Documentación Técnica del Sistema

AeroTrace

Sistema Inteligente de Análisis de Tráfico Rodado mediante Imágenes Aéreas

Equipo : NeuralLogic

Miembro : Mahsa Simaei

Fecha de entrega : 8/2/2026

Índice

1. Resumen Ejecutivo y Alcance

1.1 Descripción del Sistema

AeroTrace v2.0 es una solución integral de ingeniería de tráfico diseñada para procesar imágenes aéreas capturadas por Vehículos Aéreos No Tripulados (UAV/Drones). El sistema utiliza algoritmos avanzados de Visión Artificial (Deep Learning) basados en **YOLOv8x** para detectar, clasificar y rastrear vehículos en entornos urbanos y carreteras interurbanas mediante técnicas de **Slicing Inference (SAHI)** y **ByteTrack** para seguimiento persistente de trayectorias.

Más allá de la detección visual, el sistema integra un motor de física para calcular métricas de movilidad críticas (flujo vehicular, densidad, Nivel de Servicio - LOS según HCM 2010) y un módulo de auditoría basado en **Blockchain BSV**, garantizando la inmutabilidad y la trazabilidad de los datos generados para su uso en la administración pública.

1.2 Justificación y Valor Añadido

En el contexto de las *Smart Cities*, la gestión del tráfico carece a menudo de datos granulares y verificables. Los métodos tradicionales (bucles de inducción, cámaras fijas) son costosos y limitados geográficamente.

AeroTrace resuelve estos problemas mediante:

Ventaja	Descripción Técnica
Flexibilidad Espacial	Análisis bajo demanda en cualquier ubicación mediante drones sin infraestructura permanente
Precisión Extrema	YOLOv8x con Slicing Inference detecta vehículos de 40×40 px desde 120m de altura
Confianza Criptográfica	Hashes SHA-256 y cadena ETL compatible con BSV evitan manipulación estadística
Calibración Física	Conversión píxel→metro mediante GSD calculado con FOV y altura de vuelo

1.3 Cumplimiento de Objetivos del Hackathon

El sistema cumple estrictamente con los criterios de evaluación establecidos en las bases del hackathon:

Criterio	Implementación
Visión Artificial	YOLOv8x + SAHI + ByteTrack. Detección multicategoría (coches, motos, pesados)
Métricas de Tráfico	Densidad (veh/km²), flujo, ocupación, LOS (HCM 2010), velocidad instantánea
Blockchain BSV	SHA-256 hashing, timestamps ISO 8601, cadena ETL inmutable
Calidad y Demo	Instalación en 3 comandos, documentación completa, código modular

2. Arquitectura del Sistema

2.1 Diseño Conceptual

El sistema sigue una arquitectura modular de tubería (pipeline) de datos, desacoplando la ingesta, el procesamiento pesado y la visualización. El flujo de datos es el siguiente:

📥 **Input (Video/Imágenes)** → 🤖 **Detección AI (YOLOv8x + SAHI)** → 🎯 **Tracking (ByteTrack)** → 📊 **Métricas (TrafficEngineer)** → 🔗 **Blockchain (BSV Registry)** → 📈 **Visualización (Streamlit)**

Componente	Tecnología	Función
Frontend	Streamlit 1.29+	Interfaz web interactiva, configuración ROI, visualización en tiempo real
Core AI	Ultralytics YOLOv8x	Detección de objetos con modelo Extra Large (mAP optimizado)
Inference Optimizer	Supervision (Roboflow)	Slicing con overlapping para imágenes de alta resolución
Tracking Engine	ByteTrack	Seguimiento de IDs persistentes con oclusiones temporales
Physics Engine	TrafficEngineer (custom)	Cálculo GSD, densidad, LOS, velocidades instantáneas
Incident Detector	IncidentDetector (custom)	Detección de frenadas, vehículos detenidos, conflictos espaciales
Blockchain Layer	BSVEvidenceRegistry (custom)	Hashing SHA-256, cadena ETL, timestamps UTC
Visualización	Plotly + HTML	Dashboards interactivos exportables

2.2 Stack Tecnológico

El stack tecnológico ha sido seleccionado por su rendimiento, estabilidad y compatibilidad con datasets de UAV:

```
# Lenguaje y VersiónPython 3.9+
# Visión Artificialultralytics==8.0+
# YOLOv8 oficialsupervision==0.16+
# Slicing, tracking, anotacionesopencv-python==4.8+
# Procesamiento de imágenes
# Análisis de Datosnumpy==1.24+pandas==2.0+scipy==1.11+
# Cálculos espaciales# Visualizaciónstreamlit==1.29+plotly==5.18+
# Integridadhashlib (built-in)
# SHA-256 hashingjson (built-in)
```

