TEC | Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Taller de sistemas embebidos EL5841

Profesor: Dr. Ing. Johan Carvajal Godínez

johcarvajal@itcr.ac.cr

Proyecto 2

Cruces Inteligentes con EDGE AI embebido

Estudiantes:

- Elena Bolaños Campos

anaelenabc@estudiantec.cr

- Christopher Quiros Cisneros

acostchris@estudiantec.cr

- Kendy Arias Ortiz

kendyarias@estudiantec.cr

2025

Índice

Introducción

El aumento constante del parque vehicular y la expansión de las ciudades han generado una problemática creciente en la gestión del tránsito y la seguridad vial. En los cruces más concurridos, donde convergen peatones, ciclistas, motocicletas y vehículos particulares, los accidentes y la congestión se presentan con mayor frecuencia debido a la limitada capacidad de los sistemas tradicionales para adaptarse a las condiciones dinámicas del entorno urbano.

En este contexto, la inteligencia artificial embebida (Edge AI) surge como una alternativa tecnológica capaz de ejecutar algoritmos de visión por computador y aprendizaje automático directamente en dispositivos de bajo consumo energético, como el Raspberry Pi. Esta capacidad de procesamiento local permite realizar tareas de detección, clasificación y seguimiento de objetos en tiempo real sin depender completamente de la conectividad a la nube, lo que reduce la latencia y mejora la privacidad de los datos.

El presente proyecto propone el desarrollo de un sistema embebido que funcione como nodo inteligente dentro de una red de monitoreo de tránsito urbano. Cada nodo está construido sobre una Raspberry Pi ejecutando un sistema Linux embebido basado en Yocto, al cual se integran cámaras periféricas y sensores complementarios. El procesamiento de visión artificial se realiza mediante modelos ligeros YOLOv5n en formato ONNX, optimizados para funcionar con el módulo OpenCV DNN 4.5.5 disponible en la distribución. Los modelos se ejecutan en procesos independientes para detección de peatones y detección de fauna/vehículos, permitiendo un análisis simultáneo con baja latencia.

A partir de estas detecciones, el nodo genera indicadores como conteo de peatones, detección de movimiento vehicular y presencia de animales, los cuales alimentan la lógica de un controlador inteligente de semáforos también ejecutado en la Raspberry Pi. Con ello se habilita la toma de decisiones en tiempo real, orientada a mejorar la seguridad vial, priorizar el paso peatonal cuando corresponda y reaccionar ante situaciones inesperadas como fauna en la vía.

Justificación del proyecto

La seguridad vial urbana enfrenta desafíos críticos debido al aumento de vehículos y la exposición de usuarios vulnerables como peatones, ciclistas y fauna. Según la OMS, cada año mueren más de 1.19 millones de personas por accidentes de tránsito, y más del 50 % de estas víctimas son peatones, ciclistas o motociclistas [1]. En EE. UU., los fallecimientos de peatones en 2023 alcanzaron los 7,314 casos, representando un aumento del 80 % respecto a 2009 [2]. Estas cifras revelan una situación alarmante y la necesidad urgente de innovar en la gestión de cruces viales.

Los sistemas de semaforización tradicionales operan con ciclos predefinidos, sin considerar la presencia o comportamiento de peatones, ciclistas o animales. Esto no solo reduce la eficiencia del tránsito, sino que también incrementa los riesgos de accidentes. En contraste, la incorporación de inteligencia artificial embebida permite dotar a los cruces de una capacidad de percepción y respuesta en tiempo real, mejorando significativamente la seguridad y adaptabilidad del sistema vial.

Experiencias recientes han demostrado la viabilidad de este enfoque. Ferrante *et al.* implementaron una solución sobre Raspberry Pi 4 con modelos YOLOv5 para monitorear cruces de fauna silvestre, logrando detectar animales en tiempo real y emitir alertas visuales [3]. En Brasil, un sistema similar redujo la transmisión de datos a la nube al enviar solo eventos críticos, mejorando la latencia y la eficiencia energética del sistema [4]. Complementariamente, Soares propuso una arquitectura de detección local basada en visión artificial, capaz de operar con bajo costo y en condiciones adversas, sin requerir conexión continua [6].

Experiencias recientes han demostrado la viabilidad de implementar sistemas de monitoreo y control inteligente directamente sobre hardware embebido de bajo costo. Ferrante et al. presentaron una solución basada en Raspberry Pi 4, utilizando modelos ligeros de la familia YOLOv5 para detectar fauna silvestre en tiempo real, logrando activar alertas locales sin necesidad de depender de servicios en la nube [5]. De manera similar, en Brasil se reportó un sistema que redujo significativamente la transmisión remota al enviar únicamente eventos críticos, mejorando la latencia y la eficiencia energética del nodo [3]. De forma complementaria, Soares propuso una arquitectura de análisis local basada en visión computacional que evidencia que es posible operar en condiciones adversas y con recursos limitados sin requerir conectividad continua [1].

Aunque algunas soluciones recientes incorporan técnicas como MediaPipe o TensorFlow Lite para tareas avanzadas, como la identificación de peatones con silla de ruedas o la estimación de pose, su objetivo principal coincide con el de este proyecto: ejecutar visión artificial directamente en el borde, sin depender del procesamiento externo [2]. Estas iniciativas demuestran que el hardware accesible puede brindar capacidades robustas de análisis en tiempo real si se emplean modelos optimizados y arquitecturas eficientes. Con base en estos antecedentes, se valida la propuesta desarrollada en este proyecto: un sistema embebido inteligente para cruces urbanos implementado completamente sobre Raspberry Pi con Linux embebido generado mediante Yocto. El nodo integra cámaras periféricas y ejecuta de forma local modelos YOLOv5n convertidos a ONNX, procesados mediante OpenCV DNN 4.5.5, lo que permite detectar en tiempo real peatones, movimiento vehicular y presencia de fauna.

