

**Proyecto: “Carrito autónomo”**

**Participantes:**

**Carlos Orozco**

**Andrés Núñez**

**Jonathan Trujillo**

**Pablo Ramos**

**Supervisión y asesoría:**

**Martín Sinsel**

**Alejandro Lozano**

**12/05/2024**

**Detalles iniciales**

Inicialmente, este proyecto está planteado para tener una duración de un semestrere y se plantea que se quede como base para futuros proyectos tanto hechos por los estudiantes participando desde este punto, como futuros estudiantes inscritos a talleres tanto de robótica, como de microcontroladores, bajo la tutela de Martín Sinsel, Alejandro Lozano o cualquier otro profesor que forme parte del DESI y cuente con la autorización de por lo menos alguno de los ya mencionados supervisores.

Todos los puntos, tanto requerimientos como limitaciones planteadas para este punto del proyecto, quedan sujetos a revisión y evaluación por parte de los supervisores, con el fin de moldear y acotar el proceso de desarrollo de este proyecto, por lo que todo lo que se mencione en este documento puede evolucionar a corto y mediano plazo según se defina en ajustes y negociaciones con los profesores supervisando.

**Requerimientos y limitaciones del proyecto**

* Se está trabajando con tecnología relativamente nueva, por lo que inicialmente se busca lograr que el sistema pueda detectar y clasificar obstáculos estáticos en un ambiente controlado, con el fin de a futuroo agregar la funcionalidad de detección de obstáculos en movimiento, para la toma de decisiones en tiempo real sobre la ruta del carrito.
* Las pruebas de funcionalidad inicial se realizarán en un ambiente controlado donde variarán las posiciones de obstáculos predefinidos.
* Se plantea que la información se obtenga mientras el carrito esté en movimiento, por lo que este tendrá que operar a bajas velocidades con el fin de tomar video a un máximo de 30 fotogramas por segundo y procesarlo para la toma de decisiones.
* Los algoritmos de “pathing” del carrito se realizarán mediante el uso de información que se obtendrá de una red neuronal de tipo convolucional para identificar obstáculos en una posición relativa a la del carrito.
* La red neuronal por utilizar pasará por un proceso de “transfer learning”, por lo que se buscará tomar una red neuronal entrenada anteriormente con el fin de ahorrar tiempo entrenando solamente las capas de “SoftMax” para que el sistema pueda reconocer los obstáculos de la forma buscada.
* El sistema se montará en un armazón de carrito y se alimentará a base de una o más baterías para su operación.
* Se agregará un módulo de GPS para cuestiones de crear geocercas, que funcionarán como las coordenadas objetivo para el carrito y serán el área en el que este se detendrá.
* La mayoría de la comunicación entre periféricos y controladores se realizará por protocolo SPI con el fin de obtener los datos lo más rápido posible para su procesamiento.

**Propuesta de configuración de hardware a utilizar**

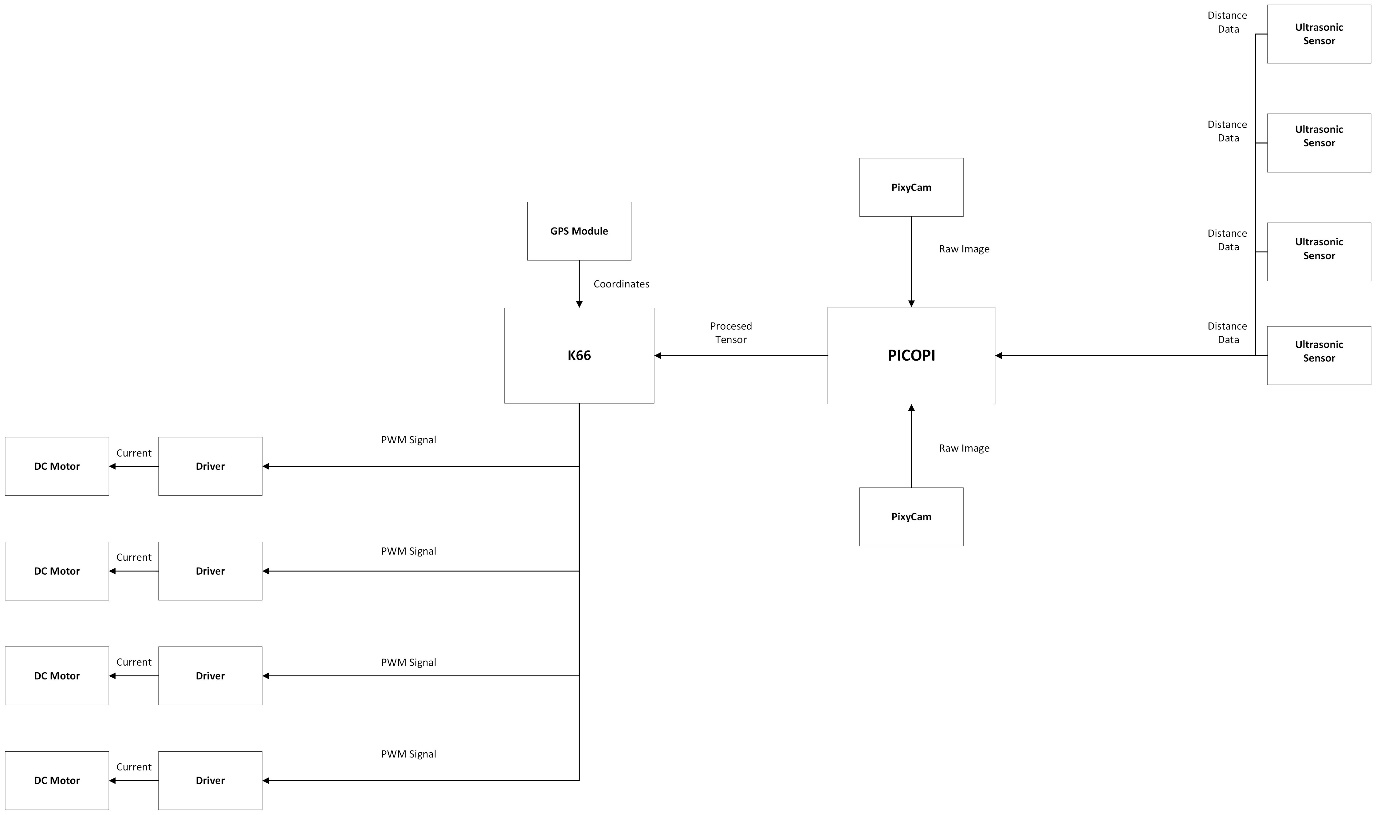


Ilustración 1 Propuesta de hardware actual

**Problemática**

El club de robótica del ITESO está tomando nuevos pasos en adaptarse a los nuevos alumnos de mecatrónica, por lo que le profesor Martin Sinsel propuso que se creara un carrito autónomo que ejecutara los algoritmos de pathing mediante redes neuronales y se guiara por GPS para poder ir de un punto a otro.