Serialización BSV

3. Módulo de Visión Artificial (Detalle Técnico)

3.1 Selección del Modelo

Se ha implementado el modelo **YOLOv8x (Extra Large)** en lugar de versiones más ligeras (Nano, Small, Medium) por las siguientes razones técnicas:

Modelo	Parámetros	mAP (COCO)	Velocidad (ms)	Decisión
YOLOv8n	3.2M	37.3%	0.99 ms	❌ Insuficiente para vistas aéreas
YOLOv8s	11.2M	44.9%	1.20 ms	❌ Precision media
YOLOv8m	25.9M	50.2%	2.42 ms	⚠️ Aceptable pero mejorable
YOLOv8l	43.7M	52.9%	3.54 ms	✅ Buena precisión
YOLOv8x	68.2M	53.9%	4.88 ms	✅ SELECCIONADO - Máxima precisión

Justificación: En imágenes aéreas capturadas a 120m de altura, un coche puede ocupar apenas 40×40 píxeles. A diferencia de aplicaciones de videovigilancia terrestre donde la velocidad es crítica, en análisis UAV batch la precisión es prioritaria. El coste computacional adicional de YOLOv8x (2.88 ms vs YOLOv8s) es despreciable comparado con el tiempo total de procesamiento por frame (~150-200 ms incluyendo tracking y métricas).

3.2 Estrategia de Inferencia: Slicing (SAHI)

El código implementa `sv.InferenceSlicer` (ver `main.py`).

Las imágenes de drones suelen ser de alta resolución (4K). Redimensionarlas a 640x640 (entrada estándar de YOLO) destruiría la información de los vehículos pequeños.

- **Solución:** La imagen se divide en recortes (*slices*) de 640x640 con un solapamiento (*overlap*) del 20%.
- **Resultado:** Se detectan vehículos que serían invisibles en una inferencia estándar.

3.3 Clasificación y Filtrado

El sistema mapea las 80 clases COCO a las categorías relevantes para movilidad urbana, filtrando objetos irrelevantes para reducir ruido computacional:

ID COCO	Clase	Umbral Conf.	Filtro Área (px²)	Filtro Aspecto (W/H)
1	bicicleta	0.20	200 - 5,000	0.3 - 1.5
2	coche	0.25	800 - 40,000	0.4 - 3.0
3	motocicleta	0.20	200 - 4,000	0.3 - 2.5

5	bus	0.35	3,000 - 80,000	1.0 - 4.0
7	camión	0.35	2,500 - 70,000	1.0 - 3.5

Se aplican filtros geométricos (DetectionFilter en main.py) para descartar falsos positivos

1. Filtro de confianza por clase: Umbrales más altos para buses y camiones (raros en dataset UAV) para reducir falsos positivos.

2. Filtro geométrico de área: Descarta detecciones con áreas en píxeles fuera del rango esperado para cada clase en vista aérea.

3. Filtro de relación de aspecto: Elimina cajas con proporciones anómalas (p.ej., un "coche" de 10:1 W/H es probablemente un error).

3.4 Tracking (Seguimiento)

Se utiliza **ByteTrack** (sv.ByteTrack) en lugar de Sort o DeepSort. A diferencia de estos, ByteTrack aprovecha las detecciones de *baja confianza* para mantener la consistencia de las trayectorias cuando los vehículos pasan bajo árboles, puentes o sufren oclusiones temporales, crítico en vistas aéreas donde los objetos pueden desaparecer parcialmente.

4. Procesamiento de Métricas de Movilidad

El núcleo de la lógica de negocio reside en la clase TrafficEngineer y VehicleTracker. No nos limitamos a contar; medimos la física del tráfico.

4.1 Calibración Espacial (GSD)

Para convertir píxeles a metros, calculamos el *Ground Sample Distance* (GSD) basándonos en parámetros físicos del UAV:

Parámetro	Valor por Defecto	Justificación
Altura de vuelo	120 m	Máximo legal para drones civiles en España/EU sin autorización especial
FOV horizontal	84°	Típico de cámaras DJI Mavic 3 (las más comunes en análisis UAV)
Resolución imagen	1920×1080 px	Full HD, estándar en dataset UAV de Kaggle

Ejemplo de cálculo: Para 1920×1080 a 120m con 84° FOV: GSD ≈ 0.14 m/px. Un vehículo de 50px de largo = 7 metros reales (coherente con un turismo estándar).

