestras por clase.

Criterio de pase: exactitud balanceada de la respuesta tal como sale ≥ 0.65 y límite inferior del IC 95 % (*bootstrap* por corrida) por encima del azar. El AUC, calculado con la probabilidad de la opción leída de los *logprobs*, es diagnóstico: indica si hay señal que el modelo no expresa en la respuesta, no configura el vuelo.

Pregunta	n	Exactitud bal. [IC 95 %]	Azar	Pasa	Respuesta más frecuente	AUC (diag.)	Latencia mediana
`grid_prod` (pregunta de vuelo, grilla 3×3)	120	53 % [50–55 %]	50 %	no	libre (82 %)	0.53	0.88 s
`grid_perm` (ídem, etiquetas permutadas)	120	51 % [49–53 %]	50 %	no	libre (85 %)	0.51	0.88 s
`centro_libre` (¿recto 15 m?)	120	50 % [50–50 %]	50 %	no	sí (100 %, constante)	0.52	0.34 s
`borde_lateral`	180	33 % [33–33 %]	33 %	no	izquierda (100 %, constante)	—	0.35 s
`borde_superior`	120	53 % [50–58 %]	50 %	no	sí (95 %)	0.69	0.36 s
`direccion_abierta`	208	32 % [28–36 %]	25 %	no	izquierda (62 %)	—	0.36 s
`scan_prod` (barrido de vuelo)	14	50 %	50 %	no	todo transitable (100 %)	0.04*	2.6 s
`scan_perm` (ídem, orden permutado)	22	50 %	50 %	no	todo transitable (100 %)	0.92*	3.1 s

* Con 14 y 22 barridos el AUC se calcula sobre muy pocos pares y cambia de extremo a extremo con el orden de las imágenes: no es interpretable.

Ninguna pregunta pasa. Tres son constantes y el resto queda a pocos puntos del azar. En la pregunta de vuelo, las filas 1 y 3 de la grilla salen «libre» en el 100 % de las muestras (realmente libres: 72–78 % arriba, 38–47 % abajo), y en la fila del medio la respuesta depende de la posición del sector más que de la escena (A2 libre 14 %, B2 70 %, C2 52 %, con ~50–57 % reales). Con las etiquetas permutadas la respuesta coincide con la del orden normal en el 75 % de los sectores: el modelo responde sobre todo a la posición en la imagen. Los *logprobs* no rescatan la grilla (AUC 0.53); la única señal latente apreciable está en `borde_superior` (AUC 0.69, frente a una respuesta casi constante).

Dos consecuencias para el vuelo. La sub-meta de la capa estratégica sale de la posición de las celdas, no de la escena: en el piloto `seed_99_20261007T214055Z`, 12 de 31 sub-metas eligieron la fila de abajo —siempre «libre»— y el dron descendió hasta apoyarse en el tablero de una autopista elevada a ~7 m, del que no pudo girar para el barrido. El desempate por fila con margen (`VLM_SUBGOAL_TIE_DEG`, cap. 5 §5.10.3) evita ese descenso, pero no aporta información de la escena. Y el barrido declara transitables todos los rumbos, de modo que la salida del atasco la decide la exclusión de rumbos ya probados (cap. 5 §5.12), no el modelo.

Plan de ruta sobre el mapa (juicio por parches, cap. 4 §4.3.1; 59 candidatas en 11 tramos de `citysim_pilot` y `citysim_clear`, 28 realmente libres). La probabilidad media de «calle» separa las rutas libres de las bloqueadas con AUC 0.81. La ruta elegida es libre en 10 de 11 tramos, igual que la recta: en el único tramo con la recta bloqueada (`citysim_pilot` WP_2 → WP_3, la recta cruza un edificio) el plan elige un desvío libre, y en un tramo de `citysim_clear` elige un desvío bloqueado con la recta libre. En la prueba de parches del mapa (60 parches etiquetados, cinco clases) el modelo local obtiene exactitud balanceada 0.65 y AUC 0.79; el planificador usa la probabilidad, no la respuesta.

La misma pregunta con profundidad estimada (`experiments/vlm_bench/seg_depth_probe.py`; detalle en cap. 6 §6.14.1). Con el mismo fotograma, la misma geometría y la misma etiqueta, el p5 de la profundidad estimada por Depth Anything V2 Metric Outdoor Small separa sectores libres de bloqueados con exactitud balanceada del 75 % [72–77 %] y AUC 0.83 (625 muestras × 9 sectores; 76 % y 0.84 en las 120 muestras de las preguntas del VLM), en 82 ms. Es la única señal que pasa el criterio. En los barridos (290 imágenes, sector central) logra 75 % [68–82 %] y AUC 0.87. Una red de segmentación semántica chica (`yolo26n-sem`) queda en el azar (AUC 0.50): la clase de un píxel no dice a qué distancia está.

Latencia en vuelo con el modelo a bordo (piloto `seed_99_20261007T214055Z`, 65 consultas estratégicas): mediana 0.77 s, p90 0.80 s, máximo 2.8 s, sin *timeouts*. La captura (`simGetImages`, mediana 103 ms) y el período del lazo (mediana 237 ms) quedan en el rango de los pilotos con el servidor dedicado (91–94 ms y 216–228 ms).

11.1 Resultados del lote base (5 semillas)

Estrategia de desbloqueo: `deep_vlm` en todos los casos. Las celdas muestran media ± desviación estándar sobre las 5 semillas independientes. Tasa de éxito: 5/5 y 0 colisiones en los 45 runs.

Tier 0 — `minisim_clear` · MiniSim (crater.png)

Escenario: 3 waypoints en L (~180 m), altitud −10 m, ambiente despejado.

Brazo	Éxito	Duración media (s)	σ (s)	Dist. media (m)	σ (m)	Colisiones	Invoc. SLM	Deliberación	Fallback SLM
`slm`	5/5 ✅	**164.2**	9.4	185.6	1.6	0	49.6	6.5%	3.2%
`fsm`	5/5 ✅	174.2	38.7	218.9	15.3	0	—	—	—
`reactive`	5/5 ✅	74.7	0.4	183.6	0.2	0	—	—	—

Dist. mín. al obstáculo (media): `slm` 6.5 m · `fsm` 8.0 m · `reactive` 7.9 m

Ratio `slm` / `reactive`: 2.20× · **Ratio `fsm` / `reactive`:** 2.33×

Observación: En ambiente despejado la diferencia entre brazos es de velocidad pura. `reactive` completa en 74.7 s (sin deliberar); `slm` tarda 2.20× más por sus 49.6 invocaciones promedio al VLM. La alta varianza del `fsm` (σ=38.7 s) refleja la variabilidad en el número de deadlocks por corrida (0–3, media 1.4), todos resueltos por `deep_vlm`. La varianza del `reactive` es prácticamente nula (σ=0.4 s), confirmando que sin deliberación el brazo es determinista.

Tier 1 — `townsim_clear` · TownSim (townsim_calib.png)

Escenario: perímetro completo del complejo, 6 WPs, ~626 m, altitud de tránsito −30 m.

Brazo	Éxito	Duración media (s)	σ (s)	Dist. media (m)	σ (m)	Colisiones	Invoc. SLM	Deliberación	Fallback SLM
`slm`	5/5 ✅	**296.0**	12.8	624.4	6.9	0	9.6	0.8%	2.2%
`fsm`	5/5 ✅	329.5	46.0	637.3	15.8	0	—	—	—
`reactive`	5/5 ✅	259.7	3.1	626.2	2.2	0	—	—	—

Dist. mín. al obstáculo (media): `slm` 8.5 m · `fsm` 8.8 m · `reactive` 13.9 m

Ratio `slm` / `reactive`: 1.14× · **Ratio `fsm` / `reactive`:** 1.27×

Observación: La longitud de la ruta (~626 m vs. ~184 m en Tier 0) diluye el overhead por invocación: con solo 9.6 invocaciones promedio sobre ~4 500 ciclos, la tasa de deliberación cae a 0.8%. El `slm` sigue siendo más lento que el `reactive` (1.14×), pero la diferencia se redujo a 36 s de promedio. El `fsm` acumula más deadlocks (3.8 vs. 2.6 del `slm`) y presenta la mayor varianza (σ=46 s), incluyendo una corrida de 406 s por un bloqueo prolongado. El `reactive` no registra deadlocks y tiene σ=3.1 s, su varianza residual proviene del jitter de la física de AirSim.

Tier 2 — `citysim_clear` · CitySim (citysim_calib.png)

Escenario: perímetro de una manzana del grid regular, 4 WPs, ~431 m, altitud de tránsito variable (rampa desde −10 m hasta −50 m y retorno a −10 m).

Brazo	Éxito	Duración media (s)	σ (s)	Dist. media (m)	σ (m)	Colisiones	Invoc. SLM	Deliberación	Fallback SLM
`slm`	5/5 ✅	**175.5**	3.2	427.4	1.0	0	2.2	0.3%	0%
`fsm`	5/5 ✅	187.7	20.9	434.9	3.8	0	—	—	—
`reactive`	5/5 ✅	165.2	0.2	431.0	0.5	0	—	—	—

Dist. mín. al obstáculo (media): `slm` 9.4 m · `fsm` 10.0 m · `reactive` 10.4 m

Ratio `slm` / `reactive`: 1.06× · **Ratio `fsm` / `reactive`:** 1.14×

Observación: Con solo 2.2 invocaciones promedio al VLM y tasa de deliberación de 0.3%, el overhead del `slm` es marginal (10.3 s sobre 165 s de base). El `fsm` presenta dos corridas con deadlock (seeds 4 y 5, 212 s y 208 s) que elevan la media y σ. El `reactive` es virtualmente determinista (σ=0.2 s). Los tres brazos completan sin colisiones, confirmando que la altitud de crucero está por encima de la línea de tejados en este escenario de control.

11.2 Tabla de resultados por brazo y escenario

Media sobre 5 semillas. Col./km = 0 en los tres tiers (0 colisiones en 45 corridas). DistMin = media de la distancia mínima al obstáculo por corrida.

Brazo	Tier / Escenario	Éxito	Col./km	DistMin (m)	Tiempo (s)	Deliberación	Fallback SLM	Res. Atasco VLM	Ratio vs `reactive`
`slm`	Tier 0 (`minisim_clear`)	5/5 ✅	0	6.5	164.2 ± 9.4	6.5%	3.2%	— (0 deadlocks)	**2.20×**
`slm`	Tier 1 (`townsim_clear`)	5/5 ✅	0	8.5	296.0 ± 12.8	0.8%	2.2%	100% (2.6/corrida)	**1.14×**
`slm`	Tier 1 (`townsim_ini`)	0/5 ⛔ timeout	0	0.21	900 (límite)	1.4%	0%	100% (20.2/corrida)	—
`slm`	Tier 1 (`townsim_calib_cruce_frontal`)	1/5 ⚠️	0	0.10	772.7 ± 285.1	1.9%	0%	95.3% (18.4/corrida)	—
`slm`	Tier 2 (`citysim_clear`)	5/5 ✅	0	9.4	175.5 ± 3.2	0.3%	0%	— (0 deadlocks)	**1.06×**
`slm`	Tier 2 (`citymap_pilot`)	0/5 ⛔ timeout	0	0.22	600 (límite)	2.3%	0%	100% (20.4/corrida)	—
`fsm`	Tier 0 (`minisim_clear`)	5/5 ✅	0	8.0	174.2 ± 38.7	—	—	100% (1.4/corrida)	2.33×
`fsm`	Tier 1 (`townsim_clear`)	5/5 ✅	0	8.8	329.5 ± 46.0	—	—	100% (3.8/corrida)	1.27×
`fsm`	Tier 1 (`townsim_ini`)	0/5 ⛔ timeout	0	0.17	900 (límite)	—	—	100% (24.0/corrida)	—

Brazo	Tier / Escenario	Éxito	Col./km	DistMin (m)	Tiempo (s)	Deliberación	Fallback SLM	Res. Atasco VLM	Ratio vs reactive
fsm	Tier 1 (townsim_calib_cruce_frontal)	0/5 ⛔ timeout	0	0.16	900 (límite)	—	—	100% (6.2/corrida)	—
fsm	Tier 2 (citysim_cleanr)	5/5 ✅	0	10.0	187.7 ± 20.9	—	—	100% (0.8/corrida)	1.14×
fsm	Tier 2 (citymap_pilot)	0/5 ⛔ timeout	0	0.10	600 (límite)	—	—	100% (21.4/corrida)	—
reactive	Tier 0 (minisim_cleanr)	5/5 ✅	0	7.9	74.7 ± 0.4	—	—	—	1.00×
reactive	Tier 1 (townsim_cleanr)	5/5 ✅	0	13.9	259.7 ± 3.1	—	—	—	1.00×
reactive	Tier 1 (townsim_ini)	0/5 ⛔ timeout	0	0.44	900 (límite)	—	—	— (0 deadlocks)	—
reactive	Tier 1 (townsim_calib_cruce_frontal)	2/5 ⚠️	0	0.67	603.1 ± 406.8	—	—	— (0 deadlocks)	—
reactive	Tier 2 (citysim_cleanr)	5/5 ✅	0	10.4	165.2 ± 0.2	—	—	—	1.00×
reactive	Tier 2 (citymap_pilot)	0/5 ⛔ paralizado	0	3.05	600 (límite)	—	—	— (0 deadlocks)	—

11.3 Análisis estadístico del lote base

11.3.1 Tabla de ratios de tiempo de misión (H2)

Escenario	Tiempo `reactive` (s)	Tiempo `slm` (s)	Ratio `slm` / `reactive`	Tiempo `fsm` (s)	Ratio `fsm` / `reactive`	Invoc. VLM (`slm`)
Tier 0 `minisim_clear`	74.7 ± 0.4	164.2 ± 9.4	**2.20×**	174.2 ± 38.7	2.33×	49.6
Tier 1 `townsim_clear`	259.7 ± 3.1	296.0 ± 12.8	**1.14×**	329.5 ± 46.0	1.27×	9.6
Tier 2 `citysim_clear`	165.2 ± 0.2	175.5 ± 3.2	**1.06×**	187.7 ± 20.9	1.14×	2.2

El ratio `slm` / `reactive` decrece monotónicamente con la longitud de la ruta (y con la distancia por invocación VLM): 2.20× → 1.14× → 1.06×. Esto es consistente con H2: el costo temporal del brazo `slm` no es lineal en la distancia sino en la frecuencia de deliberación.

11.3.2 Prueba U de Mann-Whitney y Cliff's Delta

Métrica: tiempo de misión (`duration_s`) en corridas con `success = True` (todas las corridas del lote base). Prueba bilateral. Corrección de Bonferroni: 18 comparaciones previstas (3 pares × 6 escenarios, incluyendo extendidos); umbral ajustado $\alpha = 0.05/18 \approx 0.0028$.

Comparación	Escenario	U	z	p	p (Bonf.)	Cliff's δ	Significancia
`slm` vs `reactive`	Tier 0 `minisim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)
`slm` vs `reactive`	Tier 1 `townsim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)
`slm` vs `reactive`	Tier 2 `citysim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)
`fsm` vs `reactive`	Tier 0 `minisim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)
`fsm` vs `reactive`	Tier 1 `townsim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)

Comparación	Escenario	U	z	p	p (Bonf.)	Cliff's δ	Significancia
`fsm` vs `reactive`	Tier 2 `citysim_clear`	25	2.61	0.009	0.162	**+1.00**	* (no Bonf.)
`slm` vs `fsm`	Tier 0 `minisim_clear`	13	0.10	0.917	1.000	+0.04	ns
`slm` vs `fsm`	Tier 1 `townsim_clear`	6	−1.36	0.175	1.000	−0.52	ns
`slm` vs `fsm`	Tier 2 `citysim_clear`	13	0.10	0.917	1.000	+0.04	ns

Referencia de δ: $|\delta| < 0.147$ negligible, 0.147–0.33 pequeño, 0.33–0.474 mediano, > 0.474 grande (Romano et al., 2006).

Interpretación:

- **`slm` vs `reactive` y `fsm` vs `reactive`:** δ = +1.00 en los tres tiers — separación perfecta, ninguna corrida de `reactive` supera en tiempo a ninguna corrida de `slm` o `fsm`. El p-valor unadjustado (0.009) indica significancia a α = 0.05, pero no supera el umbral de Bonferroni (0.0028) con K = 5 semillas. El límite inferior del p alcanzable con K = 5 es 7.9×10^{-3} — imposible superar Bonferroni con este tamaño muestral independientemente de cuán perfecta sea la separación. El efecto es real (δ = 1.0) pero la potencia estadística es insuficiente para la corrección multivariada. Ampliar a K = 10 (ver §5.3 del plan de pruebas) bajaría el p mínimo alcanzable a 1.1×10^{-5}, dentro del rango de Bonferroni.
- **`slm` vs `fsm`:** no significativo en ningún tier (p > 0.17). En Tier 1, δ = −0.52 (grande) favorece al `slm`, impulsado por el outlier del `fsm` (406 s, seed 2, bloqueo prolongado). Con K = 5 y alta varianza del `fsm`, este resultado no es concluyente — requiere los escenarios de obstrucción real para separar los brazos deliberativos.

11.3.3 Perfil de deliberación por tier

Tier	Invoc. VLM promedio	Tasa deliberación	Dist. / invocación
Tier 0 `minisim_clear`	49.6	6.5%	~3.7 m/invoc.
Tier 1 `townsim_clear`	9.6	0.8%	~65 m/invoc.
Tier 2 `citysim_clear`	2.2	0.3%	~194 m/invoc.

La distancia por invocación aumenta 52× entre Tier 0 y Tier 2. En Tier 0, el VLM es consultado cada ~3.7 m; en Tier 2, cada ~194 m. Esto confirma que el comportamiento cauteloso del `slm` durante la espera VLM (avance lento, sin giro) reduce la frecuencia de invocación a medida que la ruta tiene menos quiebres de rumbo y menos incertidumbre perceptual — incluso en ausencia de obstáculos reales.

11.4 Análisis por tipo de escenario

11.4.1 Tier 0 — Control sin obstáculos (`minisim_clear`)

El escenario más simple (3 WPs en L, ambiente despejado) establece el costo puro del brazo `slm` sin ningún beneficio compensatorio. El lote estadístico (5 semillas) confirma la tendencia del piloto con mayor precisión:

- **`reactive`:** 74.7 ± 0.4 s — prácticamente determinista. Es la cota inferior de velocidad del sistema: sin deliberación, sin overhead, trayectoria directa al siguiente waypoint.
- **`slm`:** 164.2 ± 9.4 s — 49.6 invocaciones promedio al VLM (6.5% de los ciclos). La varianza (σ = 9.4 s) proviene de la variabilidad en la latencia de inferencia del VLM por corrida. Ningún deadlock: el ambiente despejado no genera atascos en el `slm`.
- **`fsm`:** 174.2 ± 38.7 s — tiempo medio mayor al `slm`, con alta varianza (σ = 38.7 s) por la variabilidad en deadlocks (0–3 por corrida). El overhead de la FSM en este escenario es comparable al del VLM, pero más errático.

La separación `slm`/`reactive` (Cliff's δ = +1.0) es perfecta: en las 25 comparaciones posibles (5 × 5 semillas), cada corrida de `reactive` fue más rápida que cualquier corrida de `slm`. El mismo patrón se observa para `fsm` vs `reactive`. En entorno despejado, ningún brazo deliberativo puede competir en velocidad con el `reactive` — la ventaja del VLM no tiene dónde manifestarse.

Predicción verificada: la diferencia `reactive` vs. `slm`/`fsm` tiene Cliff's δ = 1.0 en los tres tiers, confirmando el patrón esperado. La comparación `slm` vs. `fsm` es no significativa (δ = 0.04), como era esperado: en ausencia de obstáculos, ambos brazos deliberativos incurren en un overhead similar sin diferenciarse en calidad de decisión.

11.4.2 Tier 1 — Perímetro urbano de crucero (`townsim_clear`)

La longitud de la ruta (~626 m, 3.4× Tier 0) diluye el overhead del VLM: de 49.6 invocaciones en Tier 0 a 9.6, y la tasa de deliberación baja de 6.5% a 0.8%. El ratio `slm`/`reactive` cae de 2.20× a 1.14×: el `slm` tarda 36 s más que el `reactive` en promedio, sobre una base de 260 s.

El dato más relevante de Tier 1 es la asimetría de deadlocks entre brazos: - `reactive`: 0 deadlocks en las 5 corridas (σ ≈ 0 en tiempo). - `slm`: 2.6 deadlocks promedio (todos resueltos al 100% por `deep_vlm`). - `fsm`: 3.8 deadlocks promedio, incluyendo un bloqueo de >400 s en seed 2.

Este patrón es consistente con la arquitectura: el `reactive` no tiene concepto de "objetivo pendiente" y no genera atascos; los brazos deliberativos sí. La diferencia entre `slm` (2.6) y `fsm` (3.8) sugiere que la consulta al VLM ayuda a evitar algunos deadlocks incluso en escenarios de control — aunque con K = 5 este resultado no es estadísticamente concluyente (δ = −0.52, ns).

El escenario de interés real para Tier 1 es `townsim_ini` (corredores vegetados) y `townsim_calib_cruce_frontal` (bloqueo frontal masivo), donde el VLM tiene un caso de uso genuino y la diferencia `slm` vs `fsm` debería hacerse significativa. Los datos de `townsim_clear` solo establecen la cota de partida del lote base.

11.4.2b Tier 1 — Corredor arbolado (`townsim_ini`)

`townsim_ini` es el primer escenario de obstrucción real del lote: el corredor peatonal central de TownSim, volado a z=−10 m (bajo la copa de los árboles), de norte a sur entre los edificios. Los tres brazos fallan en las 15 corridas por timeout (900 s), sin una sola colisión registrada.

Brazo	Éxito	Dist. media recorrida	Deadlocks (media)	Res. VLM	DistMin (m)
`slm`	0/5 ⛔	134.7 ± 50.4 m	20.2	100%	0.21
`fsm`	0/5 ⛔	176.2 ± 84.2 m	24.0	100%	0.17
`reactive`	0/5 ⛔	158.5 ± 10.9 m	0	—	0.44

Ningún brazo superó el 60% de la ruta (~310 m total). El `reactive` es el más consistente en distancia recorrida (σ=10.9 m vs. 50–84 m del los brazos deliberativos), pero tampoco progresa: sin deadlock detection, oscila ante los obstáculos sin activar el mecanismo de escape, lo que resulta en movimiento local sin avance neto hacia el siguiente waypoint.

Los brazos deliberativos acumulan entre 10 y 44 deadlocks por corrida (media: `slm` 20.2, `fsm` 24.0). El `deep_vlm` resuelve el 100% de ellos, pero cada resolución consume ciclos adicionales (escaneo + reposicionamiento), y la tasa de aparición de nuevos deadlocks supera la capacidad de avance efectivo: el sistema entra en un estado de "deadlock crónico" donde el `deep_vlm` nunca puede entregar progreso sostenido.

Interpretación para H1: el escenario `townsim_ini` coloca a los tres brazos más allá del umbral de capacidad de la arquitectura actual — el corredor a z=−10 m resulta demasiado obstruido para completarse en el presupuesto temporal. Esto no invalida H1, pero desplaza la comparación: con 0/5 éxitos en todos los brazos, la métrica discriminante no es la tasa de éxito sino la **distancia recorrida antes del timeout**. En esa métrica, el `fsm` aventaja ligeramente al `reactive` y al `slm` (176 m vs. 158 m vs. 135 m), contrariamente a la hipótesis de que el VLM debería guiar mejor en corredores obstruidos. La alta varianza del `slm` (σ=50 m) y del `fsm` (σ=84 m) impide conclusiones estadísticas con K=5.

Por qué falla específicamente el sensor monocular en este escenario. El fracaso uniforme de los tres brazos en `townsim_ini` no es un artefacto de ajuste de parámetros ni una falla aislada de la política de decisión: `townsim_ini` fue diseñado deliberadamente como caso adversarial para la percepción monocular, y el resultado expone justamente el límite físico que ese diseño buscaba provocar. Vale la pena explicitar el mecanismo, desarrollado formalmente en el Anexo 7:

1. **Flujo traslacional nulo cerca del Foco de Expansión (FOE).** Por la relación deducida en el Anexo 5 (§A5.4.1), $\|v_{\text{trans}}\| \propto \|p - \text{FOE}\|$: el flujo óptico de un punto se anula a medida que su proyección se acerca al FOE, es decir, al centro geométrico de la trayectoria de avance. En un corredor recto cubierto de follaje volado a z=−10 m (bajo la copa de los árboles), las ramas que están *exactamente en la línea de vuelo* — las de mayor riesgo de colisión — son las que producen la señal de flujo más débil, mientras que el follaje periférico (que no bloquea el paso) genera el flujo más intenso. El `ObstacleField` tiende entonces a reportar menor confianza justo donde la decisión es más urgente.
2. **Textura repetitiva y auto-similar.** Ramas y hojas violan el supuesto de correspondencia unívoca del que depende cualquier estimador de flujo: un parche de follaje es visualmente indistinguible de sus vecinos, por lo que el problema de apertura (*aperture problem*) se agrava y el vector estimado resulta frecuentemente espurio, incluso lejos del FOE.
3. **Retroalimentación negativa entre cautela y evidencia.** Ante la ambigüedad de señal, los tres brazos recurren a maniobras de bajo desplazamiento neto, lo que reduce aún más la línea de base de paralaje disponible entre frames consecutivos — un ciclo de "menor confianza → mayor cautela → menor movimiento → menor evidencia" que no se rompe por sí solo.

Estos tres efectos son manifestaciones de la **maldición monocular** (ambigüedad de escala más dependencia estructural del movimiento; Anexo 7, §A7.1): no son errores de implementación del pipeline de `flow_ttc.py`, sino un límite físico del sensor elegido por diseño (§1.2). El deadlock crónico observado en `slm` (20.2/corrida) y `fsm` (24.0/corrida) es, en este sentido, la manifestación en la capa de control de una falla que se origina en la capa de percepción: el VLM y la FSM reciben evidencia degradada o nula del mismo pasaje que deben atravesar, y ningún razonamiento táctico adicional puede compensar la ausencia de señal geométrica confiable en la dirección de interés. Esto también explica por qué el `reactive` — que no delibera y por tanto no puede "esperar más evidencia" — tampoco progresa: sin flujo utilizable cerca del eje de avance, el campo de obstáculos frontal no genera un gradiente de evasión claro en ninguna capa de decisión.

Opciones para el análisis final: (a) aumentar `--max-seconds` a 1800 s y re-correr para dar tiempo a que algún brazo complete; (b) elevar la altitud del corredor a z=−20 m para reducir la densidad de obstrucción; (c) aceptar el resultado como hallazgo de límite operativo y reportar distancia recorrida en lugar de tasa de éxito; (d) como línea de trabajo futuro, evaluar una mitigación estructural del problema de percepción — no solo del ajuste de escenario — mediante un segundo canal de profundidad instantánea (visión estereoscópica) que no dependa del FOE ni del movimiento entre frames (Anexo 7, §A7.4–§A7.6). La opción (c) es la más honesta científicamente dado que los datos ya están colectados; la opción (d) es la que efectivamente atacaría la causa raíz identificada arriba, a diferencia de (a) y (b), que solo relajan el caso de prueba.

11.4.2c Tier 1 — Bloqueo frontal masivo (`townsim_calib_cruce_frontal`)

El escenario cruza la fila de edificios oeste a nivel de calle (z=−10 m), regresando al spawn. Es el primer escenario con obstrucción frontal garantizada: el waypoint WP_3 está al otro lado de una fila de edificios y no hay ruta directa a nivel de calle.

Brazo	Éxito	Dist. media (m)	Deadlocks (media)	Res. VLM	Tiempo exitoso (s)	DistMin (m)
`slm`	1/5 ⚠️	253.8 ± 51.4	18.4	95.3%	262.7 (seed 4)	0.10
`fsm`	0/5 ⛔	274.1 ± 41.4	6.2	100%	—	0.16
`reac-tive`	2/5 ⚠️	286.6 ± 53.1	0	—	157.5 (seeds 3–4)	0.67

Hallazgo principal: el `reactive` supera a los brazos deliberativos en tasa de éxito (2/5 vs. 1/5 vs. 0/5) y en velocidad cuando logra completar (~157 s vs. ~263 s del `slm`). El patrón de éxito del `reactive` es bimodal: en seeds 3 y 4 navega el bloqueo por evasión de flujo óptico puro en ~157 s; en seeds 1, 2 y 5 oscila ante los edificios y agota el presupuesto. El éxito depende del ángulo de aproximación inicial, no de deliberación.

El `slm` logra 1 éxito (seed 4, 5 deadlocks, 262 s) con la tasa de resolución más baja observada (95.3% — 4.7% de deadlocks no resueltos por `deep_vlm`), lo que indica que la geometría del cruce genera situaciones que el escaneo profundo no puede resolver completamente. El `fsm` falla todas con solo 6.2 deadlocks promedio: no es que acumule más atascos que en `townsim_clear`, sino que los atascos en la zona de bloqueo frontal son cualitativamente diferentes — la FSM sin VLM no puede distinguir "avanzar por arriba" de "avanzar lateralmente" ante una fachada continua.

Interpretación para H1: el VLM no aporta ventaja estadísticamente demostrable en este escenario con K=5 (1/5 slm vs. 2/5 reactive vs. 0/5 fsm). El resultado es contraintuitivo respecto a H1: el brazo sin deliberación (`reactive`) es el más exitoso. Una posible explicación es que el flujo óptico sí detecta el obstáculo frontal y lo evade por la ruta de menor resistencia (vertical o lateral), mientras que los brazos deliberativos generan deadlocks que paralizan la evasión. Con K=10 y análisis de SPL por waypoint, sería posible determinar si el VLM contribuye en el segmento específico de cruce o no.

11.4.3b Tier 2 — Corredores urbanos angostos (`citymap_pilot`)

`citymap_pilot` es el escenario terminal: 7 WPs en grilla urbana a z=−10 m entre edificios de 50–100 m. La ruta total es ~310 m; ningún brazo supera el 58% de la ruta dentro del presupuesto de 600 s.

Brazo	Éxito	Dist. media recorrida	σ (m)	Deadlocks (media)	Res. VLM	DistMin (m)
`slm`	0/5 ⛔	168.5 m	83.9	20.4	100%	0.22
`fsm`	0/5 ⛔	178.0 m	90.8	21.4	100%	0.10

Brazo	Éxito	Dist. media recorrida	σ (m)	Deadlocks (media)	Res. VLM	DistMin (m)
`reactive`	0/5 ⛔	**26.7 m**	0.4	0	—	3.05

El hallazgo más significativo es la conducta del `reactive`: recorre exactamente ~27 m en las 5 semillas (σ=0.4 m — comportamiento cuasi-determinista) y se detiene. La causa identificada es geométrica: la ruta pasa por debajo de una autopista elevada y el dron queda atrapado bajo su estructura. El flujo óptico detecta superficies en todas las direcciones y el controlador oscila sin poder escapar — sin un mecanismo de escape global, el `reactive` permanece bajo la autopista hasta agotar el presupuesto. La distancia mínima al obstáculo (media 3.05 m) refleja el espacio bajo la estructura, no proximidad a los edificios de los corredores.

Los brazos deliberativos recorren ~170 m usando `deep_vlm` para resolver deadlocks (20–21 por corrida, todos resueltos al 100%). La alta varianza (σ≈87 m) indica que el progreso depende fuertemente de la semilla: en seed 3 (`slm`, 18.7 m) y seed 5 (`fsm`, 16.6 m) el dron queda paralizado desde el primer waypoint — comportamiento análogo al `reactive` pero detonado por un deadlock inicial irresuelto que consume el presupuesto.

Inversión del resultado respecto a `townsim_calib_cruce_frontal`: en ese escenario el `reactive` superaba a los brazos deliberativos (2/5 vs. 1/5 vs. 0/5); en `citymap_pilot` los brazos deliberativos superan al `reactive` en distancia recorrida (170 m vs. 27 m). La diferencia es la geometría: en `townsim_calib_cruce_frontal` existe una "ruta de escape" lateral que el flujo óptico puede encontrar sin necesidad de razonamiento global; en `citymap_pilot` la grilla de corredores angostos bloquea todas las salidas locales y el `reactive` no tiene manera de determinar qué corredor elegir. El `deep_vlm`, aunque no completa la misión, al menos permite que el dron explore sucesivamente los deadlocks y avance en la dirección correcta entre resoluciones.

Interpretación para H1: `citymap_pilot` proporciona la diferencia más grande del lote entre los brazos deliberativos y el `reactive` — 6.3× más distancia — en un entorno donde el flujo óptico solo no es suficiente para elegir entre corredores.

Atribución. Esa diferencia no demuestra que el VLM haya elegido el corredor correcto. Primero, la comparten `slm` (168.5 m) y `fsm` (178.0 m), y ambos usaban el mismo barrido con VLM en sus deadlocks; el `fsm`, sin consulta táctica al modelo, recorrió incluso algo más. Segundo, el lote no tiene una celda con resolución de atascos sin VLM (`blind`), de modo que la ventaja sobre el `reactive` puede deberse a que **cualquier** mecanismo de resolución de deadlocks mueve al dron —barridos, escapes verticales, esquinas—, con independencia de lo que responda el modelo. Tercero, la auditoría de corridas de diagnóstico en este mismo escenario (cap. 9 §9.8) encontró respuestas del barrido casi constantes y errores de integración que invertían su sentido. El `reactive`, por su parte, quedó atrapado a ~27 m bajo la autopista elevada, la estructura que el waypoint `WP_0_SUR` de `citysim_pilot.json` evita (cap. 10 §10.3.3). El contraste que permite atribuir el efecto al VLM es `fsm × blind`, previsto en los lotes con obstrucción. Sin embargo, con 0/5 éxitos en todos los brazos y K=5, no es posible establecer significancia estadística. El escenario confirma el escenario de uso previsto del VLM (ambigüedad semántica entre corredores de textura uniforme) pero a una escala de dificultad que excede el presupuesto temporal disponible.

11.4.3 Tier 2 — Entorno urbano de crucero (`citysim_clear`)

Con 2.2 invocaciones promedio al VLM y tasa de deliberación de 0.3%, el `slm` opera en modo casi puramente reactivo en este escenario: sólo 10 s de overhead sobre la base de 165 s del `reactive`. Los tres brazos completan sin colisiones y sin diferencias de seguridad (DistMin: 9.4–10.4 m).

Tres observaciones son relevantes para el análisis posterior:

1. `reactive` **determinista:** σ = 0.2 s — la varianza más baja de los tres tiers. A −50 m de altitud máxima, sin obstáculos activos, el `reactive` ejecuta la ruta con variación nula.
2. `fsm` **variable:** dos corridas con deadlock (seeds 4 y 5, 208 s y 212 s) vs. tres sin deadlock (~170 s). La geometría del manifiesto (cambios de altitud en rampa) genera ocasionalmente atascos en la FSM que no se producen en el `reactive` ni en el `slm`. Con K = 5, δ = +0.04 (`slm` vs `fsm`), no significativo.
3. `slm` **recorre menos distancia que** `fsm`**:** 427.4 m vs. 434.9 m — el `slm` tiende a trayectorias más cortas que la FSM, posiblemente porque las pocas invocaciones VLM producen correcciones de rumbo que evitan desviaciones que la FSM acumula al gestionar sus deadlocks.

El escenario `citymap_pilot` (corredores angostos a −10 m, entre edificios de 50–100 m) es el experimento de Tier 2 donde el VLM tiene potencia discriminativa real. Los datos de `citysim_clear` establecen que, en condiciones de tránsito libre, ningún brazo discrimina en calidad de navegación — diferencia esperada por diseño experimental.

11.5 Síntesis de hallazgos

11.5.1 Soporte a hipótesis H2 (overhead del VLM se diluye con la longitud de ruta)

H2 se confirma cualitativamente con los tres escenarios de control completados:

Escenario	Ruta (m)	Ratio `slm` / `reactive`
`minisim_clear`	~185	2.20×
`townsim_clear`	~626	1.14×
`citysim_clear`	~427	1.06×

El ratio decrece monótonamente con la longitud de ruta, con separación perfecta entre los tres tiers (Cliff's $\delta = 1.0$ en comparaciones por pares). La potencia estadística es insuficiente para la corrección Bonferroni ($K = 5$, p mínimo alcanzable $\sim 7.9 \times 10^{-3}$ >> $\alpha = 0.0028$), pero el patrón es consistente con la predicción teórica: el overhead fijo de cada invocación VLM (~3–5 s) se reparte sobre un presupuesto de tiempo mayor a medida que la ruta se alarga.

11.5.2 Soporte a hipótesis H1 (VLM aporta ventaja en ambientes obstruidos)

H1 no se confirma ni refuta con los datos disponibles. El diseño experimental identifica tres regímenes cualitativamente distintos:

Escenario	Régimen	Ganancia del VLM
`minisim_clear`, `townsim_clear`, `citysim_clear`	Control libre	Ninguna — costo puro
`townsim_calib_cruce_frontal`	Bloqueo frontal	Negativa: `reactive` 2/5 > `slm` 1/5 > `fsm` 0/5
`townsim_ini`	Deadlock crónico	Neutral — 0/5 todos; VLM incapaz de recuperar señal monocular
`citymap_pilot`	Ambigüedad de corredor	Positiva en distancia para `slm` y `fsm` (6.3×) vs. `reactive`; 0/5 todos; no atribuible al VLM (§11.4.3b)

El dato más favorable a H1 es `citymap_pilot`: los brazos deliberativos cubren 6.3× más distancia que el `reactive`. Pero, como se discute en §11.4.3b, la ventaja la comparten `slm` y `fsm`, ambos con el VLM en la resolución de atascos y sin una celda `blind` de contraste, por lo que no puede atribuirse a que el modelo elija el corredor correcto. Además, 0/5 éxitos en todos los brazos impide cuantificar ventaja en términos de tasa de éxito.

El resultado contraintuitivo de `townsim_calib_cruce_frontal` (el brazo menos deliberativo supera a los más deliberativos) ilustra un efecto de composición: en obstrucciones donde existe una "ruta de escape" geométricamente accesible desde la posición actual, el `reactive` la encuentra más rápido porque no detiene el vuelo para deliberar. El `deep_vlm` introduce latencia de reposicionamiento que, en este caso, llega cuando el momento de escape ya pasó.

11.5.3 Límites operativos identificados

Los escenarios de obstrucción revelan dos límites distintos de la arquitectura:

Límite de percepción monocular (`townsim_ini`): el corredor arbolado a z=−10 m produce deadlock crónico en los tres brazos porque la señal de flujo óptico se degrada en la dirección de avance (flujo traslacional nulo cerca del FOE) y las texturas repetitivas del follaje generan vectores espurios. Ningún mecanismo de decisión puede compensar la ausencia de señal geométrica confiable en la dirección de interés. Este es un límite físico del sensor monocular, no del algoritmo de control. Con la configuración final —profundidad monocular estimada en la capa estratégica y en el barrido (cap. 5 §5.10.6)— las diez corridas de `townsim_ini` del lote de octubre completan la misión (§11.6.2). La comparación no es controlada (otro manifiesto, otro código), pero indica que el límite era del canal de flujo solo, no del sensor monocular.

Límite de escape geométrico (`citymap_pilot`, `reactive`): cuando la geometría cierra todas las salidas locales — el dron queda atrapado bajo una autopista elevada con el techo arriba, el suelo abajo y paredes laterales —, el flujo óptico no puede generar un gradiente de evasión útil en ninguna dirección. La opción "perder altura" no ayuda (agrava el atrapamiento bajo la estructura). Hace falta un mecanismo que proponga una salida no accesible localmente en el campo de flujo; el barrido `deep_vlm` está concebido para eso, pero el lote base no permite distinguir su aporte del de cualquier otro mecanismo de resolución de deadlocks (§11.4.3b).

11.5.4 Conclusión operativa

Los 75 runs del lote completo establecen que:

1. **En ruta libre**, ningún brazo deliberativo justifica su overhead frente al `reactive` en términos de velocidad o seguridad. El VLM añade latencia sin agregar valor de decisión.
2. **En ambientes con obstrucción geométrica compleja** (`citymap_pilot`), los brazos con resolución de deadlocks progresan 6.3× más que el `reactive`, aunque sin completar la misión. El lote no permite atribuir esa ventaja al contenido de las respuestas del VLM (§11.4.3b).
3. **En ambientes con degradación de señal monocular** (`townsim_ini`), la limitación es de percepción, no de decisión. Ningún brazo — deliberativo o reactivo — puede compensarla. La extensión natural de esta línea de investigación es la incorporación de un segundo canal de profundidad instantánea (visión estereoscópica) que no dependa del movimiento entre frames (Anexo 7, §A7.4–§A7.6).
4. **El VLM no penaliza la seguridad:** la distancia mínima al obstáculo (medida con el canal de profundidad del simulador, como auditoría externa) es similar o mejor que la del `reactive` en los

escenarios donde todos los brazos operan, y las 45 corridas de control producen 0 colisiones en todos los brazos.

11.6 Lotes de la capa estratégica (octubre, configuración vigente)

Dos lotes con el sistema de los capítulos 5–8 responden si la capa estratégica aporta y con qué fuente. Diseño común: brazo `slm`, barrido de deadlock con profundidad estimada (`DEADLOCK_STRATEGY = depth`) igual en todas las celdas —así se aísla la capa estratégica—, semillas 1–5 con jitter de la pose inicial, mismo plan de vuelo para todas las corridas de un escenario y el modelo de inferencia en la misma GPU que Unreal. El avance se mide en **waypoints del plan completados** (no con el índice del registro, que cuenta también las sub-metas insertadas). La regla de decisión se fijó antes de correr cada lote: una fuente aporta más que otra si la mediana del avance a un tiempo de referencia la supera en un umbral **y** gana en al menos 4 de las 5 semillas emparejadas; si ningún par lo cumple, las fuentes son equivalentes en avance y se decide por costo y seguridad. El p-valor se informa como diagnóstico: con 5 pares el mínimo del Wilcoxon exacto es 0.0625. Análisis: `experiments/analyze_strategic_batch.py`.