**Elementos de hardware adicionales al microcontrolador K66**

* Carrito
* Encoders
* Pixy cam (2)
* GPS
* Pico pi imx8mm
* Sensores ultrasónicos (2 adelante y 2 atrás)

**Algoritmos importantes en el sistema**

Yolov5 tiny para el reconocimiento de objetos

Algoritmo de pathing (a definir)

**Fuente de energía del sistema**

Batería del carrito (Li-Po Battery pack 7.4V two cells)

Power bank para dispositivos móviles.

**Tipo de casing, medidas iniciales**

Mobile Robotics Buggy3 Kit Including RDDRONE-FMUK66:

<https://www.nxp.com/design/design-center/designs/mobile-robotics-buggy3-kit-including-rddrone-fmuk66-and-peripherals:MR-BUGGY3>

**Productos similares en el mercado**

Actualmente se están manejando pruebas en algunas ciudades de estados unidos con entregas de Uber Eats hechas por carros 100% autónomos, esta idea de entregas autónomas también está siendo extrapolada a Uber para lograr hacer viajes sin necesidad de choferes. También se ha visto implementado una versión similar al proyecto que funcionan como meseros en restaurantes, principalmente en restaurantes ubicados como japón o estados unidos.



Ilustración 2 Ejemplo de producto parecido

Link a la página del producto

<https://www.uber.com/us/es/autonomous/>

<https://www.forbes.com/sites/alanohnsman/2022/09/07/uber-taps-nuros-street-legal-robots-for-food-deliveries/?sh=36f878fc127e>

<https://dimercom.mx/robot-para-servicio-de-mesero-o-repartidor-t5lb-capacidad-de-carga-de-hasta-50-kg-y-pantalla-tactil/>

<https://eu.robotshop.com/es/collections/delivery-robots>

**Costo**

Lo único que se puede encontrar de manera accesible en venta son los robots de restaurantes los cuales se encuentran en un rango de precio de entre 150,000 pesos a 290,000 pesos mexicanos

**Principales diferencias**

Estos vehículos de entregas autónomas usan una cámara con libertad de 360° a la vez de uso de sensores y LIDAR, usan su propia IA entrenada por más de ciento de millones de horas de humanos manejando.

**Mayores retos en el desarrollo del proyecto**

Adaptar algoritmos de pathing a la salida de una red neuronal con algoritmo yolov4 tiny que procesa video en tiempo real y con este tomar decisiones de control