4.2 Métricas de Flujo y Densidad

- **Flujo Acumulado:** Se utiliza una LineZone virtual. El sistema detecta el cruce vectorial de los centroides de los vehículos sobre esta línea.

- **Densidad:** Calculada fotograma a fotograma.
  Calculada fotograma a fotograma según la fórmula estándar de densidad areal:

```
ρ = (N_vehículos / Área_ROI_m²) × 1,000,000
```

- **Nivel de Servicio (LOS):** Implementación del estándar HCM 2010 (Highway Capacity Manual). Clasifica la vía de A (Flujo libre) a F (Colapso) basándose en umbrales de densidad:

LOS	Densidad (veh/km²)	Descripción	Estado
A	0 - 14	Flujo libre	Óptimo
B	14 - 22	Flujo razonablemente libre	Estable
C	22 - 32	Flujo estable	Limitado
D	32 - 45	Flujo inestable	Denso
E	45 - 67	Flujo forzado	Saturado
F	> 67	Colapso	Crítico

4.3 Detección de Incidentes Críticos

analiza la cinemática vectorial de cada vehículo rastreado para detectar situaciones de riesgo en tiempo real:

Tipo de Incidente	Condición de Detección	Severidad	Método
Vehículo Detenido	V < 0.3 m/s durante > 3 segundos	ALTA	VehicleTracker.get_velocity()
Frenada Brusca	Aceleración < -2.5 m/s²	MEDIA	VehicleTracker.get_acceleration()
Conflicto Espacial	Distancia entre vehículos < 3.5 m	ALTA	scipy.spatial.distance.cdist()
Densidad Peligrosa	ρ > 45 veh/km²	ALTA	TrafficEngineer.calculate_density()
Densidad Crítica	ρ > 65 veh/km²	CRÍTICA	TrafficEngineer.calculate_density()

Implementación de Velocidad Instantánea: El sistema utiliza una ventana móvil de 10 frames para calcular la velocidad mediante diferencia de posiciones escaladas por GSD y divididas por el tiempo transcurrido. Esto reduce el ruido de jitter en las detecciones frame a frame.

5. Integración Blockchain BSV (Auditabilidad)

5.1 Propósito

El módulo Blockchain BSV garantiza que los datos de tráfico utilizados para multas automatizadas, planificación urbana o informes de sostenibilidad **no han sido alterados post-procesamiento**. Esto es crítico en contratos públicos donde la transparencia y la auditoría son requisitos legales.

5.2 Implementación Técnica (bsv_blockchain.py)

El sistema implementa un registro de evidencia digital mediante la clase **`BSVEvidenceRegistry`**. El flujo de datos es el siguiente:

1. Hashing de Frames (ImageEvidence)

Cada frame procesado pasa por una función de hash SHA-256. Se procesan únicamente cada 30 frames (1 segundo a 30fps) para optimizar rendimiento sin sacrificar trazabilidad.

2. Registro de Análisis (AnalysisEvidence)

Al finalizar el procesamiento, se genera un hash del diccionario de métricas completo (flujo, densidad, incidentes, etc.). Este hash vincula las conclusiones estadísticas a los frames originales.

3. Cadena de Custodia (ETL Chain)

Se crea una estructura de datos enlazada donde cada transacción contiene el hash de la transacción anterior, creando una cadena local inmutable:

```
# Transacción blockchain (bsv_blockchain.py líneas 107-129)tx_id =
uuid.uuid4().hexprev_tx = self.etl_chain[-1] if self.etl_chain else Nonetx =
BlockchainTransaction(    transaction_id=tx_id,
timestamp=datetime.utcnow().isoformat() + 'Z',  # ISO 8601 UTC
scene_id=self.scene_id,    image_hash=image_evidence.image_hash,
metrics_hash=analysis_evidence.metrics_hash,    payload=payload,
chain_stage='processed',    previous_transaction_id=prev_tx  # ENLACE DE CADENA)
```