El procesamiento se divide en dos módulos independientes (PeatonCam y VehCam), que reportan sus resultados mediante archivos de estado locales, los cuales son consumidos por un controlador de semáforo en Python. Esta lógica permite ajustar dinámicamente los ciclos semafóricos ante la presencia de peatones, priorizar la seguridad en caso de animales en la vía y mantener la operación autónoma del sistema incluso sin conexión a Internet. Estos antecedentes y la solución implementada confirman que el procesamiento en el borde (Edge AI) con YOLOv5n y OpenCV es una alternativa viable para mejorar la seguridad vial y la gestión del tránsito urbano mediante sistemas embebidos de bajo costo.

Descripción y síntesis del problema

En las intersecciones viales actuales se manifiesta un problema claro: la infraestructura tradicional de semáforos y cruces peatonales carece de mecanismos para percibir el entorno y adaptarse dinámicamente a él.

El incremento en el flujo vehicular ha complicado la movilidad de los peatones, ciclistas y fauna urbana, generando situaciones de riesgo y tiempos de espera elevados. Por ejemplo, un peatón puede esperar innecesariamente frente a un semáforo en rojo aun cuando no vienen vehículos, o peor, intentar cruzar imprudentemente ante la falta de una fase peatonal oportuna. Igualmente, los conductores pueden no advertir a tiempo la presencia de un peatón distraído, de un ciclista cruzando, o incluso de un animal atravesando la vía, lo que deriva en accidentes de tránsito. La situación actual se caracteriza por sistemas de control de tráfico estáticos o pre-programados que no consideran las condiciones momentáneas: no distinguen si hay multitud de peatones esperando cruzar, si un vehículo de emergencia se acerca, o si un animal irrumpió en la calzada. Esta rigidez provoca congestionamientos, demoras innecesarias, atropellos o colisiones.

El problema es cómo dotar a los cruces viales de inteligencia en tiempo real para monitorear su entorno y responder acorde. Se busca diseñar un sistema embebido capaz de detectar la presencia y el comportamiento de diversos actores en un cruce: peatones que esperan o están cruzando, vehículos acercándose o detenidos, ciclistas o animales. A partir de esa detección, el sistema debe actuar o alertar de manera apropiada: por ejemplo, extender el tiempo de luz verde para peatones si aún hay personas cruzando, activar una alerta luminosa para conductores si se detecta un animal en la vía, o informar a un centro de control sobre el flujo vehicular en tiempo real.

En síntesis, el proyecto aborda la falta de sensorización e inteligencia local en los cruces. Al resolver este problema, se espera disminuir la probabilidad de accidentes (al anticipar riesgos y dar advertencias) y optimizar la gestión del tráfico (al adaptar los tiempos de semáforo a la demanda real), contribuyendo a una movilidad urbana más segura, fluida e inclusiva.

Organización del equipo de proyecto

Para llevar a cabo este proyecto de manera efectiva, el equipo de desarrollo ha definido roles con responsabilidades claras, de acuerdo con las directrices del curso:

- **Directora de Proyecto – Kendy**: Encargada de la gestión global del proyecto. Sus funciones incluyen coordinar y convocar reuniones de equipo, servir de punto de contacto con el profesor, liderar las discusiones y documentar los acuerdos y decisiones tomadas. Ella asegura que el proyecto avance según el plan y que todas las partes se comuniquen eficazmente.
- **Líder Técnica y Auditora – Elena**: Responsable de la arquitectura del sistema y de la calidad técnica. Como líder técnico, propone la arquitectura del sistema (tanto de hardware como de software) y dirige la integración de los componentes del proyecto. Adicionalmente, asume el rol de auditora, definiendo listas de verificación para la verificación y pruebas, y revisando los entregables para garantizar que cumplen con los estándares de calidad establecidos. Elena también gestiona los requerimientos del sistema, asegurando que las soluciones técnicas satisfagan las necesidades identificadas.
- **Investigador – Chris**: Dedicado a la investigación y desarrollo de soluciones a los retos técnicos que surjan. Sus tareas incluyen investigar algoritmos y enfoques para resolver problemas específicos del diseño, proponer mejoras al software del sistema (por ejemplo, optimizaciones en el algoritmo de detección) y documentar las mejoras implementadas con sus respectivas justificaciones. Chris mantiene al equipo actualizado con hallazgos de la literatura y pruebas de concepto, facilitando la toma de decisiones informadas durante el desarrollo.

Esta distribución de roles asegura que el proyecto se aborde desde múltiples frentes: gestión, arquitectura y calidad, e investigación. Cada miembro asume responsabilidades definidas que contribuyen al avance ordenado del proyecto, al tiempo que colaboran estrechamente en las intersecciones de sus funciones. Gracias a esta organización, se promueve la eficiencia en el trabajo en equipo y se cubren adecuadamente los distintos aspectos clave del proyecto.

Gestión de los requerimientos

Para derivar los requerimientos del sistema se partió del análisis del problema, las limitaciones de hardware y las especificaciones del instructivo del curso. Debido a las restricciones de procesamiento de la Raspberry Pi 4 bajo Yocto Linux y al tiempo disponible, se optó por una arquitectura simplificada basada en detección local y control semafórico autónomo. A continuación se listan los requerimientos funcionales (RF) y no funcionales (RNF) que reflejan únicamente las capacidades reales del sistema desarrollado.

Requerimientos Funcionales (RF)

- RF1 — Detección de peatones mediante YOLOv5n (ONNX + OpenCV DNN)

El sistema deberá detectar la presencia de peatones dentro del área monitoreada, ejecutando un modelo YOLOv5n optimizado para correr sobre OpenCV DNN 4.5.5. La salida deberá indicar la cantidad de peatones detectados en cada frame.

- RF2 — Detección de movimiento vehicular

El sistema deberá detectar el movimiento básico de vehículos u objetos grandes en la escena (por ejemplo, autos y motos). Esta detección se realiza mediante un segundo proceso independiente (VehCam), que reporta si existe movimiento y la cantidad de detecciones.