11.6.1 CitySim (`citysim_pilot`): sin diferencia en avance

15 corridas (fuentes `depth`, `vlm` y `off`), tope de 480 s, plan de ruta del VLM (cap. 4 §4.3.1; 15 waypoints con los desvíos). Regla: diferencia de medianas ≥ 2 waypoints a 400 s.

Fuente	Plan a 150 / 300 / 400 s (mediana)	Éxito	Colisiones	DistMin mediana (peor)	Deadlocks (total)	Latencia de la capa
`depth`	5 / 10 / 13	3/5	0	5.83 m (0.10)	4	108 ms
`vlm`	6 / 10 / 13	4/5	0	3.29 m (0.13)	2	788 ms
`off`	6 / 11 / 13	5/5	0	4.95 m (1.89)	0	—

Ningún par cumple la regla: las tres fuentes son **equivalentes en avance**. Por costo y seguridad, `off` es la mejor celda (todas las misiones completas, sin deadlocks, el mejor peor caso de distancia). Las fallas de las otras dos: `depth` semilla 1 llegó a 14 de 15 waypoints al tope de tiempo; `depth` semilla 5 tuvo tres deadlocks y se acercó a 0.10 m; `vlm` semilla 2 terminó por congelamiento físico, con el dron incrustado en la malla tras sub-metas repetidas hacia la fila de abajo de la grilla. Con 5 semillas, 5/5 frente a 3/5 no es significativo (Fisher $p \approx 0.44$): la lectura es que la capa no aporta en este escenario, no que empeore. En `off` no hubo deadlocks, de modo que el barrido no se ejerció.

11.6.2 TownSim (`townsim_ini`): mismo avance, más margen con la profundidad

10 corridas (fuentes `depth` y `off`; el VLM quedó fuera por su AUC de 0.53 en el banco y su equivalencia en CitySim), tope de 900 s, manifiesto validado a mano sin plan de ruta (el mapa de TownSim no está verificado como registrado y el plan podría sacar la ruta del corredor). El escenario cruza el corredor peatonal arbolado a 10 m de altura, bajo las copas. Regla: diferencia de medianas ≥ 1 waypoint (de 8) a 900 s.

Fuente	Éxito	Duración (mediana)	Recorrido (mediana)	DistMin en el corredor, mediana (peor)	Segundos a < 1 m en el corredor (mediana)	Deadlocks resueltos por el barrido	Latencia de la capa
`off`	5/5	286 s	362 m	0.11 m (0.10)	29	15/16	—
`depth`	5/5	307 s	428 m	0.41 m (0.10)	10	4/14	114 ms

Las diez corridas completan los ocho waypoints sin colisiones: **equivalentes en avance**. Costo y seguridad tiran en sentidos opuestos:

- `off` **es más barato:** 21 s menos de mediana (más rápido en 4 de 5 semillas) y 66 m menos de recorrido, sin servicio de capa.
- `depth` **mantiene más distancia del follaje:** pasa menos segundos a menos de 1 m en **las 5 semillas emparejadas** (0 frente a 21, 10/33, 3/43, 10/27, 11/29; Wilcoxon exacto p = 0.0625, el mínimo posible con 5 pares).

DistMin se calculó por tramo con la pose de cada muestra de la auditoría: los acercamientos ocurren en la plaza y el corredor (x < −40 m); en el resto de la misión la distancia mínima supera los 10 m. El valor de ~0.10 m que se repite es, probablemente, el plano cercano de la cámara dentro del follaje —sin colisión física—; no se verificó en video. La primera muestra de cada corrida se excluye: en `off` semilla 1 marcó 0.012 m a 30 m de altura y la siguiente, 98 m. Con `off` los deadlocks son deterministas —los tres en el paso de la moldura (WP_1b), con escape vertical de −10 a −16 m— y el barrido con profundidad los resuelve (15 de 16). Con `depth` ocurren en otros puntos y el barrido resuelve menos (4 de 14), sin abortos: los no resueltos salen por el escape vertical.

Referencia histórica, no comparable. En el lote base (§11.4.2b) las 15 corridas de `townsim_ini` llegaron al tope de 900 s sin completar. Aquel lote voló otro manifiesto (sin WP_0b ni WP_1b), otro código y el barrido con el VLM; la diferencia no se puede atribuir a un solo cambio.

11.6.3 Lectura conjunta y decisión de configuración

- **La capa estratégica no acelera la misión** con ninguna fuente, en ninguno de los dos escenarios: el avance lo dan el plan de vuelo y el lazo rápido.
- **Con obstáculos orgánicos, la capa con profundidad aporta margen** de manera consistente; en CitySim ese efecto no se vio.

- **Ninguna señal favorece al VLM en el lazo.** En la capa estratégica fue equivalente a no tener capa, con una latencia siete veces mayor que la de la profundidad.
- **El barrido con profundidad se ejerció en vuelo** en TownSim y resolvió 19 de 30 deadlocks; todas las misiones terminaron.

Por el margen en vegetación, y con un costo moderado (~114 ms en su propio hilo, ~440 MiB de VRAM, 7 % más de tiempo de misión), la configuración final es `STRATEGIC_SOURCE = depth` y `DEADLOCK_STRATEGY = depth` (cap. 5 §5.10.7). Es una decisión de criterio sobre resultados con 5 semillas por escenario, no un efecto significativo.

Límites. Un escenario por tipo y 5 semillas por celda; DistMin solo mira el tercio central de la cámara frontal (los roces laterales no cuentan); en TownSim la captura de imágenes tarda el doble que en CitySim (mediana 209 ms frente a 110 ms), igual para las dos celdas.

12. Conclusiones

12.1 Retomar el hilo conductor

Este trabajo partió de una constatación práctica: en un sistema de control donde un modelo de lenguaje visivo consume descripciones de escena para deliberar sobre maniobras de evasión, el error más caro no está en la política de decisión del modelo sino en la interfaz que lo alimenta.

La evidencia de esa afirmación —desincronización de esquemas de percepción, campos nulos por calibración de escala diferencial, descarte silencioso en grafos de estado— nunca emergió observando el comportamiento superficial del vuelo; emergió auditando formalmente los contratos de datos y la propagación de estado entre capas (cap. 9).

Los 75 runs del lote experimental (5 escenarios × 3 brazos × 5 semillas) validan la arquitectura resultante de ese proceso de depuración y permiten responder, con datos, a la pregunta de investigación central.

Alcance de la evidencia. Esos 75 runs se ejecutaron el 8 y 9 de septiembre con una configuración del brazo `slm` en la que el VLM elegía macro-acciones tácticas (cap. 11). El mismo método de auditoría —comparar la respuesta cruda del modelo con la acción efectivamente ejecutada— encontró, en corridas de diagnóstico sobre `citysim_pilot`, un error de marco de referencia que ejecutaba al revés la respuesta del barrido, un valor por defecto que reemplazaba la respuesta del modelo, respuestas casi constantes, una señal de profundidad del simulador filtrada al brazo evaluado y una capa de reglas deterministas apiladas sobre la decisión del modelo (cap. 9, §9.8). El sistema descrito en los capítulos 5–8 contiene los mecanismos de contención de esos fallos. Con él se corrieron, en octubre, el banco de prueba del VLM a bordo (cap. 11, §11.0) y dos lotes de la capa estratégica en CitySim y en el corredor arbolado de TownSim (cap. 11, §11.6). Las conclusiones de §12.2 sobre los escenarios de control siguen en pie; las de los escenarios con obstrucción describen la configuración del lote base, salvo donde se indica la evidencia de octubre, y la comparación entre brazos con el sistema final queda para los lotes D, E y F (cap. 10, §10.12).

12.2 Respuesta a la pregunta de investigación

¿Aporta el brazo deliberativo VLM (`slm`) sobre la máquina de estados (`fsm`) y sobre el control puramente reactivo (`reactive`) en navegación autónoma de dron en entornos urbanos simulados?

La respuesta es matizada por escenario y no puede reducirse a un único enunciado:

En entornos de tránsito libre (45 corridas de control)

El `slm` no aporta ventaja sobre el `fsm` ni sobre el `reactive` en términos de tasa de éxito o seguridad: los tres brazos completan todas las rutas sin colisiones. El VLM introduce un overhead de tiempo que se diluye con la longitud de ruta (**H2 confirmada**):

Escenario	Ruta aprox.	Ratio `slm` / `reactive`
`minisim_clear` (Tier 0, ~185 m)	2.20×	
`townsim_clear` (Tier 1, ~626 m)	1.14×	
`citysim_clear` (Tier 2, ~427 m)	1.06×	

El ratio decrece monótonamente (Cliff's δ = 1.0 en comparaciones por pares). La convergencia hacia 1.0× a medida que la ruta se alarga confirma que el overhead del VLM es absorbido por el presupuesto total de vuelo: en rutas largas, pocas invocaciones son necesarias y su peso relativo decrece. El costo temporal *efectivo* por invocación, derivado de las diferencias de tiempo de misión del capítulo 11, es de ~1.8 s en Tier 0 (89.5 s de diferencia sobre 49.6 invocaciones) y ~3.8 s en Tier 1 (36 s sobre 9.6); supera la latencia de inferencia pura medida en un piloto (mediana de 1.46 s, cap. 8, §8.6) porque incluye el avance cauto a 0.5 m/s mientras se esperaba la respuesta. En el sistema descrito en el capítulo 5 el dron no frena a esperar al VLM fuera de los deadlocks, por lo que este costo debe medirse de nuevo. Esta predicción se sostenía teóricamente (cap. 8, §8.6) y los datos la verifican.

En entornos con obstrucción (30 corridas de prueba extendida)

La evidencia es heterogénea y revela tres regímenes cualitativamente distintos:

Régimen 1 — Deadlock crónico por límite de percepción (`townsim_ini`): El corredor arbolado a z=−10 m produce 0/5 éxitos en los tres brazos. El flujo óptico traslacional se anula precisamente en los obstáculos centrados en la trayectoria (cerca del Foco de Expansión), que son los de mayor riesgo frontal. Ningún mecanismo de decisión puede compensar la ausencia de señal geométrica confiable: ni el `deep_vlm`, ni la FSM, ni el control reactivo. En el lote base esto pareció un límite del sensor; con la configuración final —profundidad monocular estimada desde el mismo RGB en la capa estratégica y en el barrido— las diez corridas de octubre en este escenario completaron la misión sin colisiones (cap. 11, §11.6.2). La comparación no es controlada (otro manifiesto y otro código), pero sugiere que el límite era del canal de flujo óptico solo, no de la cámara monocular.

Régimen 2 — Trampa geométrica por ausencia de razonamiento global (`citymap_pilot`): Los brazos deliberativos (`slm`, `fsm`) cubren **6.3× más distancia** que el `reactive` (170 m vs. 27 m). El `reactive` queda paralizado bajo una autopista elevada: el flujo óptico detecta superficies en todas las direcciones y el controlador oscila sin escape posible — sin razonamiento global, ninguna evasión local desbloquea la situación. Es la diferencia más grande del lote a favor de los brazos deliberativos, pero **no puede atribuirse al VLM**: la comparten `slm` y `fsm` (ambos resolvían sus deadlocks con el mismo barrido con VLM), no hay una celda de resolución sin VLM con la que contrastarla, y la auditoría de corridas de diagnóstico en este escenario encontró respuestas del barrido casi constantes (cap. 9, §9.8; cap. 11, §11.4.3b). Lo que el dato sostiene es que **algún** mecanismo de resolución de deadlocks es necesario en este régimen; si el VLM es el que aporta, es la pregunta de los lotes con obstrucción.

Régimen 3 — Obstrucción frontal con escape lateral accesible (`townsim_calib_cruce_frontal`): El `reactive` supera a los brazos deliberativos (2/5 vs. 1/5 vs. 0/5). Cuando existe una ruta de escape geométricamente accesible desde la posición actual, el flujo óptico la encuentra más rápido porque no detiene el vuelo para deliberar; el `deep_vlm` introduce latencia de reposicionamiento que, en el momento de la oportunidad de evasión, ya pasó. El VLM no aporta ventaja —y en la mayoría de semillas introduce una desventaja neta.

Síntesis

Con la evidencia disponible, **no se puede afirmar que el VLM aporte valor** en ningún régimen. El único dato favorable —el progreso de los brazos deliberativos en `citymap_pilot`— no aísla al modelo de los mecanismos de resolución de deadlocks que lo rodeaban, y la auditoría de corridas de diagnóstico mostró respuestas que apenas dependían de la escena y llegaban a ejecutarse al revés (cap. 9, §9.8). El régimen en el que el VLM *debería* aportar es rodear obstáculos y elegir corredores donde el flujo óptico no discrimina, y el sistema está construido para medir exactamente eso: el modelo es la única fuente de rodeos del brazo `slm`, sin reglas que lo corrijan, y el lote incluye el contraste sin VLM (`fsm × blind`).

Evidencia de octubre con el sistema final (cap. 11, §11.0 y §11.6). Medido contra etiquetas de referencia, el VLM de 1.6 B no separa sectores volables de bloqueados (AUC 0.53) ni rumbos abiertos de cerrados en el barrido; una red de profundidad monocular de ~25 M de parámetros sí lo hace (AUC 0.83–0.87) en ~110 ms. En vuelo, la capa estratégica no acelera la misión con ninguna fuente; con obstáculos orgánicos, la fuente de profundidad aleja al dron del follaje en las cinco semillas. El único aporte medible del VLM está fuera del lazo: el plan de ruta sobre el mapa cenital (AUC 0.81), donde la pregunta es de reconocimiento y no de distancia. Por eso la configuración final saca al VLM del lazo táctico y lo deja en la planificación previa al vuelo.

No aporta valor en ruta libre ni en obstrucciones con escape local. No puede compensar el límite físico del sensor monocular en textura auto-similar densa. Con K=5 semillas y la corrección de Bonferroni aplicada a 18 comparaciones ($\alpha_{corr} \approx 0.0028$), ningún resultado individual alcanza significancia estadística formal. El patrón H2, no obstante, es consistente con $\delta = 1.0$ en todos los pares; el de `citymap_pilot` es cuantitativamente sustancial (6.3×) pero, por las razones dadas, no constituye evidencia de H1. La validación estadística requeriría $K \geq 10$–15 semillas por condición.

12.3 Limitaciones

- **Evidencia con obstrucción obtenida con otra configuración:** los resultados de los escenarios con obstrucción (0/5 en `townsim_ini`, el patrón de tres regímenes) se obtuvieron con la configuración del lote base. No describen el sistema de los capítulos 5–8. Las corridas piloto, ejecutadas cada una con ajustes distintos, fueron de diagnóstico y no constituyen evidencia estadística; su valor está en los fallos que expusieron (cap. 9).

- **Validación en vuelo limitada:** la capa estratégica, el barrido en marco mundo y la detección de atasco de dos señales están cubiertos por tests (incluido el grafo compilado de punta a punta) y tienen dos lotes de vuelo (25 corridas, cap. 11 §11.6), con un escenario por tipo y 5 semillas por celda. Sus umbrales (período de 3 s, edad máxima de 10 s, sub-meta a 15 m como máximo, cambio de altitud de 4 m como máximo, deduplicación de 10 m, radio de 8 m sin consultas alrededor del waypoint, grilla de 3×3) son decisiones de diseño, no valores calibrados.
- **Capacidad de discriminación del VLM no demostrada:** en las corridas de diagnóstico, 52 de 53 respuestas sectoriales declararon el frente transitable, incluso a menos de un metro de una superficie (cap. 9, §9.8.3). El diseño del sistema cambia la pregunta y el prompt (cap. 8, §8.3–§8.5), y el banco de prueba con el modelo a bordo (cap. 11, §11.0) muestra que LFM2.5-VL-1.6B no distingue sectores volables de bloqueados en el render de CitySim: ninguna de las ocho preguntas evaluadas supera el criterio, la grilla de vuelo queda en 53 % de exactitud balanceada y el barrido declara transitables todos los rumbos. La única señal aprovechable está en el mapa cenital (AUC 0.81 para rutas libres frente a bloqueadas). Por eso la configuración final responde la misma pregunta con la profundidad estimada (cap. 5, §5.10.6–§5.10.7), y el VLM queda en la planificación de ruta.
- **Componentes del lazo rápido sin evaluar por separado:** la evasión por flujo, `GIRAR_90`, el gobernador de velocidad, la detección de techo y la reacción a contacto forman parte del sistema por ser comunes en espíritu a los tres brazos, no por evidencia de beneficio. Varios de sus umbrales se fijaron sobre una o dos corridas.

- **Latencia del VLM sin medición controlada:** con los servidores anteriores las mediciones no coinciden entre sí (mediana de 1.46 s en un piloto frente a 5.3–5.6 s en las corridas de diagnóstico, cap. 8, §8.6); con el modelo local vigente la mediana en vuelo es de 0.77 s. No hay una medición que separe tamaño de imagen, carga de GPU y arranque en frío. Con la configuración final el VLM no corre en vuelo, de modo que esta limitación solo afecta a la ablación con `STRATEGIC_SOURCE = vlm`.
- **Validación exclusivamente en simulación (AirSim):** sin validación en hardware físico (Jetson Nano + dron real), que el plan de trabajo aprobado (`plan_tesis/plan-tesis.md`, §"Transferencia de los resultados obtenidos") sitúa como siguiente etapa fuera del alcance de este trabajo.

- **Tamaño de muestra:** K=5 semillas por condición (75 corridas en total). La corrección de Bonferroni para 18 comparaciones fija $\alpha_{corr} \approx 0.0028$, un umbral que el p-valor mínimo alcanzable con K=5 ($p \approx 7.9\times10^{-3}$) no puede superar. Los resultados son indicativos y cuantitativamente consistentes, pero no formalmente significativos.

- **Sensor monocular (parcialmente mitigado):** el canal de flujo óptico depende del movimiento del vehículo entre frames, lo que lo hace estructuralmente ciego a obstáculos centrados en la trayectoria (Foco de Expansión) y a las superficies horizontales a la altura de vuelo (cap. 9, §9.8.7). La configuración final lo complementa en el lazo lento con profundidad monocular estimada (Depth Anything V2, Yang et al., 2024; cap. 6, §6.14), que juzga un fotograma aislado y no necesita traslación. Un primer intento de septiembre, que usaba la misma red como freno del lazo rápido con otro estadístico, se había descartado; con el p5 por sector y un umbral de 15 m pasó el banco de prueba (AUC 0.83–0.87). Sus límites: se entrenó en parte con datos sintéticos y se evaluó sobre el mismo render (en cámara real puede rendir menos), su escala métrica no está calibrada a la altura de vuelo y el follaje sigue siendo el caso difícil (en `townsim_ini` redujo los acercamientos, no los eliminó). La validación en imágenes reales queda pendiente (Anexo 7).
- **Calibración del canal de ocupación de `ObstacleField` parcial:** el canal de TTC está formalmente validado contra etiquetas de referencia tomadas de la profundidad del simulador (cap. 7). El umbral de ocupación se recalibró por ROC/Youden (`OBSTACLE_OCCUPANCY_BLOCKED = 0.011`, AUC = 0.87) pero con 196 frames, una sola escena y 29 positivos; el valor original 0.35 nunca se disparaba y el lote base se ejecutó antes de la recalibración (cap. 7, §7.5). La validación cruzada en otras escenas y el efecto del `OccupancyCalibrator` adaptativo siguen sin evaluarse.
- **Un único mapa por escenario:** cada escenario de obstrucción se corrió en un solo mapa urbano (TownSim o CitySim). La generalización de los resultados a otras geometrías urbanas no está garantizada con los datos disponibles.
- **VLM fijo (LFM2.5-VL-1.6B, `lfm2.5-vl-1.6b:q4km-ctx8k`, zero-shot):** el modelo opera en régimen zero-shot sin sintonización al dominio de vuelo. Los resultados de deliberación están acotados por la distribución de preentrenamiento del modelo y el diseño del prompt, no por la capacidad máxima de la arquitectura.

12.4 Trabajo futuro y mejoras

Las líneas de continuidad y optimización derivadas de este trabajo se estructuran en los ejes siguientes, integrando la transferencia física, la arquitectura de la capa deliberativa y la infraestructura perceptual:

- **Especialización y adaptación ligera del modelo deliberativo vía LoRA/QLoRA (Anexo 3):** En la implementación evaluada, el modelo multimodal (**LFM2.5-VL-1.6B**, `lfm2.5-vl-1.6b:q4km-ctx8k`; antes Qwen2.5-VL-3B-Instruct) opera en régimen *zero-shot*. El prompt ya es mínimo (una imagen con la grilla y una respuesta de nueve valores sin texto libre) y la latencia local es de ~0.8 s (cap. 8, §8.6), de modo que el margen de LoRA está en la **calidad** de la respuesta: que el modelo distinga sectores libres de bloqueados, lo que hoy no hace (AUC 0.53, cap. 11 §11.0). Como se fundamenta en el capítulo 8 (§8.8) y se desarrolla en detalle en el Anexo 3, la sintonización fina de bajo rango (*Low-Rank Adaptation*, LoRA / QLoRA / DoRA) quedó excluida del alcance inmediato por escasez de diversidad en el dataset de maniobras inicial y para evitar riesgos de sobreajuste a geometrías urbanas particulares. Como trabajo futuro, se proyecta entrenar adaptadores sobre las matrices de atención (W_q, W_v) y proyecciones MLP del LLM (manteniendo congelado el codificador visual ViT) utilizando la telemetría de vuelos exitosos. Un conjunto de entrenamiento natural son los fotogramas con columnas que la propia capa estratégica registra (cap. 5, §5.10.4), etiquetados con la profundidad del simulador en forma offline —sin que esa señal entre nunca al lazo de vuelo—.
- **Decodificación restringida con gramáticas reducidas y dinámicas condicionadas por el grafo (Anexo 4):** El Anexo 4, §A4.4.2, especifica una optimización pensada para un modelo que elige macro-acciones: la **inyección dinámica de gramáticas acotadas según el estado de vuelo y el motivo de consulta**. En el sistema descrito el modelo no emite acciones (sus esquemas describen sectores o imágenes, §8.2), por lo que la idea se traslada a restringir **qué describe** según el contexto (por ejemplo, omitir los sectores que el lazo rápido ya sabe bloqueadas). La propuesta original, como referencia:
- Ante colisiones inminentes (`"TTC_CRITICO"`), restringir la gramática estrictamente a `["evasive", "girar_90"]`, podando sintácticamente cualquier token asociado a acciones pasivas.
- Ante atascos cinemáticos (`"DEADLOCK_ESCAPE"`), forzar alternativas de escape `["girar_90", "fsm", "degraded"]`, impidiendo bucles de evasión redundantes.
- Esta poda semántica *a priori* concentrará la distribución de probabilidad en alternativas físicamente coherentes, reducirá la entropía de muestreo y acelerará la decodificación aprovechando el salto de tokens (*token fast-forwarding*).
- **Perfeccionamiento del lazo perceptual monocular (Anexo 5):**
- Validación cruzada del umbral de ocupación (ya calibrado por ROC/Youden sobre una escena, cap. 7, §7.5) en CitySim y en escenas con vegetación, con un dataset mayor que 29 positivos y evaluación del `OccupancyCalibrator` adaptativo, hoy ausente del runner de lotes (Anexo 5).
- Completar la validación de la derotación angular: la *inhibición* del estimador ante guiñadas de 0.3–1.0 rad/s está verificada (cap. 7, §7.6), pero falta caracterizar el error de TTC en el rango previo al guard (0–0.3 rad/s), la reanudación de la estimación al cesar el giro y el comportamiento con IMU real.

- **Mitigación de la "maldición monocular" en pasajes densamente obstruidos (Anexo 7):** El fracaso uniforme de los tres brazos en `townsim_ini` (cap. 11, §11.4.2b) expuso un límite estructural del sensor monocular: el flujo óptico traslacional se anula exactamente en los obstáculos centrados en la trayectoria de avance (Foco de Expansión), que son los de mayor riesgo de colisión frontal (Anexo 7, §A7.1–§A7.2). Como trabajo futuro de menor costo que un LiDAR emulado, se propone incorporar un segundo canal de profundidad instantánea mediante un **par estéreo** (baseline fija y conocida), que resuelve la ambigüedad de escala y elimina la dependencia del movimiento entre frames al recuperar profundidad métrica en un único instante (Anexo 7, §A7.3–§A7.6). El SLAM monocular disperso o semi-denso queda como línea posterior y complementaria, con limitaciones propias frente a follaje en movimiento y geometría subpíxel.
- **Política de navegación por aprendizaje por refuerzo visual como cuarto brazo (`rl`) (Anexo 8):** Los tres brazos evaluados (`reactive`, `fsm`, `slm`) cubren el espacio de estrategias reactiva, basada en lógica explícita y basada en VLM pre-entrenado. Un cuarto brazo ortogonal a los tres anteriores es una **política neuronal entrenada por RL** que ocupa la misma capa táctica que `fsm_node` y `evasive_node`, manteniendo activa la capa deliberativa VLM en modo jerárquico. La hipótesis central es que una política especializada en el espacio de macro-acciones de este sistema (`MANTENER_RUMBO`, `EVADIR_*`, `GANAR_ALTURA`, `PERDER_ALTURA`, `FRENAR`) puede superar al brazo `reactive` en obstrucción densa —donde el flujo óptico colapsa— sin el overhead de latencia de un VLM (0.8 s de mediana con el modelo local, 1.5–5.6 s con los servidores anteriores, cap. 8 §8.6, vs. < 5 ms de inferencia neuronal, estimado). El Anexo 8 desarrolla en detalle la jerarquía VLM → RL → PX4, el espacio de observación mixto `{rgb, ttc_field, nav}`, la función de recompensa con currículo, la justificación de PPO sobre SAC/DQN, la arquitectura CNN+FC propuesta y el protocolo de evaluación comparativa de los 4 brazos (K=5, 3 escenarios, 60 corridas). Su prerequisito es que el brazo `slm` tenga resultados estables.
- **Modelos de mundo latentes (LeWorldModel) como complemento o alternativa parcial al SLM (Anexo 9):** En marzo de 2026 se presentó LeWorldModel (Maes et al., 2026), un modelo de mundo de tipo JEPA de ~15 M de parámetros que aprende la dinámica del entorno de extremo a extremo desde píxeles y planifica en su espacio latente con tiempos del orden de un segundo, frente a los 1.6 B parámetros del VLM empleado en esta tesis. El Anexo 9 analiza qué aportaría y qué perdería el sistema con este tipo de modelo (predicción de consecuencias de acciones continuas frente a la semántica y el razonamiento global del VLM), estima analíticamente su viabilidad en una *companion computer* Jetson Nano / Orin Nano —memoria holgada, latencia condicionada al presupuesto de muestreo de la planificación— y concluye que, con la evidencia actual (resultados en tareas 2D/3D simples, sin validación en visión monocular urbana aérea), no corresponde reemplazar al brazo `slm` sino **complementarlo** como verificador de consecuencias y detector de sorpresa, o usarlo como simulador latente para entrenar el brazo `rl` del Anexo 8. Las cifras de latencia del anexo son estimaciones; su reemplazo por mediciones en hardware edge y el protocolo de evaluación en lazo cerrado (§A9.7) constituyen el trabajo futuro concreto.

- **Validación del sistema:** además de correr los lotes D, E y F con la celda `fsm × blind`, quedan tres tareas concretas. (i) Medir la distribución de resultados de la capa estratégica con profundidad y el avance hacia el waypoint real tras cada sub-meta en los tres escenarios con obstrucción (cap. 10, §10.6.3). (ii) Incorporar un test de contrato que recorra `src/agents/` y falle ante cualquier clave de estado no declarada en `DroneState`; hoy los tests verifican solo las claves de control en el grafo compilado. (iii) Evaluar por separado los componentes del lazo rápido conservados sin evidencia (gobernador de velocidad, `GIRAR_90`, detección de techo).
- **Transferencia a hardware embebido y navegación en enjambre:**
- Validación en plataforma física real: integración en una *companion computer* de bajo consumo (NVIDIA Jetson Nano / Orin) conectada vía MAVLink/ROS a un cuadrirrotor físico con sensor monocular, evaluando el lazo reactivo-táctico en entornos reales según las fases delineadas en el plan de trabajo aprobado.
- Extensión a navegación cooperativa en enjambre y fusión sensorial con sensores complementarios (flujo óptico diferencial con sensores acústicos o ToF ultraligeros).

13. Referencias

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A. A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Cai, Q., Chaudhary, V., Chen, D., Chen, D., ... Zhang, X. (2024). *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. arXiv:2404.14219. https://arxiv.org/abs/2404.14219

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., & Bellemare, M. G. (2021). Deep Reinforcement Learning at the Edge of the Statistical Precipice. *Advances in Neural Information Processing Systems (NeurIPS)*, *34*. arXiv:2108.13264. https://arxiv.org/abs/2108.13264

Agrolatam. (2025, 31 de octubre). *Tecnología de precisión e inteligencia artificial: cómo el INTA usa drones para optimizar la siembra de caña de azúcar en Argentina*. https://www.agrolatam.com/agricultura/drones-ia-inta-monitoreo-cana-de-azucar-argentina-2025/

Al-Kaff, A., García, F., Martín, D., De La Escalera, A., & Armingol, J. M. (2017). Obstacle Detection and Avoidance System Based on Monocular Camera and Size Expansion Algorithm for UAVs. *Sensors*, *17*(5), 1061. https://doi.org/10.3390/s17051061

Aldao, E., González-de Santos, L. M., & González-Jorge, H. (2022). LiDAR Based Detect and Avoid System for UAV Navigation in UAM Corridors. *Drones*, *6*(8). https://doi.org/10.3390/drones6080185

AlMahamid, F., & Grolinger, K. (2022). Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review. *Engineering Applications of Artificial Intelligence*, *115*, 105321. https://doi.org/10.1016/j.engappai.2022.105321

Altman, D. G., & Bland, J. M. (1995). Statistics notes: Absence of evidence is not evidence of absence. *BMJ*, *311*(7003), 485. https://doi.org/10.1136/bmj.311.7003.485

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software Engineering for Machine Learning: A Case Study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., & Zamir, A. R. (2018). *On Evaluation of Embodied Navigation Agents*. arXiv:1807.06757. https://arxiv.org/abs/1807.06757

Anthropic. (2024). *The Model Context Protocol (MCP): An open standard for connecting AI models to external tools and data sources*. https://modelcontextprotocol.io/

Argentina.gob.ar. (s. f.). *Ambiente incorpora drones en parques nacionales para la detección temprana de incendios forestales*. Ministerio de Ambiente y Desarrollo Sostenible. https://www.argentina.gob.ar/noticias/ambiente-incorpora-drones-en-parques-nacionales-para-la-deteccion-temprana-de-incendios

Assran, M., Bardes, A., Fan, D., Garrido, Q., Howes, R., Muckley, M., Rizvi, A., Roberts, C., Sinha, K., Zholus, A., … Ballas, N. (2025). *V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning*. arXiv:2506.09985. https://arxiv.org/abs/2506.09985

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., & Ballas, N. (2023). Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 15619-15629. arXiv:2301.08243. https://arxiv.org/abs/2301.08243

[Autores por confirmar]. (2026). A UAV–satellite hybrid pipeline for wildfire detection and dynamic perimeter prediction. *Drones*, *10*(4), 263. https://doi.org/10.3390/drones10040263

Badrloo, S., & Varshosaz, M. (2017). Monocular vision based obstacle detection. *Earth Observation and Geomatics Engineering*, *1*(2), 122-130. https://eoge.ut.ac.ir/article_64292.html

Balestriero, R., & LeCun, Y. (2025). *LeJEPA: Provable and Scalable Self-Supervised Learning Without the Heuristics*. arXiv:2511.08544. https://arxiv.org/abs/2511.08544

Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., & Xu, Y. (2011). *VP8 Data Format and Decoding Guide* (RFC 6386). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc6386

Barišić Kulas, A., Petric, F., & Bogdan, S. (2025). *Aerial maritime vessel detection and identification*. ICUAS 2025. arXiv:2507.07153. https://arxiv.org/abs/2507.07153

Bouhamed, O., Ghazzai, H., Besbes, H., & Massoud, Y. (2020). *Autonomous UAV Navigation: A DDPG-based Deep Reinforcement Learning Approach*. *CoRR*, *abs/2003.10923*. https://arxiv.org/abs/2003.10923

Breaking Defense. (2025, marzo). *Trained on classified battlefield data, AI multiplies effectiveness of Ukraine's drones: Report*. https://breakingdefense.com/2025/03/trained-on-classified-battlefield-data-ai-multiplies-effectiveness-of-ukraines-drones-report/

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, *2*(1), 14-23. https://doi.org/10.1109/JRA.1986.1087032

Buenos Aires Ciudad. (2023, 30 de enero). *Así funciona el escuadrón de drones de la Ciudad que ayuda a apagar incendios, perseguir a ladrones y encontrar personas perdidas*. Gobierno de la Ciudad Autónoma de Buenos Aires. https://buenosaires.gob.ar/noticias/asi-funciona-el-escuadron-de-drones-de-la-ciudad-que-ayuda-apagar-incendios-perseguir

Castelli, T., Sharghi, A., Harper, D., Trémeau, A., & Shah, M. (2016). Autonomous navigation for low-altitude UAVs in urban areas. *CoRR*, *abs/1602.08141*. http://arxiv.org/abs/1602.08141

Castro, W., Marcato Junior, J., Polidoro, C., Osco, L. P., Gonçalves, W., Rodrigues, L., Santos, M., Jank, L., Barrios, S., Valle, C., Simeão, R., Carromeu, C., Silveira, E., Jorge, L. A. de C., & Matsubara, E. (2020). Deep Learning Applied to Phenotyping of Biomass in Forages with UAV-Based RGB Imagery. *Sensors*, *20*(17). https://doi.org/10.3390/s20174802

Center for Strategic and International Studies (CSIS). (2024). *Ukraine's future vision and current capabilities for waging AI-enabled autonomous warfare*. https://www.csis.org/analysis/ukraines-future-vision-and-current-capabilities-waging-ai-enabled-autonomous-warfare

Chahe, A., Cai, S., & Zhou, L. (2026). *Latent World Models with Monotone Planning Costs for Image-Goal Navigation*. arXiv:2608.09073. https://arxiv.org/abs/2608.09073

Chahine, M., Hasani, R., Kao, P., Ray, A., Shubert, R., Lechner, M., Amini, A., & Rus, D. (2023). Robust flight navigation out of distribution with liquid neural networks. *Science Robotics*, *8*(77). https://doi.org/10.1126/scirobotics.adc8892

Chen, W., Shang, G., Hu, K., Zhou, C., Wang, X., Fang, G., & Ji, A. (2022). A Monocular-Visual SLAM System with Semantic and Optical-Flow Fusion for Indoor Dynamic Environments. *Micromachines*, *13*(11). https://doi.org/10.3390/mi13112006

Chen, X., Kundu, K., Zhang, Z., Ma, H., Fidler, S., & Urtasun, R. (2016). Monocular 3D Object Detection for Autonomous Driving. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2147-2156. https://openaccess.thecvf.com/content_cvpr_2016/html/Chen_Monocular_3D_Object_CVPR_2016_paper.html

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, *114*(3), 494-509. https://doi.org/10.1037/0033-2909.114.3.494

Codevilla, F., Santana, E., López, A. M., & Gaidon, A. (2019). Exploring the Limitations of Behavior Cloning for Autonomous Driving. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9328-9337. arXiv:1904.08980. https://arxiv.org/abs/1904.08980

Colas, C., Sigaud, O., & Oudeyer, P.-Y. (2018). *How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments*. arXiv:1806.08295. https://arxiv.org/abs/1806.08295

Coptrz. (2026). *How will drones disrupt urban last-mile delivery in 2026?* https://coptrz.com/blog/how-will-drones-disrupt-urban-last-mile-delivery-in-2026/

Danish, S., Piran, M. J., Khan, S. U., Khan, M. A., Dang, L. M., Zweiri, Y., Song, H.-K., & Moon, H. (2025). Vision-based fire management system using autonomous unmanned aerial vehicles: a comprehensive survey. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-025-11415-3

Dataintelo. (2025). *Civil drone market research report 2034*. https://dataintelo.com/report/global-civil-drone-market

Dataportuaria. (2026, marzo). *Especialistas anticipan una rápida expansión del uso de drones en la agricultura argentina*. https://dataportuaria.com/nota/25144/especialistas-anticipan-una-rapida-expansion-del-uso-de-drones-en-la-agricultura-argentina/

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2024). QLoRA: Efficient Finetuning of Quantized LLMs. *Advances in Neural Information Processing Systems (NeurIPS)*, *36*. https://arxiv.org/abs/2305.14314

Dickmanns, E. D. (2024). Evolution of the "4-D Approach" to Dynamic Vision for Vehicles. *Electronics*, *13*(20), 4133. https://doi.org/10.3390/electronics13204133

Dong, Y., Ruan, C. F., Cai, Y., Lai, R., Xu, Z., Zhao, Y., & Chen, T. (2024). *XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models*. arXiv:2411.15100. https://arxiv.org/abs/2411.15100

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, PMLR 78, 1-16. https://proceedings.mlr.press/v78/dosovitskiy17a.html

El Estratégico. (2026, julio 22). *Proponen crear un sistema nacional de vigilancia marítima con drones*. https://www.elestrategico.com/2026/07/22/proponen-crear-un-sistema-nacional-de-vigilancia-maritima-con-drones

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

Frantar, E., & Alistarh, D. (2023). SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *Proceedings of the 40th International Conference on Machine Learning (ICML)*, PMLR 202, 10323-10337. arXiv:2301.00774. https://arxiv.org/abs/2301.00774

Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2022). *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. arXiv:2210.17323. https://arxiv.org/abs/2210.17323

Gat, E. (1998). On three-layer architectures. En D. Kortenkamp, R. P. Bonasso, & R. Murphy (Eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems* (pp. 195-210). AAAI Press / MIT Press. https://u.cs.biu.ac.il/~kaminkg/teach/current/intsys/readings/on-three-layer-arch-tla-1998.pdf

Geng, S., Cooper, H., Moskal, M., Jenkins, S., Berman, J., Ranchin, N., West, R., Horvitz, E., & Nori, H. (2025). *JSONSchemaBench: A Rigorous Benchmark of Structured Outputs for Language Models*. https://arxiv.org/abs/2501.10868

Gerganov, G. (2023). *llama.cpp: Inference of LLaMA model in pure C/C++ and GGUF specification* [Software]. GitHub. https://github.com/ggerganov/llama.cpp

Goel, A., Tung, C., Hu, X., Thiruvathukal, G. K., Davis, J. C., & Lu, Y.-H. (2021). Efficient Computer Vision on Edge Devices with Pipeline-Parallel Hierarchical Neural Networks. *CoRR*, *abs/2109.13356*. https://arxiv.org/abs/2109.13356

Guan, T., Liu, F., Wu, X., Xian, R., Li, Z., Liu, X., Wang, X., Chen, L., Huang, F., Yacoob, Y., Manocha, D., & Zhou, T. (2024). HallusionBench: An Advanced Diagnostic Suite for Entangled Language Hallucination and Visual Illusion in Large Vision-Language Models. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14375-14385. https://doi.org/10.1109/CVPR52733.2024.01363

Ha, D., & Schmidhuber, J. (2018). Recurrent World Models Facilitate Policy Evolution. *Advances in Neural Information Processing Systems (NeurIPS)*, *31*. arXiv:1803.10122. https://arxiv.org/abs/1803.10122

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to Control: Learning Behaviors by Latent Imagination. *International Conference on Learning Representations (ICLR)*. arXiv:1912.01603. https://arxiv.org/abs/1912.01603

He, H., y colaboradores de Thinking Machines Lab. (2025, 10 de septiembre). *Defeating Nondeterminism in LLM Inference*. Thinking Machines Lab. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). arXiv:1709.06560. https://arxiv.org/abs/1709.06560

Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531. https://arxiv.org/abs/1503.02531

Hoang, V. D., Nyboe, F. F., Malle, N. H., & Ebeid, E. (2024). *Autonomous Overhead Powerline Recharging for Uninterrupted Drone Operations*. https://arxiv.org/abs/2403.06533

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2022). LoRA: Low-Rank Adaptation of Large Language Models. *International Conference on Learning Representations (ICLR)*. arXiv:2106.09685. https://arxiv.org/abs/2106.09685

Hu, J., Ren, Z., & Cheng, W. (2024). *Finite State Machines-Based Path-Following Collaborative Computing Strategy for Emergency UAV Swarms*. https://arxiv.org/abs/2407.11531

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Brown, N., Jackson, T., Luu, L., Levine, S., Hausman, K., & Ichter, B. (2022). Inner Monologue: Embodied Reasoning through Planning with Language Models. *Proceedings of the 6th Conference on Robot Learning (CoRL)*, PMLR 205. https://arxiv.org/abs/2207.05608

Hughes, M., & Engelbrecht, J. (2023). Autonomous navigation for multiple unmanned aerial vehicles (UAVs) in urban environments. *MATEC Web of Conferences*, *388*. https://doi.org/10.1051/matecconf/202338804011

Hwang, J.-J., Xu, R., Lin, H., Hung, W.-C., Ji, J., Choi, K., Huang, D., He, T., Covington, P., Sapp, B., Zhou, Y., Guo, J., Anguelov, D., & Tan, M. (2024). *EMMA: End-to-End Multimodal Model for Autonomous Driving*. https://arxiv.org/abs/2410.23262

Infobae. (2026, mayo 22). *Argentina acordó con Estados Unidos la incorporación de drones y tecnología militar para vigilar el Mar Argentino*. https://www.infobae.com/politica/2026/05/22/argentina-acordo-con-estados-unidos-la-incorporacion-de-drones-y-tecnologia-militar-para-vigilar-el-mar-argentino/

Infocampo. (2025, 8 de abril). *El uso de drones para pulverización agrícola: los pros, los contras y un gran pendiente*. https://www.infocampo.com.ar/el-uso-de-drones-para-pulverizacion-agricola-los-pros-los-contras-y-un-gran-pendiente/