4. Salida Compatible BSV

El sistema genera un archivo **`evidence_log_[scene_id].json`** que cumple con el esquema necesario para ser transmitido a la red BSV mediante una API de indexación (como TAAL o Gorillapool) en fase de producción. Estructura del log:

```
{  "scene_id": "uav_sequence_001",  "location": {"lat": 40.4168, "long":
-3.7038},  "generated_at": "2026-02-08T14:30:00Z",  "statistics": {
"total_images_processed": 150,    "total_analyses": 1,    "total_transactions":
151  },  "etl_chain": [    "a3f2b1c...",    "7d9e4a2...",    "f1c8b3e..."  ],
"blockchain_transactions": [...]}
```

IMPORTANTE: Por limitaciones de tiempo del hackathon, el sistema genera las evidencias verificables localmente en archivos JSON, pero la estructura es 100% compatible con la red BSV. Los hashes son matemáticamente válidos y garantizan la integridad de los datos. En producción, estas mismas estructuras se enviarían directamente a la blockchain mediante el SDK de BSV (python-bsvx o similar), pero la lógica criptográfica ya está completamente implementada.

6. Interfaz de Usuario (Frontend)

Desarrollada con **Streamlit**, la interfaz se centra en la usabilidad para operadores urbanos.

6.1 Funcionalidades Clave

1. **Configuración de Zona (ROI):** El usuario define las coordenadas del polígono de análisis y la línea de conteo mediante *sliders* numéricos, visualizando la zona en tiempo real sobre el primer frame.
2. **Feedback en Tiempo Real:** Visualización del video procesado con capas de realidad aumentada (cajas, etiquetas, líneas) junto a KPIs actualizados frame a frame (Velocímetro de flujo, gráfico de densidad).
3. **Centro de Descargas:**
    - Video procesado (.mp4).
    - Métricas crudas (.csv).
    - Informe de incidentes (.json).
    - Log de auditoría Blockchain (.json).
    - **Dashboard HTML:** Un reporte autocontenido generado con Plotly que incluye gráficos interactivos de distribución vehicular y series temporales.

7. Guía de Instalación y Despliegue

Esta sección detalla los pasos para que el jurado pueda replicar la demo funcional.

7.1 Requisitos Previos

- Sistema Operativo: Windows 10/11, Linux (Ubuntu 20.04+) o macOS.
- Python 3.9 o superior.
- Recomendado: GPU NVIDIA con drivers CUDA (para acelerar YOLOv8), aunque funciona en CPU.

7.2 Estructura del Repositorio

/NeuralLogic_AeroTrace

/NeuralLogic_AeroTrace/BLOCKCHAIN

├── app.py # Punto de entrada Frontend (Streamlit)

├── main.py # Lógica del Backend y Visión Artificial

├── bsv_blockchain.py # Módulo de registro de evidencias

├── requirements.txt # Dependencias

└── outputs/ # Directorio generado automáticamente

7.3 Pasos de Instalación

Clonar el repositorio:

git clone https://github.com/MahsaSimaei/NeuralLogic_AeroTrace

cd C:\Repositorios\NeuralLogic_AeroTrace\BLOCKCHAIN

Instalar dependencias:

pip install -r requirements.txt

Nota: Asegúrese de instalar `ultralytics`, `streamlit`, `supervision`, `opencv-python`, `plotly`.

Ejecutar la aplicación:

streamlit run app.py

7.4 Guía de Uso de la Demo

1. El navegador abrirá automáticamente `http://localhost:8501`.
2. En la barra lateral (**Sidebar**), ajuste "Confianza del Modelo" a 0.25.
3. Seleccione la fuente: Suba un video `.mp4` o un archivo `.zip` con imágenes secuenciales.
4. Defina la **Zona de Interés**: Ajuste los sliders X/Y para delimitar la carretera.
5. Pulse **"🚀 INICIAR ANÁLISIS"**.
6. Al finalizar, navegue por las pestañas de resultados

8. Conclusiones e Impacto

AeroTrace v2.0 demuestra cómo la convergencia de la Inteligencia Artificial y la tecnología Blockchain puede transformar la gestión del tráfico urbano.

1. **Impacto Tecnológico:** Hemos logrado adaptar modelos de última generación (YOLOv8) a escenarios complejos de vista aérea utilizando técnicas de *slicing* y calibración física, superando las limitaciones de las cámaras estáticas convencionales.
2. **Transparencia:** La integración del diseño de transacciones BSV añade una capa de verdad inmutable, indispensable para la automatización de multas o la justificación de inversiones en infraestructura pública.
3. **Escalabilidad:** La arquitectura modular permite reemplazar el input de archivos por un *stream* RTSP directo de drones 5G en el futuro, habilitando la gestión de tráfico en tiempo real real.