**Esquemático del proyecto**

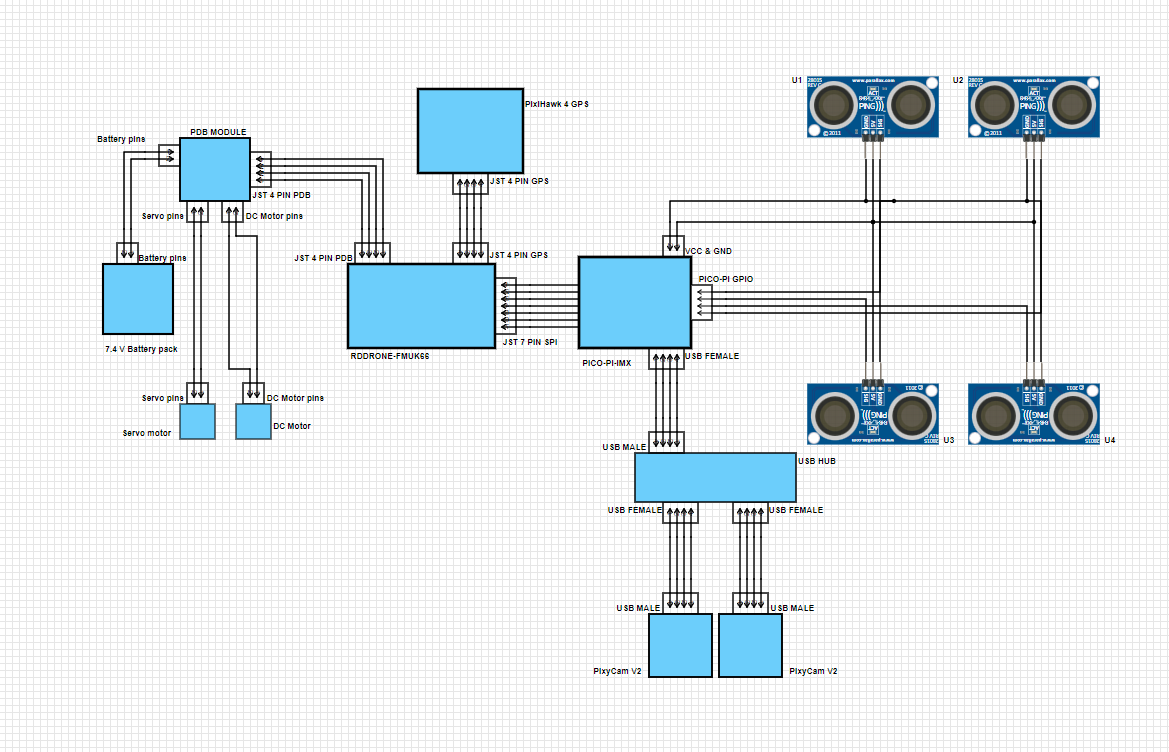


Ilustración 3 Esquemático

**Diagrama a bloques de la estructura del código (arquitectura)**

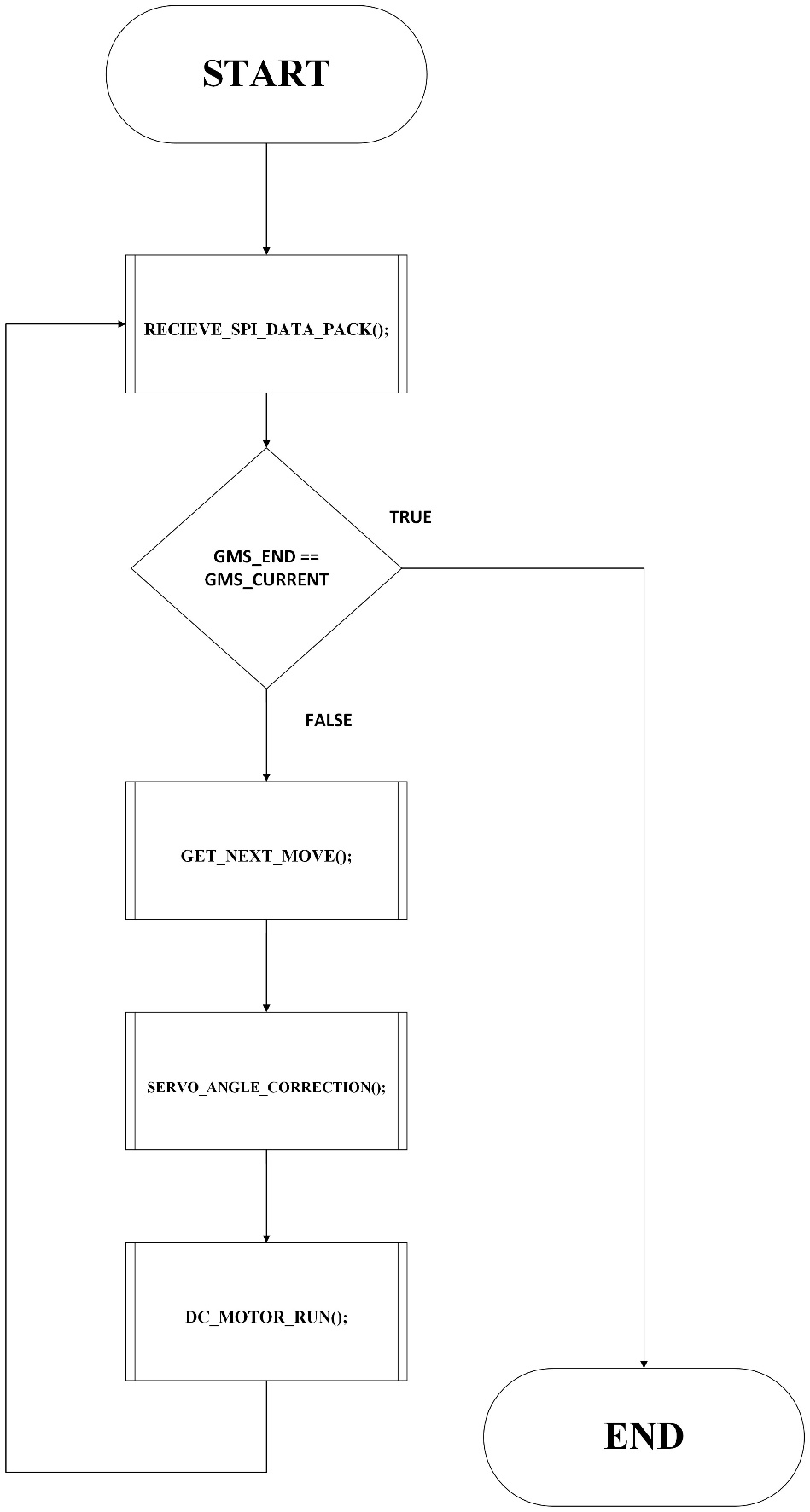


Ilustración 4 Algoritmo principal de K66

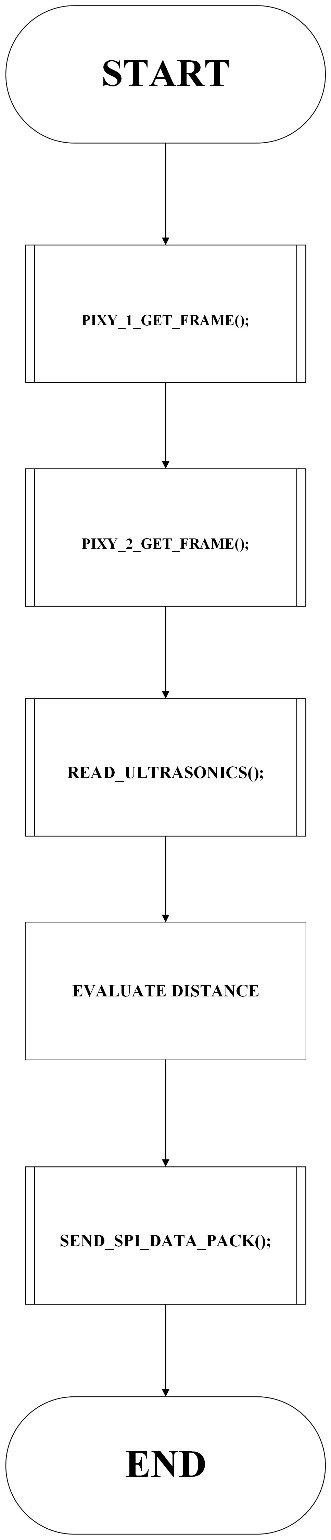


Ilustración 5 Algoritmo principal de IMX

**Códigos**

Los códigos avanzados hasta este punto son:

* Red neuronal YoloV5 nano.
* Adaptación de drivers de PixyCamV2.
* Compilación de drivers de cámara propia de IMX-PICO-PI (En caso de ser necesario).
* Adaptación de drivers de TinyGPS.
* Interpretación del Tensor.
* Transmisión de datos por SPI.
* Lectura de periféricos en K66.
* Bash file de ejecución completa en IMX.
* Servicio de ejecución en arranque para Linux.

Debido a que la mayoría de estos códigos son adaptaciones de drivers o muy pesados como para cargarlos en un comprimido, se adjuntaron imágenes como evidencia de la realización de las adaptaciones y el reentrenamiento de la red, además de una liga al repositorio privado de GITHUB en el que se encuentran estos avances.