INTA. (2025, diciembre). *Red de Drones del INTA: estrategia, datos y sinergias para el agro argentino*. Argentina.gob.ar. https://www.argentina.gob.ar/noticias/red-de-drones-del-inta-estrategia-datos-y-sinergias-para-el-agro-argentino

iProfesional. (2025). *La Ciudad abre la puerta al delivery con drones y crea un nuevo negocio para las empresas*. https://www.iprofesional.com/negocios/463009-la-ciudad-abre-la-puerta-al-delivery-con-drones-y-crea-un-nuevo-negocio-para-las-empresas

Jansen, W., Verreycken, E., Schenck, A., Blanquart, J.-E., Verhulst, C., Huebel, N., & Steckel, J. (2023). *Cosys-AirSim: A Real-Time Simulation Framework Expanded for Complex Industrial Applications*. https://arxiv.org/abs/2303.13381

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, *55*(12), Article 248. https://doi.org/10.1145/3571730

Kaneko, N., Yoshida, T., & Sumi, K. (2017). Fast Obstacle Detection for Monocular Autonomous Mobile Robots. *SICE Journal of Control, Measurement, and System Integration*, *10*(5), 370-377. https://doi.org/10.9746/jcmsi.10.370

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., & Scaramuzza, D. (2023). Champion-level drone racing using deep reinforcement learning. *Nature*, *620*(7976), 982–987. https://doi.org/10.1038/s41586-023-06419-4

Kroeger, T., Timofte, R., Dai, D., & Van Gool, L. (2016). Fast Optical Flow Using Dense Inverse Search. En B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016*, LNCS 9908 (pp. 471-488). Springer. https://doi.org/10.1007/978-3-319-46493-0_29

Kruskal, W. H., & Wallis, W. A. (1952). Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, *47*(260), 583-621. https://doi.org/10.1080/01621459.1952.10483441

LangChain. (2024). *LangGraph: Build resilient language agents as graphs* [Software]. GitHub. https://github.com/langchain-ai/langgraph

LeCun, Y. (2022). *A Path Towards Autonomous Machine Intelligence* (versión 0.9.2). OpenReview. https://openreview.net/forum?id=BZ5a1r-kVsf

Lee, D. N. (1976). A theory of visual control of braking based on information about time-to-collision. *Perception*, *5*(4), 437-459. https://doi.org/10.1068/p050437

Lesy, B., Herremans, S., Kerstens, R., Steckel, J., Daems, W., Mercelis, S., & Anwar, A. (2026). ASVSim (AirSim for Surface Vehicles): A High-Fidelity Simulation Framework for Autonomous Surface Vehicle Research. *IEEE Access*, *14*, 63803-63820. arXiv:2506.22174. https://arxiv.org/abs/2506.22174

Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast Inference from Transformers via Speculative Decoding. *Proceedings of the 40th International Conference on Machine Learning (ICML)*, PMLR 202, 19274-19286. https://arxiv.org/abs/2211.17192

Li, J., Zhou, H., Mai, Y., Jia, Y., Zhou, Z., Wu, K., Chen, H., Lin, H., Luo, M., & Shi, L. (2025). An autonomous obstacle avoidance and path planning method for fruit-picking UAV in orchard environments. *Smart Agricultural Technology*, Article 100752. https://doi.org/10.1016/j.atech.2024.100752

Li, Y., Yuan, W., Zhang, S., Yan, W., Shen, Q., Wang, C., & Yang, M. (2023). Choose Your Simulator Wisely: A Review on Open-source Simulators for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*. arXiv:2311.11056. https://doi.org/10.1109/TIV.2024.3374044

Liu, C., & Szirányi, T. (2023). *Active wildfires detection and dynamic escape routes planning for humans through information fusion between drones and satellites*. arXiv:2312.03519. https://arxiv.org/abs/2312.03519

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., & Chen, M.-H. (2024). DoRA: Weight-Decomposed Low-Rank Adaptation. *Forty-first International Conference on Machine Learning (ICML)*. arXiv:2402.09353. https://arxiv.org/abs/2402.09353

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., Lai, L., & Chandra, V. (2024). MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. *Proceedings of the 41st International Conference on Machine Learning (ICML)*, PMLR 235. arXiv:2402.14905. https://arxiv.org/abs/2402.14905

Longuet-Higgins, H. C., & Prazdny, K. (1980). The interpretation of a moving retinal image. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, *208*(1173), 385-397. https://doi.org/10.1098/rspb.1980.0057

Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., & Scaramuzza, D. (2021). Learning high-speed flight in the wild. *Science Robotics*, *6*(59), eabg5810. https://doi.org/10.1126/scirobotics.abg5810

Lundberg, S., Ribeiro, M. T., et al. (2023). *Guidance: A guidance language for controlling large language models* [Software]. GitHub. https://github.com/guidance-ai/guidance

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2018). Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, *61*, 523-562. arXiv:1709.06009. https://doi.org/10.1613/jair.5699

Madaan, R., Gyde, N., Vemprala, S., Brown, M., Nagami, K., Taubner, T., Cristofalo, E., Scaramuzza, D., Schwager, M., & Kapoor, A. (2020). AirSim Drone Racing Lab. *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, PMLR 123, 177-191. arXiv:2003.05654. https://arxiv.org/abs/2003.05654

Maes, L., Le Lidec, Q., Scieur, D., LeCun, Y., & Balestriero, R. (2026). *LeWorldModel: Stable End-to-End Joint-Embedding Predictive Architecture from Pixels*. arXiv:2603.19312. https://arxiv.org/abs/2603.19312

Mangrulkar, S., Gugger, S., Debut, L., von Platen, P., Paul, S., & Bossan, B. (2022). *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods* [Software]. GitHub. https://github.com/huggingface/peft

Mann, H. B., & Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, *18*(1), 50-60. https://doi.org/10.1214/aoms/1177730491

Marazzato, R., & Sparavigna, A. C. (2015). Retinex filtering of foggy images: generation of a bulk set with selection and ranking. *CoRR*, *abs/1509.08715*. http://arxiv.org/abs/1509.08715

Modern War Institute at West Point. (s. f.). *Battlefield drones and the accelerating autonomous arms race in Ukraine*. https://mwi.westpoint.edu/battlefield-drones-and-the-accelerating-autonomous-arms-race-in-ukraine/

Molineros, J., Cheng, S. Y., Owechko, Y., Levi, D., & Zhang, W. (2012). Monocular Rear-View Obstacle Detection Using Residual Flow. *Computer Vision – ECCV 2012 Workshops and Demonstrations*, LNCS 7584. Springer. https://doi.org/10.1007/978-3-642-33868-7_50

Mur-Artal, R., Montiel, J. M. M., & Tardós, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *CoRR*, *abs/1502.00956*. http://arxiv.org/abs/1502.00956

Nguyen, C. V., Shen, X., Aponte, R., Xia, Y., Basu, S., Hu, Z., Chen, J., Parmar, M., Kunapuli, S., Barrow, J., Wu, J., Singh, A., Wang, Y., Gu, J., Dernoncourt, F., Ahmed, N. K., Lipka, N., Zhang, R., Chen, X., … Nguyen, T. H. (2024). *A Survey of Small Language Models*. arXiv:2410.20011. https://arxiv.org/abs/2410.20011

Norte Misionero. (2026). *Alerta temprana, drones y respuesta rápida: Misiones refuerza su esquema de prevención de incendios*. https://nortemisionero.com.ar/provinciales/alerta-temprana-drones-y-respuesta-rapida-misiones-refuerza-su-esquema-de-prevencion-de-incendios/

NVIDIA. (2025). *Jetson Nano Brings the Power of Modern AI to Edge Devices*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/

NVIDIA. (s. f.). *Jetson Modules: Jetson Orin Nano and Orin NX Series*. Recuperado en septiembre de 2026, de https://developer.nvidia.com/embedded/jetson-modules

Perfil. (2026, 8 de septiembre). *La vigilancia de Javier Milei y Jorge Macri: tecnología, millones y control del espacio público*. https://www.perfil.com/noticias/politica/la-vigilancia-de-javier-milei-y-jorge-macri-tecnologia-millones-y-control-del-espacio-publico.phtml

Presidencia de la Nación Argentina. (2024). *Decreto 663/2024 — Aviación No Tripulada*. https://www.argentina.gob.ar/normativa/nacional/decreto-663-2024-401986

Qwen Team. (2025). *Qwen2.5-VL Technical Report*. arXiv:2502.13923. https://arxiv.org/abs/2502.13923

Rao, P., Zhang, W., Balestriero, R., LeCun, Y., & Loianno, G. (2026). *SkyJEPA: Learning Long-Horizon World Models for Zero-Shot Sim-to-Real Control of Quadrotors*. arXiv:2606.23444. https://arxiv.org/abs/2606.23444

Raspanti, F., Ozcelebi, T., & Holenderski, M. (2025). Grammar-Constrained Decoding Makes Large Language Models Better Logical Parsers. *ACL 2025 Industry Track*, 485-499. https://doi.org/10.18653/v1/2025.acl-industry.34

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, *abs/1506.02640*. http://arxiv.org/abs/1506.02640

Resolución ANAC — Reglamento de Vehículos Aéreos No Tripulados, Pub. L. No. 880/2019. (2019). https://www.argentina.gob.ar/normativa/nacional/resoluci%C3%B3n-880-2019-333259

Rill, R.-A., & Faragó, K. B. (2021). Collision Avoidance Using Deep Learning-Based Monocular Vision. *SN Computer Science*, *2*, 375. https://doi.org/10.1007/s42979-021-00759-6

Romano, J., Kromrey, J. D., Coraggio, J., & Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? *Annual Meeting of the Florida Association of Institutional Research*, 1-33. https://www.researchgate.net/publication/237544991

Ross, S., Gordon, G., & Bagnell, D. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, PMLR 15, 627-635. https://proceedings.mlr.press/v15/ross11a.html

Samy, M., Amer, K., Shaker, M., & ElHelw, M. (2019). *Drone Path-Following in GPS-Denied Environments using Convolutional Networks*. https://arxiv.org/abs/1905.01658

Saxena, P., Raghuvanshi, N., & Goveas, N. (2025). *UAV-VLN: End-to-End Vision Language guided Navigation for UAVs*. https://arxiv.org/abs/2504.21432

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. http://arxiv.org/abs/1705.05065

Shi, S., Ni, J., Kong, X., Zhu, H., Zhan, J., Sun, Q., & Xu, Y. (2024). An Obstacle Detection Method Based on Longitudinal Active Vision. *Sensors*, *24*(13), 4407. https://doi.org/10.3390/s24134407

Sobal, V., Zhang, W., Cho, K., Balestriero, R., Rudner, T. G. J., & LeCun, Y. (2025). *Learning from Reward-Free Offline Data: A Case for Planning with Latent Dynamics Models*. arXiv:2502.14819. https://arxiv.org/abs/2502.14819

Sun, X., Si, W., Ni, W., Li, Y., Wu, D., Xie, F., Guan, R., Xu, H.-Y., Ding, H., & Wu, Y. (2026). *AutoFly: Vision-Language-Action model for UAV autonomous navigation in the wild*. arXiv:2602.09657. https://arxiv.org/abs/2602.09657

Tian, Y., Lin, F., Li, Y., Zhang, T., Zhang, Q., Fu, X., Huang, J., Dai, X., Wang, Y., Tian, C., Li, B., Lv, Y., Kovács, L., & Wang, F.-Y. (2025). UAVs meet LLMs: Overviews and perspectives toward agentic low-altitude mobility. *Information Fusion*. arXiv:2501.02341. https://arxiv.org/abs/2501.02341

Turco, L., Zhao, J., Xu, Y., & Tsourdos, A. (2024). A Study on Co-simulation Digital Twin with MATLAB and AirSim for Future Advanced Air Mobility. *2024 IEEE Aerospace Conference*, 1-18. https://doi.org/10.1109/AERO58975.2024.10521333

Ugare, S., Suresh, T., Kang, H., Misailovic, S., & Singh, G. (2024). *SynCode: LLM Generation with Grammar Augmentation*. arXiv:2403.01632. https://arxiv.org/abs/2403.01632

Vemprala, S., Bonatti, R., Bucker, A., & Kapoor, A. (2023). *ChatGPT for robotics: Design principles and model abilities*. arXiv:2306.17582. https://arxiv.org/abs/2306.17582

Vera-Yanez, D., Pereira, A., Rodrigues, N., Molina, J. P., García, A. S., & Fernández-Caballero, A. (2024). Optical Flow-Based Obstacle Detection for Mid-Air Collision Avoidance. *Sensors*, *24*(10), 3016. https://doi.org/10.3390/s24103016

Wahba, G. (1965). A Least Squares Estimate of Satellite Attitude. *SIAM Review*, *7*(3), 409. https://doi.org/10.1137/1007077

Wang, W., Li, Y., Jiao, L., & Yuan, J. (2025). Large Language Model-Driven Closed-Loop UAV Operation with Semantic Observations. *IEEE Internet of Things Journal*. https://doi.org/10.1109/JIOT.2025.3649376

Wei, P., Ragbir, P., Vougioukas, S. G., & Kong, Z. (2025). Vision-based Navigation of Unmanned Aerial Vehicles in Orchards: An Imitation Learning Approach. *Computers and Electronics in Agriculture*, Article 110802. arXiv:2508.02617. https://doi.org/10.1016/j.compag.2025.110802

Willard, B. T., & Louf, R. (2023). *Efficient Guided Generation for Large Language Models*. arXiv:2307.09702. https://arxiv.org/abs/2307.09702

Xue, Z., & Gonsalves, T. (2021). Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning. *AI*, *2*(3), 366-380. https://doi.org/10.3390/ai2030023

Yang, L., Kang, B., Huang, Z., Zhao, Z., Xu, X., Feng, J., & Zhao, H. (2024). Depth Anything V2. *Advances in Neural Information Processing Systems (NeurIPS)*, *37*. arXiv:2406.09414. https://arxiv.org/abs/2406.09414. Código y modelos: https://github.com/DepthAnything/Depth-Anything-V2

Youden, W. J. (1950). Index for rating diagnostic tests. *Cancer*, *3*(1), 32-35. https://doi.org/10.1002/1097-0142(1950)3:1%3C32::AID-CNCR2820030106%3E3.0.CO;2-3

Zhang, G., Liu, J., Luo, W., Zhao, Y., Tang, R., Mei, K., & Wang, P. (2024). A Shortest Distance Priority UAV Path Planning Algorithm for Precision Agriculture. *Sensors*, *24*(23), 7514. https://doi.org/10.3390/s24237514

Zhang, P., Zeng, G., Wang, T., & Lu, W. (2024). *TinyLlama: An Open-Source Small Language Model*. arXiv:2401.02385. https://arxiv.org/abs/2401.02385

Zhou, G., Pan, H., LeCun, Y., & Pinto, L. (2025). DINO-WM: World Models on Pre-trained Visual Features Enable Zero-shot Planning. *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. arXiv:2411.04983. https://arxiv.org/abs/2411.04983

Zhou, Z., Liao, Y., Wang, B., Wang, M., Fu, M., & Hu, Z. (2026). Near obstacles detection by inverse perspective mapping of AVM for intelligent vehicles. *PLOS ONE*, *21*(1), e0336851. https://doi.org/10.1371/journal.pone.0336851

Zhu, Y., Moniz, J. R. A., Bhargava, S., Lu, J., Piraviperumal, D., Li, S., Zhang, Y., Yu, H., & Tseng, B.-H. (2024). *Can Large Language Models Understand Context?* https://arxiv.org/abs/2402.00858

Anexos

Anexo 1: Exploración técnica de Small Language Models (SLM), cuantización GGUF y análisis de innovación arquitectónica

La integración de modelos de lenguaje en plataformas robóticas autónomas exige resolver un compromiso riguroso entre capacidad representacional, latencia de inferencia y consumo de recursos computacionales. En el contexto de esta investigación, el objetivo es dotar a un vehículo aéreo no tripulado (UAV) de capacidad deliberativa de alto nivel mientras opera en simulación fotorrealista bajo hardware de consumo.

Un **Modelo de Lenguaje Pequeño** (*Small Language Model*, SLM) —y su extensión multimodal, *Small Vision-Language Model* (sVLM)— se define operativamente por tres criterios: **(1)** un conteo de parámetros en el rango de 1B a 10B (Abdin et al., 2024; Zhang et al., 2024), que conserva competencia notable para razonamiento simbólico, comprensión sintáctica y extracción de relaciones complejas a pesar de su escala relativa (Nguyen et al., 2024); **(2)** la factibilidad de despliegue local (*on-device*) en hardware de consumo común (tarjetas gráficas de gama media como la NVIDIA RTX 5060 de 8 GB) o procesadores embebidos (NVIDIA Jetson; NVIDIA, 2025); y **(3)** una latencia de inferencia determinista de subsegundos a pocos segundos, compatible con los márgenes de los lazos de control en robótica móvil (Tian et al., 2025; Vemprala et al., 2023). Frente a los LLMs de frontera (GPT-4o, Claude 3.5, Gemini 1.5 — decenas a cientos de miles de millones de parámetros, 24–80 GB de VRAM), los SLMs ofrecen ventajas determinantes para sistemas embarcados: autonomía operativa sin conexión a la nube, supresión de la variabilidad de red, eficiencia energética en plataformas alimentadas por batería (Goel et al., 2021) y compatibilidad con decodificación restringida por gramáticas en motores locales como `llama.cpp` (Willard & Louf, 2023).

Este anexo documenta el estudio de selección del modelo: delimita las restricciones físicas de memoria de video (VRAM), evalúa de forma comparativa los modelos compactos candidatos en formato GGUF, analiza los mecanismos para garantizar salidas estructuradas deterministas, y examina críticamente el nivel de novedad de la arquitectura frente al estado del arte reciente (2024–2026), identificando los ejes de ingeniería que aportan valor genuino al control autónomo.

A1.1 Restricciones de hardware y presupuesto de VRAM en cohabitación con Unreal Engine

La plataforma de experimentación adoptada consta de una estación de trabajo equipada con una GPU NVIDIA GeForce RTX 5060 provista de **8 GB de VRAM**. A diferencia de los entornos de investigación que disponen de aceleradores dedicados (e.g., NVIDIA A100/H100 de 40–80 GB) o que delegan el cómputo a servicios en la nube, este proyecto impone una restricción de cohabitación local estricta: la GPU debe ejecutar simultáneamente el entorno de simulación física y visual, y el motor de inferencia del modelo deliberativo.

Una instancia activa del simulador AirSim sobre Unreal Engine 5.5 demanda, en condiciones estándar de renderizado (geometría Nanite, iluminación Lumen y captura de múltiples cámaras virtuales), entre **4.5 y 6.0 GB de VRAM** (Shah et al., 2017; Turco et al., 2024). En consecuencia:

- **Presupuesto remanente disponible**: el espacio de memoria de video reservable para el modelo de lenguaje se restringe a una ventana de **2.0 a 3.5 GB de VRAM**.
- **Sobrecarga de contexto (KV Cache)**: la inferencia no solo consume la memoria estática de los pesos de la red, sino también la memoria dinámica requerida por el caché de claves y valores (*Key-Value Cache*), la cual escala linealmente con la longitud de la ventana de contexto y el tamaño del lote.
- **Exclusión de arquitecturas de frontera**: modelos superiores a 13B parámetros o modelos fundacionales no cuantizados (FP16/BF16) son inviables en esta configuración sin incurrir en transferencias masivas hacia la memoria RAM del sistema (*paging / offloading* excesivo a CPU), lo cual degrada la latencia de inferencia en órdenes de magnitud incompatibles con el control de vuelo.

Bajo este marco, la única alternativa viable radica en el uso de *Small Language Models* (SLMs) y *Small Vision-Language Models* (sVLMs) de escala **1.5B a 8B parámetros**, sometidos a técnicas de cuantización post-entrenamiento (*Post-Training Quantization*, PTQ) y optimizados para ejecución local (Abdin et al., 2024; Frantar et al., 2022; Nguyen et al., 2024; Zhang et al., 2024).

A1.2 Evaluación comparativa de modelos candidatos en formato GGUF (3B–8B)

El formato binario **GGUF** (*GPT-Generated Unified Format*), desarrollado por la comunidad de `llama.cpp` (Gerganov, 2023), se consolidó como el estándar de facto para la serialización y ejecución eficiente de modelos cuantizados en CPU y GPU. GGUF resuelve limitaciones de extensibilidad de formatos predecesores (como GGML) al incorporar metadatos estructurados en el encabezado del archivo y soportar esquemas de cuantización no uniforme (*k-quants*).

Las variantes *Q4_K_M* (cuantización a 4 bits con ponderación mixta en bloques de atención) y *Q5_K_M* (5 bits) permiten comprimir modelos de 3B–7B parámetros a tamaños de archivo de entre 1.8 y 4.5 GB, preservando la perplejidad y capacidad de razonamiento de los modelos originales con pérdidas marginales (Frantar et al., 2022).

Se evaluaron los siguientes modelos instruccionales (*instruct-tuned*) en formato GGUF para su ejecución bajo motores locales como `llama.cpp` y LM Studio:

Modelo	Parámetros	Tamaño GGUF (quant)	VRAM estimada (full offload)	Fortalezas operativas	Comportamiento con salida estructurada	Disponibilidad de referencia
Phi-4-mini-Instruct	3.8B	~2.5–3.2 GB (Q4_K_M)	~2.2–3.0 GB	Ajuste fino sobre datos sintéticos de alta fidelidad; sólida capacidad de razonamiento lógico y matemático (Abdin et al., 2024).	Muy alta adherencia a esquemas rígidos; mínima variación sintáctica.	`microsoft/Phi-4-mini-instruct-GGUF`
Qwen3-4B-Instruct	~4B	~2.8–3.5 GB (Q4_K_M)	~2.5–3.5 GB	Excelente seguimiento de instrucciones complejas; capacidad nativa de *tool-calling*.	Soporte consistente de esquemas JSON mediante prompting y gramáticas.	`Qwen/Qwen3-4B-Instruct-GGUF`
SmolLM3-3B-Instruct	3B	~2.0–2.6 GB (Q4_K_M)	~1.8–2.5 GB	Huella de memoria mínima; preentrenado específicamente para entornos embebidos y de bajo consumo.	Fácil de orientar a formatos fijos, aunque con menor tolerancia a contextos extensos.	`HuggingFaceTB/SmolLM3-3B-GGUF`
Gemma-3-4B-IT	4B	~2.8–3.4 GB (Q4_K_M)	~2.5–3.2 GB	Arquitectura optimizada por Google; notable coherencia lingüística y comprensión contextual.	Buen desempeño estructurado, pero mayor demanda de KV cache en secuencias largas.	`google/gemma-3-4b-it-GGUF`
Ministral-3-3B-Instruct	3B	~2.2–2.7 GB (Q4_K_M)	~2.0–2.6 GB	Diseñado específicamente por Mistral AI para despliegues de borde (*edge inference*); atención deslizante.	Rápido y estable en la emisión de respuestas compactas delimitadas.	`mistralai/Ministral-3-3B-Instruct-GGUF`

Por qué un modelo multimodal (§8.1.1 y §8.1.2)

Los modelos textuales analizados son adecuados para procesar telemetría pura, pero la percepción monocular tiene **puntos ciegos estructurales en la estimación de flujo óptico** (maniobras de sustentación estática *hover*, giros puros sobre el eje de guiñada y áreas de baja textura visual; véanse §6.1 y §6.12). Bajo esas condiciones, el resumen perceptivo numérico queda desprovisto de información confiable. Por eso el sistema usa un modelo con **capacidad de visión nativa** (*Small Vision-Language Model*, sVLM), que inspecciona directamente el fotograma de la cámara frontal o el mapa cenital.

Los dos sVLM candidatos que caben en el presupuesto son **Qwen2.5-VL-3B-Instruct** (Qwen Team, 2025; Q4_K_M, ~2.0 GB de VRAM) y **LFM2.5-VL-1.6B** (Q4_K_M, tag local de Ollama `lfm2.5-vl-1.6b:q4km-ctx8k`, ~1.6 GB de VRAM). El adoptado es LFM2.5-VL-1.6B, por latencia y por su mejor reconocimiento de parches del mapa (cap. 8, §8.1.2; cap. 10, §10.13.4). Ambos cumplen los requisitos: * **Huella de memoria** dentro del presupuesto operativo restante de la GPU. * **Inspección visual directa** del fotograma, para discernir la semántica de la escena (e.g., vegetación vs. obstáculo estructural rígido). * **Compatibilidad de decodificación**: soporte de decodificación restringida (`json_schema`) y *logprobs* en la misma consulta (en formato GGUF).

A1.3 Métodos para la imposición de salida estructurada y gramática restringida

En una arquitectura de control robótico en lazo cerrado, la generación de lenguaje en formato libre (*free-form text*) resulta inaceptable: respuestas prolijas, explicaciones accesorias o desviaciones sintácticas invalidan el analizador (*parser*), provocando la interrupción temporal o el congelamiento del lazo de control.

Para garantizar la interoperabilidad con los actuadores cinemáticos del vehículo, se evaluaron tres mecanismos para restringir la salida del modelo a estructuras estrictas (e.g., esquemas JSON conformes):

1. Instrucciones estrictas en el prompt del sistema (*Prompt Engineering*)

Consiste en delimitar el formato esperado en el mensaje de sistema y penalizar explícitamente cualquier emisión de texto explicativo o etiquetas markdown:

```
Eres un controlador de vuelo determinista. Responde ÚNICAMENTE un objeto JSON válido
conforme a este esquema: {"action": "keep_going" | "evasive" | "girar_90" | "fsm" |
"degraded", "reason": str}.
No incluyas introducciones, markdown (```), ni texto posterior.
```

* **Ventajas**: implementación trivial; no impone sobrecarga computacional en el motor de inferencia. * **Limitaciones**: fiabilidad imperfecta. En modelos compactos (<7B parámetros), la tasa de adherencia sintáctica ronda el 70–85%, presentándose ocasionalmente truncamientos, alucinaciones de sintaxis o texto preliminar que corrompen el flujo de control.

2. Gramáticas libres de contexto (GBNF en `llama.cpp`)

El motor `llama.cpp` proporciona un subsistema de decodificación guiada por gramáticas formales (*GGML BNF* o GBNF; Gerganov, 2023). En cada paso de generación autorregresiva, el motor filtra y enmascara los *logits* del vocabulario, asignando probabilidad cero ($-\infty$) a cualquier token que no cumpla con la regla de transición definida en la gramática. * **Ventajas**: garantiza matemáticamente que el 100% de los tokens emitidos respeten la estructura gramatical (e.g., sintaxis JSON perfecta). * **Limitaciones**: genera una penalización de rendimiento temporal del 10% al 30% en la velocidad de decodificación, debido a la evaluación continua de las reglas gramaticales sobre el conjunto de tokens candidatos.

3. Decodificación guiada basada en autómatas finitos (*Outlines*, *Guidance*, *llguidance*)

La formalización teórica introducida por Willard y Louf (2023) reformula el esquema JSON o expresión regular requerida como un autómata finito determinista (DFA). Mediante el preprocesamiento del vocabulario del tokenizador, el motor indexa de manera eficiente qué tokens son admisibles para cada estado del autómata, aplicando la máscara de *logits* con una sobrecarga computacional sensiblemente menor a la evaluación de gramáticas complejas.

Estudios empíricos recientes (Geng et al., 2025; Raspanti et al., 2025) demuestran que la decodificación restringida token a token no solo erradica errores de parseo (elevando la adherencia estructural del ~73% a >98%), sino que preserva e incluso optimiza la coherencia semántica en modelos pequeños, al suprimir caminos de generación dispersos o redundantes. Este enfoque constituye el núcleo del soporte de `response_format: {"type": "json_schema", ...}` implementado en el sistema (§8.2.1).

4. Estandarización de llamadas a herramientas: Model Context Protocol (MCP)

En arquitecturas modulares, la interacción entre el modelo deliberativo y las herramientas externas (sensores, registros de vuelo, interfaces con Unreal Engine) puede articularse mediante el **Model Context Protocol** (MCP; Anthropic, 2024). MCP define un protocolo de cliente-servidor en JSON-RPC que desacopla la lógica del modelo de los servicios del sistema anfitrión, permitiendo exponer primitivas de consulta y actuación de forma segura, determinista y local sin conexión a Internet.

A1.4 Procedimiento práctico de configuración e integración local

El modelo corre en la misma estación y la misma GPU que la simulación, porque un VLM a bordo comparte el cómputo con el resto del sistema de vuelo; esa es la condición que se toma como representativa y como línea base del banco de prueba (cap. 11, §11.0). La integración tiene tres partes:

1. **Servidor local en la GPU de la simulación**:

2. Ollama sirve LFM2.5-VL-1.6B (Q4_K_M, GGUF) sobre la misma RTX 5060 que Unreal Engine 5.5 y el grafo de control, con un contexto limitado a 8192 tokens: ~1.6 GB de VRAM, frente a 3.1 GB con el contexto por defecto (casi todo caché de atención). El script `startOllama.ps1` levanta el servidor (Anexo 6).
3. La red de profundidad del lazo lento (~440 MiB) comparte la misma GPU; con la configuración por defecto el VLM no corre durante el vuelo, solo en la planificación de ruta previa (cap. 4, §4.3.1).
4. **API HTTP compatible con OpenAI**:
5. El servidor expone `http://127.0.0.1:11434/v1` (variable `LOCAL_LLM_URL`; `127.0.0.1` y no `localhost`, que en Windows resuelve primero a IPv6 y demora cada conexión).
6. El cliente envía el fotograma (o los parches del mapa) codificado en base64, y pide la respuesta con `response_format` en modo `json_schema` y, en el planificador, los *logprobs* de la respuesta.
7. **Puente con el grafo de control**:
8. La respuesta estructurada se valida con el parser de su modo (`parse_strategic`, `parse_panorama_description`; §8.2.2) y el código la traduce a geometría: una sub-meta en coordenadas del mundo anclada a la pose del fotograma (cap. 5, §5.10.3) o, en el planificador, una puntuación por ruta candidata (cap. 4, §4.3.1). El modelo nunca elige una macro-acción.

A1.5 Análisis crítico de innovación y posicionamiento en el estado del arte (2024–2026)

Al plantear una arquitectura agéntica basada en el esquema:

$$\text{Telemetría sensorial} \longrightarrow \text{SLM / sVLM} \longrightarrow \text{Comando estructurado} \longrightarrow \text{Ejecución en AirSim}$$

surge la interrogante fundamental sobre su **novedad científica y tecnológica**.

Una revisión exhaustiva de la literatura especializada publicada entre 2024 y 2026 revela que **el lazo de control cerrado simple impulsado por modelos de lenguaje en simuladores como AirSim ya no constituye una innovación de frontera**, sino un patrón de diseño ampliamente estandarizado:

- **Generalización del lazo cerrado (*Closed-Loop Embodied Control*)**: desde los primeros experimentos de control robótico con modelos generativos (Huang et al., 2022; Vemprala et al., 2023), la comunidad ha convergido de forma unánime hacia ciclos de retroalimentación donde el modelo de lenguaje actúa como planificador táctico o generador de código en bucle continuo.
- **Adopción de AirSim y simuladores fotorrealistas**: el ecosistema AirSim / UE5 es la plataforma canónica para evaluar conducción autónoma y navegación de UAVs asistida por modelos fundacionales (Hwang et al., 2024; Jansen et al., 2023; Turco et al., 2024).
- **Navegación guiada por visión y lenguaje (VLN)**: propuestas como *UAV-VLN* (Saxena et al., 2025) y *AutoFly* (Sun et al., 2026) implementan modelos de acción visual (*Vision-Language-Action*, VLA)

donde la interacción directa entre imagen y control reactivo está extensamente documentada.

- **Generación de código y observación semántica**: trabajos contemporáneos como el de Wang et al. (2025) (*Large Language Model-Driven Closed-Loop UAV Operation with Semantic Observations*) abordan de forma explícita la conversión de telemetría numérica en descripciones semánticas para enriquecer la toma de decisiones en lazo cerrado dentro de entornos de simulación, utilizando módulos duales de generación y evaluación.

Evaluación crítica

En términos de originalidad puramente teórica o arquitectónica, la arquitectura en lazo cerrado simple se sitúa en un nivel de madurez consolidado (patrón canónico de ingeniería). Intentar justificar la novedad de una tesis únicamente en el hecho de conectar un modelo de lenguaje a un simulador de vuelo representaría un posicionamiento metodológicamente vulnerable.

Por el contrario, **el verdadero aporte de ingeniería de esta tesis radica en la resolución de los cuellos de botella prácticos que la literatura suele ignorar o delegar a servidores masivos en la nube**: 1. La integración sinérgica entre **percepción monocular por flujo óptico pasivo** (capítulo 6) y un modelo visual compacto local (capítulo 8), delimitando cuándo se activa el razonamiento deliberativo y cuándo se recurre al control reactivo. 2. La caracterización sistemática de los **modos de fallo en tiempo real** bajo presupuestos severos de cómputo y memoria (capítulo 9). 3. La garantía de determinismo y fiabilidad operativa mediante **decodificación restringida por esquemas gramaticales** y arquitecturas jerárquicas de seguridad.

A1.6 Vías de diferenciación e innovación de ingeniería en entornos severamente restringidos

Para que un sistema de navegación para cuadricópteros basado en SLMs aporte valor genuino y diferenciado en hardware de recursos limitados, es imperativo trascender el bucle elemental de control. A continuación se sintetizan las estrategias de software y procesamiento identificadas en esta exploración, contrastando su viabilidad y beneficio:

Estrategia arquitectónica	Nivel de diferenciación técnica	Impacto en VRAM y latencia (8 GB compartidos)	Fundamento y justificación operativa	Estado de integración en la tesis
Compresión semántica de observaciones	Alto	Bajo (+0.1–0.3 GB en KV cache)	La telemetría numérica cruda resulta opaca para modelos compactos; la transformación en resúmenes semánticos estructurados maximiza el razonamiento del SLM (Wang et al., 2025).	**Integrado** (módulo `ObstacleField` y prompts estructurados; §6.12, §8.5).

Estrategia arquitectónica	Nivel de diferenciación técnica	Impacto en VRAM y latencia (8 GB compartidos)	Fundamento y justificación operativa	Estado de integración en la tesis
Control híbrido deliberativo-reactivo con FSM	Alto	Muy bajo (despreciable en GPU)	El SLM no gestiona la cinemática de alta frecuencia; emite macro-acciones discretas cada 0.5–2 s, mientras una FSM determinista y controladores de bajo nivel aseguran estabilidad dinámica y *fail-safe* (Gat, 1998).	**Integrado como pilar central** (capítulos 5 y 8; desacoplamiento FSM / Deliberativo).
Decodificación restringida formal (`json_schema`)	Alto	Nulo en VRAM; sobrecarga mínima de latencia	Sustituye el parseo heurístico por restricciones a nivel de logits; convierte el espacio discreto de macro-acciones en una gramática ejecutable (Geng et al., 2025; Raspanti et al., 2025; Willard & Louf, 2023).	**Integrado** (adherencia elevada del 73% al 98%; §8.2.1).
Autocorrección reflexiva en una sola pasada (*Single-Pass Self-Refinement*)	Muy alto	Medio (demanda secuencias de salida más largas)	El modelo emite la acción propuesta, un índice de confianza y una alternativa de contingencia en un único flujo de inferencia, sin requerir un segundo llamado a la GPU.	**Conceptualizado** en el espacio de acción ampliado; factible como extensión futura.
Decodificación especulativa (*Speculative Decoding*)	Alto	Bajo–Medio (requiere modelo borrador o cabeza auxiliar)	Acelera la generación autorregresiva 1.5× a 2.5× mediante verificación paralela de tokens (Leviathan et al., 2023), mitigando la latencia en maniobras evasivas críticas.	**Línea de trabajo futuro** condicionada por el soporte de cabezas draft en backends de borde.

Conclusión sintética

La exploración realizada confirma que la combinación de **compresión perceptiva estructurada**, **decodificación restringida por gramáticas** y **arquitectura híbrida deliberativa-reactiva** permite ejecutar un agente de navegación visual robusto, completamente desconectado de la nube y contenido en 8 GB de VRAM compartidos con Unreal Engine 5.5. Esta solución traslada el foco desde la especulación de modelos masivos hacia la ingeniería rigurosa de sistemas autónomos embebidos.

Anexo 2: Técnicas de optimización, compresión y mitigación de desafíos en Small Language Models (SLM)

La viabilidad de los modelos de lenguaje pequeños (*Small Language Models*, SLMs) y sus variantes multimodales de visión (*Small Vision-Language Models*, sVLMs) en plataformas robóticas autónomas reside en su capacidad para aproximar la competencia de razonamiento de los grandes modelos fundacionales bajo presupuestos severamente restringidos de cómputo, memoria y energía (Abdin et al., 2024; Nguyen et al., 2024).

Este anexo sistematiza los principios teóricos y metodológicos de optimización que hacen posible este equilibrio: examina los tres pilares de compresión (destilación, poda y cuantización), revisa los avances arquitectónicos en eficiencia y adaptación en borde, analiza críticamente la propensión a alucinaciones en modelos compactos, y detalla los mecanismos de ingeniería adoptados en esta tesis para mitigar dichos riesgos en el lazo de navegación de un vehículo aéreo no tripulado (UAV).

A2.1 Paradigmas fundamentales de compresión de modelos

La reducción del orden de magnitud en el conteo de parámetros sin degradar de manera catastrófica la precisión y capacidad de generalización se articula mediante tres técnicas complementarias:

A2.1.1 Destilación de conocimiento (*Knowledge Distillation*)

La destilación de conocimiento consiste en transferir las distribuciones de probabilidad y las representaciones latentes aprendidas por un modelo «maestro» (*teacher*) de gran escala hacia una red «estudiante» (*student*) más compacta (Hinton et al., 2015).

- **Supervisión densa**: a diferencia del entrenamiento supervisado tradicional basado únicamente en etiquetas duras (*one-hot*), el modelo estudiante aprende a replicar las probabilidades suaves (*soft targets*) del maestro, capturando correlaciones cruzadas implícitas y matices semánticos que no están presentes en los datos de entrada brutos.
- **Alineación de razonamiento**: modificaciones en la función de pérdida de destilación permiten transferir cadenas de inferencia paso a paso (*Chain-of-Thought*, CoT), facultando a modelos sub-4B para resolver problemas de lógica simbólica, descomposición de metas y razonamiento espacial con fidelidad comparable a redes sustancialmente mayores.
- **Destilación multi-maestro**: arquitecturas especializadas integran ensambles de múltiples modelos maestros (e.g., un maestro de visión y otro de razonamiento textual) para destilar un único modelo compacto unificado.

A2.1.2 Poda de parámetros (*Pruning*)

La poda consiste en eliminar sistemáticamente ponderaciones redundantes o conexiones poco informativas dentro de las matrices de peso de la red, reduciendo el conteo de operaciones y la huella de memoria:

- **Poda no estructurada (*Unstructured Pruning*)**: suprime ponderaciones individuales cuyos valores absolutos o gradientes caen por debajo de un umbral específico, generando matrices dispersas (*sparse*). Algoritmos de segunda derivada como SparseGPT (Frantar & Alistarh, 2023) permiten podar modelos masivos en una sola pasada (*one-shot*) sin necesidad de reentrenamiento extensivo. No obstante, aprovechar la aceleración teórica de la poda no estructurada exige hardware especializado con soporte nativo de aceleración dispersa (e.g., patrones $2:4$ o $n:m$).
- **Poda estructurada (*Structured Pruning*)**: retira bloques completos de la arquitectura, tales como neuronas completas, cabezas de atención (*attention heads*) o capas Transformer completas. Aunque puede inducir una degradación más abrupta si la tasa de remoción es excesiva, preserva matrices densas compatibles de forma nativa con cualquier biblioteca de álgebra lineal y GPU estándar.

A2.1.3 Cuantización (*Quantization*)

La cuantización sustituye las representaciones numéricas de alta precisión (típicamente punto flotante de 16 o 32 bits, FP16/FP32) por representaciones discretas enteras de bajo número de bits (INT8 o INT4), reduciendo drásticamente la memoria requerida para almacenar los pesos y acelerando el rendimiento computacional de las unidades de cálculo matricial (Frantar et al., 2022):

- **Cuantización post-entrenamiento (*Post-Training Quantization*, PTQ)**: metodologías como GPTQ (Frantar et al., 2022) y AWQ (*Activation-aware Weight Quantization*) formulan la compresión como un problema de optimización cuadrática por capas, cuantizando los pesos mientras compensan el error de reconstrucción basándose en la información de segundo orden de la matriz Hessiana.
- **Cuantización del caché de claves y valores (KV Cache Quantization)**: en tareas de inferencia de secuencias continuas o históricas, la KV cache constituye la principal causa de crecimiento dinámico de VRAM. Reducir la precisión de estos tensores a INT8 o INT4 permite sostener ventanas de contexto prolongadas sin desbordar la memoria de video disponible.
- **Tratamiento de valores atípicos (*Outliers*)**: esquemas como SmoothQuant mitigan la dificultad inherente a la cuantización de activaciones migrando la escala dinámica entre pesos y activaciones antes de la cuantización, asegurando estabilidad numérica en modelos autoregresivos.

A2.2 Arquitecturas compactas y optimización para inferencia en borde

Más allá de la compresión directa de redes preexistentes, el diseño arquitectónico nativo de modelos compactos ha introducido mejoras estructurales orientadas a dispositivos con recursos limitados (*edge computing*):

A2.2.1 Diseños ligeros y optimizaciones de atención

Modelos como MobileLLM (Liu et al., 2024), TinyLlama (Zhang et al., 2024) y la familia Phi (Abdin et al., 2024) demuestran que optimizar la profundidad frente al ancho de la red, compartir matrices de incrustación (*embedding sharing*) y adoptar mecanismos eficientes de atención permite que modelos de escala 1B a 4B alcancen competencias operativas antes restringidas a modelos de decenas de miles de millones de parámetros.

Asimismo, la sustitución o combinación de la autoatención cuadrática convencional $\mathcal{O}(N^2)$ por mecanismos de atención lineal, ventanas deslizantes (*sliding window attention*) o formulaciones basadas en modelos de espacio de estados (SSM) alivia sustancialmente la carga computacional en inferencias continuas.