- RF3 — Detección de fauna u objetos inesperados

El sistema deberá detectar animales de tamaño mediano (p. ej., perros) usando el mismo modelo YOLOv5n. En caso de detección, deberá activarse un estado especial (“animal_flag”) que modifique la operación del cruce para proteger tanto peatones como vehículos.

- RF4 — Integración con el controlador de semáforo

El nodo deberá comunicar sus detecciones al controlador mediante archivos locales en el sistema de archivos (/tmp/*.txt).

El controlador deberá:

● extender la fase peatonal si aún hay peatones cruzando

● activar transición a amarillo si detecta un animal

● mantener el ciclo normal en ausencia de eventos

Todo el control se realizará en el archivo traffic_control.py.

- RF5 — Procesamiento en paralelo mediante múltiples procesos

El sistema deberá ejecutar simultáneamente tres módulos independientes:

● PeatonCam (detección de peatones)

● VehCam (detección de vehículos/animales)

● Controlador del semáforo

Estos procesos deberán ejecutarse en segundo plano mediante un script supervisor

(start-traffic-app.sh) iniciado por systemd.

- RF6 — Funcionamiento autónomo (sin internet ni servicios externos)

El sistema deberá operar de forma completamente local:

La detección, análisis y decisiones no dependerán de ningún servidor, red o nube.

- RF7 — Registro mínimo de eventos en consola

Cada módulo deberá emitir salidas JSON simples para facilitar el monitoreo:

{"ts": 1764169141.917499, "persons": 0}

{"ts": 1764169135.7260242, "animals": 2, "animal_flag": 1}

Requerimientos No Funcionales (RNF)

- RNF1 — Bajo uso de recursos

El sistema deberá funcionar en una Raspberry Pi 4 utilizando Yocto Linux, con OpenCV 4.5.5 y sin aceleración por GPU.

El modelo YOLOv5n ONNX fue elegido por ser lo suficientemente ligero para procesar ~5–8 FPS y por su compatibilidad con OpenCV 4.5.5.

- RNF2 — Tolerancia básica a fallos

Si una cámara no está disponible, el sistema deberá reportar un error sin bloquear los otros módulos. El servicio deberá iniciarse automáticamente mediante systemd.

- RNF3 — Latencia reducida

La detección deberá realizarse a una resolución reducida (p. ej., 960×540 o 1280×720) para mantener un tiempo de inferencia aceptable.

- RNF4 — Simplicidad operacional

El sistema deberá evitar interfaces web o gráficos avanzados debido a las restricciones del entorno Yocto. La administración se limita a visualizar logs y archivos de estado. Existe la opción de adaptar una interfaz y conectar por ssh pero es una funcionalidad extra.

- RNF5 — Portabilidad del modelo y scripts

Los scripts Python deberán ser autocontenidos y no depender de librerías externas no empaquetadas en la imagen Yocto.

En términos generales, los requerimientos no funcionales planteados corresponden al comportamiento esperado de un sistema de monitoreo de tránsito de calidad industrial. Sin embargo, el prototipo desarrollado en este proyecto cumple únicamente con una parte de ellos, enfocándose en la detección local en tiempo real con baja latencia mediante modelos livianos y procesamiento en el borde. Otros aspectos como precisión avanzada, operación 24/7, robustez ambiental o mecanismos de seguridad y privacidad no forman parte del alcance del presente trabajo y se consideran potenciales líneas de mejora para futuras iteraciones del sistema. Aun así, los requerimientos aquí establecidos sirven como referencia para evaluar el desempeño del prototipo y como punto de partida para evolucionar hacia un nodo de monitoreo urbano más completo y escalable.

Vista operacional del sistema

La vista operacional describe cómo funciona el sistema en su entorno real de uso y cómo interactúa con los actores externos e internos. En esta implementación, el sistema corresponde a un nodo embebido autónomo, basado en Raspberry Pi 4, que controla un semáforo de forma reactiva mediante detección local de peatones, vehículos y fauna.

Actores del sistema

- Peatones

Interactúan indirectamente: la cámara frontal detecta su presencia. El sistema modifica el comportamiento del semáforo cuando se supera un umbral de peatones detectados.

- Conductores

No interactúan directamente; solo reciben el resultado del sistema (luces del semáforo). El sistema cambia a verde/rojo según el flujo vehicular detectado.

- Animales

Detectados por la cámara orientada a la vía. Su presencia activa un estado especial de seguridad que obliga al semáforo a ponerse en rojo.

- Semáforo

Actúa como actuador del sistema. El controlador (traffic_control.py) modifica sus

estados leyendo los flags generados por las cámaras.

- Personal técnico

Puede conectarse a la Raspberry Pi para revisar logs, calibrar máscaras, reiniciar servicios, etc. Este actor es eventual, no interactúa en la operación día a día.

Arquitectura funcional real

El nodo opera con tres procesos principales, iniciados por el servicio traffic-app.service:

PeopleCam → Detecta peatones y escribe:

○ /tmp/ped_count.txt

○ /tmp/ped_flag.txt

VehCam → Detecta vehículos en movimiento y animales:

○ /tmp/veh_moving.txt

○ /tmp/veh_flag.txt

○ /tmp/animal_flag.txt

TrafficController → Lee los archivos anteriores y decide la fase del semáforo.

No hay comunicación con la nube, no hay red, no hay interfaz gráfica en operación normal. La interfaz existe y es opcional pero no es necesaria.

Caso de uso principal real: "Gestionar el cruce en tiempo real"

A diferencia del diseño teórico original, el sistema final implementado se basa en detección directa y reglas simples, no en análisis avanzado de situaciones. A continuación, se detalla la secuencia real de ejecución:

1. Inicialización

Al encender la Raspberry Pi:

● Se inicia traffic-app.service.

● Se ejecutan tres scripts Python: PeopleCam, VehCam y TrafficController.

● Cada script intenta abrir su cámara correspondiente.