Liga al repositorio: <https://github.com/729244/CARRITO_PROJECT>



Ilustración 6 Imagen base tomada en filtro de entrada de estudiantes de ITESO

Ilustración 7 Imagen del tensor que procesó la illustración 4

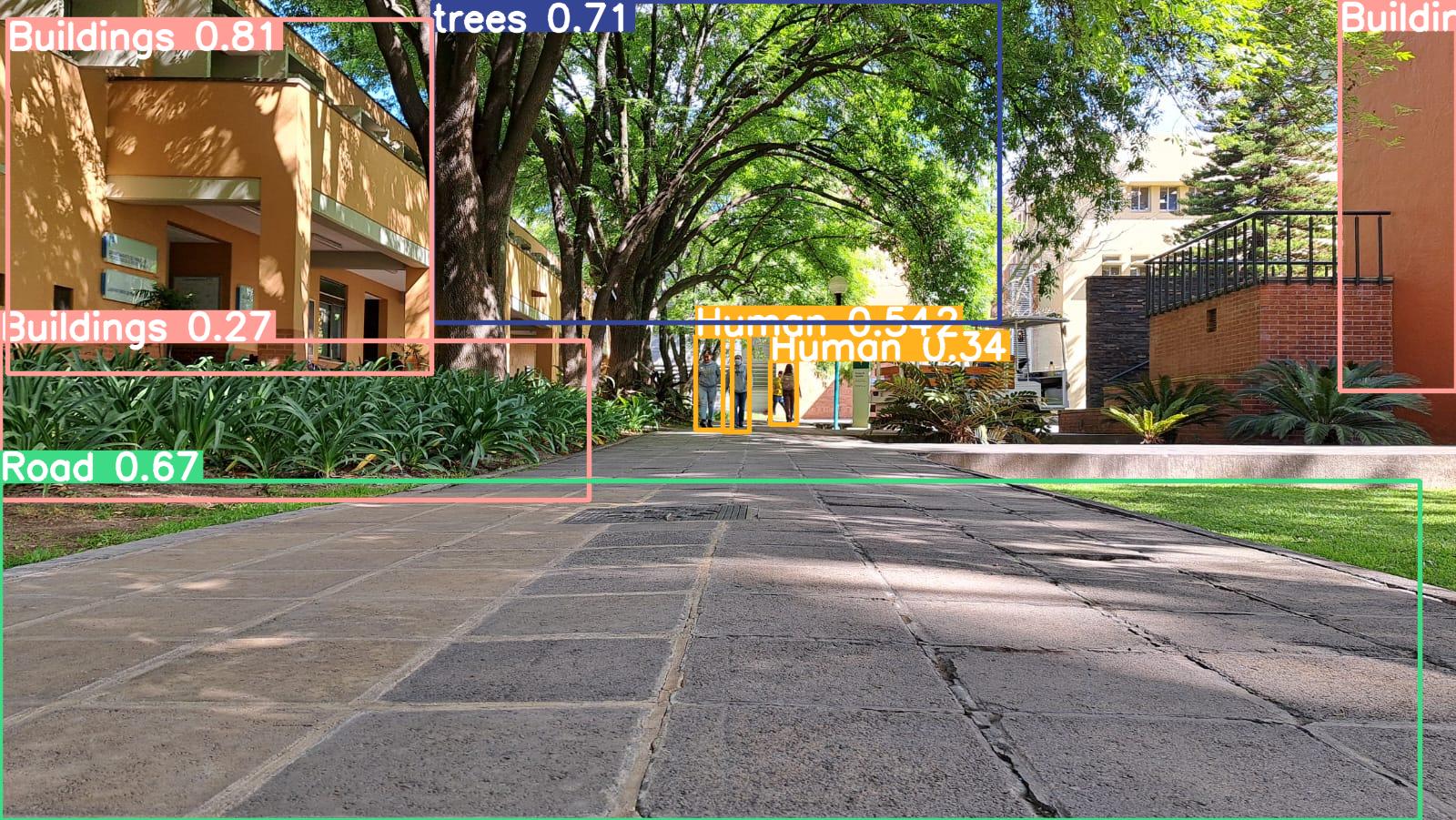


Ilustración 8 Imagen procesada por tensor en entrada filtro edificio I

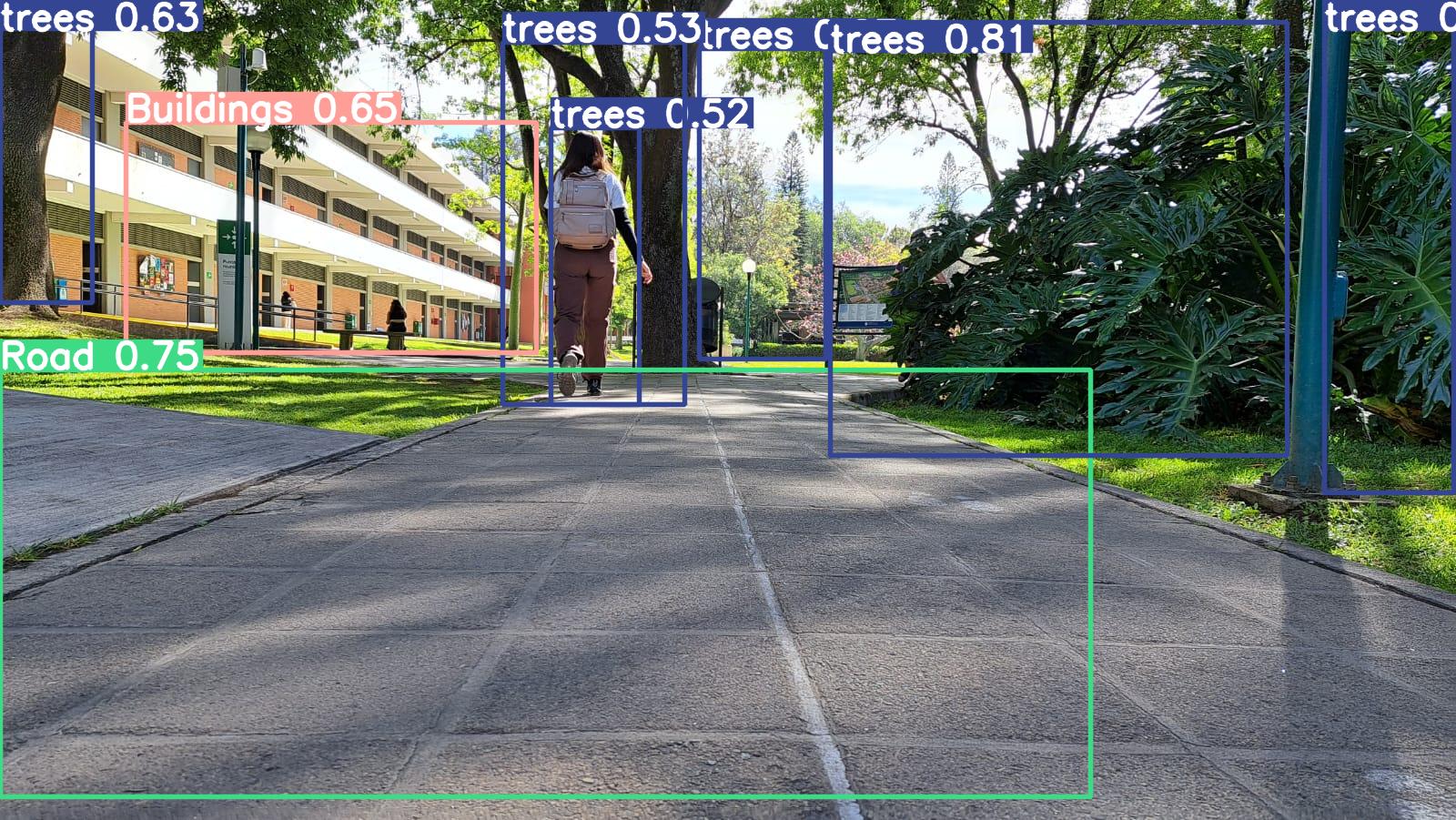


Ilustración 9 Imagen procesada por tensor en entrada filtro edificio I en un ángulo distinto