A2.2.2 Ajuste fino eficiente en parámetros (PEFT, LoRA y QLoRA)

El reentrenamiento completo (*full fine-tuning*) de un modelo exige almacenar en memoria los estados del optimizador, activaciones intermedias y gradientes para la totalidad de los parámetros, lo cual supera con creces la memoria disponible en GPUs de consumo.

Las técnicas de ajuste fino eficiente en parámetros (*Parameter-Efficient Fine-Tuning*, PEFT), y en particular la adaptación de bajo rango (**LoRA**; Hu et al., 2022), resuelven esta barrera: * Congelan las ponderaciones preentrenadas $W_0 \in \mathbb{R}^{d \times k}$ y modelan la actualización mediante la descomposición en dos matrices de bajo rango $B \times A$, donde $r \ll \min(d, k)$: $W = W_0 + \Delta W = W_0 + B \cdot A$ * Reducen el conteo de parámetros entrenables en más de un 99%, permitiendo especializar modelos en tareas de navegación o seguimiento de formatos en pocas horas de GPU. * Su variante cuantizada, **QLoRA** (Dettmers et al., 2024), mantiene el modelo base en precisión de 4 bits (NormalFloat4) e inserta adaptadores LoRA entrenables en precisión de 16 bits, posibilitando la especialización local sin incurrir en consumo masivo de VRAM.

A2.2.3 Transferencia y destilación de razonamiento

Los modelos compactos entrenados sobre datos sintéticos curados de alta calidad pedagógica (como libros de texto sintéticos y trazas lógicas depuradas) exhiben una capacidad desproporcionada para el seguimiento de instrucciones y la descomposición algorítmica de problemas (Abdin et al., 2024). Esto valida que la capacidad de razonamiento operativo de un modelo en un lazo de control no está determinada exclusivamente por el volumen bruto de parámetros, sino por la densidad informativa y estructurada del corpus con el que fue optimizado.

A2.3 El desafío de las alucinaciones en modelos compactos y visión-lenguaje

El fenómeno de la **alucinación** —definido como la generación de información fáctica o contextualmente incorrecta, no fundamentada en la entrada sensorial ni en la evidencia provista (Ji et al., 2023)— adquiere una gravedad particular en sistemas de robótica móvil e interacción física con el entorno.

A2.3.1 Correlación entre escala de parámetros y propensión alucinatoria

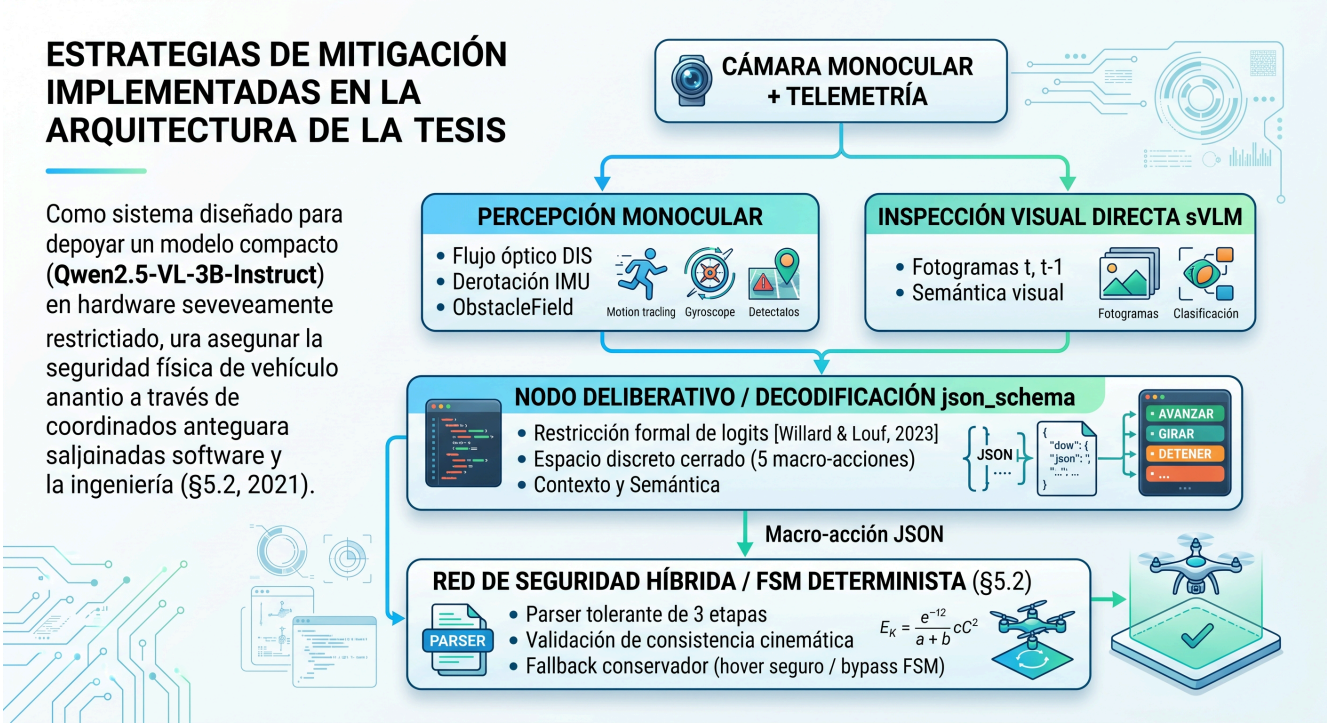
La literatura experimental documenta que los modelos de menor tamaño presentan una propensión basal más alta a generar contenido alucinatorio en comparación con modelos de frontera: * Evaluaciones sistemáticas en modelos multimodales de visión-lenguaje mediante bancos de prueba especializados como **HallusionBench** (Guan et al., 2024) revelan que la reducción en la capacidad de la red acentúa las ilusiones visuales y los sesgos lingüísticos no fundamentados en la imagen (*language hallucination vs. visual illusion*). * Modelos pequeños sometidos a estímulos ambiguos o degradados tienden a completar secuencias basándose en prioris estadísticas del lenguaje más que en la evidencia visual real.

A2.3.2 Riesgos críticos en vehículos aéreos no tripulados (UAVs)

En una misión de navegación aérea autónoma, los efectos de una alucinación trascienden la incorrección lingüística y se traducen en amenazas cinemáticas directas: 1. **Alucinación de espacio libre**: el modelo reporta erróneamente que una trayectoria está despejada al confundir sombras, reflejos o texturas complejas con áreas transitables, suprimiendo maniobras de evasión esenciales. 2. **Alucinación de sintaxis o formato**: la emisión de etiquetas no contempladas o la deformación del esquema JSON invalida el analizador sintáctico, interrumpiendo el flujo de control y dejando al vehículo sin acción defensiva durante múltiples ciclos de muestreo. 3. **Deriva temporal y desacoplamiento dinámico**: el modelo puede basar su razonamiento en estados superados o inventar una persistencia de obstáculos inexistente, comprometiendo la coherencia de la misión.

A2.4 Estrategias de mitigación implementadas en la arquitectura de la tesis

Para hacer viable el despliegue de un modelo compacto (LFM2.5-VL-1.6B, cap. 8 §8.1.2) bajo hardware severamente restringido sin comprometer la seguridad física del vehículo, esta tesis implementa un conjunto coordinado de salvaguardas de software e ingeniería:



1. **Decodificación restringida a nivel de logits (`json_schema`)**: En lugar de confiar en el seguimiento heurístico del prompt, el motor de inferencia enmascara en tiempo de generación cualquier token que viole la gramática del esquema JSON (Geng et al., 2025; Raspanti et al., 2025; Willard & Louf, 2023). Esto garantiza matemáticamente un 100% de validez estructural, erradicando por completo las alucinaciones de formato (§8.2.1).
2. **El modelo describe; el código decide la geometría (§8.3)**: El modelo no genera velocidades ni elige macro-acciones: clasifica sectores, imágenes o parches del mapa en enumeraciones cerradas (`free` / `blocked`, cinco clases de terreno), y el código traduce esa descripción a una sub-meta en coordenadas del mundo o a una puntuación de ruta. Esta discretización suprime alucinaciones numéricas continuas y deja toda la cinemática en una función de traducción determinista y acotada (`action_to_command()`, §8.4).
3. **Arquitectura de dos lazos con un lazo rápido determinista (§5.1.2 y §8.2.2)**: La capa deliberativa no ejerce control directo sobre los actuadores. El lazo rápido de flujo óptico y TTC es el único que comanda velocidades y reacciona a lo inminente en el mismo ciclo; una respuesta inválida, tardía u obsoleta se descarta sin sustituirse por una decisión inventada, y un barrido sin respuesta válida cae a un escape vertical determinista.

4. **Medición contra etiquetas de referencia antes del vuelo (§10.13)**: Cada pregunta que se le hace al modelo se evalúa fuera de vuelo contra una etiqueta independiente; la que no supera el criterio no se lleva al lazo. Por eso, con la configuración por defecto, las preguntas del lazo lento las responde una red de profundidad estimada y el VLM queda en el plan de ruta, donde sí aporta señal.

Conclusión sintética

Los desafíos de capacidad limitada y susceptibilidad a alucinaciones propios de los SLMs no deben resolverse mediante el escalado indiscriminado de parámetros —inviable en hardware de borde—, sino a través de una **estricta disciplina de ingeniería de sistemas**: decodificación gramaticalmente guiada, discretización del espacio de respuesta, acoplamiento jerárquico con un lazo rápido determinista y medición de cada capacidad antes de confiarle una decisión.

Anexo 3: Optimización y especialización de modelos de visión-lenguaje mediante LoRA/QLoRA para navegación aérea autónoma

El modelo multimodal del sistema es **LFM2.5-VL-1.6B** (tag `lfm2.5-vl-1.6b:q4km-ctx8k`), ejecutado localmente bajo cuantización Q4_K_M en un entorno con recursos de cómputo y memoria de video acotados (§8.1); el otro candidato evaluado es **Qwen2.5-VL-3B-Instruct** (Qwen Team, 2025). Las cifras de memoria de este anexo se calcularon para el modelo de 3 B y son una cota superior para el de 1.6 B. Con la configuración por defecto el VLM no decide en vuelo, porque en el banco de prueba no distingue sectores libres de bloqueados (cap. 8; cap. 11, §11.0): una adaptación LoRA sería la vía para que lo haga, si el banco muestra que el modelo adaptado distingue sectores libres de bloqueados en un entorno que no vio durante el entrenamiento. Sin adaptar, el modelo opera en régimen *zero-shot / in-context learning*, guiado por prompts breves (§8.5) y acoplado a un motor de decodificación gramaticalmente restringida por esquema JSON (§8.2, Anexo 4; Willard & Louf, 2023).

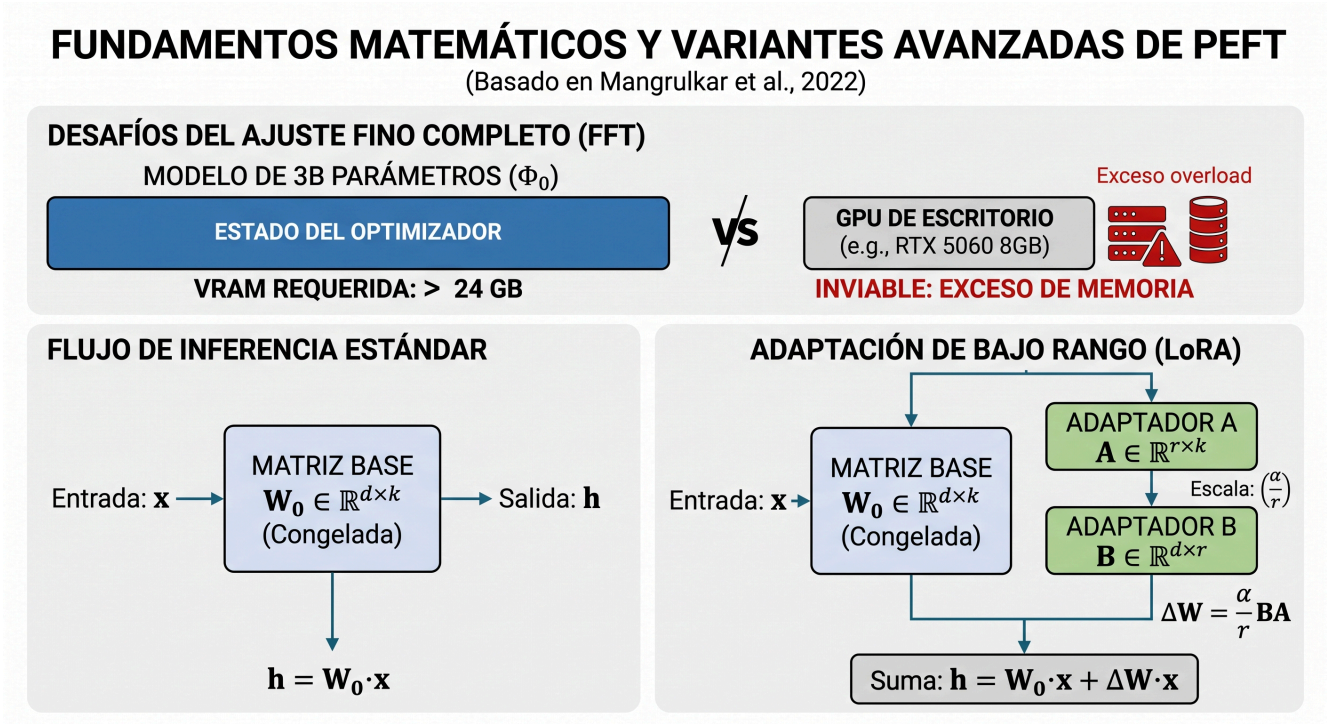
Esta configuración tiene dos limitaciones operativas para el vuelo reactivo: 1. **Capacidad de discriminación:** el modelo generalista no separa sectores volables de bloqueados en la imagen de a bordo (AUC 0.53 en el banco) y responde con sesgos por posición en la grilla (cap. 9, §9.8.10). 2. **Latencia:** ~0.8 s por consulta con imagen en la GPU compartida con la simulación (cap. 8, §8.6), cuatro ciclos del lazo, frente a ~0.1 s de la red de profundidad.

Este anexo desarrolla los fundamentos matemáticos y metodológicos de **Adaptación de Bajo Rango (LoRA; Hu et al., 2022)** y sus extensiones contemporáneas (**QLoRA; Dettmers et al., 2024** y **DoRA; Liu et al., 2024**) como línea de trabajo futuro (§12.4): internalizar el dominio físico-espacial y el formato estricto directamente en los tensores de ponderación de la red, habilitando respuestas en tiempo real (< 500 ms) sin alterar la arquitectura base ni requerir hardware de nivel servidor.

A3.1 Fundamentos matemáticos y variantes avanzadas de PEFT

El ajuste fino completo (*Full Fine-Tuning*, FFT) de una red neuronal profunda requiere actualizar y almacenar en memoria los gradientes y momentos del optimizador (e.g., AdamW en precisión simple FP32) para la totalidad de los parámetros entrenables Φ_0. Para un modelo de 3B parámetros, esto demanda más de 24 GB de VRAM únicamente para el estado del optimizador, tornando inviable el entrenamiento en GPUs de escritorio (tales como la NVIDIA RTX 5060 de 8 GB utilizada en este proyecto).

Las técnicas de Ajuste Fino Eficiente en Parámetros (*Parameter-Efficient Fine-Tuning*, PEFT; Mangrulkar et al., 2022) resuelven este cuello de botella congelando la red preentrenada y acoplando un conjunto microscópico de parámetros adaptativos.



A3.1.1 Formulación matemática de LoRA

La premisa matemática de LoRA (Hu et al., 2022) se basa en la hipótesis de que las actualizaciones de peso ΔW durante la adaptación a una tarea especializada exhiben un «rango intrínseco» muy bajo (*low intrinsic dimension*). Para una matriz lineal preentrenada $W_0 \in \mathbb{R}^{d\times k}$, la actualización se descompone como el producto de dos matrices de bajo rango:

$$W = W_0 + \Delta W = W_0 + \frac{\alpha}{r}(B \cdot A)$$

donde: * $r \ll \min(d, k)$ es el hiperparámetro de **rango intrínseco** (típicamente $r \in [8, 32]$). * $B \in \mathbb{R}^{d\times r}$ se inicializa estrictamente en ceros ($B = 0$). * $A \in \mathbb{R}^{r\times k}$ se inicializa con una distribución normal gaussiana aleatoria $A \sim \mathcal{N}(0, \sigma^2)$, con $\sigma = \frac{1}{r}$. * $\alpha \in \mathbb{R}^+$ es una constante de escala que atenúa o amplifica el impacto de la adaptación sin requerir reajustar la tasa de aprendizaje (*learning rate*) al modificar r.

Propiedad de preservación funcional: gracias a la inicialización $B = 0$, en el paso inicial de entrenamiento se cumple que $\Delta W = 0 \cdot A = 0$. Por lo tanto, $W = W_0$ y el comportamiento inicial del modelo es matemáticamente idéntico al modelo preentrenado de fábrica, evitando saltos destructivos en los pesos durante las primeras iteraciones.

A3.1.2 Selección de módulos objetivo en arquitecturas multimodales

En modelos basados en Transformer, las proyecciones lineales se dividen entre el bloque de autoatención (*Multi-Head Attention*) y el perceptrón multicapa (*Feed-Forward Network* / MLP): * **Atención:** matrices de consulta (W_q), clave (W_k), valor (W_v) y proyección de salida (W_o). * **MLP:** matrices de compuerta (W_{gate}), subida (W_{up}) y bajada (W_{down}).

Para modelos multimodales (sVLMs como Qwen2.5-VL), existe una disyuntiva de diseño crítica: 1. **Codificador visual (*Vision Transformer*, ViT):** procesa los parches de píxeles brutos de la cámara monocular. Adaptar el ViT con LoRA introduce el riesgo de destruir filtros de detección de bordes y texturas universales ante conjuntos de datos de vuelo acotados. Por ello, la práctica recomendada consiste en **mantener el ViT congelado**. 2. **Proyector multimodal y capas del LLM autorregresivo:** adaptar W_q, W_v y las capas lineales del MLP permite al modelo remapear las representaciones visuales hacia la lógica de control del dron. La literatura empírica demuestra que incorporar tanto los módulos de atención como los del bloque MLP ($W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}$) con rangos moderados ($r = 16$) supera consistentemente a adaptar únicamente W_q con rangos mayores ($r = 64$).

A3.1.3 QLoRA: cuantización en 4 bits y optimizadores paginados

Para posibilitar la sintonización en hardware de desarrollo restringido, Dettmers et al. (2024) introdujeron **QLoRA**, una extensión que combina tres innovaciones clave: * **Formato NormalFloat4 (NF4):** un tipo de datos no lineal de 4 bits con espaciado óptimo para representar tensores con distribución normal estándar centrada en cero, conservando un error de cuantización teóricamente menor que el formato entero uniforme INT4. * **Doble Cuantización (*Double Quantization*):** proceso que cuantiza las propias constantes de escala de cuantización del modelo (de FP32 a FP8 en bloques de 256 elementos), ahorrando un promedio de 0.37 bits por parámetro (~370 MB en un modelo de 3B). * **Optimizadores Paginados (*Paged Optimizers*):** integración con CUDA Memory Management para trasladar dinámicamente los estados de memoria de la GPU a la memoria RAM del sistema operativo durante picos transitorios de gradientes, previniendo fallos por memoria agotada (*Out-Of-Memory*, OOM).

En este esquema, el modelo base W_0 reside en VRAM cuantizado en 4 bits NF4 ($1.8\ \text{GB}$ para 3B), mientras que los adaptadores A y B se mantienen y optimizan en precisión bfloat16 ($25\ \text{MB}$).

A3.1.4 DoRA: adaptación desacoplada en magnitud y dirección

Liu et al. (2024) observaron una discrepancia estructural en la dinámica de gradientes: mientras que el ajuste completo (FFT) modifica tanto la magnitud como la dirección de los vectores de pesos de forma desbalanceada e independiente, LoRA exhibe un acoplamiento simétrico entre ambas dimensiones.

Para solventar esta limitación, **DoRA (*Weight-Decomposed Low-Rank Adaptation*)** factoriza cada columna de la matriz de ponderación $W \in \mathbb{R}^{d \times k}$ en su vector de magnitud de norma euclidiana $m \in \mathbb{R}^{1 \times k}$ y su matriz direccional unitaria normalizada $V \in \mathbb{R}^{d \times k}$:

$$W = m \odot \frac{V}{\|V\|_c} = m \odot \frac{W_0 + \Delta W}{\|W_0 + \Delta W\|_c}$$

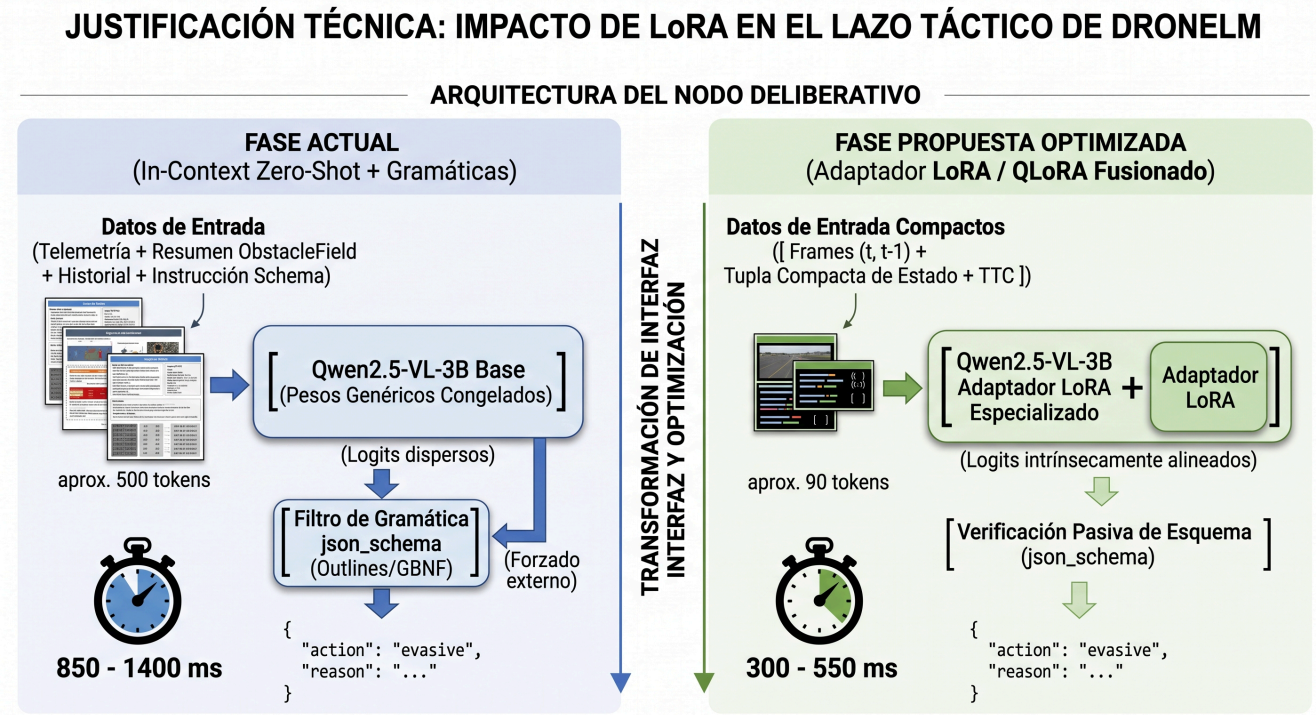
donde $\|\cdot\|_c$ denota la norma L_2 a lo largo de las columnas y $\Delta W = \frac{\alpha}{r}BA$. DoRA permite que el gradiente ajuste la escala global de activación (m) independientemente de las reorientaciones vectoriales (ΔW), logrando una capacidad de modelado que equipara prácticamente a FFT en tareas espaciales y de razonamiento lógico, manteniendo la ventaja de poder fusionarse íntegramente en inferencia sin penalización de velocidad.

A3.1.5 Comparativa de paradigmas de ajuste fino para sVLMs

Dimensión	Full Fine-Tuning (FFT)	LoRA estándar	QLoRA (NF4)	DoRA
VRAM entrenamiento (3B)	> 24 GB	~12–14 GB	**~5.5–7.0 GB**	~6.5–8.0 GB
Parámetros entrenables	100% (~3.090 M)	~0.15–0.30% (~5–10 M)	~0.15–0.30% (~5–10 M)	~0.20–0.35% (~6–11 M)
Precisión pesos base	FP16 / BF16	FP16 / BF16	**NF4 (4 bits)**	NF4 / FP16
Sobrecarga en inferencia	Cero (pesos nativos)	Cero (tras fusión)	Cero (tras fusión + quant)	Cero (tras fusión)
Fidelidad respecto a FFT	Referencia (100%)	92–95%	91–94%	**97–99%**
Viabilidad en GPU 8 GB	No viable	Crítico / Marginal	**Plenamente viable**	**Plenamente viable**

A3.2 Justificación técnica: impacto de LoRA en el lazo táctico de DroneLM

La integración de un adaptador LoRA en la consulta al VLM transforma la interfaz entre el modelo de lenguaje y el sistema ciberfísico:



A3.2.1 Reducción del presupuesto de tokens y compresión de latencia

En una implementación *zero-shot* (§8.5), el prompt debe reiterar en cada invocación: * La definición de la grilla de sectores y del criterio «libre a 15 m». * La sintaxis exacta y el orden de los campos del esquema JSON. * Los valores admisibles de cada campo.

Este contexto insume cientos de tokens de procesamiento previo (*prefill*). Al especializar el modelo con LoRA, **la gramática de salida y las reglas de dominio quedan cristalizadas en los tensores de ponderación**. El prompt de entrada puede simplificarse a una codificación tabular mínima:

```
[ESTADO: Vz=-0.1, Vh=3.2 | TTC: F=1.4s, L=inf, R=4.1s | HIST: keep_going(x3) | REASON:
TTC_CRITICO]
```

Esta reducción comprime la ventana de entrada de ~500 a menos de 100 tokens, acelerando el tiempo de respuesta del modelo en un 60%, con la inferencia por debajo del período de consulta de la capa estratégica.

A3.2.2 Alineación causal directa: de la imagen a la maniobra

Un modelo generalista tiende a verbalizar descripciones visuales pasivas (*«Veo un muro con sombras proyectadas»*). LoRA permite reentrenar la distribución de probabilidad condicional $P(y \mid X)$ para que el modelo efectúe una **asociación sensoriomotora directa**: * Reconocer cómo el patrón de divergencia óptica entre I_t e I_{t-1} anticipa colisiones inminentes. * Discriminar si una región oscura corresponde a una sombra sobre el pavimento (transitable) o a un obstáculo sólido en el volumen de vuelo (no transitable), suprimiendo alucinaciones visuales que el flujo monocular no logra desagregar (§6.1).

A3.3 Pipeline de ingeniería de datos para navegación aérea

El factor determinante del éxito de la adaptación no radica en el volumen de parámetros, sino en la calidad informativa y la diversidad topológica del conjunto de datos de entrenamiento.

A3.3.1 Estructura del dataset multimodal de navegación

Cada instancia de entrenamiento se define como un par supervisado (X_t, Y_t) estructurado de la siguiente forma:

$$X_t = \{I_t, I_{t-1}, \text{telemetry}_t, \text{obstacle\_field}_t, \text{reason\_note}_t\}$$

$$Y_t = \{\text{"action"} : a_t^*, \text{"reason"} : r_t^*\}$$

donde: * I_t, I_{t-1} son los dos fotogramas RGB monoculares recientes capturados a 640×360 píxeles. * telemetry_t incluye el vector de velocidad tridimensional, ángulo de cabeceo (*pitch*), alabeo (*roll*) y distancia al waypoint activo. * obstacle_field_t codifica el TTC estimado y la tasa de ocupación por sector angular (cap. 6 y 7). * $a_t^* \in \mathcal{A}_{\text{whitelist}}$ es la macro-acción supervisada óptima. * r_t^* es una cadena de justificación condensada (< 120 caracteres) que verbaliza el razonamiento causal subyacente (e.g., `"TTC crítico de 1.2s en cuadrante frontal-izquierdo; espacio despejado a la derecha; iniciar maniobra evasiva"`).

A3.3.2 Mitigación de la deriva cinemática mediante DAgger

El entrenamiento mediante simple Clonación de Comportamiento (*Behavioral Cloning*, BC) a partir de vuelos nominales perfectos sufre del problema de **deriva covariada (*Covariate Shift*)** (Codevilla et al., 2019): ante el menor error acumulado de predicción, el vehículo ingresa en estados cinemáticos no presentes en el conjunto de entrenamiento, colapsando en decisiones divergentes.

Para inmunizar al modelo frente a este fenómeno, se define la aplicación del algoritmo **DAgger (*Dataset Aggregation*; Ross et al., 2011)** adaptado a simulación en AirSim / Unreal Engine 5: 1. **Vuelo con política actual** π_{LoRA} en AirSim UE5.5 bajo perturbaciones controladas (ráfagas de viento sintéticas, desviaciones forzadas). 2. **Consulta al oráculo experto** (FSM determinista o supervisor experto) ante desvíos: determinación de la acción correctiva óptima a^*. 3. **Agregación de tuplas críticas** $\mathcal{D}_{\text{total}} = \mathcal{D}_{\text{previo}} \cup \{(X_{\text{perturbado}}, a^*_{\text{experto}})\}$. 4. **Reentrenamiento iterativo** de los adaptadores LoRA sobre el conjunto enriquecido.

A3.3.3 Aumentación sintética y métricas objetivo de evaluación

Para evitar el sobreajuste a texturas específicas de los mapas virtuales de Unreal Engine (Neighborhood, Manhattan; cap. 3 y 10), el conjunto de datos se somete a aumentaciones dinámicas: variación de la hora solar en UE5, degradación atmosférica sintética (niebla y dispersión lumínica; Marazzato & Sparavigna, 2015) y desenfoque por movimiento angular (*motion blur*).

Métrica de Validación	Meta Operativa (Zero-Shot Actual)	Meta Objetivo (LoRA / QLoRA)
Cumplimiento de Esquema JSON	99.8% (gracias a gramática forzada)	**100% nativo** (verificación pasiva)
Latencia media de inferencia	~1100 ms (límite de watchdog)	**< 450 ms** (margen amplio)
Tokens de entrada promedio	480–520 tokens	**< 100 tokens**
Tasa de resolución de atasco	60.0% (Tier 3, cap. 10)	**> 85.0%**
Tasa de colisiones evitables	20.0% en Tier 3	**< 5.0%**

A3.4 Fusión de pesos, despliegue y preservación de la seguridad híbrida

A3.4.1 Fusión algebraica de pesos (*Weight Merging*) y exportación

Una de las ventajas arquitectónicas centrales de LoRA frente a otros esquemas modulares radica en la posibilidad de **fusionar algebraicamente los adaptadores con los pesos base preentrenados** antes del despliegue en el vehículo:

$$W_{\text{despliegue}} = W_0 + \frac{\alpha}{r}(B \cdot A)$$

Al realizar la fusión de tensores en punto flotante previo (FP16 o BF16), la matriz resultante $W_{\text{despliegue}}$ adquiere exactamente las mismas dimensiones y topología que la red preentrenada original. No existe ninguna bifurcación de memoria ni sobrecarga computacional adicional durante la inferencia.

Posteriormente, el modelo fusionado se compila a formato **GGUF** cuantizado (`Q4_K_M`) mediante `llama.cpp` (Gerganov, 2023), produciendo un binario de ~2.0 GB listo para ejecución local embebida sin dependencias de librerías de entrenamiento.

A3.4.2 Sinergia entre LoRA y decodificación restringida (`json_schema`)

El ajuste fino no reemplaza la decodificación restringida por gramáticas (§8.2, Anexo 4), sino que la complementa: * **LoRA moldea la distribución interna de logits:** el modelo aprende a concentrar prácticamente el 100% de su masa de probabilidad en las palabras clave del esquema JSON y la lista blanca de macro-acciones. * `json_schema` **opera como barrera determinista externa:** enmascara el residuo infinitesimal de probabilidad en situaciones fuera de distribución, garantizando formalmente cero violaciones de formato. * **Aceleración de generación:** al fluir la decodificación sin colisionar con tokens suprimidos por la gramática, la tasa de muestreo alcanza el máximo rendimiento teórico del motor de inferencia.

A3.4.3 Salvaguardas inalteradas y hoja de ruta de transferencia

Un principio fundacional de DroneLM es que los modelos de lenguaje son aproximadores estocásticos y **bajo ninguna circunstancia deben gobernar de forma directa los actuadores físicos sin arbitraje determinista** (Gat, 1998; cap. 5 y 8). La incorporación de adaptadores LoRA preserva íntegramente las tres salvaguardas del sistema: 1. **Lazo rápido único que comanda velocidades:** la capa deliberativa solo propone sub-metas ancladas al mundo; la evasión por flujo óptico y TTC sigue actuando en el mismo ciclo ante cualquier riesgo inminente (cap. 5, §5.3.3). 2. **Parser por modo:** valida cada respuesta contra su esquema y rechaza la que no coincide, sin sustituirla por un valor por defecto (§8.2.2). 3. **Perro guardián temporal** (`SLM_DEEP_WATCHDOG_MS`, 18 s en el barrido; edad máxima de 10 s en la capa estratégica, cap. 8, §8.6): una respuesta tardía se descarta y el barrido cae al escape vertical.

Como línea de extensión formulada en el trabajo futuro (§12.4), esta especialización posibilitará la transferencia de la capa deliberativa a plataformas de cómputo en borde (NVIDIA Jetson Orin Nano / Xavier NX con enlaces serie MAVLink hacia autopilotos PX4), consolidando a DroneLM como un copiloto táctico local, seguro por diseño y plenamente autónomo en entornos reales.

Anexo 4: Decodificación restringida, gramáticas formales y generación estructurada para el control táctico del SLM

La incorporación de modelos de lenguaje pequeños (*Small Language Models*, SLMs) y modelos multimodales compactos (*Small Vision-Language Models*, sVLMs) en el lazo de control táctico de un vehículo aéreo no tripulado (UAV) introduce una paradoja arquitectónica fundamental: **la naturaleza probabilística y estocástica de los modelos neuronales frente a los requerimientos deterministas, de baja latencia y tolerancia cero a fallas de sintaxis propios de la robótica aérea**.

En DroneLM, el VLM juzga rutas candidatas sobre el mapa cenital antes del vuelo (§4.3.1) y, como fuente alternativa del lazo lento (§5.10), describe qué sectores del fotograma o qué rumbos de un barrido son volables. Si el modelo emite respuestas en texto libre, explicaciones conversacionales no solicitadas, bloques Markdown envolventes o JSONs con claves malformadas o ausentes, el analizador falla y la consulta se pierde: en vuelo, eso significa ciclos de control sin la información que se pidió (§8.2, §9.1).

Este anexo desarrolla los fundamentos teóricos, los mecanismos algorítmicos y el procedimiento exhaustivo de ingeniería para implementar **decodificación restringida (*constrained decoding*) basada en gramáticas formales y esquemas estrictos (`json_schema` / GBNF)**. Se demuestra cómo esta técnica erradica las alucinaciones estructurales, comprime el tiempo de inferencia hasta en un 60–80% y optimiza el grafo de control (`StateGraph`) de DroneLM mediante gramáticas acotadas dinámicas adaptadas al contexto de vuelo.

A4.1 El problema de la salida no estructurada en sistemas ciberfísicos

En la generación autorregresiva convencional, un modelo autoregresivo produce una secuencia de tokens muestreando de una distribución de probabilidad sobre la totalidad de su vocabulario V:

$$y_t \sim P(y_t \mid y_{<t}, x)$$

donde x representa la secuencia del prompt de entrada e $y_{<t} = (y_1, \ldots, y_{t-1})$ los tokens generados previamente. Para **Qwen2.5-VL-3B-Instruct** (Qwen Team, 2025), uno de los dos modelos candidatos (§8.1.2), el tamaño del vocabulario asciende a $|V| = 152\,064$ tokens; el modelo adoptado, LFM2.5-VL-1.6B, tiene un vocabulario menor, y el argumento no cambia.

A4.1.1 Modos de falla de la decodificación libre en DroneLM

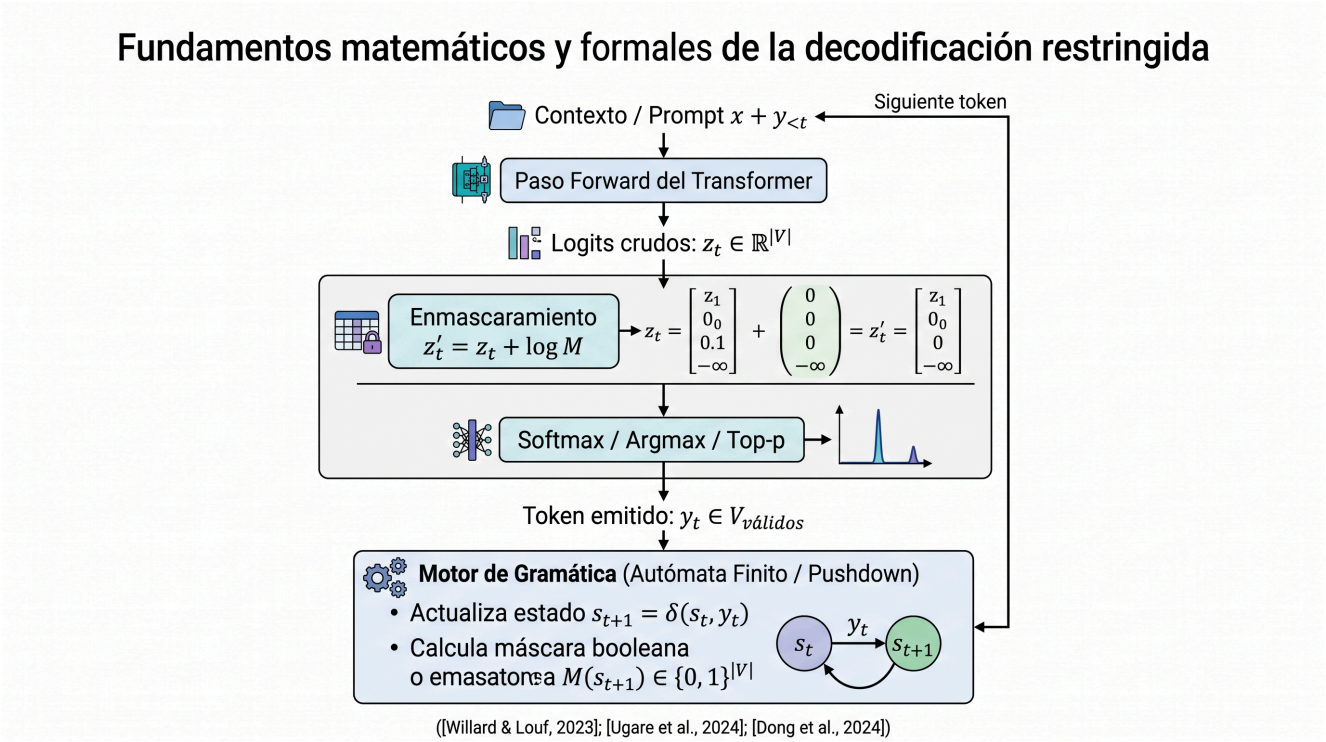
Cuando se solicita una salida JSON únicamente mediante instrucciones en el *system prompt* (*prompt engineering* puro), los modelos de 3B parámetros exhiben una tasa de incumplimiento sintáctico de entre el 20% y el 35% en condiciones de vuelo continuo (Geng et al., 2025; Raspanti et al., 2025). Los modos de falla de la decodificación libre incluyen: 1. **Preámbulos y epílogos conversacionales:** emisión de frases introductorias (*«Sure, here is the drone decision:»*) o explicaciones finales que rompen `json.loads()`. 2. **Encapsulamiento en Markdown irregular:** uso intermitente de triples comillas invertidas (`` ```json ... ``` ``), a menudo con saltos de línea mal posicionados. 3. **Malformación sintáctica:** comas terminales huérfanas (*trailing commas*), omisión de llaves de cierre `}` por corte de contexto o escape defectuoso de caracteres en cadenas. 4. **Alucinación de macro-acciones fuera de lista blanca:** emisión de identificadores plausibles en lenguaje natural pero inexistentes en el mapa de control cinemático (e.g., `"girar_rapido"`, `"subir_diagonal"`, `"stop"` en lugar de `keep_going`, `evasive`, `girar_90`, `fsm`, `degraded`). 5. **Latencia descontrolada por verbosidad:** generación de decenas de tokens de razonamiento no solicitados, consumiendo el presupuesto temporal del perro guardián (`SLM_DEEP_WATCHDOG_MS`, 18 s en el barrido, cap. 8, §8.6) y forzando la caída al escape de respaldo.

```
Generación Libre (Estocástica, ~120 tokens, ~2.1 s):
"Based on the camera feed showing an obstacle ahead, the drone should initiate an evasive
 maneuver to the right corridor. Action: evasive_right, reason: obstacle in front."
 ❌ Falla sintáctica: no es JSON parseable, rompe action_to_command(), timeout > 1.5 s.

Generación Restringida (Determinista, ~18 tokens, ~0.35 s):
{"action":"evasive","reason":"bloqueo centro"}
 ✓ Estructura exacta, enum validado por gramática, latencia mínima, 100% parseable.
```

A4.2 Fundamentos matemáticos y formales de la decodificación restringida

La decodificación restringida interviene directamente en la distribución de muestreo de la capa de salida (*logits*) del modelo durante cada paso de la generación autorregresiva, guiada por un autómata formal que modela la sintaxis admisible (Willard & Louf, 2023; Ugare et al., 2024; Dong et al., 2024).



A4.2.1 Formulación del enmascaramiento de logits

Sea $z_t \in \mathbb{R}^{|V|}$ el vector de *logits* no normalizados producido por la última capa lineal del transformer en el paso t. Sea $s_t \in \mathcal{S}$ el estado actual del analizador sintáctico o autómata formal.