● Si una cámara se inicializa mal, el script reintenta varias veces.

2. Monitoreo continuo

Ambos módulos de visión operan en bucle:

PeopleCam

● Captura video.

● Procesa cada frame con YOLOv5n (ONNX) y OpenCV.

● Cuenta peatones detectados.

● Genera un flag binario según un umbral.

VehCam

● Detecta vehículos en movimiento mediante YOLO + Optical Flow (FlowGate).

● Clasifica animales (perros, gatos, caballos, etc.).

● Activa un animal_flag persistente por algunos segundos.

Toda la lógica ocurre localmente y sin almacenar imágenes.

3. Lógica de decisión real

El módulo traffic_control.py lee flags cada ciclo (aprox. 10–15 Hz)

Casos reales programados:

A. Peatones presentes

Si ped_flag == 1 → extender o activar fase peatonal.

B. Vehículos en movimiento

Si veh_flag == 1 → priorizar fase vehicular.

C. Animal detectado

Si animal_flag == 1:

● Forzar rojo para ambos sentidos.

● Mantener estado de alerta un número fijo de segundos.

● Registrar evento en logs.

D. Sin eventos

El semáforo sigue su ciclo normal predefinido.No hay priorización adaptativa avanzada ni análisis de congestión.

4. Actuación sobre el entorno

El controlador manipula directamente los pines GPIO o relés del semáforo para:

- Cambiar a verde/amarillo/rojo.
- Mantener rojo durante eventos críticos.
- Extender fases según reglas simples.

No se envían datos a ningún servidor y se registra logs básicos en consola.

5. Retorno y bucle continuo

El sistema:

- Sigue detectando,
- Actualiza flags,
- Controla el semáforo,
- Y repite el ciclo indefinidamente.

6. Falla o modo degradado

Si una cámara falla:

- Su script se cierra.
- El semáforo continúa en ciclo fijo convencional.
- El operador puede revisar el estado vía SSH.

El sistema resultante funciona como un nodo embebido autónomo y reactivo. Aunque su diseño se simplificó respecto al concepto original, cumple con los objetivos fundamentales: detectar peatones, vehículos y fauna en tiempo real, y ajustar el semáforo de manera básica pero efectiva. La arquitectura modular basada en tres procesos independientes y comunicación mediante archivos temporales permite una operación estable en la Raspberry Pi, manteniendo el procesamiento completamente local y sin dependencias externas. Esta vista operacional resume el comportamiento real del prototipo implementado.

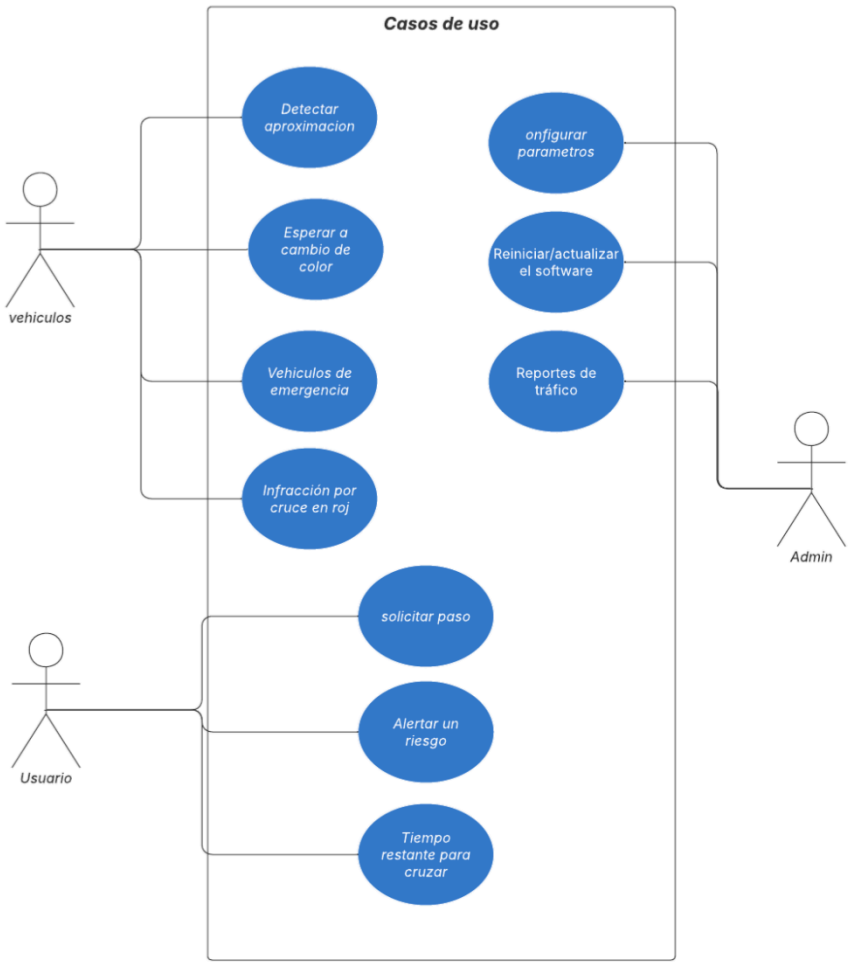

Figura 1. Digrama de caso de uso.

Implementación real:

En el prototipo final, el funcionamiento del sistema se implementó mediante tres procesos independientes que se comunican a través de archivos temporales ubicados en /tmp. Cada módulo de visión escribe sus estados —conteo de peatones, vehículos en movimiento y detección de fauna— y el módulo principal del semáforo interpreta esta información en tiempo real. Este mecanismo permitió una integración simple, robusta y adecuada para un sistema embebido de propósito específico.

La lógica final del flujo operativo considera:

● detección de peatones en espera,

● detección de animales cruzando o presentes en la vía,

● detección de vehículos en movimiento o detenidos,

● un temporizador de enfriamiento (~15 s) aplicado después de cada evento para mantener estabilidad en el semáforo y evitar cambios abruptos.