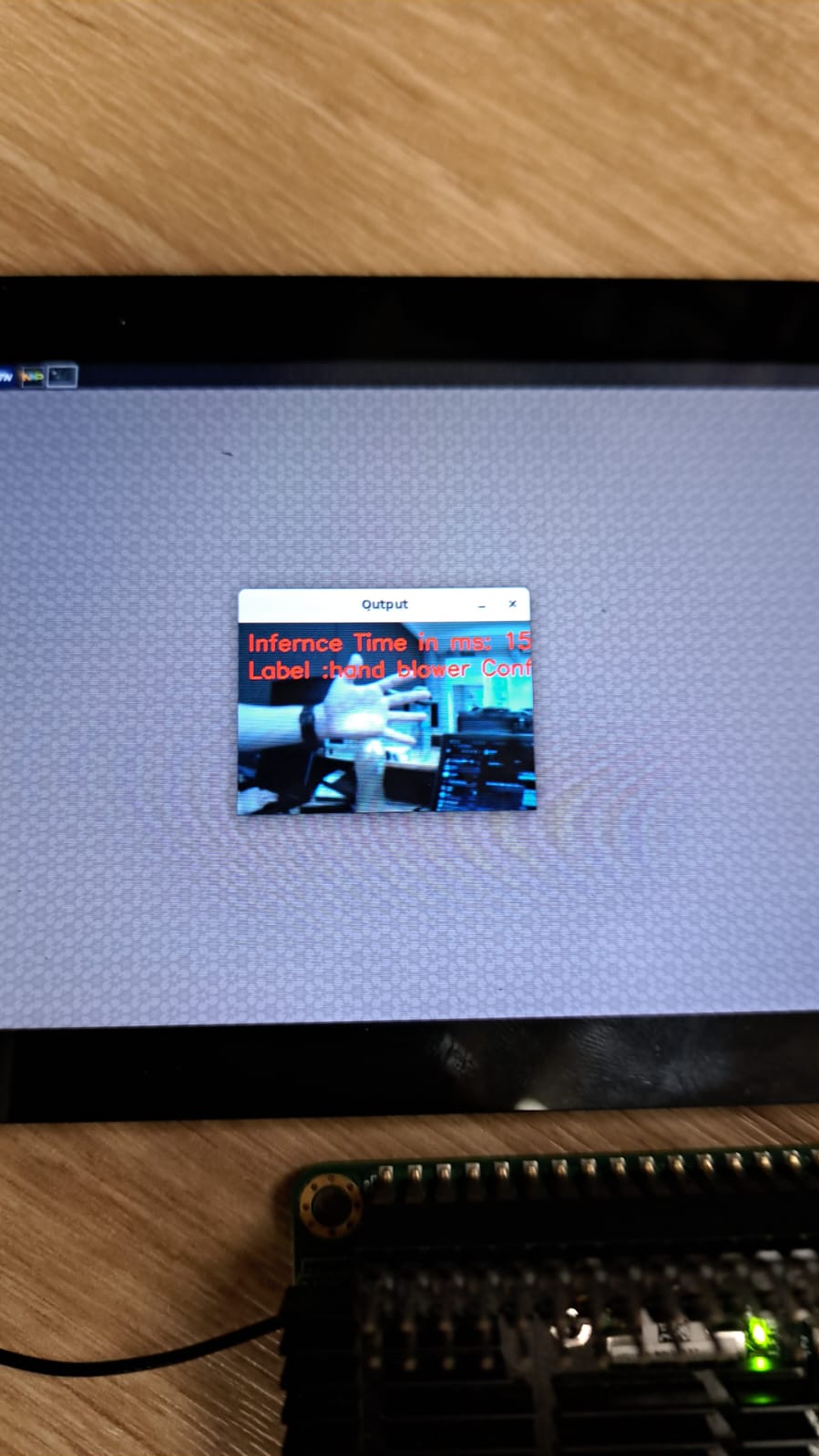


Ilustración 10 Imagen procesada por un tensor dentro de la tarjeta de desarrollo IMX-PICO-PI

**Descripción de las secciones que se consideran más complejas tanto en hardware como en software**

Por la parte de hardware solamente hay dos problemas que en realidad tienen una solución sencilla, el montaje de una segunda cámara y el reacomodo de las tarjetas, la propuesta de solución a la parte del acoplamiento de la segunda cámara es hacer una segunda base para cámara para imprimir en 3D con la que sea posible montar ambas cámaras al carro y por la parte de la reorganización de la tarjeta es posible despegar la cinta de la *K66* y reacomodarla o intercambiar los *PCBs* del carro para mover las tarjetas de lugar.

En software los problemas se vuelven algo más complicados, ya que la integración de todos los periféricos es un tema complicado para hacer el algoritmo de pathing ya que es necesario generar un empaquetado desde la *IMX* para pasar los datos por *SPI* a la *K66* y obtener las señales de *PWM* necesarias.

En el caso del algoritmo de pathing, se está evaluando la posibilidad de realizarlo utilizando ecuaciones de control para automatizar todo el proceso o si este se hará guiado solamente por las coordenadas y la brújula del *GPS*, esto representa un problema ya que es necesario afinar detalles del tensor para darle más precisión a la información contenida en las imágenes y la falta de conocimiento de estos temas puede ser un gran contratiempo.

El último problema o riesgo que se pudo identificar es la reversa cuando el carro se encuentre con obstáculos que no pueda sortear de forma fácil, al igual que con el algoritmo de pathing, se está evaluando si solamente se va a utilizar solamente la información de los sensores ultrasónicos o si se va a desarrollar un algoritmo más complejo para que la reversa funcione de una forma más efectiva y así el carro pueda replantear el camino que tomará sin necesidad de hacer un mapeo completo del terreno.

**Aplicación real**

La aplicación concebida para este proyecto, desde un inicio fue dejar una base de un carrito autónomo para el club de robótica, en este caso la idea es preparar un prototipo con todo lo necesario para realizar un sistema autónomo que controle un carro, a través de dos tarjetas para distribuir la carga de procesamiento donde, el procesamiento de imagen es realizado dentro de una tarjeta con un microprocesador *IMX*, mediante el uso de una red neuronal y el procesamiento de otros periféricos y pathing, se distribuye en una *K66* para que el procesamiento sea más eficaz.