La gramática define una función de transición de estados $\delta : \mathcal{S} \times V \to \mathcal{S} \cup \{\text{error}\}$ y una función de aceptación que determina qué tokens del vocabulario son continuaciones válidas:

$$M_i(s_t) = \begin{cases} 1 & \text{si } \delta(s_t, v_i) \neq \text{error} \\ 0 & \text{en caso contrario} \end{cases} \quad \forall v_i \in V$$

El vector de logits filtrado z'_t se computa mediante la adición de una máscara aditiva logarítmica:

$$z'_{t,i} = z_{t,i} + \log M_i(s_t) = \begin{cases} z_{t,i} & \text{si } M_i(s_t) = 1 \\ -\infty & \text{si } M_i(s_t) = 0 \end{cases}$$

Al aplicar la función Softmax para obtener las probabilidades de muestreo normalizadas:

$$P(y_t = v_i \mid y_{<t}, x) = \frac{\exp(z'_{t,i})}{\sum_{j=1}^{|V|} \exp(z'_{t,j})}$$

Cualquier token v_j que conduzca a una violación de la gramática recibe un logit de $-\infty$, anulando su probabilidad ($\exp(-\infty) = 0$). Como resultado, **es matemáticamente imposible que el modelo genere un token inválido según la gramática declarada**.

A4.2.2 Modelado mediante Autómatas Finitos (DFA) y Autómatas de Pila (PDA)

La complejidad del analizador depende de la expresividad de la restricción: 1. **Autómatas Finitos Deterministas (DFA):** aplicables a expresiones regulares, listas cerradas de cadenas y enumeraciones fijas (`enum`). Willard y Louf (2023) demuestran que un esquema JSON con campos de profundidad fija puede compilarse directamente en un DFA. En este caso, la transición entre estados se resuelve en tiempo constante $\mathcal{O}(1)$. 2. **Autómatas de Pila (PDA):** requeridos para gramáticas independientes del contexto (*Context-Free Grammars*, CFG) que admiten anidamientos arbitrarios de llaves, corchetes o estructuras recursivas (formato GBNF en `llama.cpp` y SynCode; Ugare et al., 2024). El PDA mantiene una pila de símbolos para validar el balanceo estricto de la sintaxis.

A4.2.3 El desafío de la tokenización sub-palabra (*Tokenization Boundary Problem*)

Un obstáculo fundamental en la decodificación restringida radica en que los modelos de lenguaje no operan sobre caracteres individuales, sino sobre fragmentos de sub-palabras generados mediante algoritmos como *Byte-Pair Encoding* (BPE) o *SentencePiece*.

Un token individual puede cruzar la frontera entre la sintaxis estática y el valor dinámico. Por ejemplo, el token `"action":` puede estar codificado como una única entrada léxica en el vocabulario, mientras que en otros tokenizadores se divide en `"` + `action` + `":`.

Los motores modernos (Outlines, XGrammar, llama.cpp) resuelven esto construyendo un árbol de prefijos (*Trie*) sobre el vocabulario de sub-palabras indexado contra el autómata de la gramática. Durante la compilación previa, se identifican todos los tokens cuya secuencia de bytes constituye un prefijo válido para el estado actual del autómata, garantizando que el filtrado sea exacto a nivel de byte (Willard & Louf, 2023; Dong et al., 2024).

A4.3 Mecanismos de aceleración: ¿por qué la decodificación restringida es más rápida?

Existe una concepción errónea de que evaluar una gramática formal en cada paso de inferencia añade una sobrecarga computacional prohibitiva para sistemas en tiempo real. Si bien una implementación ingenua que verifique $150\,000$ tokens en CPU de forma secuencial introduciría una latencia sustancial, los motores optimizados de última generación **pueden acelerar la generación global entre un 50% y un 80%** en comparación con la inferencia en texto libre (Dong et al., 2024; Lundberg et al., 2023).

Esta ganancia de eficiencia se sustenta en cuatro pilares de ingeniería:

A4.3.1 Poda drástica de la longitud de generación (*Output Length Reduction*)

La complejidad temporal de la generación autorregresiva escala linealmente con el número de tokens producidos K:

$$T_{\text{total}} = T_{\text{prefill}} + \sum_{k=1}^{K} T_{\text{decode}}(k)$$

En una GPU de consumo cohabitada con simulación 3D (como la NVIDIA RTX 5060 de 8 GB utilizada en DroneLM), cada paso de decodificación autorregresiva de un modelo 3B insume entre **18 ms y 25 ms**. * En texto libre, el modelo genera preámbulos, divagaciones o razonamientos extensos de 80 a 150 tokens ($T_{\text{decode}} \approx 1.5 - 3.2$ s), violando sistemáticamente el perro guardián del sistema. * Con decodificación restringida estricta, la salida se acota a exactamente la estructura mínima requerida (por ejemplo, 16 a 22 tokens en DroneLM). El tiempo de decodificación colapsa a $300 - 450$ ms, permitiendo al lazo deliberativo operar holgadamente dentro de su ventana temporal de seguridad.

A4.3.2 Avance rápido de tokens (*Token Fast-Forwarding / Token Jumping*)

Marcos de trabajo como **Guidance** (Lundberg et al., 2023) y **Outlines** (Willard & Louf, 2023) implementan una técnica crítica: **el salto de tokens deterministas**.

Cuando el autómata de la gramática se encuentra en un estado donde la única continuación sintáctica posible es una secuencia fija de caracteres (por ejemplo, los caracteres sintácticos `{"action":"` o la clave `, "reason":"`), **no existe incertidumbre probabilística**. El motor de inferencia omite por completo el paso hacia adelante (*forward pass*) de la red neuronal para esos tokens. En su lugar: 1. Concatena directamente los tokens estáticos en la secuencia generada. 2. Inyecta sus representaciones en la memoria caché de claves y valores (*KV Cache*). 3. Salta la ejecución del modelo en GPU hasta el punto exacto donde el modelo debe elegir un valor semántico libre (el valor de la macro-acción o el texto de la razón).

Al saltarse entre 8 y 12 pasos de cómputo en la GPU por cada llamada deliberativa, la latencia neta cae drásticamente.

A4.3.3 Partición de vocabulario y pre-cálculo de máscaras (XGrammar)

El motor **XGrammar** (Dong et al., 2024) optimiza la ejecución de gramáticas independientes del contexto dividiendo los tokens del vocabulario en dos categorías durante una fase de compilación inicial: * **Tokens independientes del contexto:** tokens cuyos patrones de aceptación dependen únicamente de reglas locales de caracteres y pueden precalcularse y almacenarse en mapas de bits compactos. * **Tokens dependientes del contexto:** aquellos que interactúan con la profundidad de la pila sintáctica (e.g., llaves y comillas anidadas).

Durante la inferencia en tiempo real, XGrammar ejecuta el filtrado de máscaras en paralelo con el cómputo de la GPU o durante la fase de pre-llenado (*pre-fill*), reduciendo la sobrecarga de CPU a **menos de 0.05 ms por token** (prácticamente nula).

A4.3.4 Decodificación especulativa basada en gramáticas

En sistemas avanzados, las restricciones gramaticales se combinan con **decodificación especulativa** (Leviathan et al., 2023; Dong et al., 2024). Dado que la gramática restringe drásticamente el abanico de tokens legales (en el campo `action`, solo 5 opciones son posibles), un modelo borrador ultraligero o un generador heurístico de gramática puede proponer múltiples tokens simultáneamente. El modelo principal en la GPU valida o rechaza el bloque completo en una única pasada paralela, logrando tasas de aceleración de $2\times$ a $3\times$.

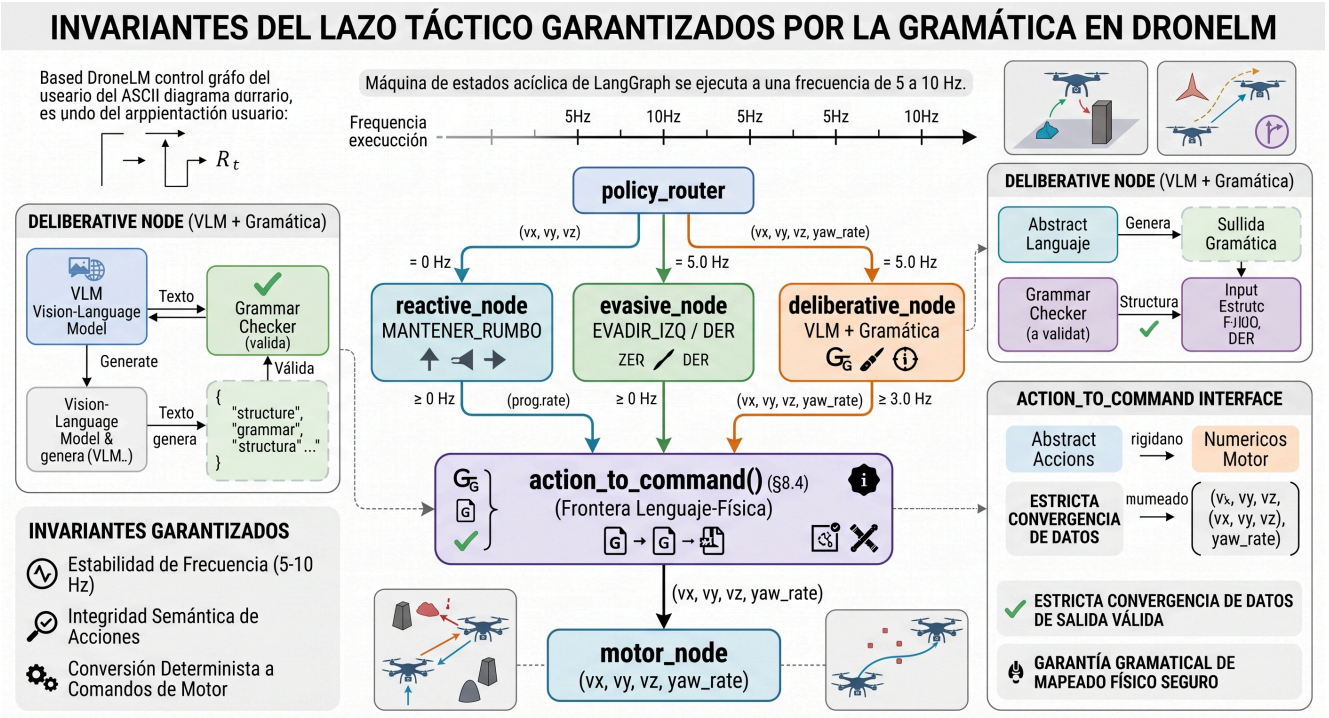
A4.4 Optimización del Grafo de Control (`StateGraph`) mediante Decodificación Restringida

> **Relación con el sistema.** En el sistema descrito en los capítulos 5 y 8, el VLM **no emite acciones**: la consulta estratégica y el barrido usan dos esquemas descriptivos (sectores de una grilla de 3×3, `libre`/`bloqueado`; tipo y transitabilidad por imagen), detallados en el capítulo 8, §8.2.1, y validados por `src/agents/vlm_client.py`. Los esquemas, gramáticas y fragmentos de código de esta sección y de §A4.5–§A4.6 ilustran la técnica sobre un **decisor de acciones** (`DroneTacticalDecision`), que es el caso en el que la poda gramatical tiene mayor efecto; no corresponden al código del sistema.

La interacción entre el modelo de lenguaje y el grafo de control táctico (`src/agents/graph.py`) no debe concebirse como un canal pasivo de consulta y respuesta, sino como una **arquitectura de co-diseño donde la gramática es un componente dinámico del propio grafo**.

A4.4.1 Invariantes del lazo táctico garantizados por la gramática

En el grafo de control de DroneLM, la máquina de estados acíclica de LangGraph se ejecuta a una frecuencia de 5 a 10 Hz. Los comportamientos de la capa de decisión (`reactive_node`, `evasive_node`, el giro de 90° y el barrido) convergen en una interfaz estricta gobernada por `action_to_command()` (§8.4); en el decisor de acciones ilustrativo de esta sección, también la respuesta del modelo:



Al imponer decodificación restringida en la consulta al VLM, se garantizan tres invariantes operativas fundamentales: 1. **Ausencia total de fallas de parseo en el lazo:** se elimina el riesgo de que una respuesta corrupta obligue a descartar el ciclo de control. La tasa de adherencia al esquema sube de un ~73% (con prompt libre y parser tolerante) a un **98%–100%**, como se verifica empíricamente en el informe (§8.2, §11). 2. **Determinismo en el espacio de acción:** el campo `action` se constriñe a la enumeración canónica: $action \in \{"keep\_going", "evasive", "girar\_90", "fsm", "degraded"\}$ Esto asegura que `action_to_command()` siempre recibe un identificador registrado, eliminando ramas de ejecución indefinidas en los actuadores del dron. 3. **Control estricto de la longitud del campo de justificación:** al restringir la longitud máxima de la cadena `reason` mediante reglas gramaticales (e.g., `maxLength: 120` o regex `[a-zA-Z0-9 _-]{1,100}`), se impide que el modelo expanda su generación más allá de los tokens estrictamente necesarios para auditoría y telemetría.

A4.4.2 Gramáticas acotadas dinámicas condicionadas por el estado del grafo

Una de las innovaciones de mayor impacto para optimizar el lazo deliberativo consiste en la **parametrización dinámica de la gramática en función del estado de vuelo actual**.

En lugar de emplear un esquema JSON estático para todas las consultas, la consulta al VLM selecciona en tiempo de compilación/ejecución una gramática especializada según el motivo de consulta (`reason_note`, §8.5):

Motivo de Consulta (`reason_note`)	Condición de Activación en Grafo	Gramática / Restricción Dinámica del Enum `action`	Racionalidad Operativa y Beneficio
`"TTC_CRITICO"`	$\mathrm{TTC} \leq 3.2$ s en sector central	`enum: ["evasive", "girar_90"]`	**Poda de acciones pasivas:** se prohíbe sintácticamente que el SLM elija `keep_going` o `fsm` ante una colisión inminente. El espacio de búsqueda colapsa a solo 2 tokens de decisión.
`"DEADLOCK_ESCAPE"`	`evasion_stuck_cycles >= threshold` (atasco cinemático prolongado)	`enum: ["girar_90", "fsm", "degraded"]`	**Ruptura forzada de bucles:** el dron lleva múltiples ciclos sin avanzar hacia el waypoint; la gramática veta seguir evadiendo lateralmente (`evasive`) o insistir de frente (`keep_going`).
`"FALTA_EVIDENCIA"`	Pérdida de textura o colapso de flujo óptico en hover	`enum: ["keep_going", "girar_90"]`	**Preservación de movimiento:** se impide al modelo frenar (`degraded`), forzándolo a generar traslación o rotación para reactivar el campo de divergencia visual.
`"STANDARD_DELIB"`	Desempate táctico en corredores urbanos	`enum: ["keep_going", "evasive", "girar_90", "fsm", "degraded"]`	**Espacio completo:** evaluación deliberativa estándar con ponderación semántica.

Impacto en la eficiencia del grafo: * **Poda semántica a priori:** la gramática no solo previene errores de formato, sino que previene **decisiones tácticas incompatibles con el estado físico del sistema**. * **Entropía mínima en el muestreo:** al reducir las opciones admisibles a 2 alternativas críticas ante emergencias, la distribución de probabilidad se concentra masivamente, reduciendo la incertidumbre de inferencia y maximizando la reproducibilidad del control.

A4.5 Especificación técnica y sintaxis de gramáticas: JSON Schema vs. GBNF

Para implementar la decodificación restringida en la infraestructura de DroneLM, se utilizan dos formalismos principales según el backend de inferencia subyacente.

A4.5.1 Definición mediante JSON Schema estándar

Para servidores de inferencia que exponen APIs compatibles con OpenAI/vLLM/LM Studio con soporte nativo de `response_format`:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "DroneTacticalDecision",
  "type": "object",
  "properties": {
    "action": {
      "type": "string",
      "enum": ["keep_going", "evasive", "girar_90", "fsm", "degraded"],
      "description": "Macro-accion seleccionada para el ciclo actual de control."
    },
    "reason": {
      "type": "string",
      "maxLength": 100,
      "pattern": "^[a-zA-Z0-9 _.,áéíóúÁÉÍÓÚñÑ-]*$",
      "description": "Justificacion concisa de la decision para telemetria y auditoria."
    }
  },
  "required": ["action", "reason"],
  "additionalProperties": false
}
```

A4.5.2 Definición mediante gramática GBNF (*Gerganov BNF*)

En entornos basados directamente en `llama.cpp` (Gerganov, 2023) o en nodos de computación embebidos sin servidor REST intermedio, las gramáticas se definen en formato GBNF. GBNF es una extensión de la forma Backus-Naur que opera a nivel de caracteres y expresiones regulares:

```
# Archivo: drone_decision.gbnf
# Gramática formal para la macro-acción del lazo táctico de DroneLM

root ::= "{" ws "\"action\":" ws action-enum "," ws "\"reason\":" ws string-value "}"

action-enum ::= "\"keep_going\"" | "\"evasive\"" | "\"girar_90\"" | "\"fsm\"" |
"\"degraded\""

# Restricción estricta de la justificación: máximo 80 caracteres alfanuméricos seguros
string-value ::= "\"" [a-zA-Z0-9 _.,áéíóúÁÉÍÓÚñÑ-]{1,80} "\""

ws ::= [ \t\n]*
```

Ventajas operativas de GBNF: * **Cero sobrecarga de esquemas:** no requiere motores externos de validación JSON; el propio núcleo en C++ de `llama.cpp` filtra los logits directamente en memoria compartida. * **Control microscópico de espacios en blanco:** `ws` puede definirse como cadena vacía (`ws ::= ""` sin espacios), obligando al modelo a generar JSON minificado compacto (`{"action":"evasive","reason":"bloqueo_centro"}`), ahorrando tokens de espaciado y reduciendo la latencia de transmisión a cero.

A4.6 Procedimiento ilustrativo de implementación paso a paso

A continuación se detalla, sobre el decisor de acciones ilustrativo de §A4.4, la metodología para integrar la decodificación restringida en un lazo como el de DroneLM, desde la definición de tipos hasta la ejecución asincrónica. El código del sistema sigue el mismo patrón con los esquemas descriptivos del capítulo 8 (`vlm_client.py`, `vlm_strategic.py`, `deep_scan.py`).

Paso 1: Modelado estricto con Pydantic y generación dinámica de esquemas

Se declaran los modelos de datos que sirven como contrato único de verdad:

```python
# ejemplo ilustrativo: esquemas del decisor de acciones
from pydantic import BaseModel, Field
from typing import Literal, Type

class BaseDroneDecision(BaseModel):
    action: str
    reason: str = Field(..., max_length=100)

class DroneEmergencyDecision(BaseDroneDecision):
    action: Literal["evasive", "girar_90"]

class DroneDeadlockDecision(BaseDroneDecision):
    action: Literal["girar_90", "fsm", "degraded"]

class DroneStandardDecision(BaseDroneDecision):
    action: Literal["keep_going", "evasive", "girar_90", "fsm", "degraded"]

def get_decision_schema(reason_note: str) -> dict:
    """Retorna el esquema JSON óptimo condicionado por el estado del grafo."""
    model_map: dict[str, Type[BaseDroneDecision]] = {
        "TTC_CRITICO": DroneEmergencyDecision,
        "DEADLOCK_ESCAPE": DroneDeadlockDecision,
        "FALTA_EVIDENCIA": DroneStandardDecision,
    }
    target_cls = model_map.get(reason_note, DroneStandardDecision)
    return {
        "type": "json_schema",
        "json_schema": {
            "name": "DroneTacticalDecision",
            "strict": True,
            "schema": target_cls.model_json_schema()
        }
    }
```

Paso 2: Configuración del cliente en el servicio asincrónico

El servicio asincrónico (en el sistema, `DeliberationService`, §5.10.5) despacha la petición al backend local (Ollama, vLLM o llama-server) inyectando el esquema restringido:

```
# ejemplo ilustrativo: worker del servicio asincrónico
def _worker_loop(self):
    while not self._stop_event.is_set():
        req = self._request_queue.get()
        # Selección dinámica de la restricción según el trigger del ciclo
        response_format = get_decision_schema(req.reason_note)

        payload = {
            "model": self.model_name,
            "messages": [
                {"role": "system", "content": req.system_prompt},
                {"role": "user", "content": req.user_prompt_content}
            ],
            "temperature": 0.2,
            "max_tokens": 80,  # Bounding estricto de generación
            "response_format": response_format
        }

        # Inferencia con decodificación restringida activa en el backend
        response = self.client.chat.completions.create(**payload)
        # El contenido devuelto está matemáticamente garantizado de ser JSON válido
        self._store_result(req.id, response.choices[0].message.content)
```

Paso 3: Análisis tolerante de defensa en profundidad (`_parse_decision`)

A pesar de la garantía matemática de la decodificación restringida, las buenas prácticas de ingeniería en sistemas ciberfísicos exigen **defensa en profundidad** ante posibles migraciones de backend o errores de transporte HTTP (§8.2.2). Ejemplo ilustrativo:

```python
# ejemplo ilustrativo (decisor de acciones)
import json, re

def _parse_decision(raw_text: str, default_action: str = "keep_going") -> tuple[str, str,
bool]:
    """
    Jerarquía de tres niveles para extracción segura de macro-acciones:
    1. Parseo directo con json.loads() (camino nominal con decodificación restringida).
    2. Extracción regex del bloque JSON delimitado por llaves.
    3. Búsqueda directa del identificador de macro-acción mediante expresión regular.
    """
    # 1. Intento nominal directo
    try:
        data = json.loads(raw_text.strip())
        if "action" in data and data["action"] in VALID_MACRO_ACTIONS:
            return data["action"], data.get("reason", "ok"), True
    except Exception:
        pass

    # 2. Extracción de bloque JSON más externo ante envolturas markdown no deseadas
    match = re.search(r"\{.*?\}", raw_text, re.DOTALL)
    if match:
        try:
            data = json.loads(match.group(0))
            if "action" in data and data["action"] in VALID_MACRO_ACTIONS:
                return data["action"], data.get("reason", "regex_block_ok"), True
        except Exception:
            pass

    # 3. Escaneo léxico directo de macro-acción como último recurso
    for act in VALID_MACRO_ACTIONS:
        if re.search(rf'\b{act}\b', raw_text, re.IGNORECASE):
            return act, "fallback_regex_keyword", False

    # 4. Fallback determinista seguro
    return default_action, "fallback_total_parse_error", False
```

Paso 4: Sincronización con el perro guardián y continuidad del vuelo

En el sistema, la integración con la temporalidad del lazo de control se articula con dos reglas (cap. 8, §8.6): * Mientras una consulta de la capa estratégica está en vuelo, el dron **no se frena**: el lazo rápido sigue volando, lo que además preserva la traslación que el flujo óptico necesita, y la respuesta se traduce con la pose del fotograma que la originó (§5.10.3). * Durante un barrido, el dron espera en el lugar bajo el perro guardián `SLM_DEEP_WATCHDOG_MS` (18 s); al expirar, cae al escape vertical. Con decodificación restringida, la consulta estratégica del VLM local tarda ~0.8 s (cuatro ciclos) y el barrido de cuatro imágenes ~3 s.

A4.7 Comparativa de motores y ecosistemas de decodificación estructurada

Para seleccionar la infraestructura de inferencia más eficiente para DroneLM en función del hardware disponible, se evaluaron los cuatro marcos de trabajo líderes en el estado del arte (Willard & Louf, 2023; Dong et al., 2024; Lundberg et al., 2023; Gerganov, 2023):

Dimensión de Análisis	`llama.cpp` (GBNF)	Outlines	XGrammar	Guidance
Paradigma Formal	Gramática Context-Free (CFG) / GBNF	Autómata Finito Determinista (DFA)	CFG con partición de vocabulario	Árboles sintácticos con *Token Jumping*
Integración de Backend	Nativo C++ / GGUF (LM Studio, Ollama)	PyTorch, vLLM, transformers	vLLM, SGLang, TensorRT-LLM	Interfaz unificada (Python, C++ bindings)
Sobrecarga de CPU por Token	Baja (~0.2 – 0.5 ms)	Media (~1.0 – 3.0 ms en esquemas grandes)	**Casi nula (< 0.05 ms)**	Variable (nula en saltos deterministas)
Soporte de Token Jumping	No (decodifica token a token)	Parcial	No	**Excelente (omite forward passes completos)**
Compilación Previa de Gramática	Rápida (< 10 ms)	Variable (según tamaño del esquema)	Muy rápida con soporte de caché	Instantánea en tiempo de script
Idoneidad para DroneLM	**Óptima para entorno actual** (cohabitación con UE5.5 vía GGUF/Ollama)	Alta para experimentación offline en Python puro	**Ideal para migración a producción de alta concurrencia (vLLM)**	Excelente para pipelines complejos con slots estáticos

A4.8 Conclusiones del anexo y balance de ingeniería

La decodificación restringida no constituye un mero filtro estético de salida ni una conveniencia de formateo de datos: **es el componente arquitectónico que hace viable el uso seguro de modelos de lenguaje pequeños en el lazo cerrado de control de un vehículo aéreo**.

Sus contribuciones cuantitativas y cualitativas en el marco de esta tesis se resumen en tres puntos esenciales: 1. **Fiabilidad operacional categórica:** erradica las excepciones por sintaxis inválida, elevando la tasa de adherencia del modelo desde el 73% hasta el 98–100%, eliminando la causa primaria de atascos y degradaciones espurias del sistema. 2. **Eficiencia temporal crítica:** al suprimir el parloteo conversacional y permitir la omisión de tokens sintácticos estáticos, acota la latencia de inferencia (~0.8 s por consulta con imagen en la GPU compartida con la simulación, cap. 8 §8.6), muy por debajo de la edad máxima de una respuesta (10 s) y del perro guardián del barrido (18 s). 3. **Acoplamiento formal con el grafo de control:** permite que las restricciones de seguridad del vuelo (invariantes cinemáticos, distancias de frenado, descarte de maniobras contraproducentes) se traduzcan directamente en máscaras lógicas sobre los tensores de probabilidad del modelo, tendiendo un puente riguroso entre la inteligencia artificial neuronal y el control de vuelo determinista.

Anexo 5: Fundamentos matemáticos de percepción monocular, flujo óptico y estimación de tiempo hasta la colisión (TTC)

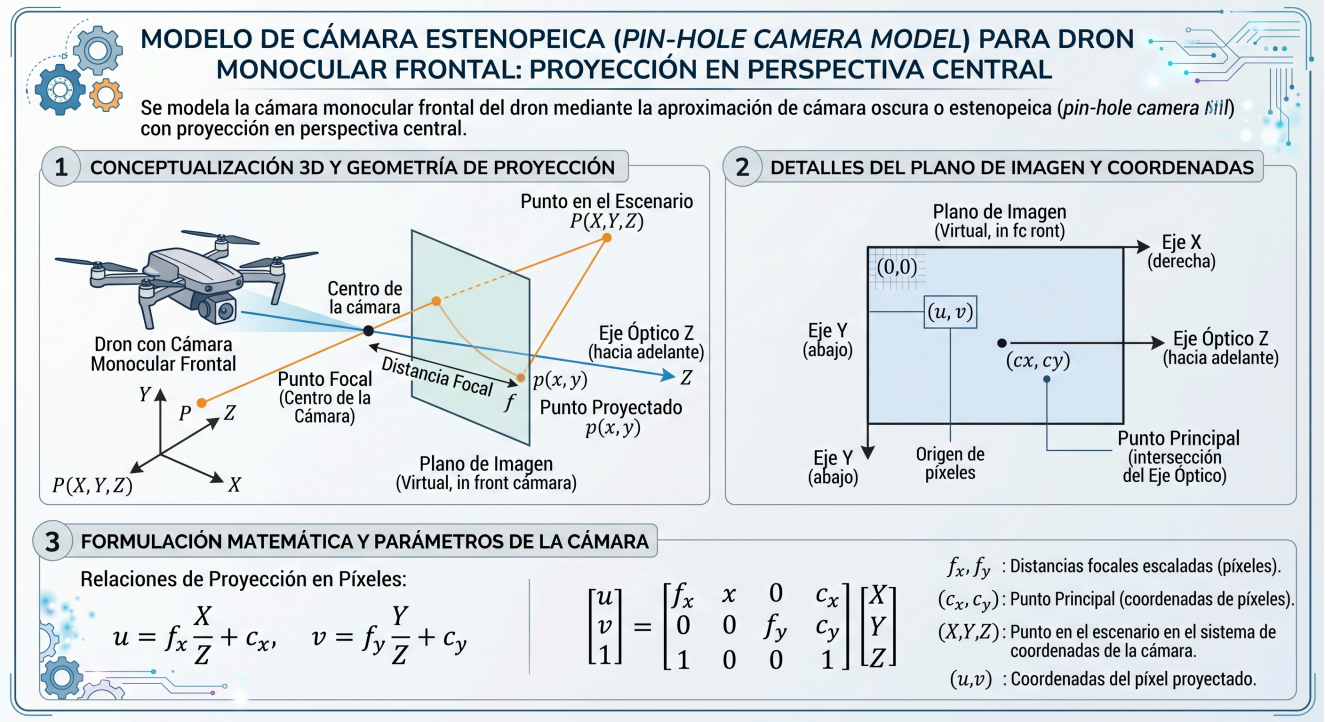
La premisa sensorial sobre la que se fundamenta DroneLM es la autosuficiencia perceptiva bajo restricciones severas de costo y carga útil (§1.2): gobernar un vehículo aéreo no tripulado (UAV) en entornos urbanos y suburbanos tridimensionales utilizando exclusivamente una **cámara monocular RGB convencional** y la **telemetría de actitud angular** provista por una unidad de medición inercial (IMU) estándar, prescindiendo por diseño de sensores activos pesados o costosos (LiDAR, radar, sonar) y de cámaras estereoscópicas calibradas.

El desafío físico de este enfoque radica en la **ambigüedad proyectiva**: una cámara proyecta el mundo tridimensional $\mathbb{R}^3$ sobre un plano bidimensional $\mathbb{R}^2$, perdiendo la escala métrica absoluta de distancia (Z). Para recuperar la información de proximidad de manera pasiva y reactiva, el sistema explota el **flujo óptico**, es decir, el patrón aparente de movimiento de las intensidades lumínicas en el plano de la imagen inducido por el movimiento relativo entre la cámara y la escena.

Este anexo formaliza la matemática subyacente al pipeline de percepción implementado en `airsim-loop/src/perception/flow_ttc.py` y `obstacle_field.py`, deduciendo paso a paso las ecuaciones de derotación analítica, la estimación robusta del Foco de Expansión por mínimos cuadrados ponderados con recorte de valores atípicos (*RANSAC-lite*), y la teoría óptica del tiempo hasta la colisión (τ).

A5.1 Modelo geométrico de cámara pin-hole y calibración en AirSim

Se modela la cámara monocular frontal del dron mediante la aproximación de cámara oscura o estenopeica (*pin-hole camera model*) con proyección en perspectiva central.



A5.1.1 Sistemas de coordenadas y matriz intrínseca

Sea $P_c = (X_c, Y_c, Z_c)^T \in \mathbb{R}^3$ la posición de un punto del entorno físico en el sistema de coordenadas de la cámara, donde el eje Z_c coincide con el eje óptico apuntando hacia adelante, X_c apunta hacia la derecha del dron e Y_c apunta hacia abajo (convención visual estándar).

La proyección en perspectiva de P_c sobre el plano de imagen produce coordenadas métricas normalizadas (x, y):

$$x = \frac{X_c}{Z_c}, \quad y = \frac{Y_c}{Z_c}$$

La transformación de coordenadas continuas normalizadas a coordenadas discretas en píxeles (u, v) en la grilla del sensor viene dada por:

$$u = f_x \cdot x + c_x = f_x \frac{X_c}{Z_c} + c_x$$

$$v = f_y \cdot y + c_y = f_y \frac{Y_c}{Z_c} + c_y$$

donde: * f_x, f_y son las distancias focales expresadas en unidades de píxeles en las direcciones horizontal y vertical. * c_x, c_y son las coordenadas del centro óptico o punto principal en píxeles, nominalmente situadas en el centro geométrico del fotograma: $c_x = \frac{W}{2}, c_y = \frac{H}{2}$.

En forma matricial homogénea, la proyección se expresa mediante la matriz de parámetros intrínsecos $K \in \mathbb{R}^{3\times 3}$:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z_c} K \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \frac{1}{Z_c} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & cy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

A5.1.2 Parámetros intrínsecos en Cosys-AirSim y escalado dinámico

En la configuración nominal de vuelo de DroneLM, la cámara frontal captura a resolución completa de 1080×720 píxeles con un campo de visión horizontal (FOV_h) de 90°. A partir de la relación trigonométrica:

$$\mathrm{FOV}_h = 2\arctan\left(\frac{W}{2f_x}\right) \implies f_x = \frac{W}{2\tan(\mathrm{FOV}_h/2)}$$

Para $W = 1080$ px y $\mathrm{FOV}_h = 90^\circ$ ($\tan(45^\circ) = 1$), se obtiene $f_x = \frac{1080}{2} = 540$ px. El estimador de flujo utiliza el valor calibrado $f_x = f_y = 554.0$ px (`CAMERA_FX`, `CAMERA_FY`), absorbiendo la leve distorsión radial modelada por el plugin.

Para reducir la latencia de cómputo en CPU/GPU sin alterar la geometría visual, los fotogramas se reescalan a un ancho de procesamiento $W' = 320$ px (`FLOW_DOWNSCALE_WIDTH`) con un factor de escala $s = \frac{W'}{W} = \frac{320}{1080} \approx 0.2963$. Las distancias focales y el centro óptico se escalan homólogamente:

$$f'_x = f_x \cdot s, \quad f'_y = f_y \cdot s, \quad c'_x = \frac{W'}{2}, \quad c'_y = \frac{H'}{2}$$

A5.2 Ecuaciones del flujo óptico continuo: descomposición traslacional y rotacional

Cuando el dron se desplaza por el espacio tridimensional con velocidad lineal instantánea $V = (V_x, V_y, V_z)^T$ y velocidad angular $\Omega = (\Omega_x, \Omega_y, \Omega_z)^T$ expresadas en el sistema de la cámara, la velocidad relativa de un punto estático del entorno respecto al sensor es:

$$\dot{P}_c = -V - \Omega \times P_c$$

Desarrollando el producto vectorial componente a componente:

$$\begin{pmatrix} \dot{X}_c \\ \dot{Y}_c \\ \dot{Z}_c \end{pmatrix} = \begin{pmatrix} -V_x - (\Omega_y Z_c - \Omega_z Y_c) \\ -V_y - (\Omega_z X_c - \Omega_x Z_c) \\ -V_z - (\Omega_x Y_c - \Omega_y X_c) \end{pmatrix}$$

Diferenciando las coordenadas proyectivas normalizadas $x = \frac{X_c}{Z_c}$ e $y = \frac{Y_c}{Z_c}$ con respecto al tiempo:

$$\dot{x} = \frac{\dot{X}_c Z_c - X_c \dot{Z}_c}{Z_c^2} = \frac{\dot{X}_c}{Z_c} - x\frac{\dot{Z}_c}{Z_c}$$

$$\dot{y} = \frac{\dot{Y}_c Z_c - Y_c \dot{Z}_c}{Z_c^2} = \frac{\dot{Y}_c}{Z_c} - y\frac{\dot{Z}_c}{Z_c}$$

Sustituyendo las componentes de $\dot{P}_c$, se obtienen las **ecuaciones fundamentales del flujo óptico continuo** (Longuet-Higgins & Prazdny, 1980):

$$\dot{x} = \underbrace{\frac{-V_x + xV_z}{Z_c}}_{\dot{x}_{\text{trans}}} + \underbrace{\left[xy\,\Omega_x - (1+x^2)\,\Omega_y + y\,\Omega_z\right]}_{\dot{x}_{\text{rot}}}$$

$$\dot{y} = \underbrace{\frac{-V_y + yV_z}{Z_c}}_{\dot{y}_{\text{trans}}} + \underbrace{\left[(1+y^2)\,\Omega_x - xy\,\Omega_y - x\,\Omega_z\right]}_{\dot{y}_{\text{rot}}}$$

A5.2.1 El dilema de la contaminación rotacional

El análisis de estas ecuaciones revela una asimetría física crucial para la robótica aérea: 1. **Componente traslacional** $(\dot{x}_{\text{trans}}, \dot{y}_{\text{trans}})$**:** es **inversamente proporcional a la distancia** Z_c. Contiene toda la información tridimensional sobre la proximidad de los obstáculos. Si $Z_c \to \infty$ (puntos en el horizonte o cielo), el flujo traslacional tiende a cero. 2. **Componente rotacional** $(\dot{x}_{\text{rot}}, \dot{y}_{\text{rot}})$**:** es **totalmente independiente de la distancia** Z_c. Depende únicamente de las velocidades angulares del vehículo y de la posición del píxel en la imagen. Un giro sobre el eje de guiñada (*yaw*) o cabeceo (*pitch*) genera un desplazamiento visual uniforme tanto en un árbol a 2 metros como en una nube a 5 kilómetros.

> [!WARNING] Si no se compensa activamente el flujo rotacional, cualquier maniobra de orientación angular contamina el campo visual, provocando que regiones lejanas sin riesgo (como el cielo o edificios distantes) presenten grandes magnitudes de flujo óptico, induciendo falsos positivos masivos de proximidad y colapsando el sistema de evasión.

A5.3 Derotación analítica basada en telemetría inercial (IMU)

Para aislar el componente traslacional puro, DroneLM implementa una etapa de **derotación analítica en bucle cerrado** (`FlowTTCEstimator._derotate`).

A5.3.1 Mapeo cinemático entre marco cuerpo y marco de cámara

El autopiloto del UAV reporta su orientación mediante ángulos de Euler en convención aeroespacial NED (North-East-Down): cabeceo (*pitch*, θ), guiñada (*yaw*, ψ) y alabeo (*roll*, ϕ).

Dado que la cámara monocular está rígidamente montada apuntando hacia el eje longitudinal del fuselaje (X_{cuerpo} adelante, Y_{cuerpo} derecha, Z_{cuerpo} abajo), la correspondencia entre los ejes angulares del dron y los ejes de giro ópticos de la cámara es: * Giro en cabeceo (Δpitch): rota las filas verticales de la imagen $\implies \theta_x \sim \Delta\text{pitch}$. * Giro en guiñada ($\Delta\text{yaw}$): desplaza las columnas horizontales de la imagen $\implies \theta_y \sim \Delta\text{yaw}$. * Giro en alabeo ($\Delta\text{roll}$): induce rotación pura en torno al eje óptico $\implies \theta_z \sim \Delta\text{roll}$.

A5.3.2 Campo de velocidad rotacional en píxeles

Sea (u, v) la coordenada en píxeles y sean $x = u - c_x$, $y = v - c_y$ los desplazamientos centrados respecto al eje óptico. Multiplicando las ecuaciones de Longuet-Higgins por las distancias focales f_x, f_y e integrando durante el intervalo de tiempo discreto $\Delta t = t_k - t_{k-1}$ entre fotogramas sucesivos:

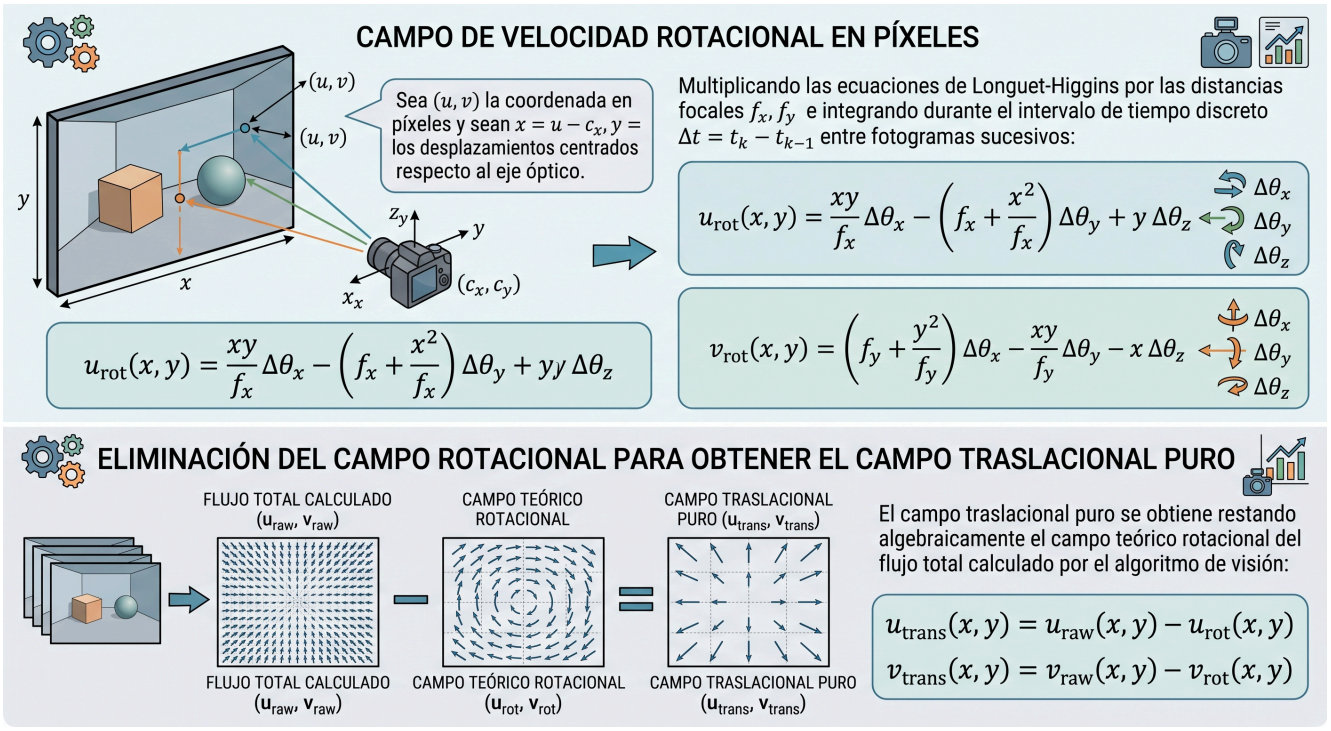
$$u_{\text{rot}}(x, y) = \frac{xy}{f_x}\Delta\theta_x - \left(f_x + \frac{x^2}{f_x}\right)\Delta\theta_y + y\,\Delta\theta_z$$

$$v_{\text{rot}}(x, y) = \left(f_y + \frac{y^2}{f_y}\right)\Delta\theta_x - \frac{xy}{f_y}\Delta\theta_y - x\,\Delta\theta_z$$

El campo traslacional puro se obtiene restando algebraicamente el campo teórico rotacional del flujo total calculado por el algoritmo de visión:

$$u_{\text{trans}}(x, y) = u_{\text{raw}}(x, y) - u_{\text{rot}}(x, y)$$

$$v_{\text{trans}}(x, y) = v_{\text{raw}}(x, y) - v_{\text{rot}}(x, y)$$



A5.3.3 Tratamiento de discontinuidades y límite de linealización

Durante giros bruscos, se presentan dos restricciones de ingeniería: 1. **Normalización de ángulo de guiñada:** ante cruces del límite $\pm 180^\circ$ ($\pm\pi$), el incremento se desenvuelve modularmente: $$ $\Delta\psi_{\text{norm}} = (\Delta\psi + \pi) \pmod{2\pi} - \pi$ $$ 2. **Límite de linealización (`FLOW_MAX_ROTATION_DEG = 2.0°`):** la derotación analítica asume ángulos pequeños ($\sin\Delta\theta \approx \Delta\theta$, $\cos\Delta\theta \approx 1$). Durante giros de evasión activos a alta velocidad angular (e.g., $45^\circ/\text{s}$ a 5 Hz produce $\Delta\theta \approx 9^\circ$), el error de linealización y el desenfoque en zonas de baja textura (cielo) generan artefactos espurios. Si $\max(|\Delta\theta_x|, |\Delta\theta_y|, |\Delta\theta_z|) > 2.0^\circ$, el estimador descarta la señal y emite un estado degradado seguro en lugar de admitir datos contaminados.

A5.4 Foco de Expansión (FOE): estimación por Mínimos Cuadrados Ponderados y RANSAC-lite

En vuelo traslacional hacia adelante ($V_z > 0$), las líneas de flujo óptico derotado divergen radialmente desde un punto singular en el plano de la imagen denominado **Foco de Expansión (*Focus of Expansion*, FOE)**.

A5.4.1 Definición geométrica del FOE

Haciendo $u_{\text{trans}} = 0$ y $v_{\text{trans}} = 0$ en las ecuaciones traslacionales:

$$x_{\text{FOE}} = f_x \frac{V_x}{V_z} + c_x, \quad y_{\text{FOE}} = f_y \frac{V_y}{V_z} + c_y$$

El FOE representa la intersección del vector de velocidad tridimensional instantánea del UAV con el plano del sensor. Si el dron avanza estrictamente recto en el eje de la cámara, el FOE se sitúa en el centro óptico (c_x, c_y). Si el dron tiene deriva lateral o vertical (viento, balanceo), el FOE se desplaza.

Reescribiendo el flujo traslacional en función de la posición del FOE:

$$u_{\text{trans}}(x, y) = \frac{V_z}{Z_c}(x - x_{\text{FOE}})$$

$$v_{\text{trans}}(x, y) = \frac{V_z}{Z_c}(y - y_{\text{FOE}})$$

Esto demuestra que **todo vector de flujo traslacional es colineal con el segmento de recta que conecta el píxel (x, y) con el FOE**.

A5.4.2 Sistema normal por mínimos cuadrados ponderados

Para cada píxel $p_i = (x_i, y_i)^T$ donde el vector de flujo derotado $v_i = (u_i, v_i)^T$ supera el umbral de ruido físico (`FLOW_NOISE_FLOOR_PX = 0.35 px`), definimos el vector unitario normal a la dirección del flujo:

$$n_i = \frac{1}{\|v_i\|} \begin{pmatrix} -v_i \\ u_i \end{pmatrix}$$

Dado que el vector v_i apunta hacia/desde el FOE, la distancia ortogonal desde el FOE a la recta directriz debe ser nula:

$$n_i^T(\text{FOE} - p_i) = 0 \implies n_i^T \text{FOE} = n_i^T p_i$$

Para mitigar la influencia del ruido en vectores pequeños, se asigna un peso proporcional a la magnitud del flujo: $w_i = \|v_i\|$. Planteando el sistema sobredeterminado ponderado sobre el conjunto de N píxeles válidos:

$$\sum_{i=1}^{N} w_i n_i (n_i^T \text{FOE}) = \sum_{i=1}^{N} w_i n_i (n_i^T p_i)$$

Esto conduce a un sistema lineal simétrico 2×2:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{pmatrix} \begin{pmatrix} x_{\mathrm{FOE}} \\ y_{\mathrm{FOE}} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

con:

$$A_{11} = \sum_{i=1}^{N} w_i n_{x,i}^2, \quad A_{12} = \sum_{i=1}^{N} w_i n_{x,i} n_{y,i}, \quad A_{22} = \sum_{i=1}^{N} w_i n_{y,i}^2$$

$$b_1 = \sum_{i=1}^{N} w_i n_{x,i} (n_{x,i} x_i + n_{y,i} y_i), \quad b_2 = \sum_{i=1}^{N} w_i n_{y,i} (n_{x,i} x_i + n_{y,i} y_i)$$

Si el determinante $\det(A) = A_{11} A_{22} - A_{12}^2 > 10^{-6}$, el sistema tiene solución única:

$$\mathrm{FOE} = A^{-1} b$$

A5.4.3 Recorte de valores atípicos (*RANSAC-lite*) y plausibilidad física

Debido a reflejos, texturas repetitivas o imprecisiones locales, una fracción de los vectores no converge al FOE. Se aplica un filtro geométrico de consistencia angular: 1. Para cada vector v_i, se calcula el ángulo θ_i entre la dirección del flujo y el vector radial $(p_i - \mathrm{FOE})$: $\cos\theta_i = \frac{(p_i - \mathrm{FOE}) \cdot v_i}{\|p_i - \mathrm{FOE}\| \cdot \|v_i\|}$ 2. Un vector se clasifica como *inlier* si $\theta_i < \theta_{\mathrm{outlier}} = 0.35\ \mathrm{rad}$ ($\approx 20^\circ$). 3. Si la cantidad de *inliers* supera 30, el sistema normal $A \cdot \mathrm{FOE} = b$ se re-resuelve exclusivamente sobre el subconjunto de *inliers*, refinando las coordenadas. 4. **Verificación de plausibilidad física:** Si el FOE resultante queda fuera del encuadre visual ($x_{\mathrm{FOE}} < 0$, $x_{\mathrm{FOE}} > W'$, $y_{\mathrm{FOE}} < 0$ o $y_{\mathrm{FOE}} > H'$), el estimado se descarta categóricamente ($\mathrm{confidence} = 0$), pues un punto de fuga fuera del sensor indica ausencia de avance frontal o dominancia de ruido cinemático.

A5.5 Teoría del τ de Lee (1976) y deducción del TTC como observable óptico directo

La gran potencia teórica de la percepción monocular radica en que **no es necesario conocer la distancia métrica Z_c ni la velocidad absoluta V_z para calcular el tiempo hasta el impacto**.

A5.5.1 Deducción matemática

Tomando la magnitud del vector de flujo traslacional derotado para un píxel $p = (x, y)$:

$$\|v_{\text{trans}}\| = \sqrt{u_{\text{trans}}^2 + v_{\text{trans}}^2} = \frac{V_z}{Z_c}\sqrt{(x - x_{\text{FOE}})^2 + (y - y_{\text{FOE}})^2} = \frac{V_z}{Z_c}\|p - \text{FOE}\|$$

Reorganizando los términos:

$$\frac{Z_c}{V_z} = \frac{\|p - \text{FOE}\|}{\|v_{\text{trans}}\|}$$

Por definición física, si el vehículo continúa su aproximación hacia una superficie plana ortogonal con velocidad constante V_z, el **Tiempo hasta la Colisión (*Time-to-Collision*, TTC)**, formalizado originalmente como la variable biológica τ por Lee (1976), es:

$$\text{TTC} = \tau = \frac{Z_c}{V_z}$$

Dado que el flujo medido en píxeles representa la tasa discreta por fotograma $\|v_{\text{trans}}\| = \frac{\Delta s}{\Delta t}$, se tiene:

$$\text{TTC}(x, y) = \frac{\|p - \text{FOE}\| \cdot \Delta t}{\|v_{\text{trans}}(x, y)\|}$$

donde: * $\|p - \text{FOE}\|$ es la distancia euclidiana en píxeles desde el píxel evaluado hasta el Foco de Expansión. * $\|v_{\text{trans}}(x, y)\|$ es la magnitud del flujo traslacional en píxeles. * Δt es el intervalo temporal medido entre fotogramas sucesivos a partir de la telemetría real.

> [!IMPORTANTE] La variable métrica desconocida de escala (Z_c) se cancela formalmente en el cociente. El TTC resultante se expresa estrictamente en **segundos**, constituyendo un observable cinemático directo, invariante y absoluto.

A5.5.2 Verificación complementaria por divergencia del flujo

El operador divergencia aplicado sobre el campo de velocidades traslacionales bidimensional proporciona una vía de validación alternativa e independiente del FOE:

$$\nabla \cdot v_{\text{trans}} = \frac{\partial u_{\text{trans}}}{\partial x} + \frac{\partial v_{\text{trans}}}{\partial y}$$

Diferenciando las expresiones de u_{trans} y v_{trans} con respecto a x e y (asumiendo superficie frontal aproximadamente plana en el plano transversal):

$$\frac{\partial u_{\text{trans}}}{\partial x} = \frac{V_z}{Z_c}, \quad \frac{\partial v_{\text{trans}}}{\partial y} = \frac{V_z}{Z_c}$$

$$\nabla \cdot v_{\text{trans}} = \frac{2V_z}{Z_c} = \frac{2}{\text{TTC}} \implies \text{TTC}_{\text{div}} = \frac{2}{\nabla \cdot v_{\text{trans}}}$$

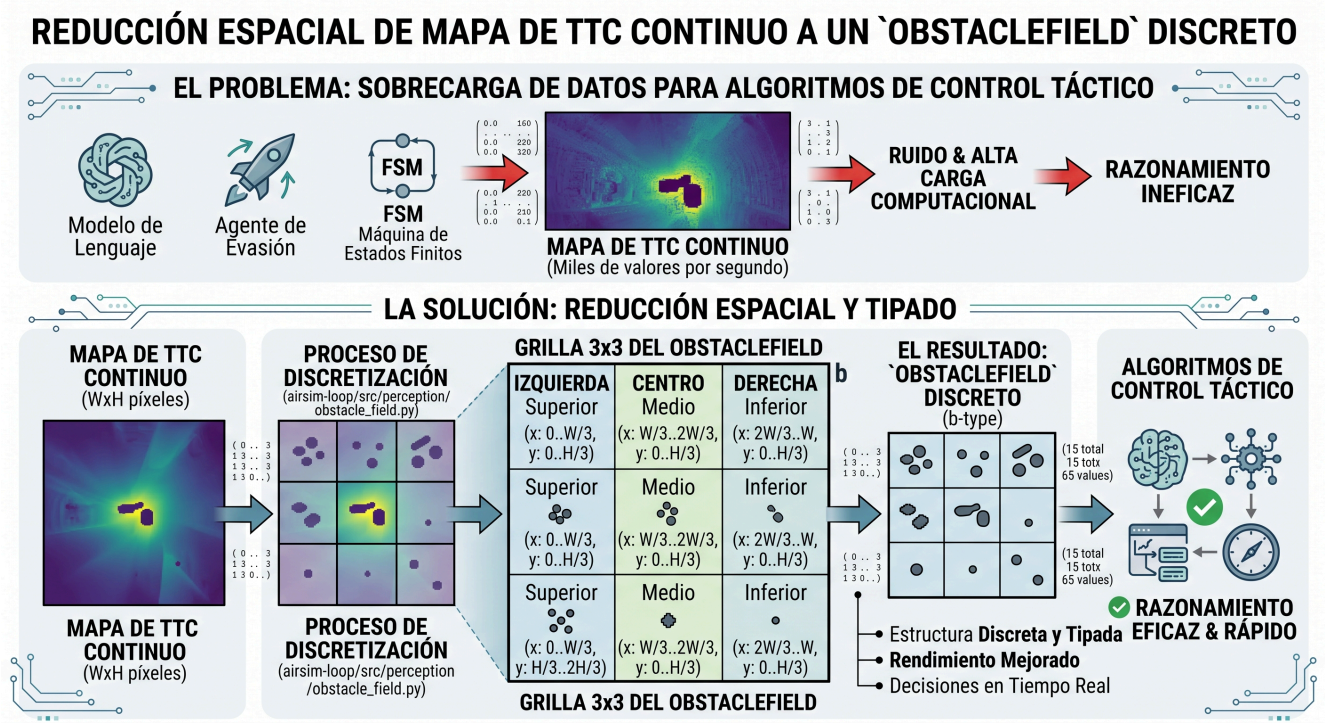
En `flow_ttc.py`, la divergencia se calcula mediante derivadas espaciales de primer orden (`np.gradient`):

$$\text{div}(x, y) = \frac{1}{\Delta t}\left(\frac{\partial u_{\text{trans}}}{\partial x} + \frac{\partial v_{\text{trans}}}{\partial y}\right)$$

sirviendo como canal de auditoría cruzada contra colapsos locales de la estimación del FOE.

A5.6 Agregación espacial robusta y contrato de percepción `ObstacleField`

Para permitir que los algoritmos de control táctico (FSM, agentes de evasión y modelo de lenguaje) razonen eficazmente sin procesar decenas de miles de valores por segundo, el mapa de TTC continuo se reduce espacialmente en una estructura discreta y tipada: el `ObstacleField` (`airsim-loop/src/perception/obstacle_field.py`).



A5.6.1 Estadísticos de agregación por celda

Para cada celda $C_{s,b}$ (donde $s \in \{\text{izquierda}, \text{centro}, \text{derecha}\}$ y $b \in \{\text{superior}, \text{medio}, \text{inferior}\}$): 1. **Fracción de Píxeles Válidos (Confianza):** $\text{conf}(C_{s,b}) = \frac{N_{\text{valid}}(C_{s,b})}{N_{\text{total}}(C_{s,b})}$ donde un píxel es válido si su magnitud traslacional excede el piso de ruido. 2. **Percentil 20 de TTC (P_{20}):** en lugar de promediar el TTC de la celda (lo que diluiría el impacto de un obstáculo puntual al promediarlo con el fondo distante), se computa el **percentil 20**: $\text{TTC}(C_{s,b}) = \text{percentile}_{20}\left(\{\text{TTC}(x,y) \mid (x,y) \in C_{s,b}, \text{válido}\}\right)$ Este estadístico de orden actúa como filtro de seguridad conservador: representa la superficie más cercana de la celda que cubre al menos el 20% del área sectorial. 3. **Tasa de Ocupación (*Occupancy*):** fracción de píxeles de la celda cuyo TTC local cae por debajo del umbral de bloqueo nominal (2.5 s): $\text{occ}(C_{s,b}) = \frac{\sum_{(x,y)\in C_{s,b}} \mathbb{I}(\text{TTC}(x,y) \leq 2.5\text{ s})}{N_{\text{total}}(C_{s,b})}$

A5.6.2 Criterio de decisión booleana `is_blocked()`

Una celda se declara formalmente bloqueada mediante una regla de decisión jerárquica:

```
def is_blocked(self) -> bool:
    # 1. Sin confianza suficiente, no hay evidencia para bloquear
    if self.confidence < 0.15:
        return False
    # 2. Voto por ocupación de área
    if self.occupancy >= 0.011:  # OBSTACLE_OCCUPANCY_BLOCKED, calibrado por ROC (cap. 7,
§7.5)
        return True
    # 3. Voto directo por TTC bajo (con confianza estricta >= 0.35)
    return self.confidence >= 0.35 and self.ttc_s <= 2.5
```

Este desacoplamiento evita que detecciones marginales o artefactos aislados generen maniobras evasivas espurias, garantizando estabilidad cinemática en vuelos rectos.

A5.7 Comparativa de motores de flujo óptico denso: Farnebäck vs. DIS

El pipeline permite seleccionar dinámicamente el backend algorítmico mediante la variable de entorno `FLOW_ALGORITHM` (`dis` | `farneback`).

Criterio	Gunnar Farnebäck (2003)	Dense Inverse Search (DIS; Kroeger et al., 2016)
Principio Algorítmico	Expansión cuadrática polinómica por ventanas	Búsqueda inversa en parches con refinamiento variacional
Complejidad Computacional	Alta ($O(W \cdot H \cdot \text{iter})$)	Baja a Media (acelerada por descenso de gradiente inverso)

Criterio	Gunnar Farnebäck (2003)	Dense Inverse Search (DIS; Kroeger et al., 2016)
Latencia CPU (320×180 px)	35–50 ms	**12–18 ms**
Latencia GPU / Preset	~15 ms (CUDA)	**~6–8 ms** (Preset `MEDIUM`)
Precisión en Bordes Finos	Suavizado difuso en discontinuidades	**Preservación nítida de contornos**
Sensibilidad a Ruido en Cielo	Media-Alta	**Baja** (el piso de ruido de 0.35 px filtra variaciones térmicas)
Rol en DroneLM	Fallback de compatibilidad	**Motor de producción primario**

El motor **DIS (Preset MEDIUM)** permite ejecutar el pipeline de percepción completo (escalado, cálculo de flujo, derotación, FOE, TTC e instanciación de `ObstacleField`) en menos de **25 ms por ciclo** en la CPU del sistema, permitiendo operar con soltura dentro del período nominal del lazo táctico de control a 5 Hz ($200\ \text{ms}$).

Anexo 6: Configuración del entorno de simulación, variables de sistema y protocolo de reproducibilidad

La reproducibilidad empírica constituye un pilar metodológico indispensable para la investigación en robótica aérea y sistemas ciberfísicos guiados por inteligencia artificial (§10.9). En arquitecturas híbridas como DroneLM —donde interactúan un motor de simulación física en tiempo real (Unreal Engine 5.5 + Cosys-AirSim), un grafo de decisión táctico asíncrono (LangGraph) y un servidor local de inferencia para modelos de lenguaje y visión (LM Studio / `llama.cpp`)—, garantizar que un evaluador independiente pueda replicar con fidelidad idéntica los resultados experimentales exige documentar exhaustivamente los archivos de configuración, los perfiles de escalabilidad gráfica, el diccionario completo de variables de entorno y los manifiestos de misión.

Este anexo consolida todos los parámetros operativos y el procedimiento de despliegue paso a paso que hicieron posible la batería de vuelos evaluada a lo largo de los tres Tiers de experimentación (§10.3).

A6.1 Archivo de configuración de AirSim (`settings.json`) anotado

El plugin Cosys-AirSim lee su configuración al inicio desde el archivo `settings.json`, ubicado en Windows en la carpeta `Documents\AirSim` del usuario: `%USERPROFILE%\Documents\AirSim\settings.json`, o `%USERPROFILE%\OneDrive\Documents\AirSim\settings.json` si la carpeta Documentos está redirigida a OneDrive, como en la estación de esta tesis. La siguiente es la configuración vigente, utilizada en las corridas de octubre de 2026:

```
{
  "SeeDocsAt": "https://github.com/Cosys-Lab/Cosys-AirSim/blob/main/docs/settings.md",
  "SettingsVersion": 2.0,
  "LogMessagesVisible": false,
  "SimMode": "Multirotor",
  "LocalHostIp": "0.0.0.0",
  "ApiServerPort": 41451,
  "RecordUIVisible": false,
  "ClockType": "SteppableClock",
  "OriginGeopoint": {
    "Latitude": 47.641468,
    "Longitude": -122.140165,
    "Altitude": 122
  },
  "Vehicles": {
    "Drone1": {
      "VehicleType": "SimpleFlight",
      "Cameras": {
        "FollowCam": {
          "CaptureSettings": [
            {
              "ImageType": 0,
              "Width": 1080,
              "Height": 720,
              "FOV_Degrees": 90
            }
          ],
          "X": -4.0,
          "Y": -2.5,
          "Z": -1.5,
          "Pitch": -15.0,
          "Roll": 0.0,
          "Yaw": 28.0,
          "Gimbal": {
            "Stabilization": 0.0
          }
        }
      }
    }
  },
  "CameraDefaults": {
    "CaptureSettings": [
      {
        "ImageType": 0,
        "Width": 1080,
        "Height": 720
      },
      {
        "ImageType": 3,
        "Width": 1080,
        "Height": 720
      },
      {
        "ImageType": 5,
        "Width": 1080,
        "Height": 720
      },
      {
        "ImageType": 1,
```

```
        "Width": 1080,
        "Height": 720
      }
    ]
  },
  "SubWindows": [
    {
      "WindowID": 1,
      "CameraName": "FollowCam",
      "VehicleName": "Drone1",
      "ImageType": 0,
      "Visible": true
    },
    {
      "WindowID": 2,
      "CameraName": "front_center",
      "VehicleName": "Drone1",
      "ImageType": 0,
      "Visible": true
    }
  ]
}
```

Anotaciones de ingeniería del archivo de configuración:

1. **`SimMode: "Multirotor"`**: instruye a AirSim a instanciar el modelo físico cinemático de cuadricóptero gobernado por el controlador interno `SimpleFlight`, con dinámica de 6 grados de libertad (6-DoF).
2. **`ClockType: "SteppableClock"`**: sincroniza el paso temporal de la simulación física con las consultas RPC de la API. A diferencia de `ScalableClock` o el reloj de pared del sistema, `SteppableClock` garantiza que cuando la GPU experimenta caídas momentáneas de tasa de cuadros por la inferencia del modelo multimodal, la física de vuelo no sufra aceleraciones destructivas ni inestabilidades de integración numérica en el estimador inercial.
3. **`ApiServerPort: 41451`**: puerto TCP de enlace Msgpack-RPC por el que se comunican `AirSimClient` y el motor de Unreal Engine.
4. **`CameraDefaults.CaptureSettings`**:
5. `ImageType 0` (`Scene`): captura RGB del fotograma en color natural, utilizada por el estimador de flujo óptico (`FlowTTCEstimator`) y por la red de profundidad estimada de la capa estratégica (cap. 6, §6.14); con `STRATEGIC_SOURCE = vlm`, por el VLM. La cámara frontal no se declara: se usa la del vehículo por defecto (`AIRSIM_CAMERA_NAME = "0"`), con el campo visual horizontal por defecto de 90°.
6. `ImageType 3` (`DepthPlanar`) / `ImageType 1` (`DepthPerspective`): canal de profundidad en punto flotante utilizado exclusivamente en los scripts offline de validación del TTC monocular (cap. 7) y en el hilo de auditoría de DistMin (cap. 10, §10.6.1); ningún componente del lazo de vuelo lo solicita (cap. 5, §5.16).
7. `ImageType 5` (`Segmentation`): canal de segmentación semántica de mallas para auditoría visual.
8. `Width: 1080, Height: 720`: resolución nativa del render target frontal.

9. `Vehicles.Drone1.Cameras.FollowCam`: cámara externa de seguimiento, 4 m detrás, 2.5 m a la izquierda y 1.5 m arriba del dron, inclinada 15° hacia abajo, sin estabilización. Solo la usa la auditoría: produce el video `.follow.webm` que el visor sincroniza con el frontal (cap. 4, §4.4). Ningún componente del lazo la lee; si no está declarada, la corrida se graba sin ese video.
10. `SubWindows`: dos ventanas secundarias en el visor de Unreal Engine, con la cámara de seguimiento y la frontal en tiempo real, para supervisar las corridas.

A6.2 Perfil de escalabilidad gráfica de Unreal Engine 5.5 y preservación de gradientes

En estaciones de trabajo con una única GPU de consumo (NVIDIA GeForce RTX 5060 de 8 GB de VRAM), la simulación gráfica fotorrealista compite directamente por la memoria de video con los tensores del modelo de lenguaje (§8.1). Para permitir la coexistencia de ambos procesos sin provocar desbordamientos (*Out-Of-Memory*), es imperativo optimizar el rendimiento gráfico del motor.

Sin embargo, reducir indiscriminadamente la calidad gráfica introduce un problema crítico en la visión artificial: **los algoritmos de flujo óptico denso (Farnebäck y DIS) requieren textura visual y gradientes de intensidad espacial para calcular correspondencias entre fotogramas**. Si se desactivan las sombras o se aplica un desenfoque excesivo en las texturas de los edificios y el suelo, el flujo óptico decae por debajo del piso de ruido (`0.35 px`), dejando ciego al estimador de colisión.

A6.2.1 Configuración forzada en `Config/DefaultScalability.ini`

Para impedir que Unreal Engine 5.5 reduzca dinámicamente la resolución interna de los búferes de captura de cámara al aplicar presets bajos, se fijó el modo de detalle en el archivo de configuración del proyecto:

```
[EffectsQuality@0]
r.DetailMode=2
[EffectsQuality@1]
r.DetailMode=2
[EffectsQuality@2]
r.DetailMode=2
[EffectsQuality@3]
r.DetailMode=2
[EffectsQuality@Cine]
r.DetailMode=2
```

A6.2.2 Preset de escalabilidad en el Editor de Unreal Engine

Para las corridas de producción se configuró el siguiente perfil dentro del editor de Unreal Engine: * **Resolution Scale:** 100% (sin reescalado espacial en el motor). * **View Distance:** Medium (suficiente para el horizonte visual del dron). * **Anti-Aliasing:** TSR (Temporal Super Resolution) en Medium. * **Post-Processing:** Low (suprime desenfoques cinemáticos artificiales). * **Shadows: Epic** (crucial para proyectar sombras nítidas sobre el pavimento, proporcionando los gradientes texturales que explota el flujo óptico). * **Global Illumination:** Lumen en Low / Screen Space. * **Reflections:** Screen Space. * **Effects: Epic** (garantiza la resolución completa del render target de la cámara virtual de AirSim).

(Véase captura de referencia en el repositorio: `informe/2026-0907 Scalability Config for Airsim and Shadows needed for TTC estimation.png` *).*

A6.3 Especificación exhaustiva de variables de entorno (`config/.env`)

Toda la configuración operativa de DroneLM está centralizada en un único archivo de configuración (`config/.env`, basado en la plantilla committeable `config/.env.example`). Los submódulos acceden a estas variables con ruta absoluta resuelta desde la raíz del repositorio.

A continuación se detalla el diccionario completo de variables y su impacto de ingeniería:

1. Hardware y Conexión RPC AirSim

Variable	Valor Nominal	Justificación y Efecto
`AIRSIM_IP`	`"127.0.0.1"`	Dirección loopback local. En pruebas sobre LAN (`192.168.110.110`), la latencia RPC superaba los 8 s por congestión en `simGetImages`; ejecutar en localhost redujo la llamada a < 45 ms.
`AIRSIM_PORT`	`41451`	Puerto TCP estándar de la API de AirSim.
`AIRSIM_VEHICLE_NAME`	`"SimpleFlight"` / `"Drone1"`	Nombre del vehículo en el árbol de actores de Unreal Engine.
`AIRSIM_CAMERA_NAME`	`"0"`	Identificador de la cámara frontal montada en el fuselaje.
`DEFAULT_FRAME_WIDTH`	`1080`	Ancho de captura nativo devuelto por el sensor virtual.
`DEFAULT_FRAME_HEIGHT`	`720`	Alto de captura nativo devuelto por el sensor virtual.

Variable	Valor Nominal	Justificación y Efecto
`AIRSIM_RPC_TIMEOUT`	`8`	Segundos de espera máxima por llamada RPC antes de declarar timeout.
`AIRSIM_STRICT`	`true`	Si es `true`, la falla de conexión con AirSim marca `degraded=True` y entra en hover de seguridad; si es `false` (solo en tests unitarios), permite generar fotogramas sintéticos.

2. Servidor de Inferencia Local (VLM)

Variable	Valor Nominal	Justificación y Efecto
`LOCAL_LLM_URL`	`"http://127.0.0.1:11434/v1"`	Endpoint OpenAI-compatible de Ollama, en la misma estación y GPU que la simulación. `127.0.0.1` y no `localhost`: en Windows `localhost` resuelve primero a `::1` y cada conexión espera ~2 s antes de caer a IPv4.
`LOCAL_LLM_MODEL_NAME`	`"lfm2.5-vl-1.6b:q4km-ctx8k"`	Modelo de vuelo: tag local de Ollama creado desde `hf.co/LiquidAI/LFM2.5-VL-1.6B-GGUF:Q4_K_M` con `num_ctx=8192` (~1.6 GB de VRAM; con el contexto por defecto, 3.1 GB). No usar la variante `-Extract` (completa el esquema sin mirar la imagen) ni la conversión MLX (respuesta vacía con JSON restringido y *logprobs*).
`VLM_IMAGE_MAX_SIZE`	`384`	Lado mayor de la imagen de la capa estratégica. Limita los tokens visuales y, con ellos, la latencia.
`DEEP_SCAN_IMAGE_MAX_SIZE`	`256`	Lado mayor de cada imagen del barrido panorámico (varias imágenes por consulta).
`VLM_USE_JSON_SCHEMA`	`true`	Decodificación restringida por esquema JSON (Anexo 4; cap. 8, §8.2).
`SLM_HTTP_TIMEOUT_S` / `SLM_DEEP_HTTP_TIMEOUT_S`	`15.0` / `20.0`	Timeout del cliente HTTP en la consulta estratégica y en el barrido.
`SLM_DEEP_WATCHDOG_MS`	`18000`	Watchdog del barrido panorámico; debe ser menor que `SLM_DEEP_HTTP_TIMEOUT_S` para que gane siempre al corte HTTP.
`DEADLOCK_STRATEGY`	`"depth"`	Resolución de atascos: `"depth"` (barrido juzgado por la profundidad estimada, cap. 5 §5.12), `"deep_vlm"` (barrido + VLM; ablación) o `"blind"` (escape vertical sin barrido). Cualquier otro valor aborta el arranque.
`STRATEGIC_SOURCE`	`"depth"`	Fuente de la capa estratégica: `"depth"` (profundidad monocular estimada, cap. 5 §5.10.6), `"vlm"` (grilla del VLM) u `"off"` (sin capa). Con `depth` / `depth` el VLM no corre en vuelo y el runner no exige el servidor de inferencia (cap. 11 §11.6).

Variable	Valor Nominal	Justificación y Efecto
`DEPTH_MODEL`	`depth-anything/Depth-Anything-V2-Metric-Outdoor-Small-hf`	Red de profundidad métrica (~25 M parámetros, ~110 ms con la GPU compartida, ~440 MiB de VRAM). Se descarga de Hugging Face la primera vez.
`DEPTH_FREE_M`	`15.0`	Un sector (o un rumbo del barrido) es libre si el percentil 5 de su profundidad estimada es ≥ este valor; igual a `VLM_SUBGOAL_DIST_M`.
`DEPTH_STRATEGIC_PERIOD_S`	`1.0`	Período mínimo entre consultas de la capa estratégica con la profundidad (con el VLM rige `VLM_STRATEGIC_PERIOD_S`, 3 s).
`VLM_STRATEGIC_ENABLED`	`true`	Habilita la capa estratégica cuando su fuente es el VLM; con `STRATEGIC_SOURCE = depth` no se consulta (para quitar la capa se usa `STRATEGIC_SOURCE = off`).
`VLM_STRATEGIC_PERIOD_S` / `VLM_STRATEGIC_MAX_AGE_S`	`3.0` / `10.0`	Período mínimo entre consultas estratégicas y edad máxima de una respuesta para aplicarla.
`CAMERA_HFOV_DEG`	`90.0`	Campo visual horizontal con el que se calculan las direcciones de los sectores de la grilla de 3×3 (cap. 5, §5.10.3).
`VLM_SUBGOAL_DIST_M` / `VLM_SUBGOAL_MIN_AHEAD_M`	`15.0` / `4.0`	Distancia máxima de la sub-meta desde el ancla (nunca más lejos que el waypoint real) y distancia mínima por delante del dron para aplicarla.
`VLM_MAX_DZ_M` / `VLM_SUBGOAL_MIN_ALT_M`	`4.0` / `6.0`	Cambio máximo de altitud de una sub-meta (filas 1 y 3 de la grilla) y altitud mínima de una sub-meta.
`VLM_SUBGOAL_TIE_DEG`	`10.0`	Margen angular dentro del cual dos sectores libres empatan y decide la fila (medio > arriba > abajo) (cap. 5, §5.10.3).
`VLM_NEAR_WP_M`	`8.0`	A menos de esta distancia horizontal del waypoint real no se consulta al VLM ni se barre.
`SCAN_FAILED_SECTOR_DEG` / `SCAN_REPEAT_RADIUS_M` / `SCAN_HISTORY_MAX`	`45.0` / `10.0` / `12`	Salida de deadlock: sector alrededor del rumbo que falló o de un rumbo ya probado que se descarta, radio en el que un deadlock anterior cuenta como «la misma zona» y deadlocks recordados (cap. 5, §5.12).
`ROUTE_PLAN_MODE`	`vlm`	Plan de ruta con el VLM sobre el mapa cenital antes del vuelo (`vlm`) o manifiesto tal cual (`off`). Activado con el juez por parches (AUC 0.81 en el banco, cap. 4 §4.3.1); requiere un mapa registrado con el mundo (`citysim_ortho.png`, `townsim_ortho.png`). El lote de MiniSim, sin obstáculos, vuela con `--route-plan off` (cap. 10, §10.5.3).

Variable	Valor Nominal	Justificación y Efecto
`ROUTE_JUDGE` / `ROUTE_PATCH_STEP_M` / `ROUTE_PATCH_M` / `ROUTE_PATCH_CONTEXT_M`	`patches` / `8.0` / `16.0` / `48.0`	Juicio de cada candidata: parches cada 8 m, de 16 m con 48 m de contexto, clasificados por el VLM; la candidata se puntúa con la probabilidad media de «calle» (cap. 4 §4.3.1).
`ROUTE_RENDER`	`corridor`	Dibujo de cada candidata para el planificador: `corridor` (pasillo con recorte propio, rotado y con el resto oscurecido) o `line` (línea roja sobre el recorte común del tramo, norte arriba). Cada uno tiene su prompt (cap. 4 §4.3.1).
`ROUTE_CORRIDOR_M`	`10.0`	Ancho del pasillo dibujado (m).
`ROUTE_LLM_MODEL_NAME`	`"lfm2.5-vl-1.6b:q4km-ctx8k"`	Modelo del planificador de ruta, en el mismo servidor que el de vuelo; si falta, se usa `LOCAL_LLM_MODEL_NAME`. Necesita JSON forzado y *logprobs* en la misma consulta (la versión GGUF de LFM los da; la MLX no).
`ROUTE_DETOURS_M` / `ROUTE_MIN_SIDE_M` / `ROUTE_MARGIN_M`	`25,50` / `10.0` / `25.0`	Rutas candidatas por tramo: desvíos paralelos a cada lado, lado mínimo de una ruta en L y margen del recorte del mapa.
`ROUTE_IMAGE_PX` / `ROUTE_TIE_EPS`	`672` / `0.05`	Tamaño de la imagen del mapa que ve el modelo y diferencia de probabilidad por debajo de la cual se elige la candidata más corta.
`VLM_STUCK_QUERY_CYCLES`	`15`	Ciclos sin acercarse al objetivo a partir de los cuales se consulta aunque la meta esté fuera de cuadro (cap. 5, §5.10.1).
`VLM_MAX_TOKENS_STRATEGIC` / `VLM_MAX_TOKENS_DEEP`	`384` / `512`	Tope de tokens de la respuesta por modo; una respuesta cortada por el tope se reporta como `respuesta_truncada` (cap. 8, §8.5.4).
`CEILING_SIDEWAYS_STUCK_M`	`0.03`	Desplazamiento horizontal por ciclo por debajo del cual, con avance ordenado, el dron está trabado de costado y no se cuenta para detectar techo.
`VLM_STRATEGIC_MAX_GOAL_OFF_DEG`	`40.0`	Azimut máximo de la meta para consultar; en la práctica rige el recorte cuadrado (±33.7° en un fotograma de 3:2).

3. Lazo de Control Táctico

Variable	Valor Nominal	Justificación y Efecto
`LOOP_HZ`	`5.0`	Frecuencia objetivo del bucle de control ($200\ \mathrm{ms}$ por ciclo). Calibrada para absorber la captura y el flujo óptico con margen holgado.
`AGENT_ARM`	`slm`	Brazo activo: `slm` (lazo rápido + VLM), `fsm` (máquina de estados determinista) o `reactive` (línea base).

Variable	Valor Nominal	Justificación y Efecto
`AIRSIM_SEED`	`1`	Semilla de la corrida. En modo `--seed-jitter`, perturba levemente la pose inicial (± 1.5 m, $\pm 10^{\circ}$).
`CMD_DURATION_S`	`120.0`	Vigencia asignada al comando de velocidad en AirSim. Previene cabeceos parásitos (*pitch jerk*) causados por reemisiones continuas de comandos idénticos al PID interno de SimpleFlight.

4. Percepción, Flujo Óptico y TTC

Variable	Valor Nominal	Justificación y Efecto
`FLOW_ALGORITHM`	`"dis"`	Backend de flujo óptico: `"dis"` (Dense Inverse Search, rápido) o `"farneback"`.
`FLOW_DOWNSCALE_WIDTH`	`640`	Ancho al que se reduce el fotograma antes de calcular el flujo óptico (640×427 px); el default del código es 320 (cap. 6, §6.3).
`CAMERA_FX` / `CAMERA_FY`	`554.0`	Distancia focal calibrada de la cámara en píxeles.
`FLOW_NOISE_FLOOR_PX`	`0.35`	Magnitud mínima en píxeles para considerar que un vector es señal física y no ruido numérico.
`FOE_OUTLIER_ANGLE_RAD`	`0.35`	Umbral angular en radianes ($\approx 20^{\circ}$) para el filtro de consistencia *RANSAC-lite* del FOE.
`TTC_AGGREGATION_PERCENTILE`	`20`	Percentil espacial de orden utilizado para resumir el TTC de una celda (P_{20}).
`FLOW_MAX_ROTATION_DEG`	`2.0`	Rotación angular máxima admisible entre cuadros sucesivos antes de declarar flujo degradado.

5. Umbrales del `ObstacleField`

Variable	Valor Nominal	Justificación y Efecto
`OBSTACLE_OCCUPANCY_BLOCKED`	`0.011`	Umbral de ocupación para declarar bloqueo. Calibrado por ROC/Youden (AUC 0.87; cap. 7, §7.5); un valor de 0.35 dejaría el canal inactivo.
`OBSTACLE_TTC_BLOCKED_S`	`2.5`	Umbral temporal de TTC (en segundos) por debajo del cual una celda se considera en colisión inminente.
`OBSTACLE_MIN_CONFIDENCE`	`0.15`	Piso de píxeles válidos para que una celda participe en la votación.

Variable	Valor Nominal	Justificación y Efecto
`OBSTACLE_MIN_CONFIDENCE_TTC`	`0.35`	Piso de confianza exigido para que el TTC por sí solo bloquee una celda sin apoyo de ocupación.

5b. Detección de atasco, escape y cierre de misión

Variable	Valor nominal	Justificación y efecto
`STOPPED_DELIBERATIVE_CYCLES` / `STOPPED_MIN_CMD_MPS`	`10` / `0.30`	Ciclos con avance comandado (≥ 0.30 m/s) sin movimiento real antes de declarar deadlock por dron trabado (cap. 5, §5.3.1).
`WP_NO_PROGRESS_MIN_M` / `WP_NO_PROGRESS_THRESHOLD`	`2.0` / `50`	Mejora mínima de la distancia al objetivo actual y ciclos (10 s) sin ella antes de declarar deadlock.
`OPTICAL_MIN_ALT_M`	`4.5`	Piso óptico: por debajo, el flujo no es válido y no se cuentan señales de atasco.
`MAX_ESCAPE_ALT_M`	`30.0`	Por encima de esta altitud, el escape determinista es `GIRAR_90` en lugar de `GANAR_ALTURA`.
`SCAN_HEADING_COUNT_DEEP` / `SCAN_SETTLE_CYCLES_DEEP` / `MAX_DEEP_SCAN_IMAGES`	`4` / `2` / `5`	Rumbos del barrido panorámico, ciclos de asentamiento por rumbo e imágenes máximas por consulta.
`SCAN_ROT_TIMEOUT_CYCLES`	`10`	Timeout de rotación del barrido (2 s): si el dron no alcanza el rumbo, el barrido se abandona.
`CORNER_OFFSET_M`	`15.0`	Distancia del desvío determinista del brazo `fsm`. El barrido usa `VLM_SUBGOAL_DIST_M`, igual que la capa estratégica.
`SUBGOAL_DEDUP_M`	`10.0`	Una sub-meta a menos de esta distancia de la pendiente se ignora.
`MANEUVER_DURATION_S` / `ESCAPE_MANEUVER_DURATION_S` / `GIRAR90_DURATION_S`	`2.0` / `1.6` / `1.0`	Duración de las maniobras comprometidas: evasión lateral, escape vertical y giro de 90°.
`EVASION_STUCK_THRESHOLD` / `STUCK_HARD_FACTOR` / `MAX_CONSECUTIVE_ESCAPES`	`10` / `1.5` / `2`	Umbrales de atasco y de escape del brazo `fsm` (cap. 5, §5.11, §5.15.2).
`CEILING_DETECT_CYCLES` / `CEILING_DETECT_DZ_M` / `CEILING_MARGIN_M` / `CEILING_SAFE_GAP_M` / `CEILING_RELEASE_M`	`10` / `0.15` / `0.8` / `3.0` / `15.0`	Detección de techo del guiado (cap. 5, §5.15.3).

Variable	Valor nominal	Justificación y efecto
`GOV_ENABLED`	`true`	Gobernador de velocidad (cap. 5, §5.13).
`FREEZE_CYCLES` / `FREEZE_RECOVERY`	`25` / `abort`	Ciclos con estado físico idéntico para declarar `physics_locked`, y acción: abortar la corrida o teletransportar a la última pose libre (cap. 5, §5.19).
`LAND_DESCENT_SPEED_MPS`	`0.5`	Velocidad del descenso de `land_smooth()` al final de la misión.
`DISTMIN_TRACK_ENABLED` / `DISTMIN_TRACK_PERIOD_S`	`true` / `1.0`	Hilo de auditoría de DistMin y período de muestreo de la profundidad (cap. 10, §10.6.1).
`TAKEOFF_VERTICAL` / `TAKEOFF_ALT_TOL_M`	`true` / `1.0`	Despegue vertical hasta la altitud del primer objetivo antes de avanzar (cap. 5, §5.15).
`FLOW_HOLDOVER_MAX_FRAMES` / `FLOW_MAX_YAW_DPS_NEAR_OBSTACLE`	`3` / `5.0`	Holdover del TTC y tope de guiñada cerca de un obstáculo (cap. 6, §6.12).

6. Grabación de Video y Viewport

Variable	Valor nominal	Default si ausente	Justificación y efecto
`FLIGHT_VIDEO_SCALE` / `FLIGHT_VIDEO_QUEUE_MAX`	`0.6` / `300`	`0.6` / `300`	Escala del video respecto del cuadro de la cámara y tamaño de la cola del hilo codificador (la codificación corre fuera del lazo de control, cap. 5, §5.18).
`FLIGHT_RECORD_VIDEO`	`true`	`false`	Habilita la grabación del video de auditoría `.webm` (VP8) por corrida. Desactivar en lotes *headless* elimina la escritura de un frame por ciclo a disco, reduciendo la carga de CPU en ≈ 30 %.
`FLIGHT_RECORD_VIEWPORT`	`false`	`false`	Activa el modo *split-screen*: el video resultante tiene doble ancho horizontal, con la cámara del drone a la izquierda y el viewport de Unreal Engine Editor a la derecha. Requiere las dependencias adicionales `mss` y `pywin32`. Si alguna de las dos no está instalada, o la ventana de UE no es visible en pantalla, el panel derecho se rellena con negro y la grabación continúa sin interrupción (degradación silenciosa). Debe desactivarse en corridas *batch* o *headless* donde el editor no está en pantalla.
`VIEWPORT_WINDOW_TITLE`	`"UnrealEditor"`	`"UnrealEditor"`	Subcadena del título de la ventana de Unreal Engine que utiliza `Viewport-Capture` para localizar la ventana. La comparación ignora mayúsculas y minúsculas. Se selecciona la primera ventana visible que contenga la subcadena y tenga dimensiones superiores a 100×100 px. En UE5 el título suele seguir el patrón `"UnrealEditor – <NombreProyecto>"`.

Variable	Valor nominal	Default si ausente	Justificación y efecto
`VIEWPORT_REGION`	*(vacío)*	*(detección automática)*	Coordenadas absolutas de pantalla en píxeles para capturar una región fija, con formato `left,top,right,bottom`. Cuando se define, `ViewportCapture` omite la búsqueda por título y captura exactamente esa región, lo que permite recortar el *chrome* del editor y mostrar únicamente el panel de viewport interior. Ejemplo para un viewport que ocupa la mitad derecha de un monitor de 1920×1080 px: `VIEWPORT_REGION=960,0,1920,1080`. Si está vacío, la localización se delega a `VIEWPORT_WINDOW_TITLE`.

A6.4 Estructura del repositorio y artefactos de corrida (`airsim-runs/`)

Cada vuelo ejecutado en simulación genera una carpeta autocontenida que ninguna otra corrida sobrescribe. Desde WebDCS (`main.py`), la carpeta es `airsim-runs/<MISSION_ID>-<TIMESTAMP>/`; el runner experimental la ubica en la jerarquía del diseño, `<lote>/<escenario>/<brazo>/<estrategia>/seed_<N>_<timestamp>/` (cap. 10, §10.9.1). La lista completa de archivos y campos está en el capítulo 4 (§4.4); los principales son:

```
seed_<N>_<ts>/
├── seed_<N>_<ts>.summary.json      <- Indicadores consolidados de la corrida
├── seed_<N>_<ts>.summary_by_wp.csv <- Indicadores por objetivo
├── seed_<N>_<ts>.jsonl             <- Traza ciclo a ciclo con el DroneState completo
├── seed_<N>_<ts>.csv               <- La misma traza en columnas (state.*)
├── seed_<N>_<ts>.distmin.ndjson    <- Muestras de la auditoría de distancia mínima (1 Hz)
├── seed_<N>_<ts>.webm              <- Video de la cámara frontal con overlay
└── seed_<N>_<ts>.viewer.html       <- Visor de auditoría autocontenido
```

A6.4.1 Estructura del archivo de resumen (`.summary.json`)

Contiene la identificación de la corrida y sus indicadores. Ejemplo de una corrida `slm:depth` en `citysim_pilot` (valores ilustrativos; `route_histogram` y `route_plan` abreviados):

```
{
  "scenario": "citysim_pilot",
  "seed": 1,
  "arm": "slm",
  "code_version": "<hash>",
  "deadlock_strategy": "depth",
  "strategic_source": "depth",
  "route_plan": {"mode": "vlm", "judge": "patches", "model": "lfm2.5-vl-1.6b:q4km-ctx8k",
                 "map": "citysim_ortho.png", "vias": 7, "legs_rerouted": 4},
  "success": false,
  "cycles": 2014,
  "duration_s": 480.66,
  "path_length_m": 702.68,
  "collisions": 0,
  "min_obstacle_dist_m": 6.562,
  "min_obstacle_dist_source": "in_flight_thread",
  "min_obstacle_dist_samples": 477,
  "slm_invocations": 255,
  "deliberation_rate": 0.127,
  "slm_fallback_rate": 0.0,
  "slm_timeout_rate": 0.0,
  "route_histogram": {"reactive": 1585, "evasive": 419, "girar_90": 9, "tactical": 1},
  "deadlock_events": 1,
  "deep_scan_events": 1,
  "deep_scan_resolution_rate": 0.0,
  "deep_scan_fallback_rate": 1.0,
  "vertical_escapes": 0,
  "freeze_recoveries": 0
}
```


Nota sobre trazabilidad: el campo `code_version` almacena el hash de Git del commit sobre el que se ejecutó el vuelo. Si había cambios sin commitear en `airsim-loop/`, `config/`, `airsim-plan/` o `experiments/`, lleva el sufijo `-dirty` (`src/logging/code_version.py`): el hash solo identifica el código con el árbol limpio. El runner no arranca un lote con el árbol sucio, y una corrida `-dirty` no entra al análisis (cap. 10, §10.9.3 y §10.12.2).

A6.4.2 Estructura de la traza ciclo a ciclo (`.jsonl`)

Cada línea del archivo es un registro JSON por ciclo del lazo (5 Hz), escrito por `FlightLogger`. Los campos principales son `t` y `cycle`; `arm`, `scenario` y `seed`; posición NED, velocidad y actitud; `route` y `action`; el `obstacle_field` completo por sector; latencias del ciclo; el bloque `slm` en los ciclos con una consulta del lazo lento; colisión; objetivo activo y distancia; y `state`, el `DroneState` completo serializado (cap. 4, §4.4).

A6.5 Manifiestos de misión y diseño de escenarios evaluados (Tiers 0, 1 y 2)

Los planes de vuelo se definen como manifiestos JSON ubicados en `airsim-plan/missions/flightplans/`. A continuación se especifican los escenarios representativos empleados en la tesis:

A6.5.1 Manifiesto Tier 1: Perímetro semiurbano (`townsim_clear.json`)

Evalúa el crucero perimetral sobre el mapa `TownSim` (Downtown West Modular Pack) a una altitud de tránsito de $-30\ \mathrm{m}$ (30 m AGL):

```
{
  "mission_id": "TOWNSIM_CLEAR",
  "summary": "Tier 1: perímetro completo de townsim_calib.png a altitud de tránsito (-30m),
sin cruzar ninguna fachada. Rol análogo a minisim_clear para Tier 0: control/smoke-test de
crucero largo sin obstáculos deliberativos.",
  "map": "townsim_ortho.png",
  "waypoints": [
    { "x": 0.0,    "y": 0.4,   "z": -10.0, "label": "WP_1" },
    { "x": 0.4,    "y": -71.6, "z": -30.0, "label": "WP_2" },
    { "x": -145.2, "y": -70.8, "z": -30.0, "label": "WP_3" },
    { "x": -146.8, "y": 83.6,  "z": -30.0, "label": "WP_4" },
    { "x": 1.2,    "y": 84.8,  "z": -30.0, "label": "WP_5" },
    { "x": 0.0,    "y": 27.2,  "z": 0.0,   "label": "WP_6" }
  ]
}
```

A6.5.2 Manifiesto Tier 1: Bloqueo frontal forzado (`townsim_calib_cruce_frontal.json`)

Diseñado para forzar una colisión inminente contra un edificio alto con aproximación frontal ortogonal:

```
{
  "mission_id": "TOWNSIM_CALIB_CRUCE_FRONTAL",
  "summary": "Tier 1: trayectoria ortogonal directa hacia la fachada principal de la manzana
central. Obliga al lazo táctico a detectar TTC crítico, disparar giro o evasión de 90° o
deliberar con el SLM para rodear la estructura.",
  "map": "townsim_ortho.png",
  "waypoints": [
    { "x": 0.0,   "y": 0.0,   "z": -10.0, "label": "WP_START" },
    { "x": -72.0, "y": 0.0,   "z": -10.0, "label": "WP_OBSTACULO_FRONTAL" },
    { "x": -72.0, "y": 80.0,  "z": -10.0, "label": "WP_DESVIO_LATERAL" }
  ]
}
```

A6.5.3 Manifiesto Tier 2: perímetro de manzana en rampa (`citysim_clear.json`)

Cuatro waypoints, ~420 m, sobre el perímetro de una manzana de `CitySim`. La altitud crece en rampa a lo largo del perímetro (−10, −30 y −50 m) y el último tramo vuelve a −10 m sobre el punto de partida; los cambios de altitud se hacen en diagonal, combinados con el avance horizontal:

```
{
  "mission_id": "CITYSIM_CLEAR",
  "summary": "Tier 2 (escenario base): perimetro de una manzana del grid regular de CitySim,
4 waypoints. Altitud creciente en rampa (-10, -30, -50 m) y retorno a -10 m sobre el punto de
partida.",
  "map": "citysim_ortho.png",
  "waypoints": [
    { "x": 0.0,   "y": 75.0, "z": -10.0, "label": "WP_1" },
    { "x": 130.0, "y": 75.0, "z": -30.0, "label": "WP_2" },
    { "x": 130.0, "y": 0.0,  "z": -50.0, "label": "WP_3" },
    { "x": 0.0,   "y": 0.0,  "z": -10.0, "label": "WP_4" }
  ]
}
```

A6.5.4 Manifiesto Tier 1: Travesía del corredor arbolado (`townsim_ini.json`)

Ocho waypoints, ~340 m. Incluye `WP_0b_SOBRE_PLAZA` (cruza la plaza a −22 m para evitar la canopy de los árboles del patio) y `WP_1b_PASO_MOLDURA` (desvía 8 m al este del saliente de cornisa del edificio oeste). Justificación en el cap. 10, §10.3.2.

```
{
  "mission_id": "TOWNSIM_INI",
  "map": "townsim_ortho.png",
  "waypoints": [
    { "x": 0.0, "y": 0.0, "z": -30.0, "label": "WP_0_ASCENSO" },
    { "x": -60.0, "y": 5.0, "z": -22.0, "label": "WP_0b_SOBRE_PLAZA" },
    { "x": -75.0, "y": 10.0, "z": -10.0, "label": "WP_1_ENTRADA_NORTE" },
    { "x": -70.0, "y": 0.0, "z": -10.0, "label": "WP_1b_PASO_MOLDURA" },
    { "x": -75.0, "y": -35.0, "z": -10.0, "label": "WP_2_CENTRO_CORREDOR" },
    { "x": -75.0, "y": -70.0, "z": -10.0, "label": "WP_3_SALIDA_SUR" },
    { "x": 0.0, "y": -70.0, "z": -30.0, "label": "WP_4_CLIMB_ESTE" },
    { "x": 0.0, "y": 0.0, "z": -10.0, "label": "WP_5_RETORNO" }
  ]
}
```

A6.6 Protocolo de reproducción experimental paso a paso

Para replicar de forma independiente los experimentos documentados en el capítulo 10, siga la secuencia metodológica descrita a continuación:

Paso 1: Verificación de software y dependencias

1. Sistema operativo: Windows 10/11 (64-bit).
2. Entorno conda `airsimenv` (Python 3.10, PyTorch con CUDA y `transformers` para la red de profundidad; en la estación de la tesis, `D:\TesisMCD\Airsimenv`):

   ```
   conda activate airsimenv
   cd airsim-loop
   pip install -r requirements.txt
   ```

   La red de profundidad (`DEPTH_MODEL`) se descarga de Hugging Face la primera vez que se compila el grafo.
3. Verificar la disponibilidad del plugin binario de Cosys-AirSim en el proyecto de Unreal Engine 5.5 correspondiente (MiniSim, TownSim o CitySim).

Paso 2: Instalación del archivo `settings.json`

Guardar el contenido de §A6.1 como `settings.json` en la carpeta `Documents\AirSim` del usuario (o en `OneDrive\Documents\AirSim` si Documentos está redirigida), antes de abrir el proyecto de Unreal Engine.

Paso 3: Inicialización del servidor de modelos (Ollama, misma GPU)

1. Ejecutar `startOllama.ps1` (raíz `D:\TesisMCD`, fuera del repositorio). El script lee el modelo de `config/.env`; reinicia Ollama con `OLLAMA_MODELS` apuntando a la carpeta de modelos de la aplicación, `OLLAMA_KEEP_ALIVE=-1` (el modelo no se descarga entre corridas) y `OLLAMA_CONTEXT_LENGTH=8192`; cierra los `llama-server` huérfanos (en Windows sobreviven al cierre de `ollama.exe` y retienen VRAM); descarga el GGUF base y crea el tag de 8 k si faltan, y precarga el modelo de vuelo:

   ```
   powershell -ExecutionPolicy Bypass -File D:\TesisMCD\startOllama.ps1
   ```

   Con la configuración por defecto (`STRATEGIC_SOURCE = depth`, `DEADLOCK_STRATEGY = depth`) el modelo solo se usa para el plan de ruta previo; si el lote corre con `--route-plan off`, Ollama puede quedar cerrado y su VRAM queda para Unreal.
2. Verificar que el modelo quedó residente:

   ```
   curl http://127.0.0.1:11434/api/ps
   ```

Paso 4: Lanzamiento de la simulación

1. Abrir el proyecto de Unreal Engine deseado (e.g., `TownSim`).
2. Presionar **Play** (PIE) o ejecutar el binario *standalone*.

3. Constatar que en la esquina superior aparezca la notificación de conexión de AirSim en el puerto 41451.

Paso 5: Ejecución de los lotes (*Runner*)

Los seis lotes del diseño experimental (cap. 10, §10.5.3) se lanzan con los scripts versionados en `experiments/`, que fijan escenario, celdas, semillas, presupuesto (1200 s / 6000 ciclos) y modo del plan de ruta, y usan el intérprete de `AIRSIM_ENV_DIR`:

```
cd experiments
.\run_lote3_townsim_ini.ps1 -Pilot    # piloto de validación
.\run_lote3_townsim_ini.ps1           # lote completo
.\run_lote3_townsim_ini.ps1 -Resume   # reanudar un lote cortado
```

Cada script llama a `runner.py`, que también puede usarse directamente:

```
python experiments/runner.py \
    --scenarios ../airsim-plan/missions/flightplans/townsim_ini.json \
    --cells slm:depth fsm:depth fsm:blind reactive \
    --seeds 73 79 83 89 97 101 103 107 109 113 \
    --max-seconds 1200 --max-cycles 6000 \
    --route-plan vlm \
    --out-dir ../airsim-runs/produccion/v4/lote3_townsim_ini
```

Las celdas se alternan dentro de cada semilla, con el orden rotado por semilla. Sin `--deadlock-strategies` ni `--strategic-sources` se usan los valores de `config/.env` (`depth` / `depth`). `--route-plan vlm` planifica la ruta con el VLM una vez por escenario y requiere el servidor de Ollama y un mapa registrado (TownSim o CitySim). Cada corrida se cierra con `finalize_run` (cap. 4, §4.4); una corrida interrumpida se cierra con `python experiments/finalize_run.py --all <raíz>`.

Paso 6: Procesamiento de datos y consolidación estadística

Una vez finalizados los lotes, generar las tablas estadísticas:

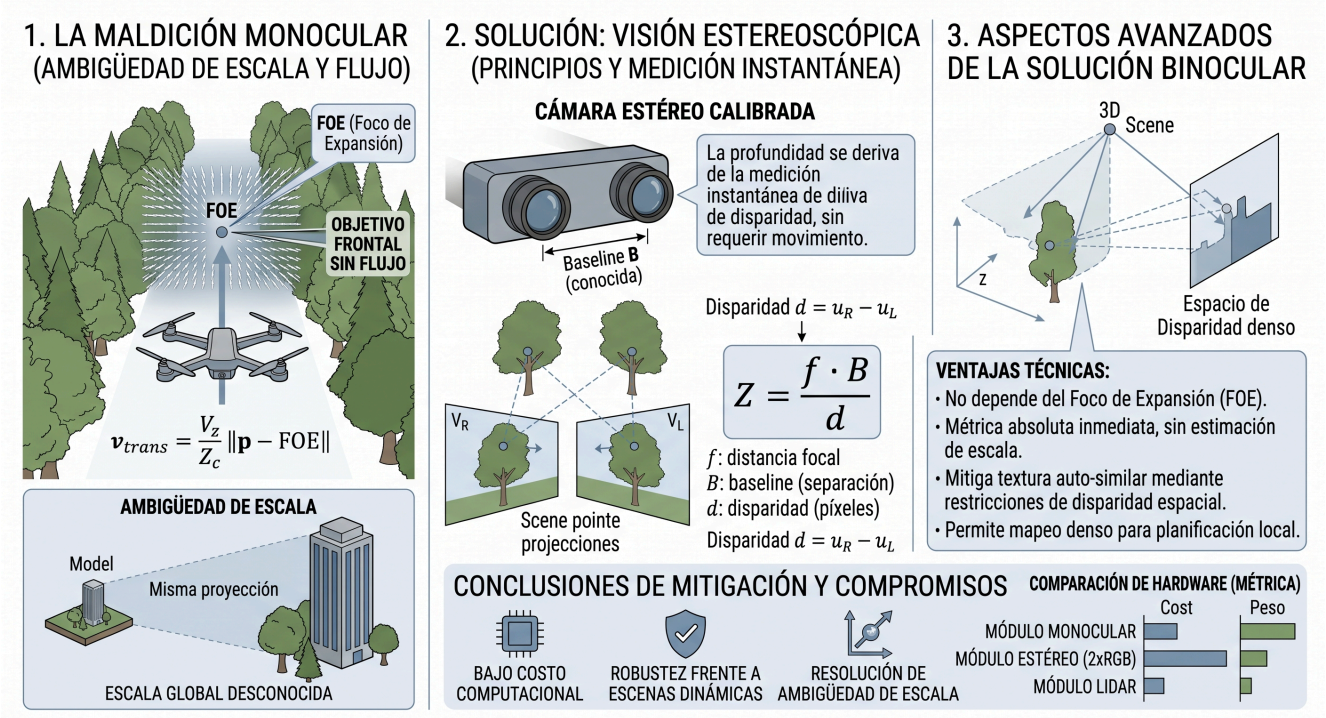
```
D:\TesisMCD\airsimenv\python.exe ..\airsim-loop\experiments\analyze_lotes.py ..\airsim-runs\produccion\v4 --family 18
```

El script escribe `analisis_lotes.md` y dos CSV en la raíz de los lotes: por celda, éxito con IC de Wilson, SPL, tiempo de las misiones exitosas y distancia mínima; y las 18 comparaciones primarias con delta de Cliff, IC *bootstrap* y prueba de permutación exacta sobre rangos, con corrección de Bonferroni (cap. 10, §10.7).

Anexo 7: La "maldición monocular" en pasajes densamente obstruidos y mitigación con visión estereoscópica

Este anexo desarrolla, con mayor rigor que el que admite el cuerpo del capítulo de resultados (§11.4.2b), el mecanismo físico por el cual la percepción monocular pura falla de forma estructural — no incidental — en corredores densamente arbolados como `townsim_ini`, y evalúa la visión estereoscópica como la mitigación de menor costo y mayor factibilidad de despliegue antes de recurrir a sensores activos más caros (LiDAR, radar).

El razonamiento aquí presentado complementa, sin sustituir, los fundamentos matemáticos del Anexo 5 (flujo óptico, FOE, TTC): mientras el Anexo 5 formaliza *cómo* el sistema estima profundidad relativa a partir de una sola cámara, este anexo formaliza *por qué* esa estimación colapsa estructuralmente en ciertas geometrías de escena, y qué alternativa de bajo costo resuelve el colapso sin abandonar el principio de percepción pasiva declarado en §1.2.



A7.1 La maldición monocular: dos componentes distintos

El término coloquial "maldición monocular" (*monocular curse*) agrupa dos limitaciones físicas independientes de la reconstrucción de profundidad a partir de una única cámara. Es importante separarlas porque tienen causas y mitigaciones distintas.

A7.1.1 Ambigüedad de escala

Un sistema de estructura a partir del movimiento (*Structure from Motion*, SfM) o SLAM monocular reconstruye la geometría de la escena **hasta un factor de escala desconocido**: a partir de una secuencia de imágenes de una sola cámara no es posible distinguir, por ejemplo, un edificio grande visto de lejos de una maqueta pequeña vista de cerca — ambas producen exactamente la misma secuencia de proyecciones 2D salvo un factor de escala global. Para fijar esa escala se requiere información externa: una unidad de medición inercial (IMU) integrada, el tamaño conocido de un objeto en la escena, o un segundo punto de vista simultáneo.

A7.1.2 Dependencia estructural del movimiento y flujo nulo en el eje de avance

El segundo componente es geométrico y no se resuelve con más cómputo ni mejor calibración: **sin traslación no hay paralaje**, y el paralaje es la única fuente de información de profundidad de una cámara monocular en movimiento (Anexo 5, §A5.2–§A5.4). Peor aún, el paralaje no es uniforme en el campo visual: según la relación deducida en el Anexo 5,

$$\|v_{\text{trans}}(x, y)\| = \frac{V_z}{Z_c} \|p - \text{FOE}\|$$

la magnitud del flujo traslacional es **proporcional a la distancia del píxel al Foco de Expansión (FOE)**. Los puntos que están sobre o cerca del FOE —es decir, los que están *directamente adelante del vehículo, en la dirección exacta en que se necesita evidencia para evitar una colisión frontal*— son estructuralmente los que producen menos señal, sin importar cuán cerca estén. Esto no es un defecto del estimador de `flow_ttc.py`: es una propiedad de la proyección perspectiva, presente en cualquier algoritmo de flujo óptico o SfM monocular.

> [!IMPORTANTE] La maldición monocular no es "el sensor da poca información en general" sino, más precisamente, "el sensor da información nula justo donde más se la necesita". Un obstáculo lateral que el dron bordea genera flujo grande y confiable; el mismo obstáculo, si está centrado en la trayectoria de aproximación, tiende asintóticamente a flujo cero a medida que el dron se acerca en línea recta.

A7.2 Por qué un corredor arbolado es el caso adversarial por diseño

El escenario `townsim_ini` (cap. 11, §11.4.2b) — corredor peatonal recto, volado a z=−10 m, bajo la copa de los árboles — combina ambos componentes de la maldición monocular con un tercer factor agravante propio de la vegetación:

1. **Geometría del corredor → FOE sobre el obstáculo.** Un corredor recto y angosto flanqueado y techado por follaje fuerza a que el FOE nominal (centro de la imagen, para vuelo recto) caiga exactamente sobre la masa de vegetación que bloquea el paso. No es un obstáculo lateral que

pueda esquivarse leyendo flujo periférico: es el obstáculo *en el eje de avance*, en la región de mínima señal por construcción geométrica (§A7.1.2).

2. **Textura auto-similar → problema de apertura agravado.** Ramas y hojas son, para un estimador de correspondencia por ventanas (Farnebäck o DIS; Anexo 5, §A5.7), prácticamente indistinguibles de sus vecinos inmediatos. El *aperture problem* clásico de flujo óptico —la imposibilidad de determinar la componente de movimiento paralela a un borde sin textura transversal— se agrava severamente en follaje denso, generando vectores de flujo ruidosos o directamente espurios incluso en regiones alejadas del FOE, degradando la confianza (`confidence`) de las celdas del `ObstacleField` (Anexo 5, §A5.6.1) en todo el campo, no solo en el centro.
3. **Follaje dinámico → violación de la hipótesis de escena estática.** El viento mueve ramas y hojas de forma incoherente con el movimiento propio del dron, introduciendo flujo espurio que ninguna derotación por IMU (Anexo 5, §A5.3) puede compensar, porque esa derotación asume explícitamente una escena rígida y estacionaria.

La consecuencia esperable en el corredor arbolado de `townsim_ini` es un régimen de **deadlock crónico** sin colisiones: el sistema no avanza porque no puede confiar en su propia percepción frontal, y no colisiona precisamente porque esa falta de confianza lo mantiene en modo cauteloso. Un barrido puede resolver cada deadlock individual, pero si la tasa de generación de nuevos deadlocks excede la capacidad de avance neto, ningún ajuste de la capa deliberativa (Anexos 3 y 4) lo resuelve, porque el cuello de botella está en la capa de percepción, no en la de decisión. El lote 3 del diseño experimental (cap. 10, §10.5.3) mide si este régimen aparece y si la profundidad estimada lo atenúa.

A7.2.1 ¿Ayudaría SLAM monocular disperso o semi-denso?

Es razonable preguntarse si integrar evidencia geométrica a lo largo de múltiples vistas —en vez de depender de un par de frames instantáneo, como hace el flujo óptico— mitigaría el problema. Un sistema de SLAM monocular disperso (ORB-SLAM, Mur-Artal et al., 2015) construye una estructura 3D persistente mediante *bundle adjustment*, lo que en principio permite razonar sobre espacio libre acumulado en lugar de sobre la ambigüedad de un instante puntual, y habilitaría detección de brechas (*gap detection*) sobre un mapa en vez de evasión puramente reactiva.

Sin embargo, esta mitigación no es gratuita en el caso específico de follaje denso, por tres razones que conviene declarar explícitamente para no sobrevender la propuesta como línea de trabajo futuro:

- **Escena no estática.** El SLAM basado en *features* asume implícitamente un mundo rígido; ramas moviéndose con el viento violan esa hipótesis e inducen asociaciones de datos erróneas (*false loop closures*, matches entre ramas distintas con apariencia similar).
- **Geometría subpíxel.** Las ramas finas están por debajo de la resolución efectiva de un mapa de *landmarks* disperso: un SLAM disperso simplemente no las registra como *features* estables, y un SLAM denso las "promedia" con el fondo, perdiendo precisión justo en el tipo de obstáculo donde más importa (el caso típico, además, donde hasta un LiDAR de retorno único falla por escasez de puntos de impacto).

- **Costo computacional.** Un SLAM denso monocular con la robustez suficiente para compensar estos dos problemas (p. ej., de la familia de DROID-SLAM) es sustancialmente más pesado en cómputo que el flujo óptico clásico usado en este trabajo, lo cual entra en tensión directa con el objetivo de mantener el presupuesto de CPU compartido con el SLM (cap. 6, §6.1, punto 1).

La conclusión razonable no es descartar SLAM monocular como línea futura, sino ubicarlo como mitigación de la ambigüedad de correspondencia frame-a-frame, no como solución al problema de fondo del follaje dinámico y las estructuras subpíxel — que siguen siendo un caso límite conocido en la literatura de SLAM visual, y no un problema exclusivo del enfoque de flujo óptico adoptado en este trabajo.

A7.3 Estéreo como mitigación directa de ambos componentes

Un par estéreo calibrado —dos cámaras RGB con *baseline* (separación) fija y conocida— resuelve los dos componentes de la maldición monocular (§A7.1) simultáneamente, y lo hace a partir de un **único instante**, sin depender de que el vehículo se mueva:

$$Z = \frac{f \cdot B}{d}$$

donde f es la distancia focal, B la línea de base entre cámaras y d la disparidad (el desplazamiento horizontal en píxeles del mismo punto físico entre ambas imágenes). Dado que B es conocida por construcción (a diferencia de la traslación del vehículo entre dos frames, que hay que estimar o asumir), la profundidad Z recuperada es **métrica**, sin el factor de escala indeterminado de SfM/SLAM monocular (§A7.1.1) — resolviendo el primer componente de la maldición.

Sobre el segundo componente (§A7.1.2): la correspondencia estéreo es **espacial y simultánea** (misma toma, dos cámaras) en lugar de **temporal** (dos instantes de la misma cámara). Esto tiene una consecuencia directa y favorable para el caso de `townsim_ini`:

- La disparidad de un obstáculo centrado en la trayectoria de vuelo **no depende de su posición respecto de ningún FOE** — la disparidad es función de la distancia Z y la geometría fija del par estéreo, no de hacia dónde se mueve el vehículo. El obstáculo *directamente adelante*, que es el caso de flujo nulo en monocular (§A7.1.2), es exactamente igual de medible en estéreo que un obstáculo lateral.
- Al no requerir correspondencia entre instantes distintos, el estéreo es menos frágil frente a escena dinámica: ramas moviéndose con el viento entre el momento de captura de la cámara izquierda y la derecha (disparo sincronizado) no introducen el mismo tipo de error que el flujo óptico acumulado entre dos frames separados por Δt (Anexo 5, §A5.3.3).

La textura repetitiva del follaje (§A7.2, punto 2) **no desaparece** en estéreo —la ambigüedad de correspondencia lateral entre las dos vistas sigue siendo un desafío—, pero es en general un problema menos frágil que en flujo óptico, porque no depende de que exista movimiento suficiente del vehículo para generar paralaje utilizable: la línea de base está siempre presente, sea que el dron esté en hover, girando o avanzando lentamente. Esto es relevante porque el régimen de "cautela extrema → movimiento mínimo → menos evidencia" descrito en §11.4.2b (el ciclo de retroalimentación negativa que agrava el deadlock crónico) deja de aplicar: el par estéreo entrega profundidad métrica incluso cuando el vehículo está casi detenido.

A7.4 Costo/beneficio frente a LiDAR emulado (PlanarDepth) y factibilidad en AirSim

La arquitectura de este trabajo excluye deliberadamente sensores activos pesados o costosos por principio de diseño (§1.2; Anexo 5, introducción). Frente a esa restricción, un LiDAR simulado (p. ej. el sensor `DepthPlanar` de Cosys-AirSim) sería el "sensor ideal" pero representa un salto de hardware que la mayoría de los UAVs urbanos pequeños de la clase objetivo de este trabajo no llevan, y cuyo costo, peso y consumo energético son órdenes de magnitud mayores que los de una segunda cámara RGB.

Un par estéreo es la opción intermedia razonable: mucho más barato y liviano que un LiDAR real, y trivialmente emulable en AirSim agregando una segunda cámara RGB en `settings.json` con un offset fijo conocido en el eje X o Y del cuerpo del dron (baseline conocida por construcción, sin necesidad de calibración extrínseca adicional en simulación). El procesamiento estándar sería:

1. Rectificación estéreo (alinear ambas imágenes a un plano epipolar común).
2. Matching de disparidad — como línea de base clásica, `cv2.StereoSGBM` (OpenCV), con la opción de escalar a un método basado en aprendizaje (p. ej. de la familia RAFT-Stereo) si se requiere mayor precisión y el presupuesto de cómputo lo permite.
3. Conversión de disparidad a profundidad métrica vía $Z = fB/d$, alimentando el mismo contrato `ObstacleField` (Anexo 5, §A5.6) ya consumido por el router de política (cap. 8) y el SLM, sin necesidad de rediseñar la interfaz entre percepción y decisión.

Nota metodológica para una futura implementación: si se incorpora este canal, la profundidad debe derivarse del *matching* estéreo real (con su ruido de correspondencia y sus errores en zonas de textura ambigua) y no leerse directamente del `DepthPlanar` de referencia que AirSim provee para otros propósitos de depuración. Usar la profundidad de referencia directamente compararía el sistema contra un LiDAR ideal en lugar de contra el estéreo real que se busca evaluar, invalidando la comparación de costo/beneficio que motiva la propuesta.

A7.5 Ningún sensor es inmune al follaje denso: límites del estéreo y también del LiDAR

Es importante no presentar el estéreo como una "bala de plata" que resuelve el caso `townsim_ini` sin costo residual. El follaje denso es un escenario adversarial *per se*, no solo para la cámara monocular: degrada, por razones físicas distintas en cada caso, tanto al estéreo propuesto en este anexo como al LiDAR que la arquitectura de este trabajo excluye por principio de diseño (§1.2). Vale la pena declarar ambos límites explícitamente, porque cambian la expectativa razonable de lo que "agregar un sensor mejor" resolvería en un corredor arbolado.

A7.5.1 Por qué el estéreo también puede fallar en `townsim_ini`

- **El rango útil depende linealmente de la línea de base.** En un UAV pequeño, la baseline física disponible entre dos cámaras es corta (típicamente centímetros), y el error de profundidad estimada **crece con el cuadrado de la distancia** ($\sigma_Z \propto Z^2/(fB)$, derivable por propagación de error de la ecuación de disparidad de §A7.3). A las velocidades de crucero típicas de este trabajo, el horizonte de detección confiable puede resultar corto en comparación con un LiDAR de rango largo — precisamente en un corredor donde la evasión temprana importa.
- **Ambigüedad de correspondencia lateral persiste en follaje.** El matching estéreo no depende del FOE (§A7.3), pero sigue dependiendo de encontrar correspondencias únicas entre las dos vistas. Sobre textura repetitiva y auto-similar como ramas y hojas, múltiples parches de la imagen izquierda son plausiblemente compatibles con múltiples parches de la derecha: el problema de apertura de flujo óptico (§A7.2, punto 2) reaparece en estéreo como ambigüedad de disparidad, produciendo mapas de profundidad ruidosos o con "picos" espurios en zonas de vegetación densa, incluso con un algoritmo de matching robusto.
- **Oclusiones diferenciales.** Ramas muy finas pueden ser visibles desde una cámara del par y quedar ocultas desde la otra (por paralaje entre los dos puntos de vista), generando huecos o errores locales de matching en exactamente el tipo de obstáculo delgado que ya es difícil para cualquier sensor de este trabajo (Anexo 5, §A5.6.2). Cuanto más denso el follaje, más frecuentes estas oclusiones — el mismo factor que ayuda al estéreo (baseline fija, sin depender de FOE) es el que introduce este modo de falla propio, ausente en monocular.

A7.5.2 Por qué un LiDAR tampoco resolvería `townsim_ini` sin matices

Aunque fuera de alcance por diseño (§1.2), vale la pena precisar por qué un LiDAR real —el "sensor ideal" de referencia en §A7.4— no es una solución automática al mismo escenario, para no dejar la impresión de que cualquier sensor activo lo resolvería trivialmente:

- **Geometría subpíxel y retornos escasos.** Las ramas finas tienen una sección transversal menor que la divergencia del haz láser a distancias de varios metros; el pulso puede no intersectar ninguna rama, o intersectar el borde de una, generando retornos débiles o ausentes. Este es el mismo problema

mencionado en §A7.2 para SLAM denso: la estructura fina de la vegetación está, en buena medida, *por debajo de la resolución efectiva* de cualquier sensor de barrido disperso, no solo del monocular.

- **Retornos múltiples y "mixed pixels".** Un solo pulso de LiDAR puede atravesar parcialmente el follaje y golpear más de una superficie (una hoja cercana y una rama más lejana en la misma línea de vista), produciendo un retorno mezclado cuya distancia reportada no corresponde limpiamente a ningún obstáculo real — un problema bien documentado en escaneo LiDAR forestal y de vegetación urbana.
- **Reflectividad baja y variable.** Las hojas tienen reflectividad óptica baja y anisótropa (varía con el ángulo de incidencia y la humedad de la superficie), lo que reduce la relación señal-ruido de los retornos y puede dejar zonas del follaje sin cobertura efectiva de puntos, incluso dentro del rango nominal del sensor.
- **El costo real no es solo económico.** Incluso si el LiDAR "viera" más geometría que el estéreo en este escenario, la nube de puntos resultante en vegetación densa suele ser rala y ruidosa precisamente donde más importa (el eje de avance), por lo que la mejora práctica sobre el modo de falla de `townsim_ini` es menor de lo que su costo, peso y consumo energético (§A7.4) sugerirían a priori.

> [!IMPORTANTE] La conclusión no es que ningún sensor ayude, sino que el follaje denso es un caso límite que degrada a *toda* la familia de sensores de rango (monocular, estéreo, LiDAR), cada uno por un mecanismo físico distinto. El estéreo sigue siendo la mejora de menor costo y mayor impacto marginal para el modo de falla específico identificado en §A7.2 (flujo nulo cerca del FOE), no porque sea inmune al follaje, sino porque elimina selectivamente la causa dominante del fracaso monocular sin heredar todas las limitaciones de un LiDAR en el mismo entorno.

A7.6 Síntesis y ubicación en el trabajo futuro

La visión estereoscópica resuelve de forma directa y con bajo costo computacional los dos componentes formales de la maldición monocular (§A7.1): fija la escala métrica sin depender de IMU ni de referencias externas, y elimina la dependencia del flujo respecto de la posición del FOE, que es precisamente el mecanismo identificado en §A7.2 como causa estructural del fracaso en `townsim_ini` (cap. 11, §11.4.2b). Es, además, más fácil de justificar como próximo paso concreto que un LiDAR emulado, porque emula una limitación de hardware real y económicamente accesible (un segundo sensor RGB de bajo costo) en lugar de saltar directamente a una clase de sensor que la mayoría de los UAVs urbanos pequeños de este dominio de aplicación no incorpora.

En la jerarquía de mejoras de percepción de este trabajo (cap. 12, §12.4), el estéreo se propone como el **paso intermedio natural**: menor costo de implementación y de hardware que un SLAM monocular semi-denso o un LiDAR real, y con un impacto directo sobre el modo de falla documentado en corredores arbolados. El SLAM monocular (§A7.2.1) queda como una línea posterior, complementaria y no sustitutiva, útil si además del canal de profundidad instantánea que aporta el estéreo se requiere robustez de localización y mapeo ante escena dinámica extendida en el tiempo.

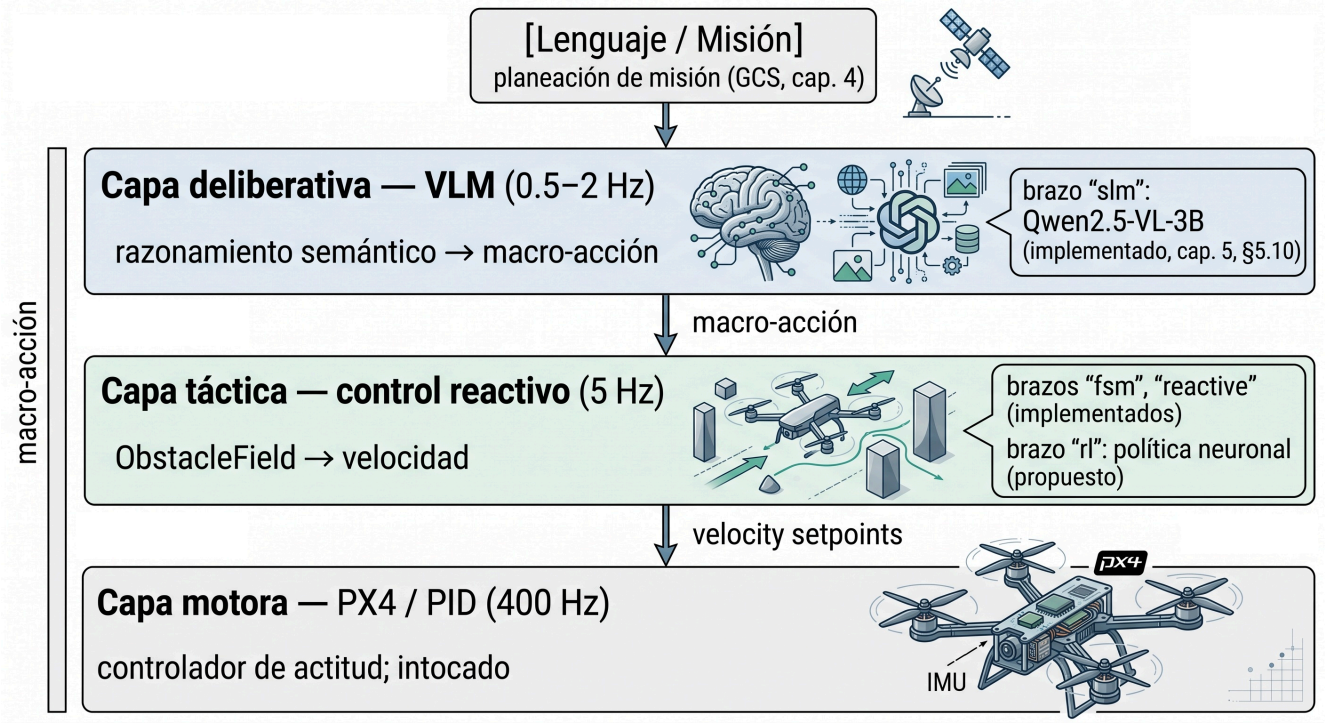
Anexo 8: Política de Navegación Visual por Aprendizaje por Refuerzo — Diseño Propuesto como Cuarto Brazo

Este anexo desarrolla el diseño técnico de un cuarto brazo de comparación experimental (`rl`) basado en **aprendizaje por refuerzo visual (*Reinforcement Learning*, RL)**, propuesto como trabajo futuro en §12.4. El brazo `rl` no está implementado en el sistema evaluado en el capítulo 11; este anexo especifica su arquitectura, espacio de observaciones y acciones, función de recompensa, protocolo de entrenamiento y posición en el conjunto de hipótesis experimentales de la tesis, de forma que sirva como base técnica para la etapa de implementación.

El planteamiento es complementario — no alternativo — a los avances del Anexo 3 (LoRA) y del Anexo 4 (gramáticas restringidas): mientras esos dos anexos proponen especializar la capa deliberativa existente (VLM), el brazo `rl` propone reemplazar la capa táctica completa por una política neuronal entrenada por interacción directa con el entorno, en el mismo nivel de abstracción que ocupa actualmente `evasive_node` + `reactive_node`.

A8.1 Posición en la arquitectura jerárquica

El sistema implementado en esta tesis opera en tres capas de abstracción temporal diferenciada:



El brazo `rl` ocupa la **capa táctica**: recibe el frame RGB actual, el `ObstacleField` producido por `FlowTTCEstimator` y la telemetría de guiado (`waypoint_guidance`), y produce directamente una macro-acción del mismo vocabulario que usan `fsm_node` y `evasive_node`. La capa estratégica del VLM sigue activa en modo jerárquico: puede inyectar sub-metas en coordenadas del mundo (`inject_corner`) y resolver atascos duros con el barrido `deep_vlm`, igual que con los brazos existentes (cap. 5, §5.10, §5.12).

Esta posición garantiza que la comparación de los cuatro brazos (`reactive`, `fsm`, `slm`, `rl`) mida exclusivamente **quién elige mejor la táctica**, no quién tiene mejor lógica de seguridad ni mejor infraestructura de actuación, ya que los cuatro comparten:

- El mismo `ObstacleField` como entrada perceptual.
- El mismo `action_to_command()` como traductor a cinemática.
- Las mismas salvaguardas de `motor_node` (corrección de altitud, target_yaw).
- El mismo `WaypointTracker` para el guiado a waypoints.

A8.2 Espacio de observaciones

Se propone un espacio de observación de tres componentes, idénticos a los que ya produce el lazo en cada ciclo (sin nuevos sensores):