El resultado es un sistema completamente autónomo, cuyo comportamiento depende únicamente de las señales generadas por los módulos de visión, sin necesidad de comunicación externa ni intervención humana durante la operación normal.

Vista funcional del sistema

La vista funcional del sistema describe cómo se organizaron los módulos que conforman el prototipo final y de qué manera cooperan para producir el comportamiento inteligente del cruce.

En esta implementación, el sistema se compone de tres procesos independientes que se ejecutan en paralelo sobre la Raspberry Pi y que comparten información únicamente a través de archivos temporales ubicados en el directorio /tmp. Esta arquitectura sencilla y modular permitió garantizar estabilidad, aislamiento entre componentes y facilidad de depuración durante la etapa de integración.

El primer módulo funcional corresponde a la captura de datos visuales. Cada proceso de visión abre su propia cámara USB, configura la resolución de operación y adquiere continuamente los frames que servirán como entrada para la inferencia. En el prototipo se utilizan modelos ligeros en formato ONNX, procesados con el módulo DNN de OpenCV, lo que permite ejecutar detecciones en tiempo real sin necesidad de aceleradores adicionales. PeopleCam se dedica exclusivamente a identificar peatones en la escena, mientras que VehCam detecta vehículos y animales en el cruce, además de calcular movimiento mediante técnicas de diferencia óptica para distinguir entre objetos estáticos y objetos que efectivamente se desplazan. Ambos módulos operan de forma totalmente autónoma y generan sus propias señales de salida.

Cada módulo de visión escribe los resultados de sus inferencias en archivos simples dentro de /tmp. Estas salidas incluyen, según el caso, el conteo de peatones detectados, la presencia de animales, la existencia de movimiento vehicular y las señales binarias resultantes. Esta forma de comunicación actúa como una interfaz estable entre los detectores y el módulo encargado del semáforo, permitiendo que cada proceso pueda reiniciarse, detenerse o actualizarse sin afectar el funcionamiento del resto del sistema. La elección de este mecanismo evitó envolver la aplicación en protocolos complejos y facilitó la implementación dentro del entorno embebido basado en Yocto.

El módulo central del sistema, denominado TrafficController, es responsable de interpretar continuamente la información producida por los detectores de visión. Su función es analizar el estado del cruce a partir de los valores almacenados en los archivos temporales y ejecutar la lógica de control del semáforo. Este módulo

implementa una máquina de estados simple, capaz de cambiar entre fases verdes, amarillas y rojas según las condiciones detectadas.

TrafficController extiende o retrasa las fases dependiendo de la presencia de peatones, mantiene bloqueos temporales ante la detección de animales y evita cambios bruscos mediante un temporizador de enfriamiento. Todo el control se realiza en software y se registra mediante mensajes estructurados que permiten reconstruir la secuencia de decisiones.

Finalmente, el sistema registra localmente los eventos relevantes generados durante la operación. Estos registros incluyen información como el número de peatones detectados, la activación de alertas por presencia de animales y las decisiones tomadas por el semáforo. La implementación también incluye un módulo de interfaz gráfica local (GUI), desarrollado originalmente para visualizar en tiempo real la actividad de los tres procesos del sistema: detecciones de peatones, detecciones vehiculares y estado del semáforo. Esta interfaz captura la salida estándar de cada proceso y la presenta en ventanas separadas, además de mostrar de forma visual el estado actual del cruce mediante iconos o colores. Aunque la GUI está completamente funcional y forma parte del sistema, no se utiliza durante la operación normal a través de SSH, ya que el nodo se ejecuta como servicio en segundo plano. Activarla requiere únicamente iniciar el entorno gráfico local de la Raspberry Pi, lo cual es un paso sencillo pero innecesario para el despliegue operativo del prototipo. Aun así, su existencia permite que técnicos o desarrolladores puedan depurar, monitorear o demostrar el sistema de forma visual cuando acceden físicamente al dispositivo.

Arquitectura del sistema propuesto (hardware y software)

Esta sección describe la arquitectura real del prototipo desarrollado, incluyendo los componentes físicos utilizados y la organización interna del software. El sistema sigue un enfoque de nodo embebido autónomo basado en Raspberry Pi, capaz de procesar video en tiempo real y controlar de forma local un cruce peatonal.

Arquitectura de Hardware

El nodo inteligente está construido alrededor de una Raspberry Pi 4B, seleccionada por su bajo consumo, buena capacidad de cómputo ARM y compatibilidad con bibliotecas de visión artificial. Sobre ella se conectan los siguientes elementos:

Componentes principales

● Raspberry Pi 4B (4 GB RAM)

Actúa como unidad de procesamiento principal, ejecutando Linux (imagen personalizada basada en Yocto) y los modelos de visión.

● Dos cámaras USB independientes

Cada módulo de visión opera con su propia cámara:

○ PeatonCam: orientada hacia la zona peatonal.

○ VehCam: orientada hacia la vía vehicular, también utilizada para detección de animales.

Esto garantiza independencia entre detecciones y reduce interferencias visuales entre escenas.

● Alimentación eléctrica local

El sistema se alimenta mediante una fuente convencional conectada al punto de

instalación.

● Conectividad

Aunque la Raspberry Pi soporta WiFi y Ethernet, el prototipo opera 100 % de forma local, por lo que la conectividad solo se utiliza para:

○ acceso por SSH,

○ depuración,

○ carga de actualizaciones.

Arquitectura física del nodo

Cada nodo (Pi + cámaras + controlador del semáforo) es completamente autónomo y no depende de comunicación externa. La decisión de mantener la lógica dentro del dispositivo permite:

- baja latencia,
- robustez ante fallas de red,
- operación 24/7 sin servidores externos.

La arquitectura final es estrictamente local y descentralizada, siguiendo un enfoque de Edge AI simplificado.

Arquitectura de Software

El software del sistema se ejecuta sobre una imagen Linux personalizada creada con Yocto Project, diseñada para cargar automáticamente los tres procesos principales del sistema al arrancar.