Las aplicaciones principales que estos sistemas tienen en el mundo real es la de realizar envíos de productos mediante robots, en lugares que cuentan con la infraestructura necesaria para su funcionamiento. Otra posible aplicación, aunque a un nivel mucho más avanzada, son las aplicaciones de piloto automático de ciertos automóviles eléctricos que cuentan con esta funcionalidad.

**Explicación de funcionamiento**

El funcionamiento de todo el sistema se divide en dos ejecuciones distintas en distintas tarjetas ambas con diferentes procesos, por un lado toda la parte de procesamiento de imagen dentro de la *IMX* y por el otro lado, los procesos de la *K66*, entre los que se encuentra el procesamiento del *GPS*, los sensores ultrasónicos, la brújula, el control del *PWM* de los motores y finalmente el algoritmo de pathing.

**Obtención de imagen**

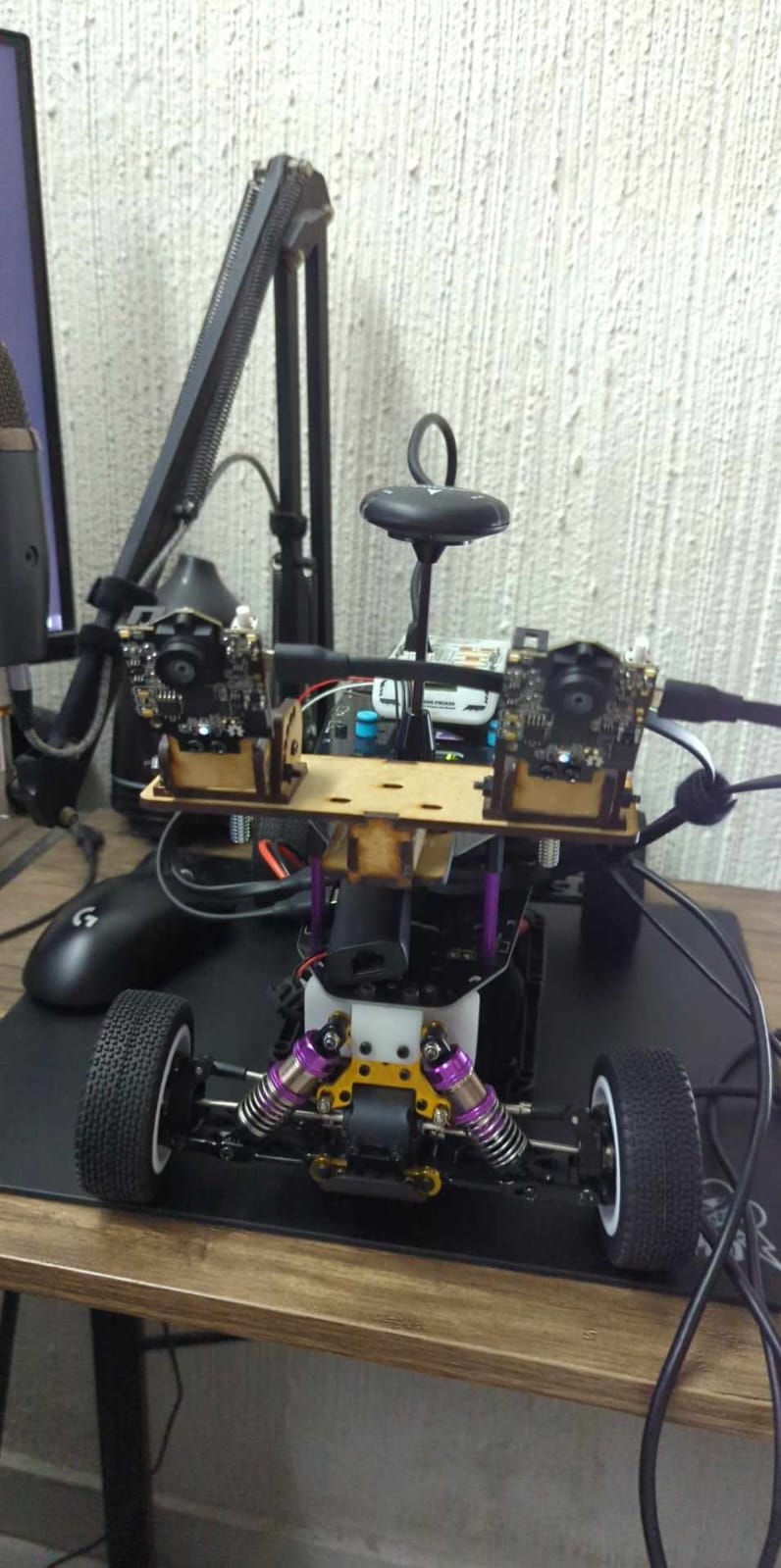
Para el acondicionamiento y procesamiento de las imágenes, se tomaron dos cámaras *Pixy2*, las cuáles se comunicaron mediante *USB* para extraer frames completos desde dos procesos disntintos que se ejecutan en un código de *C++.*

Ilustración 11 Carrito con el montaje de las cámaras

Ambas cámaras fueron montadas al carro con una base de *MDF* que fue diseñada a medida, con el fin de poder hacer ajustes en los ángulos de la visión de los sensores. Con las cámaras montadas, se procedió a conectar un *HUB* de *USB* para poder asignar identificadores distintos dentro del programa y poder repartir el proceso de la obtención de cada frame en forks, dentro de los cuáles, se almacena el frame en un archivo dentro de la *IMX* y se utilizan dos *PIPES* distintos para señalizar que ambos frames se tomaron y que la tarjeta está lista para el procesamiento.

**Procesamiento de imagen**

Por la parte del procesamiento de imagen, se utilizó una red neuronal *YoloV5* *nano* de *Ultralytics* que fue re-entrenada mediante *transfer* *learning* utilizando un *dataset* personalizado de 199 imágenes de los caminos del ITESO con 6 distintas clases catalogadas que abarcan:

* Building (Edificios y muros en general).
* Fall (Escalones).
* Furniture (Contenedores de basura, mapas de la universidad y postes de luz).
* Natural Object (Árboles, arbustos y plantas en general).
* OffRoad (Caminos fuera de alcance, jardineras).
* PersonObject (Personas).