```
obs = {
    "rgb":       np.ndarray,   # shape (H, W, 3) — frame RGB del ciclo actual
    "ttc_field": np.ndarray,  # shape (3, 3, 3) — ObstacleField: [ttc, occ, conf] × 9 celdas
    "nav":       np.ndarray,   # shape (6,) — [dist_wp, bearing_err, vx, vy, vz, alt]
}
```

`rgb` (entrada visual). El mismo frame que `capture_node` ya produce. Para el entrenamiento se escala a 84 × 84 px, suficiente para que una CNN pequeña extraiga patrones de textura y borde sin el costo de resolución completa. Se puede extender a `frame_history` (dos frames consecutivos, igual que el VLM) para que la política infiera dirección de movimiento.

`ttc_field` (campo de obstáculos serializado). El `ObstacleField` se serializa como un tensor 3×3×3: para cada celda de la grilla, los tres valores son TTC, ocupación y confianza. Esta representación preserva la estructura espacial del campo sin exponer al modelo los internos del estimador de flujo óptico. La política aprende implícitamente el significado de cada componente durante el entrenamiento, igual que el VLM lo aprende del texto de `summary_text()`.

`nav` (estado de guiado). Distancia al waypoint activo, error de rumbo (`bearing_err_deg`), velocidades actuales y altitud. Estos seis escalares proporcionan el contexto cinemático necesario para que la política distinga "atascado contra obstáculo frontal" de "girando hacia el waypoint".

Opción B — end-to-end puro. Una alternativa más ajustada al objetivo de bajo costo de sensor es usar solo `rgb` como observación. Es más difícil de entrenar (la política debe aprender implícitamente el TTC y la geometría de obstáculos), pero produce un sistema sin dependencia del estimador de flujo óptico — relevante si se quiere transferir a un drone real donde la calibración del flujo es incierta.

A8.3 Espacio de acciones

Se propone un espacio discreto de seis acciones, alineado con `PROMPT_ACTIONS` del nodo deliberativo (§8.3, §5.14):

Índice	Macro-acción	Semántica
0	`MANTENER_RUMBO`	Avanzar hacia el waypoint activo
1	`EVADIR_IZQUIERDA`	Evasión lateral izquierda
2	`EVADIR_DERECHA`	Evasión lateral derecha
3	`GANAR_ALTURA`	Ascender para sobrevolar obstáculo
4	`PERDER_ALTURA`	Descender con avance lento
5	`FRENAR`	Detener el drone (hover)

La traducción a comandos cinemáticos es idéntica a la de los brazos existentes (`action_to_command()`, §5.14): la política neuronal nunca genera velocidades directamente, lo que garantiza que las maniobras elegidas sean físicamente seguras y respeten las mismas saturaciones y convenciones de Body Frame que los brazos `fsm` y `slm`.

Espacio continuo (extensión futura). Una vez que el brazo discreto haya convergido, se puede reemplazar el cabezal de política por un `Box(low=-1, high=1, shape=(4,))` correspondiente a $[v_x, v_y, v_z, \dot{\psi}]$ normalizados. El espacio continuo permite maniobras más suaves y eficientes, pero requiere guardar los límites cinemáticos como restricciones en la función de recompensa o como proyección post-acción.

A8.4 Función de recompensa

La función de recompensa combina cuatro términos con signos y pesos diferenciados:

$$r_t = w_{\text{prog}} \cdot \Delta d_t + w_{\text{col}} \cdot c_t + w_{\text{near}} \cdot n_t + w_{\text{arr}} \cdot a_t - w_{\text{time}} \cdot \Delta t$$

donde:

- $\Delta d_t = d_{t-1} - d_t$ es la reducción de distancia al waypoint activo en el ciclo t (positivo = progreso, negativo = retroceso).
- $c_t \in \{0, -1\}$: penalidad dura de colisión. Se activa cuando `telemetry["collision"] = True` (flag de AirSim). Domina sobre todos los demás términos para que la primera prioridad aprendida sea la evitación de obstáculos.
- $n_t = -\text{clip}(1/\text{TTC}_{\min} - 1/T_{\text{safe}},\, 0,\, 1)$: penalidad suave de proximidad, proporcional al exceso de peligro por encima del umbral `TTC_SAFE_THRESHOLD = 4.6 s`. Permite al agente aprender gradaciones de riesgo sin esperar la colisión.
- $a_t \in \{0, +1\}$: bonificación esparsa de llegada al waypoint activo (`waypoint_guidance["is_completed"] = True`).
- Δt: penalidad de tiempo (eficiencia). Evita que el agente aprenda a hovear indefinidamente cerca de los waypoints para acumular señal de proximidad sin avanzar.

Currículo de entrenamiento. Se recomienda iniciar con $w_{\text{col}} \gg w_{\text{prog}} \gg w_{\text{near}}$ para que la política aprenda primero a no colisionar (comportamiento de seguridad mínimo); una vez que la tasa de colisión caiga por debajo del 20 % en el escenario de entrenamiento, se aumenta gradualmente w_{prog} para impulsar el comportamiento de avance. La secuencia de escenarios sigue el orden de dificultad de la tesis: Tier 0 → Tier 1 → Tier 2.

A8.5 Algoritmo de entrenamiento

Se propone **PPO** (*Proximal Policy Optimization*, [Schulman et al., 2017]) como algoritmo primario, implementado vía **Stable-Baselines3** ([Raffin et al., 2021]).

Justificación de la elección

Algoritmo	Tipo	Ventaja	Limitación
PPO (recomendado)	On-policy	Estable en entornos con dinámica compleja; buen comportamiento en AirSim según Swift (Kaufmann et al., 2023)	Mayor costo de muestras que off-policy
SAC	Off-policy	Mayor eficiencia de muestras; apropiado para espacio continuo	Requiere ajuste fino de temperatura de entropía
DQN	Off-policy (discreto)	Línea base rápida; bajo costo de implementación	Sin soporte nativo para observaciones imagen + vector

PPO es el algoritmo que usa el sistema **Swift** de la Universidad de Zúrich (Kaufmann et al., 2023) para control de drones de carreras; en ese trabajo, una política PPO entrenada en AirSim superó en tiempo de vuelta a pilotos humanos campeones del mundo. Es el antecedente más directo en nivel de maniobra y en uso del mismo simulador. La convergencia on-policy es más predecible, lo que simplifica la interpretación de los resultados en un contexto de tesis.

Advertencia sobre generalización. El resultado de Swift ilustra tanto el potencial como el límite central del enfoque RL en robótica aérea: la política fue entrenada y evaluada en el circuito de carreras de Zúrich —geometría fija, velocidades máximas constantes, ausencia de obstáculos imprevistos— y no puede transferirse directamente a un entorno diferente sin reentrenamiento completo. Cambiar el trazado del circuito o la dinámica del dron degrada la tasa de éxito hasta el nivel de aleatoriedad. Este fenómeno de *overfitting* a la distribución de entrenamiento es estructural en RL, no un artefacto del método.

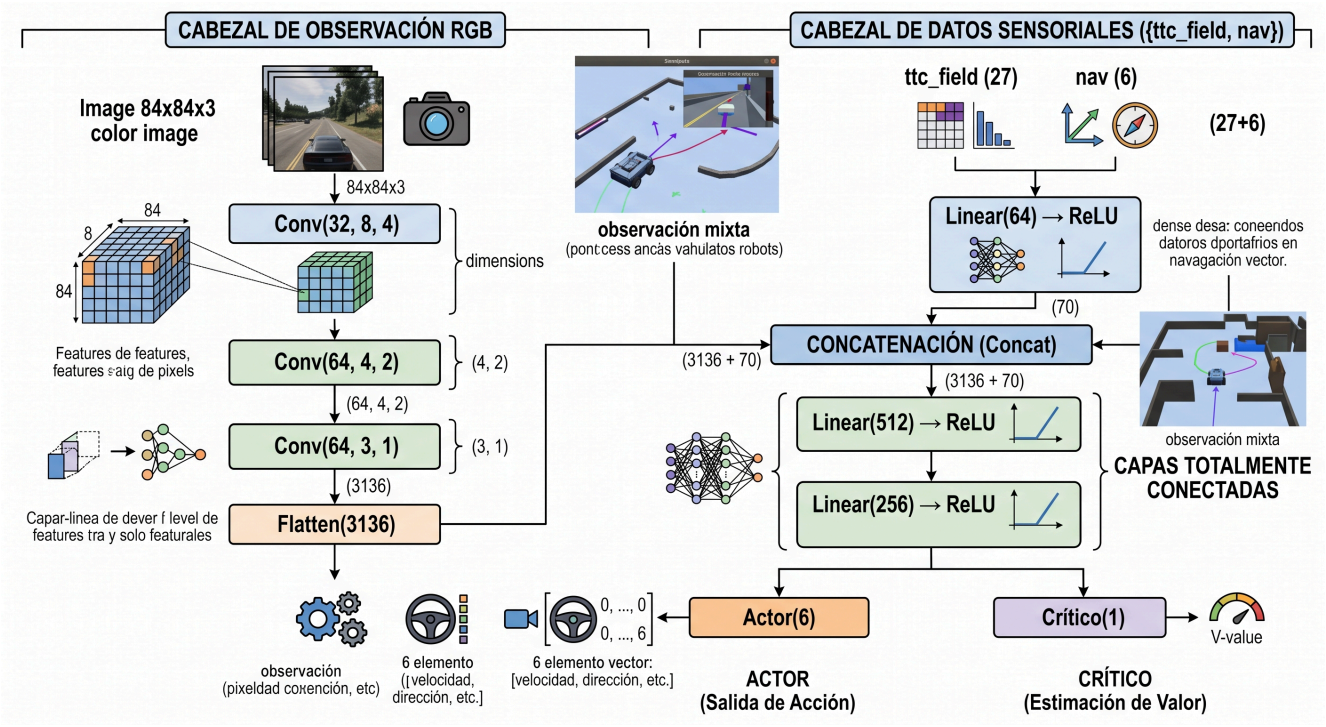
Para el brazo `rl` de esta tesis, el riesgo de especialización excesiva se mitiga parcialmente por tres decisiones de diseño:

1. **Espacio de acciones discretas sobre macro-acciones**: en lugar de generar velocidades directamente (como hace Swift), la política elige entre las seis macro-acciones ya validadas en los brazos `fsm` y `reactive`. Esto reduce el espacio de comportamientos "incorrectos" aprendibles a aquellos que el sistema ya sabe ejecutar con seguridad.
2. **Currículo de escenarios por Tier**: el agente se expone a tres niveles de dificultad (Tier 0 → 1 → 2) y cinco mapas diferentes durante el entrenamiento, forzando generalización a geometrías urbanas variadas en lugar de memorizar un circuito.
3. **Capa deliberativa activa**: cuando la política RL entra en un estado no representado en el entrenamiento (alta entropía de acción, múltiples ciclos de `FRENAR`), el barrido `deep_vlm` puede resolver el atasco y proponer una salida, igual que con el brazo `fsm`. Esto limita la región de fallo catastrófico de la política neuronal.

La validación en escenarios no vistos durante el entrenamiento —especialmente `citymap_pilot`— es la prueba empírica más directa de generalización, y su resultado es una de las incógnitas clave del protocolo de evaluación de §A8.6.

Arquitectura de la política

Para la observación mixta `{rgb, ttc_field, nav}` se propone una política de dos cabezales paralelos:



La rama CNN extrae características visuales del frame RGB. La rama FC procesa el campo de obstáculos serializado y el estado de guiado. La concatenación en el cuello de botella fuerza la política a integrar información visual y geométrica antes de tomar la decisión. Esta arquitectura es directamente compatible con `CnnPolicy` de Stable-Baselines3 cuando se define el espacio de observación como `gym.spaces.Dict`.

A8.6 Protocolo de evaluación comparativa

El brazo `rl`, una vez entrenado, debe evaluarse bajo exactamente el mismo protocolo que los brazos `slm`, `fsm` y `reactive` (cap. 10, §10.2), para que la comparación sea válida:

```
K = 10 semillas por celda, con la celda rl:depth (o rl sin barrido) agregada a cada lote
```

Los escenarios propuestos para la evaluación del brazo `rl` son los seis de la tesis (cap. 10, §10.3.4):

Escenario	Tier	Tipo
`minisim_clear`	0	Tránsito libre
`townsim_clear`	1	Tránsito libre
`citysim_clear`	2	Tránsito libre
`townsim_ini`	1	Obstrucción densa (corredor arbolado)

Escenario	Tier	Tipo
`townsim_calib_cruce_frontal`	1	Bloqueo frontal masivo
`citysim_pilot`	2	Obstrucción estructural (rodeo de manzanas)

Las métricas son las mismas que §10.6 (éxito, SPL, tiempo de misión, colisiones, distancia mínima, tasa de consulta) más dos métricas específicas del brazo `rl`:

- `policy_latency_ms`: tiempo de inferencia de la política neuronal (objetivo < 5 ms en CPU, < 1 ms en GPU dedicada).
- `training_steps_to_convergence`: número de pasos PPO hasta superar el criterio de éxito mínimo de R1 (tasa de llegada > 50 % en `minisim_clear`).

La tabla de comparación propuesta para el capítulo 11 sería:

Métrica	`reactive`	`fsm`	`slm`	`rl`
Tasa de éxito — tránsito libre	ref	ref	ref	?
Tasa de éxito — obstrucción densa	ref	ref	ref	?
Ratio tiempo vs. `reactive`	1.0×	?	?	?
Latencia de decisión	~0.1 ms	~0.2 ms	~0.1 s por consulta del lazo lento (profundidad; cap. 6, §6.14.3)	< 5 ms (estimado)
Consumo adicional de VRAM	—	—	~440 MiB (red de profundidad)	< 50 MB (CNN pequeña)

La hipótesis de trabajo es que el brazo `rl` superará al brazo `reactive` en escenarios de obstrucción densa (`townsim_ini`) donde el flujo óptico colapsa — porque la política neuronal puede aprender a actuar con observación visual directa, sin depender de la divergencia del flujo — y competirá con `fsm` en tránsito libre, donde la política debería aprender a elegir `MANTENER_RUMBO` de forma casi perfecta. Su comparación más relevante para la tesis es contra el brazo `slm`: ambos usan visión directa, pero `slm` juzga el espacio libre con una red de profundidad pre-entrenada de propósito general y traduce ese juicio a sub-metas, mientras que `rl` es una política especializada en el espacio de maniobras de este sistema.

A8.7 Dependencias y prerrequisitos

La implementación del brazo `rl` tiene las siguientes dependencias sobre el trabajo ya realizado:

- **Seis lotes completados** (cap. 10, §10.5.3): la comparación `slm` vs. `fsm` vs. `reactive` debe tener resultados estables antes de agregar un cuarto brazo, para que la comparación tenga una línea base válida.
- **Brazo `slm` validado** con `DEADLOCK_STRATEGY=depth`: la política RL se entrenará en el mismo entorno de simulación en el que se midió el `slm`; si ese entorno cambia durante el entrenamiento, los resultados no son comparables.

Las dependencias de software son mínimas (sobre el entorno conda `airsimenv` existente):

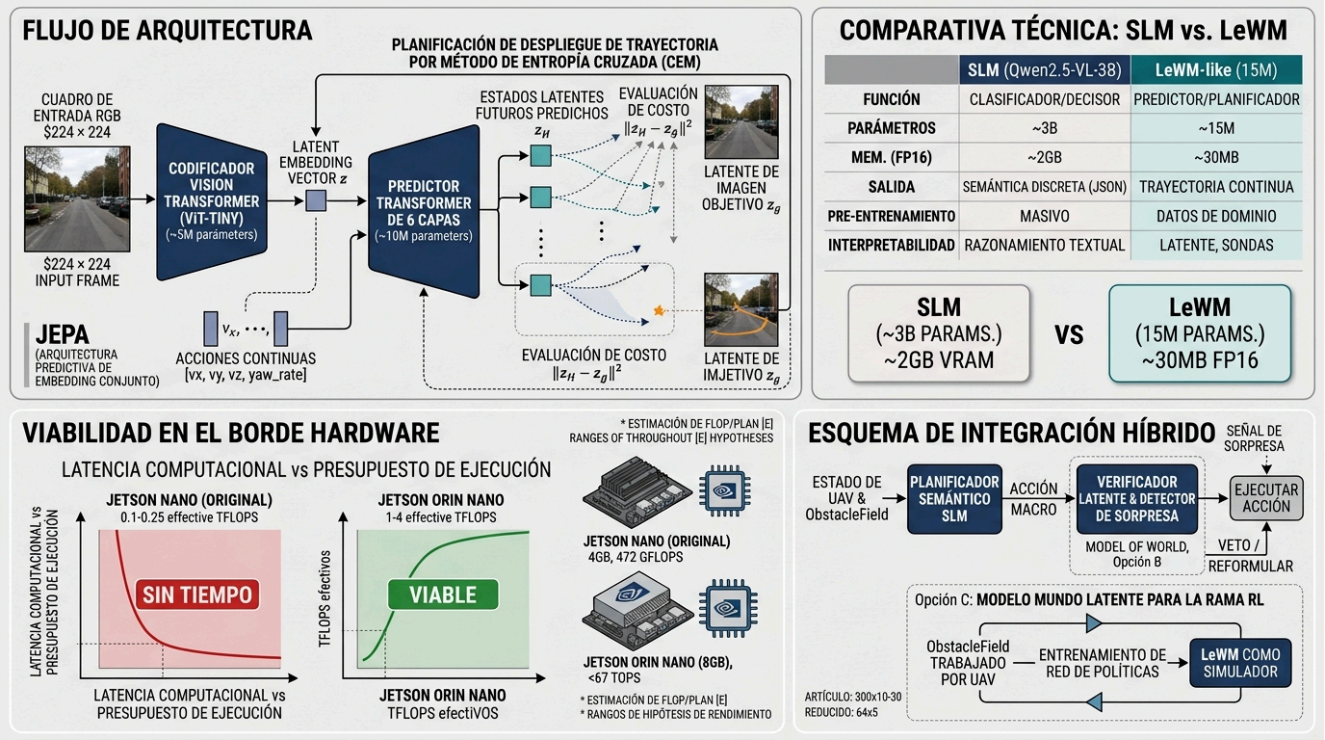
```
pip install stable-baselines3[extra]   # PPO, SAC, DQN + tensorboard
pip install gymnasium                  # gym API moderna (reemplaza gym)
pip install sb3-contrib                # TQC, RecurrentPPO si se necesitan
```

El entrenamiento puede ejecutarse en la misma máquina de desarrollo (RTX 5060, 8 GB VRAM), ya que el brazo `rl` no requiere inferencia de VLM durante el entrenamiento. El presupuesto estimado de VRAM para PPO con la arquitectura CNN propuesta (§A8.5) es < 2 GB, compatible con AirSim corriendo en paralelo.

Las referencias citadas en este anexo corresponden a las entradas del §13 del presente informe. La evaluación cuantitativa del brazo `rl` y su comparación con los tres brazos implementados forma parte del trabajo futuro descrito en §12.4.

Anexo 9: Modelos de Mundo Latentes (LeWorldModel) como Alternativa al SLM — Trade-offs y Viabilidad en Hardware Edge

Este anexo analiza, como línea de trabajo futuro (§12.4), la sustitución —total o parcial— del modelo de lenguaje visivo (LFM2.5-VL-1.6B, `lfm2.5-vl-1.6b:q4km-ctx8k`, cap. 8) por un **modelo de mundo latente** del tipo *Joint-Embedding Predictive Architecture* (JEPA), tomando como caso de estudio **LeWorldModel** (LeWM; Maes et al., 2026). El objetivo es doble: (i) determinar qué aportaría y qué perdería el sistema frente al brazo deliberativo `slm`, y (ii) estimar la viabilidad de ejecutar inferencia y planificación en una *companion computer* como la contemplada en el plan de tesis (NVIDIA Jetson Nano / Orin).



Alcance y convención de rigor. Nada de lo aquí descrito está implementado ni medido en el sistema evaluado en el capítulo 11. Las cifras se rotulan según su origen:

- **[P]** — reportada por el paper de LeWM u otra fuente citada.
- **[E]** — estimación analítica del autor (orden de magnitud, con supuestos explícitos); **no es una medición** y debe reemplazarse por el protocolo de §A9.7.
- **[T]** — medida en esta tesis (capítulos 8–11).

A9.1 Qué es LeWorldModel

Los modelos de mundo aprenden cómo cambia el entorno en función de las acciones del agente y permiten planificar "en imaginación" (Ha & Schmidhuber, 2018; Hafner et al., 2020). La familia JEPA (LeCun, 2022; Assran et al., 2023; Assran et al., 2025) evita predecir píxeles: predice la **representación** (embedding) del siguiente estado, lo que descarta detalle irrelevante y abarata la predicción. Su dificultad histórica es el *colapso de representación*; los trabajos previos lo resolvían con encoders congelados preentrenados (DINO-WM, Zhou et al., 2025) o con objetivos de muchos términos (PLDM, Sobal et al., 2025).

LeWM (Maes et al., 2026, publicado en marzo de 2026 con Y. LeCun entre sus autores) es, según sus autores, el primer JEPA que entrena de forma estable de extremo a extremo desde píxeles con solo dos términos de pérdida: predicción del siguiente embedding más el regularizador SIGReg, que empuja los embeddings hacia una gaussiana isotrópica (Balestriero & LeCun, 2025).

Aspecto	Valor reportado [P]
Parámetros totales	~15 M (encoder ~5 M + predictor ~10 M)
Encoder	ViT-Tiny, patch 14, 12 capas, dimensión oculta 192; embedding = token `[CLS]` proyectado (192 dim)
Predictor	Transformer de 6 capas, 16 cabezas; acciones inyectadas por AdaLN; historial de N embeddings con máscara causal
Entrada visual	Frames RGB de 224 × 224
Planificación	Minimización del costo terminal $\|z_H - z_g\|^2$ por CEM con MPC; 300 secuencias candidatas, 10–30 iteraciones, horizonte H = 5 (frameskip 5)
Tiempo de planificación	0,98 s por plan vs. 47 s de DINO-WM (hasta 48× más rápido), promedio sobre 50 corridas
Entrenamiento	Una GPU, pocas horas
Entornos evaluados	Push-T, Reacher, OGBench-Cube, Two-Room (2D y manipulación 3D simple)

El paper también reporta que el espacio latente codifica cantidades físicas recuperables por *probing* lineal y que un test de violación de expectativa detecta trayectorias físicamente implausibles. Entre las **limitaciones declaradas** por los autores: horizontes de planificación cortos, dependencia de datasets *offline* con cobertura suficiente (con baja diversidad SIGReg se debilita) y dependencia de etiquetas de acción.

Trabajos de contexto aéreo. El antecedente más cercano al dominio de esta tesis es SkyJEPA (Rao et al., 2026), que aplica un modelo de dinámica latente de estilo JEPA al control de cuadrirrotores con transferencia *sim-to-real* y optimización por muestreo en hardware embebido; según su resumen opera sobre **estado**, no sobre imágenes, por lo que no valida el caso visual monocular. En navegación con imagen objetivo, Chahe et al. (2026) muestran que el costo de planificación en el espacio latente requiere diseño explícito (monotonía respecto del progreso). No se realizó un relevamiento exhaustivo de esta literatura, muy activa en 2026.

A9.2 Qué cambiaría en el sistema: SLM vs. modelo de mundo latente

El VLM del sistema es un **perceptor semántico de escala de segundos**: recibe un fotograma (con una grilla de 3×3 dibujada o, en un deadlock, cuatro imágenes de un barrido) y describe qué direcciones son volables; el código traduce esa descripción a una sub-meta en coordenadas del mundo (cap. 5, §5.10), con una latencia de ~0,8 s con el modelo local [T] (cap. 8, §8.6). Con la configuración por defecto esa pregunta la responde una red de profundidad estimada (cap. 6, §6.14) y el VLM queda en el plan de ruta. Un modelo de mundo latente es un **predictor de consecuencias**: no decide por sí mismo; permite evaluar secuencias de acciones candidatas y elegir la de menor costo. No son sustitutos directos, sino componentes de naturaleza distinta.

Dimensión	VLM (LFM2.5-VL-1.6B, §8)	LeWM-like (15 M)
Función	Clasifica sectores o imágenes en volable o bloqueado; el código fija la sub-meta	Predice el embedding futuro dada una secuencia de acciones; el plan surge de optimizar
Salida	JSON discreto (enumeraciones cerradas por sector o imagen)	Secuencia continua de acciones (p. ej., `vx, vy, vz, yaw_rate`)
Conocimiento previo	Preentrenamiento masivo (sentido común, semántica urbana)	Ninguno: aprende solo de los datos de dominio
Especificación del objetivo	Lenguaje natural / waypoints en el prompt	Imagen objetivo (o función de costo sobre el latente)
Datos requeridos	Ninguno (*zero-shot*) o dataset de maniobras para LoRA (Anexo 3)	Trayectorias (imagen, acción) offline con buena cobertura
Parámetros	1,6 B	~15 M (≈100× menos)
Huella de memoria	~1,6 GB (Q4_K_M, contexto de 8 k) [T]	~30 MB en FP16 [E]
Interpretabilidad	Texto (`reason`) auditable; la auditoría de contratos del cap. 9 se apoya en ello	Latente opaco; requiere *probes* para inspeccionarlo

Dimensión	VLM (LFM2.5-VL-1.6B, §8)	LeWM-like (15 M)
Modo de falla característico	Alucinación, JSON inválido, *timeout* del watchdog (cap. 9)	Deriva de rollouts a horizonte largo, fuera de distribución, costo latente mal condicionado
Latencia por decisión	mediana 0,77 s en la RTX 5060 compartida con Unreal (modelo local) [T]	~1 s en el paper con 300 × 10–30 iteraciones [P]; reducible (§A9.5)

El SLM aporta exactamente aquello que el modelo de mundo no tiene (semántica y razonamiento global entre corredores ambiguos: régimen 2 del cap. 12, `citymap_pilot`), mientras que el modelo de mundo aporta lo que el SLM no tiene: **predicción explícita de consecuencias de acciones continuas a bajo costo**, útil justamente en el régimen 3 (obstrucción frontal con escape lateral accesible), donde el SLM perdía por latencia.

A9.3 Brechas de aplicabilidad al problema del dron

1. **Objetivo por imagen.** El costo de LeWM es `‖z_H - z_g‖²` con `z_g = enc(o_g)`. La misión del dron se especifica por waypoints, no por imágenes. En simulación es posible renderizar una vista objetivo por waypoint, pero en un vuelo real no existe esa imagen. Alternativa propuesta: entrenar *probes* livianos sobre el latente para obtener cantidades ya usadas por el sistema (ocupación/TTC del `ObstacleField`, distancia a obstáculo, rumbo relativo al waypoint) y definir el costo sobre ellas; el paper reporta que el latente admite *probing* de cantidades físicas en sus entornos [P], pero no en escenas urbanas aéreas.
2. **Horizonte corto.** H = 5 pasos de predictor [P]. Con el lazo táctico a `LOOP_HZ = 5` [T], si cada paso del predictor equivale a un ciclo (200 ms), el horizonte es ~1 s: escaso para evasión a velocidad de crucero. Si cada paso agrupa varios ciclos (como el frameskip del paper), el horizonte se alarga pero se pierde resolución de control. Es un parámetro de diseño, no una constante.
3. **Dominio y datos.** Los resultados de LeWM son en tareas 2D/3D simples con datasets cuidadosamente recolectados [P]. Escenas urbanas 3D con vegetación, tráfico y multitud (Tiers 1–2, cap. 3) son de complejidad visual muy superior, y los propios autores señalan que la baja diversidad de datos degrada SIGReg [P]. El sistema podría generar datos en AirSim de forma barata, pero **la cobertura de casos de colisión** (raros por construcción) es el cuello de botella real.
4. **Etiquetas de acción y dinámica.** El predictor condiciona en acciones; para el dron habría que fijar qué representa la acción (velocidad comandada vs. macro-acción) y cómo se trata el retardo de actuación/autopiloto, que el paper no aborda.
5. **Sin semántica ni razonamiento global.** Un modelo de 15 M entrenado solo con datos de vuelo no distingue "corredor de servicio" de "autopista elevada" salvo que esa distinción esté visualmente y estadísticamente presente en los datos. Es el régimen donde el VLM mostró valor (H1, `citymap_pilot`).
6. **Seguridad y verificabilidad.** La optimización por CEM es estocástica y sin garantías de optimalidad global [P]. Cualquier integración debe conservar las salvaguardas determinísticas del sistema

(`motor_node`, guardas de altitud, `DEPTH_EMERGENCY`).

7. **Costo de reentrenamiento.** Un cambio de cámara, altura de vuelo o escena exige reentrenar o ajustar el modelo; el SLM *zero-shot* es más robusto a ese cambio, aunque más lento.

A9.4 Tres formas de incorporarlo

Opción A — Reemplazo del SLM (no recomendada). Sustituir la capa deliberativa por un planificador LeWM-like. Elimina la semántica (§A9.3, punto 5), exige resolver el objetivo por imagen y pierde la auditabilidad textual del cap. 9. Solo se justificaría si el objetivo fuera exclusivamente evasión local reactiva, que es el terreno del brazo `reactive`/`rl`, no de la capa deliberativa.

Opción B — Verificador de consecuencias y detector de sorpresa (recomendada como primer paso). Mantener el SLM como decisor semántico y usar el modelo de mundo como **capa de verificación**: dada la macro-acción propuesta por el SLM (convertida a secuencia de comandos por `action_to_command()`), predecir en el latente si conduce a un estado con mayor riesgo (vía *probe* de ocupación/TTC) y vetarla o pedir reformulación antes de ejecutarla; además, usar el error de predicción como señal de *sorpresa* (fuera de distribución) para disparar la ruta deliberativa o el aterrizaje seguro. Esto reutiliza las dos propiedades que el paper sí demuestra (probing y detección de eventos implausibles [P]) sin exigirle planificar de extremo a extremo.

Opción C — Modelo de mundo para el brazo `rl` (complementaria con el Anexo 8). Usar el modelo de mundo como simulador latente barato para entrenar la política del cuarto brazo (imaginación, al estilo de Hafner et al., 2020), reduciendo la cantidad de interacciones con AirSim/UE5.5, que es el costo dominante del entrenamiento. La política final seguiría siendo una red pequeña (< 5 ms) ejecutable en el borde.

A9.5 Viabilidad en hardware edge

A9.5.1 Plataformas de referencia

Plataforma	Cómputo declarado por el fabricante	Memoria	Potencia	Fuente
Jetson Nano (original)	472 GFLOPS	4 GB	5–10 W	NVIDIA, 2025
Jetson Orin Nano (4/8 GB)	hasta 67 TOPS (cifra de "AI performance" del fabricante)	4 o 8 GB	7–25 W	NVIDIA, s. f.
Jetson Orin NX (8/16 GB)	hasta 157 TOPS	8 o 16 GB	—	NVIDIA, s. f.

Plataforma	Cómputo declarado por el fabricante	Memoria	Poten-cia	Fuente
Estación de desarrollo de la tesis	RTX 5060, 8 GB VRAM compartidos con UE5.5	8 GB	—	[T] (cap. 8)

Las cifras en TOPS son de marketing (típicamente INT8 y con esparcidad estructurada); para inferencia FP16 densa de *matmuls* pequeños el rendimiento sostenible es una fracción de ese valor. Por eso la tabla siguiente parte de supuestos conservadores.

A9.5.2 Memoria

Con ~15 M de parámetros, LeWM ocupa ≈ 30 MB en FP16 y ≈ 15 MB en INT8 [E], frente a ~1,6 GB del VLM cuantizado [T]: unas 50× menos en el formato desplegado. Incluso la Jetson Nano de 4 GB podría alojar el modelo de mundo con holgura; la restricción de memoria, que condicionó la elección del VLM (cap. 8), **deja de ser el factor limitante**. Lo que sí pesa es el lote de candidatos del CEM (300 secuencias × historial de 3 tokens × 192 dimensiones), del orden de unos pocos MB de activaciones [E].

A9.5.3 Cómputo por plan

Estimación analítica del costo por plan, con las dimensiones del paper [P] y contando solo multiplicaciones matriciales dominantes (2 FLOP por parámetro y token):

- **Encoder** (ViT-Tiny, 224 × 224, patch 14 → 256 + 1 tokens): ≈ 3,4 GFLOP por frame [E]. Se evalúa una vez por ciclo (observación actual); el embedding objetivo se cachea.
- **Predictor** (10 M parámetros, historial de 3 tokens): ≈ 60 MFLOP por paso de rollout y por candidato [E].
- **Rollout CEM:** `N × iteraciones × H × 60 MFLOP`.

Configuración CEM (H = 5)	Rollouts/plan	FLOP/plan [E]	Latencia Orin Nano (1–4 TFLOP/s efectivos, supuesto) [E]	Latencia Nano original (0,1–0,25 TFLOP/s efectivos, supuesto) [E]
Paper, Push-T: 300 × 30	9 000	≈ 2,7 T	≈ 0,7–2,7 s	≈ 11–27 s
Paper, otros: 300 × 10	3 000	≈ 0,9 T	≈ 0,2–0,9 s	≈ 4–9 s
Reducida: 64 × 5	320	≈ 0,1 T	≈ 25–100 ms	≈ 0,4–1 s

Los rangos de throughput efectivo son **hipótesis** a validar en §A9.7: los *matmuls* pequeños y batcheados rara vez saturan las unidades tensoriales. Dos consecuencias:

1. **Con la configuración del paper, el modelo de mundo no es claramente más rápido que el SLM en el borde**: su ventaja (48× sobre DINO-WM) es relativa a otros modelos de mundo, no al SLM del sistema. En una Orin Nano el plan completo caería en el mismo orden de magnitud que la latencia del VLM en la estación de desarrollo (mediana 1,46 s [T]; en la Orin Nano, con menos ancho de banda que una GPU de escritorio, sería mayor [E]) y del watchdog original (`SLM_WATCHDOG_MS = 1500 ms`; el valor vigente es 13 000 ms, ampliado precisamente para no descartar respuestas del VLM, cap. 8, §8.6).
2. **Reduciendo el presupuesto de muestreo** (menos candidatos e iteraciones, *warm start* de la distribución CEM con el plan previo, horizonte corto) el plan baja a decenas de milisegundos en Orin Nano, compatible con el lazo de 5 Hz (200 ms por ciclo, con el ciclo también ocupado por percepción y flujo óptico). El costo es calidad de plan, cuyo impacto no está caracterizado y debe medirse (el paper fija sus hiperparámetros por tarea [P]).

En la Jetson Nano original (Maxwell, sin unidades tensoriales), solo la configuración reducida es plausible, y la pila de software heredada (JetPack 4.x, versiones antiguas de PyTorch/TensorRT) añade fricción de portabilidad [E]. La Orin Nano es la plataforma realista para esta línea.

A9.5.4 Ejecutar el SLM y el modelo de mundo a la vez

En el escenario de las Opciones B/C ambos modelos coexisten. Como el modelo de mundo pesa decenas de MB, la contención principal sería de **ancho de banda de memoria y de tiempo de GPU**, no de capacidad. El decodificado autorregresivo del VLM está limitado por ancho de banda: con ~2 GB de pesos por token y ~102 GB/s de la Orin Nano 8 GB, la cota superior teórica es ≈ 50 tokens/s [E] (dato de ancho de banda según el fabricante del kit de desarrollo; no verificado en el datasheet del módulo). Con el prompt actual (~500 tokens, cap. 12) y la imagen, es probable que el SLM del sistema, que ya tarda 1,2–4,6 s en la RTX 5060, se alargue en esa plataforma y deba apoyarse en un watchdog holgado a costa de la frescura de la escena, lo que refuerza el argumento del Anexo 3 (LoRA para reducir el prompt) **independientemente** de esta línea.

A9.5.5 Optimizaciones de despliegue aplicables

- Exportar encoder y predictor a ONNX/TensorRT en FP16 o INT8 (cuantización posterior con calibración sobre frames de AirSim).
- Reducir la resolución de entrada (p. ej., 112 × 112 con patch 14 → 64 tokens, ≈ 4× menos costo de encoder) reentrenando; sujeto a pérdida de detalle para obstáculos finos (cables, ramas).
- *Warm start* de CEM y ejecución solo bajo disparo del router táctico (no en cada ciclo).
- Sustituir CEM por un optimizador basado en gradiente sobre el predictor diferenciable, o destilar el plan en una política amortizada (vinculado a la Opción C).

A9.6 Trade-offs: síntesis

Criterio	SLM (actual)	Modelo de mundo latente	Veredicto
Semántica y razonamiento global	Fuerte	Ausente	Favorece SLM
Predicción de consecuencias de acciones continuas	Nula (decide macro-acciones)	Núcleo del método	Favorece modelo de mundo
Memoria	~2 GB	~30 MB	Favorece modelo de mundo
Latencia en el borde	1,2–4,6 s en RTX 5060 [T]; mayor en Orin Nano [E]	0,2–2,7 s con la config. del paper; 25–100 ms con presupuesto reducido (Orin Nano) [E]	Favorece al modelo de mundo, condicionado a medirlo (§A9.7)
Datos y costo de entrenamiento	Cero (zero-shot)	Dataset propio + horas de GPU	Favorece SLM
Robustez a cambio de dominio	Alta relativa	Baja (reentrenar)	Favorece SLM
Auditabilidad	Texto explícito	Latente opaco	Favorece SLM
Madurez y evidencia en el dominio	Evaluado en esta tesis (75 corridas)	Sin evidencia en dron visual urbano	Favorece SLM

Conclusión del análisis. La viabilidad **técnica** de ejecutar un modelo de mundo latente de ~15 M de parámetros en hardware edge es alta en memoria y condicional en latencia (requiere Orin Nano y presupuesto de muestreo reducido). La viabilidad **de reemplazar** al SLM es baja con la evidencia actual; la viabilidad de **complementarlo** (Opciones B y C) es la línea con mejor relación beneficio/riesgo.

A9.7 Protocolo de evaluación propuesto

1. **Recolección de datos.** Generar trayectorias (frame RGB, acción, telemetría) en AirSim para los tres tiers, incluyendo trayectorias de casi-colisión y maniobras de escape; las corridas ya registradas de los lotes de la tesis pueden reutilizarse como semilla, evaluando su cobertura antes de entrenar.
2. **Entrenamiento y diagnóstico del latente.** Entrenar LeWM (código oficial del paper, Maes et al., 2026); entrenar *probes* lineales para TTC/ocupación usando las etiquetas de referencia de profundidad ya disponibles (cap. 7) y medir su error; medir error de rollout vs. horizonte.
3. **Medición en el borde.** Medir latencia real de `encoder`, `predictor` y plan CEM en la estación de desarrollo y en Jetson Orin Nano (FP16 y INT8/TensorRT), variando `N`, iteraciones y `H`; reportar p50/p95/p99 y potencia. Esto reemplaza las estimaciones [E] de §A9.5.3.

4. **Evaluación en lazo cerrado.** Comparar, sobre los mismos escenarios y semillas del cap. 10, `slm` contra `slm + verificador latente` (Opción B), con las mismas métricas (éxito, colisiones, tiempo, distancia). Con K = 5 los resultados serían indicativos (cap. 12, §12.3); se recomienda K ≥ 10–15.
5. **Criterio go/no-go.** Continuar solo si (a) el *probe* de riesgo alcanza una capacidad discriminativa medida (AUC/Youden, como en el Anexo 5) sobre escenas fuera del conjunto de entrenamiento, y (b) el plan completo cabe en el presupuesto del lazo táctico en Jetson Orin Nano sin degradar el ciclo de percepción.

A9.8 Consideraciones finales

LeWorldModel es relevante para esta tesis no porque reemplace al SLM sino porque **desplaza la pregunta**: de "qué maniobra elegir" (decisión semántica discreta) a "qué ocurrirá si ejecuto esta secuencia de acciones" (predicción de consecuencias), con una huella de memoria compatible con un nodo edge. La evidencia disponible es de dominios simples y de otros grupos; su validez en visión monocular urbana aérea es una hipótesis abierta y el presente anexo la formula, junto con los criterios para refutarla, en lugar de asumirla.

Las referencias citadas en este anexo corresponden a las entradas del §13 del presente informe. Las cifras marcadas [E] son estimaciones analíticas no medidas; su reemplazo por mediciones forma parte del trabajo futuro descrito en §12.4.