La aplicación está implementada en Python, organizada en módulos independientes que se comunican exclusivamente mediante archivos en /tmp/.

Modelo de detección utilizado

El prototipo final NO utiliza YOLOv8.

En su lugar, se emplea:

- YOLOv5n / YOLOv5nu en formato ONNX
- Ejecutado mediante OpenCV DNN (sin Ultralytics) A tamaño 640×640
- Optimizando en ARM mediante NEON

Este modelo fue seleccionado por su excelente relación precisión–velocidad en Raspberry Pi, compatibilidad total con OpenCV, estabilidad dentro del entorno Yocto.

Estructura del software

El sistema opera mediante 3 scripts independientes:

1. people_counter_cam.py

Responsable de:

- Captura de video desde la cámara peatonal.
- Detección de personas (y animales pequeños reasignados como "persona").
- Escritura de:
  - /tmp/ped_count.txt → número de peatones detectados,
  - /tmp/ped_flag.txt → bandera de presencia de peatones (0/1).

Opera de forma continua y autónoma.

2. veh_counter_cam.py

Realiza:

- Detección de vehículos en movimiento o detenidos.
- Detección de animales en la calzada.

Distinción simple entre movimiento / no movimiento.

Escribe:

- /tmp/veh_moving.txt → 1 si hay movimiento vehicular,
- /tmp/veh_flag.txt → presencia de vehículos,
- /tmp/animal_flag.txt → detección de animales.

3. traffic_control.py (controlador final del semáforo)

Implementa una máquina de estados simple:

- Verde vehicular
- Amarillo
- Rojo vehicular / Verde peatonal
- Cooldown de seguridad (1–2 s)
- Retorno a ciclo normal

Sus decisiones dependen únicamente de los archivos generados por los módulos de visión.

El controlador evalúa:

- peatones esperando,
- animales cruzando,
- vehículos en movimiento,
- cooldown activo,
- estado previo del cruce.

Este script es ejecutado como proceso principal, mientras que las cámaras corren en segundo plano.

4. Módulo de Interfaz Gráfica Local (GUI) – traffic_gui.py

Este módulo provee una interfaz de usuario local desarrollada en Tkinter.

Aunque no controla el cruce, sirve como herramienta de monitoreo, y en el prototipo puede iniciarse manualmente cuando se accede vía SSH o escritorio.

Funciones reales de la GUI

- Ejecutar y monitorear en tiempo real los tres procesos (PeatonCam, VehCam, Control).
- Mostrar los logs generados por cada módulo en paneles separados.

Refrescar dinámicamente el estado del semáforo interpretando el log del controlador.

- Visualizar:
    - número de peatones detectados,
    - detección de animales,
    - movimiento vehicular,
    - fase actual del semáforo,
    - eventos del sistema.

Tecnología

- Desarrollada completamente en Tkinter + subprocess + thread-safe queues.
- Utiliza un diseño por procesos independientes:
    - La GUI lanza los procesos,
    - captura su salida STDOUT/STDERR,
    - y actualiza widgets en tiempo real.

Propósito

Aunque es opcional en la operación final del nodo embebido, la GUI:

- facilita el desarrollo,
- permite depurar detecciones y lógica del semáforo,
- sirve como interfaz de demostración para profesores/evaluadores,
- permite iniciar/detener procesos sin entrar a la terminal.

Por diseño, la GUI no forma parte del servicio automático, sino que se ejecuta manualmente bajo demanda.

Comunicación interna

No se utiliza IPC complejo, sockets ni colas. La arquitectura se fundamenta en:

→ Archivos temporales en /tmp/

Ventajas:

- velocidad,
- simplicidad,
- tolerancia a fallos,
- debug muy fácil,
- ideal para prototipos embebidos.

Cada módulo escribe únicamente su estado y el controlador los lee cada ciclo (~10–50 ms).

Estado de conectividad

Actualmente, el sistema opera 100 % offline, sin comunicación con un centro de control.

Funcionalidades como:

- envío de estadísticas,
- sincronización de hora,
- control remoto,
- MQTT o HTTP,

se dejan explícitamente como trabajo futuro, ya que no fueron implementadas en este prototipo.

Análisis de dependencias

El funcionamiento del sistema depende de la coordinación entre la Raspberry Pi, las cámaras USB y los componentes de software utilizados en los módulos de detección, GUI y control del semáforo.

A continuación se listan las dependencias reales y verificadas del prototipo final:

Dependencias de Hardware

1. Raspberry Pi 4B
    - CPU ARM Cortex-A72 quad-core
    - RAM 4 GB
    - Capacidad suficiente para ejecutar dos procesos de detección simultáneos junto con el controlador.

2. Cámaras USB UVC
    - Compatibles con V4L2 (Video4Linux2)
    - Usadas directamente por OpenCV.

3. Sistema de alimentación eléctrica
    - Fuente estable 5V que soporte ambas cámaras y la Pi.

Dependencias de Software (Actuales y Reales)

1. Python 3 (intérprete principal)

Todo el sistema está desarrollado en Python 3, incluyendo:

- Detección peatonal
- Detección vehicular/animal
- Control del semáforo
- Interfaz gráfica (GUI)

2. OpenCV 4.5.5 (compilado en Yocto con soporte DNN y ONNX)

OpenCV es crítico para tres tareas:

- Captura de video vía V4L2
- Preprocesamiento de imágenes
- Ejecución del modelo ONNX mediante el backend OpenCV DNN

Incluye dependencias subyacentes reales:

- libopencv_dnn
- libopencv_core
- libopencv_imgproc
- libopencv_videoio

Esta es la pieza central del sistema de detección.