Con este requisito cumplido, se utilizó la herramienta *Google* *Colab* *Research* donde se importó la red neuronal y se escribieron los archivos *.JSON* para direccionar y repartir los datos de entrenamiento, validación y de prueba, para ser relacionados con las clases del *dataset.*

Ilustración 12 Json file de la red neuronal

Con todos estos constraints cumplidos, se hizo el entrenamiento de la red, la red principal se configuró utilizando *PyTorch* para un tamaño de imagen de [416, 416] pixeles a 100 épocas de entrenamiento.

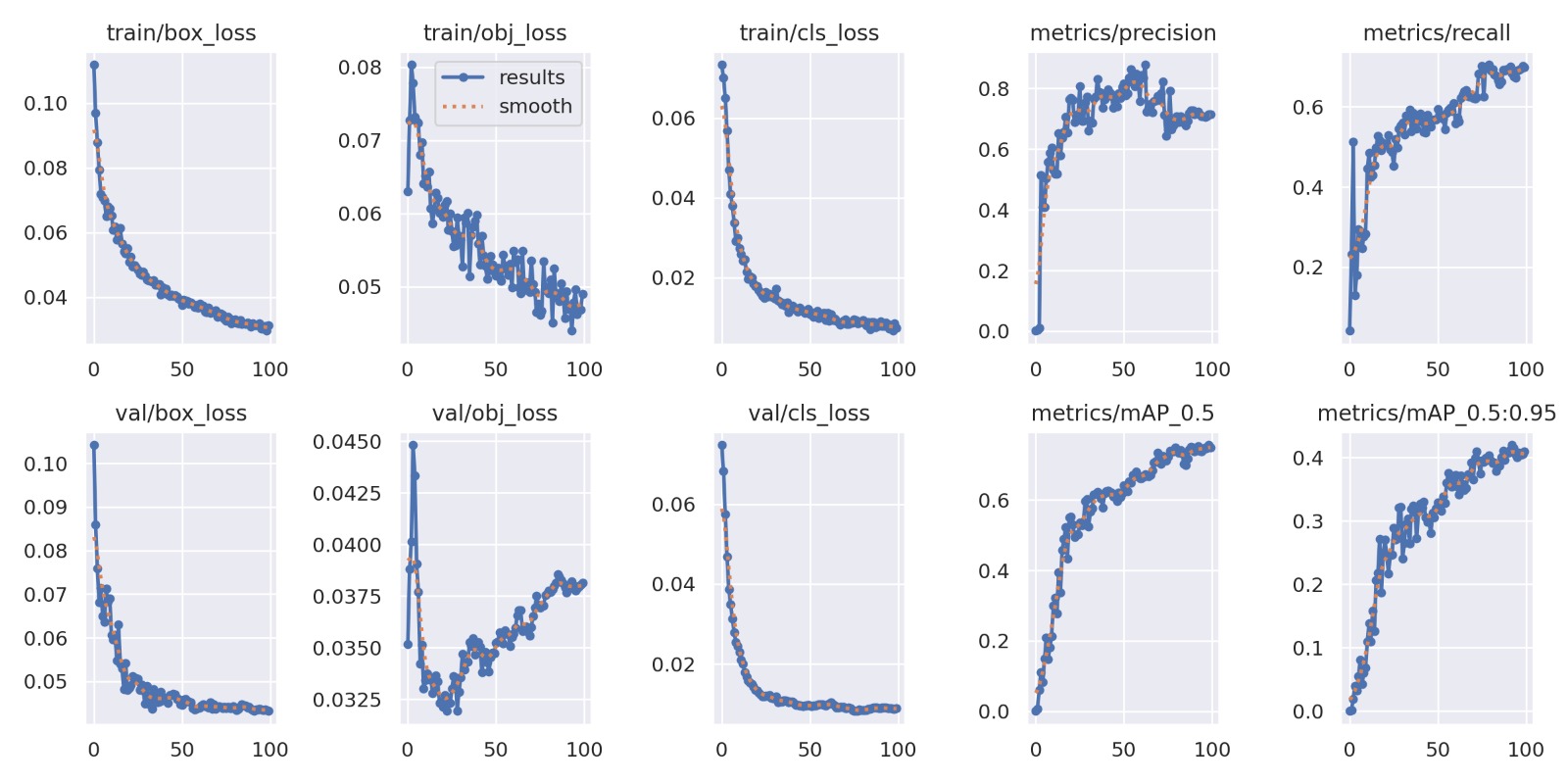
Con la red entrenada, se procedió a hacer un *port* de la red para *TensorFlowLite*, que es una biblioteca de modelos experimentales optimizada para correr redes neuronales en dispositivos de media/baja capacidad de procesamiento.

Ilustración 13 Gráficas de rendimiento de la red neuronal

El resultado de esto, fue una red neuronal más ligera, con precisiones un poco más bajas que requirió ciertos ajustes para tener un funcionamiento adecuado.

Ilustración 14 Malfuncionamiento de la red en TFLite

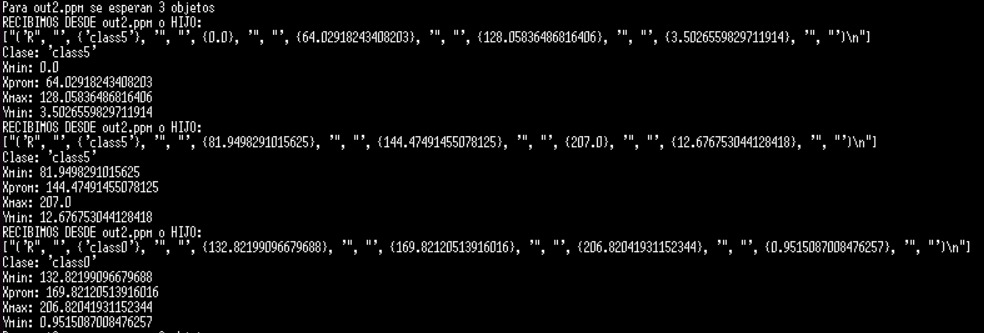
Después de los ajustes realizados, se modificó un código en Python proporcionado por la plataforma de *Ultralythics* para tomar la red reducida y hacer el procesamiento de imagen utilizando *TorchHub*, que es parte de la biblioteca de *PyTorch*, para definir que la red a utilizar es la versión reducida de *TFLite*.

Ilustración 15 Procesamiento de imagen en la red

Como se puede observar en la illustración 13. Se realizaron adecuaciones a la salida del tensor, para sacar los puntos mínimos de los objetos en *x* y *y*, el punto promedio en *x* y la clase a la que estos pertenecen. Todo esto realizado utilizando la librería *Pandas* en *Python*.