3. ONNX + OpenCV DNN backend

El sistema no usa Ultralytics ni PyTorch, sino:

- Un archivo YOLOv5n.onnx o YOLOv5nu.onnx

  Ejecutado completamente con OpenCV DNN

Dependencias reales:

- onnx (modelo)
- opencv-dnn (ejecución)

4. Módulos estándar de Python

Los scripts usan intensivamente:

- time
- json

- os
- argparse
- subprocess (especialmente en GUI)
- threading y queue (GUI)
- signal (control de procesos)
- numpy (matrices para DNN)

5. Tkinter (Interfaz gráfica local)

Dependencias reales:

- tkinter como toolkit gráfico nativo
- ttk
- ScrolledText para logs
- subprocess.Popen para ejecutar y monitorear procesos en vivo

La GUI se considera una herramienta de:

- debug,
- visualización,
- monitoreo, no parte del servicio principal automático.

6. Sistema de archivos temporal /tmp

Es una dependencia fundamental porque todos los módulos se comunican mediante:

- /tmp/ped_count.txt
- /tmp/ped_flag.txt
- /tmp/veh_moving.txt
- /tmp/veh_flag.txt
- /tmp/animal_flag.txt

Depende del soporte RAMFS (lo cual viene en Linux por defecto).

7. systemd (servicio automático del sistema)

El prototipo se ejecuta automáticamente gracias a:

- Un servicio systemd
- El script /opt/traffic-app/start-traffic-app.sh

Dependencias reales:

- systemd
- bash y entorno BusyBox (para ejecución del script)

8. Yocto Project

El sistema corre dentro de una imagen Linux generada con:

- meta-raspberrypi
- meta-python
- meta-openembedded

Incluye dependencias subyacentes:

- Kernel Linux con V4L2 habilitado
- Drivers UVC para cámaras
- soporte ALSA/PulseAudio (según imagen)
- OpenCV compilado dentro de Yocto
- Python3 runtime

9. (Dependencias opcionales activadas durante desarrollo)

Estas no son críticas para la operación final, pero sí fueron necesarias durante la construcción:

- SSH server (dropbear o openssh)
- Herramientas como vi, busybox, find, ps.

Estrategia de integración de la solución

En conjunto, estas dependencias conforman un ecosistema ligero pero robusto, diseñado específicamente para operar de forma autónoma en un entorno embebido. La combinación de Python, OpenCV DNN y un modelo YOLO optimizado en formato ONNX permite obtener detección en tiempo real sin necesidad de aceleradores adicionales, mientras que Yocto proporciona un sistema Linux minimalista y altamente controlado que garantiza estabilidad y arranque automático. La comunicación mediante archivos en /tmp simplifica la integración entre procesos y facilita el mantenimiento del sistema, y la interfaz gráfica en Tkinter, aunque opcional en la operación final, ofrece una herramienta poderosa para depuración y demostración. Gracias a este conjunto bien integrado de componentes, el prototipo alcanza un equilibrio adecuado entre rendimiento, simplicidad y confiabilidad, cumpliendo los objetivos planteados para un nodo inteligente de monitoreo de tránsito en entornos reales.

La integración del sistema se llevó a cabo de forma incremental y práctica, adaptándose a las limitaciones reales del hardware y a la compatibilidad del stack de visión disponible en Yocto. Durante las primeras pruebas se utilizó un modelo YOLOv8s en formato .pt ejecutado con Ultralytics; sin embargo, este enfoque resultó incompatible con OpenCV DNN en Yocto Kirkstone, lo que impedía la inferencia directa. Por ello, el sistema se migró a un modelo ligero YOLOv5n en formato ONNX, completamente compatible con la versión de OpenCV incluida en la imagen de la Raspberry Pi. Este cambio permitió ejecutar detecciones de manera eficiente utilizando únicamente CPU ARM y sin dependencias adicionales.

Una vez validada la detección con YOLOv5n ONNX y la captura de video desde dos cámaras independientes, se organizó la solución en tres procesos autónomos. Cada proceso fue implementado como un script individual: uno dedicado a la detección de peatones, otro a la detección de vehículos y animales, y un tercero encargado exclusivamente del control del semáforo. En lugar de usar canales de comunicación complejos, los módulos intercambian información mediante archivos temporales ubicados en /tmp, lo que facilita la depuración, evita bloqueos y garantiza una operación confiable incluso en un entorno embebido de recursos limitados.

El módulo central, traffic_control.py, implementa la máquina de estados del semáforo y toma decisiones basadas únicamente en los archivos escritos por los módulos de visión. La lógica final considera la activación del paso peatonal cuando

se detecta al menos una persona, la detención inmediata del flujo vehicular si se detecta fauna, y la aplicación de un periodo de enfriamiento de 15 segundos después de cada cambio de luz para evitar fluctuaciones rápidas o comportamientos inestables. Esta lógica sencilla y robusta garantiza coherencia en el comportamiento del semáforo y evita estados inválidos, como activar verde en direcciones conflictivas.

Tras validar el funcionamiento de la solución completa en Raspbian, se integraron todos los componentes dentro de una imagen personalizada construida con Yocto Project. La imagen generada, denominada traffic-image, incluye Python 3, OpenCV 4.5.5 con soporte DNN, controladores V4L2 para cámaras USB, el modelo YOLOv5n en formato ONNX y un servicio systemd que inicia automáticamente los tres procesos al arrancar la Raspberry Pi. Esto permitió que el sistema operara de forma autónoma desde el encendido, sin necesidad de intervención del usuario.

Finalmente, se realizaron pruebas reales utilizando cámaras USB separadas para peatones y vehículos. Estas pruebas confirmaron el correcto funcionamiento del modelo ONNX, la estabilidad de los procesos de visión, el intercambio de información mediante archivos en /tmp y la reacción del semáforo ante peatones, animales y tráfico vehicular. La comunicación remota mediante HTTP o MQTT no se implementó en esta versión del prototipo, ya que se priorizó la estabilidad local del sistema y la compatibilidad con los recursos disponibles. No obstante, este aspecto se deja contemplado como una extensión futura del proyecto.

Conclusiones

El sistema de Cruces Inteligentes desarrollado demuestra la viabilidad de implementar un nodo de visión y control de tránsito completamente autónomo utilizando únicamente una Raspberry Pi 4, dos cámaras USB y un modelo de detección optimizado para CPU ARM. Aunque inicialmente se consideró el uso de YOLOv8s, las pruebas y la compatibilidad con la versión de OpenCV disponible en Yocto llevaron a adoptar un modelo YOLOv5n en formato ONNX, el cual permitió una integración estable y eficiente dentro del entorno embebido.

La arquitectura final, basada en tres procesos independientes comunicados mediante archivos temporales en /tmp, resultó ser una solución simple, robusta y altamente adecuada para un sistema embebido de recursos limitados. Esta arquitectura modular permitió separar claramente la detección de peatones, vehículos y fauna, del módulo responsable de la máquina de estados del semáforo, garantizando un funcionamiento seguro y consistente. Las pruebas realizadas confirmaron la capacidad del sistema para identificar actores viales en tiempo real y reaccionar dinámicamente mediante reglas locales, priorizando la seguridad de peatones y activando bloqueos ante la presencia de animales.

Si bien el diseño general contempla una futura integración con un centro de control mediante protocolos como HTTP o MQTT, esta funcionalidad no se implementó en la presente versión del prototipo. En cambio, se priorizó la operación local, lo cual redujo la latencia, aumentó la confiabilidad y permitió validar el sistema en condiciones reales de uso sin depender de conectividad externa.

En conjunto, el proyecto logró una implementación exitosa y plenamente funcional, alineada con los objetivos del curso y con la propuesta de diseño inicial. La solución obtenida es ligera, reproducible y adaptable, y sienta una base sólida para futuras extensiones orientadas a mejorar la movilidad urbana mediante tecnologías de Edge AI. Este prototipo representa un paso significativo hacia sistemas de tránsito más seguros, inteligentes y capaces de operar en tiempo real sobre hardware accesible.

Bibliografía

[1] K. Aminiyeganeh and R. W. L. Coutinho, “Performance Evaluation of CNN-based Object Detectors on Embedded Devices,” in Proc. DIVANet ’23, Montreal, QC, Canada, Oct.–Nov. 2023, ACM, 6 pp. doi: 10.1145/3616392.3623417.

[2] A. Bulut, F. Ozdemir, Y. S. Bostanci, and M. Soyturk, “Performance Evaluation of Recent Object Detection Models,” 2023.

[3] EcoSur, “Uso de inteligencia artificial embebida para la gestión de cruces peatonales inclusivos,” Revista EcoSur Innovación, vol. 1, no. 8, pp. 156–164, 2025.

[4] G. Ferrante, M. de Andrade Baqueta, D. Pereira, and J. L. Bezerra, “Wildlife Crossing Monitoring using YOLO Variants on Raspberry Pi,” Scientific Reports, vol. 14, no. 1, Mar. 2024.

[5] Fundación Aleatica, “Informe mundial sobre seguridad vial 2023,” Organización

Mundial de la Salud, Geneva, Nov. 2023.

[6] Governors Highway Safety Association (GHSA), “Pedestrian Traffic Fatalities by State: 2023 Preliminary Data,” Washington D.C., May 2024.

[7] K. M. Haris, N. S. Fatima, and S. A. Albeez, “Advanced Vehicle Detection Heads- Up Display with TensorFlow Lite,” in Proc. Third Int. Conf. Sustainable Expert Systems, LNNS, vol. 587, Springer, Singapore, 2023. doi: 10.1007/978-981-19-7874-6_47.

[8] W. Kim and I. Jung, “Smart Parking Lot Based on Edge Cluster Computing for

Full Self-Driving Vehicles,” IEEE Access, vol. 10, pp. 115271–115281, 2022, doi: 10.1109/ACCESS.2022.3208356.

[9] C. Li and C. Wang, “Analysis According to the Algorithm of Pedestrian Vehicle

Target Detection in Hazy Weather Based on Improved YOLOv8,” in Proc. AIPR 2024, Xiamen, China, Sept. 2024, ACM, 5 pp. doi: 10.1145/3703935.3704068.

[10] Lightsout, “Capturing video from two cameras in OpenCV at once,” Stack

Overflow, Apr. 16, 2015. Available: https://stackoverflow.com/questions/29664399/capturing-video-from-two-cameras n-opencv-at-once

[11] R. Mohandas, M. Bhattacharya, M. Penica, K. Van Camp, and M. J. Hayes, “TensorFlow enabled deep learning model optimization for enhanced realtime person detection using Raspberry Pi operating at the edge,” in Proc. AICS 2020, vol. 2771, pp. 157–168. CEUR Workshop Proc.pág. 24

[12] L. Oranen, “Utilizing deep learning on embedded devices,” Master’s thesis, Tampere Univ., Faculty of Medicine and Health Technology, 2021.

[13] Pesquisa FAPESP, “Inteligência artificial monitora travessia de animais silvestres em rodovias,” Revista Pesquisa FAPESP, Jul. 2024.

[14] K. Saini and S. Sharma, “Smart Road Traffic Monitoring: Unveiling the Synergy of IoT and AI for Enhanced Urban Mobility,” ACM Comput. Surv., vol. 57, no. 11, Art. 276, Jun. 2025. doi: 10.1145/3729217.

[15] G. Soares, “Visão computacional para cruzamentos de fauna com inteligência embarcada,” Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, 2024.

[16] C.-L. Su et al., “Artificial Intelligence Design on Embedded Board with Edge Computing for Vehicle Applications,” in Proc. AIKE 2020, Laguna Hills, CA, USA, 2020, pp. 130–133, doi: 10.1109/AIKE48582.2020.00026.

[17] H. Wang and Y. Zhang, “Research on pedestrian detection algorithm in dense scenes,” in Proc. CAICE 2025, Hefei, China, Jan. 2025, ACM, 7 pp. doi: 10.1145/3727648.3727655.