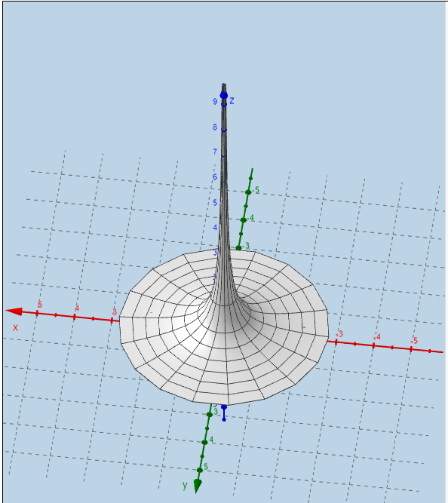
Con estos datos extraídos, se notó un problema en el cálculo de la distancia a través de las cámaras y este fue que, al tratar de tomar la distancia en cámaras inadecuadas para este funcionamiento y sin sensores extras para el apoyo de este cálculo, no es posible cuadrar una distancia con un procentaje de error aceptable, ya que la función que definiría la distancia en una imagen con ese grado de profundidad, se convierte en una función discreta de Torriceli[[1]](#footnote-2), o trompeta de Gabriel, la cuál es una variación de la rotación de y cuenta con un volumen finito, dentro de una superficie infinita, a la cual era necesario realizarle una derivada.

Ilustración 16 Simulación de función de Torriceli en geógebra

Debido a este asunto, se decidió hacer un algoritmo bioinspirado simple, en el que se promedian las coordenadas de los objetos con ambas cámaras, emulando la función de los ojos humanos, con el que se determina si un obstáculo se encuentra cerca o lejos, si es pequeño o grande y si se encuentra a la derecha, izquierda o está centrado en la dirección a la que se dirige el carro.

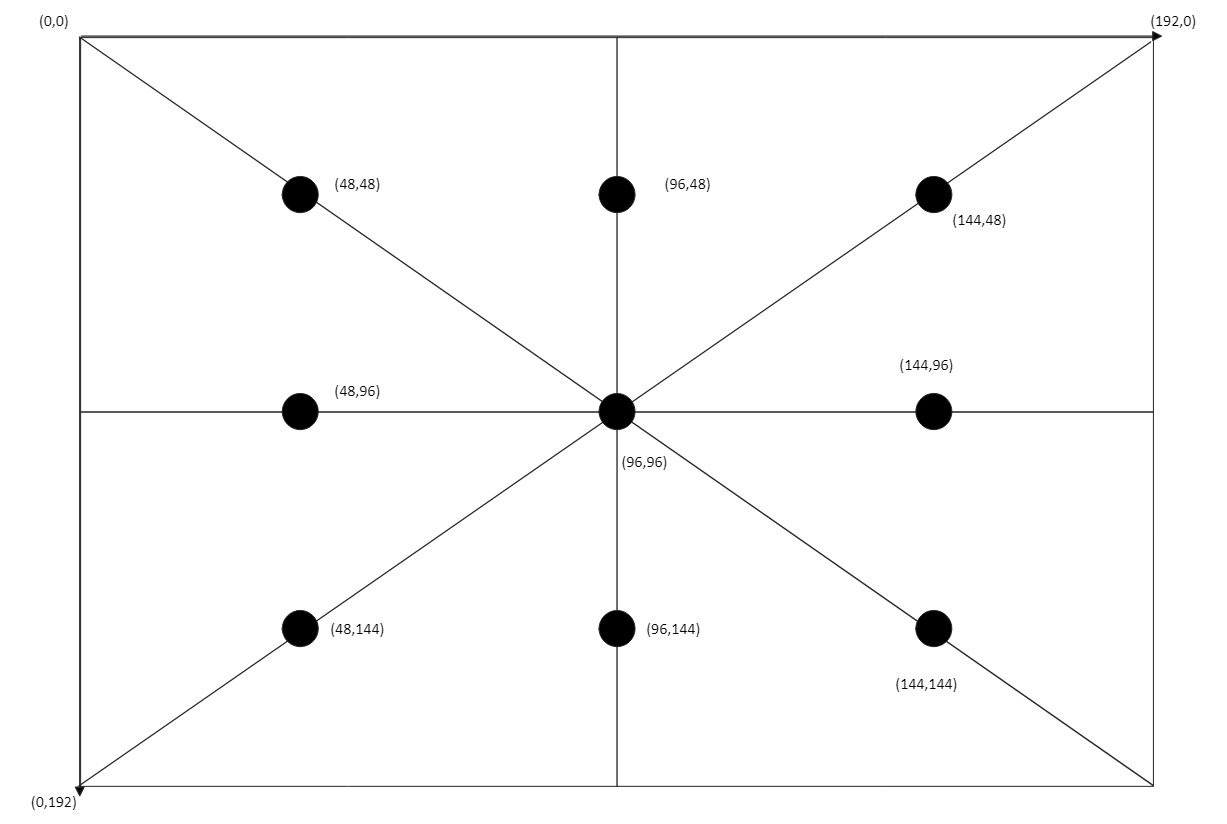


Ilustración 17 Diagrama bidimensional de coordenadas de imagen

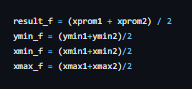
Tomando como referencia el diagrama de la Illustración 17. Se diseñó un algoritmo de decisiones en el que a través de estas coordenadas en el plano bidimensional en el que se toman rangos de pixeles para establecer posiciones y tamaños relativos para la toma de desiciones, además de descartar los objetos solamente visibles para una de las dos cámaras, ya que esto significa que el objeto está muy cargado hacia un lado específico, y soluciona el problema de la posible desincronización de las cámaras al asignar los identificadores de *USB* cuando se conecta el sistema completo.

Ilustración 18 Promedio de valores de posición relativa.

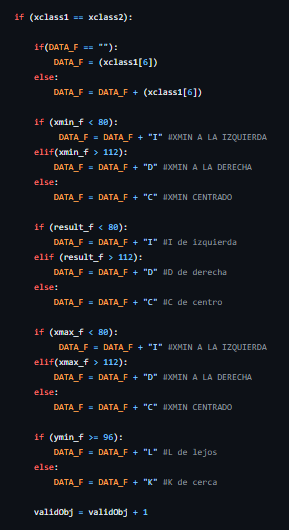


Ilustración 19 Algoritmo de toma de decisiones basada en promedios de distancia relativa.

**Comunicaciones**

Las transferencias entre tarjetas se realizaron mediante empaquetados de datos preinterpretados y convertidos a strings, que fueron pasados mediante *PIPES* a un progama en *C* dentro de la *IMX*, para realizar transferencias por el protocolo de comunicaciones *SPI* a una frecuencia de 400kHz.

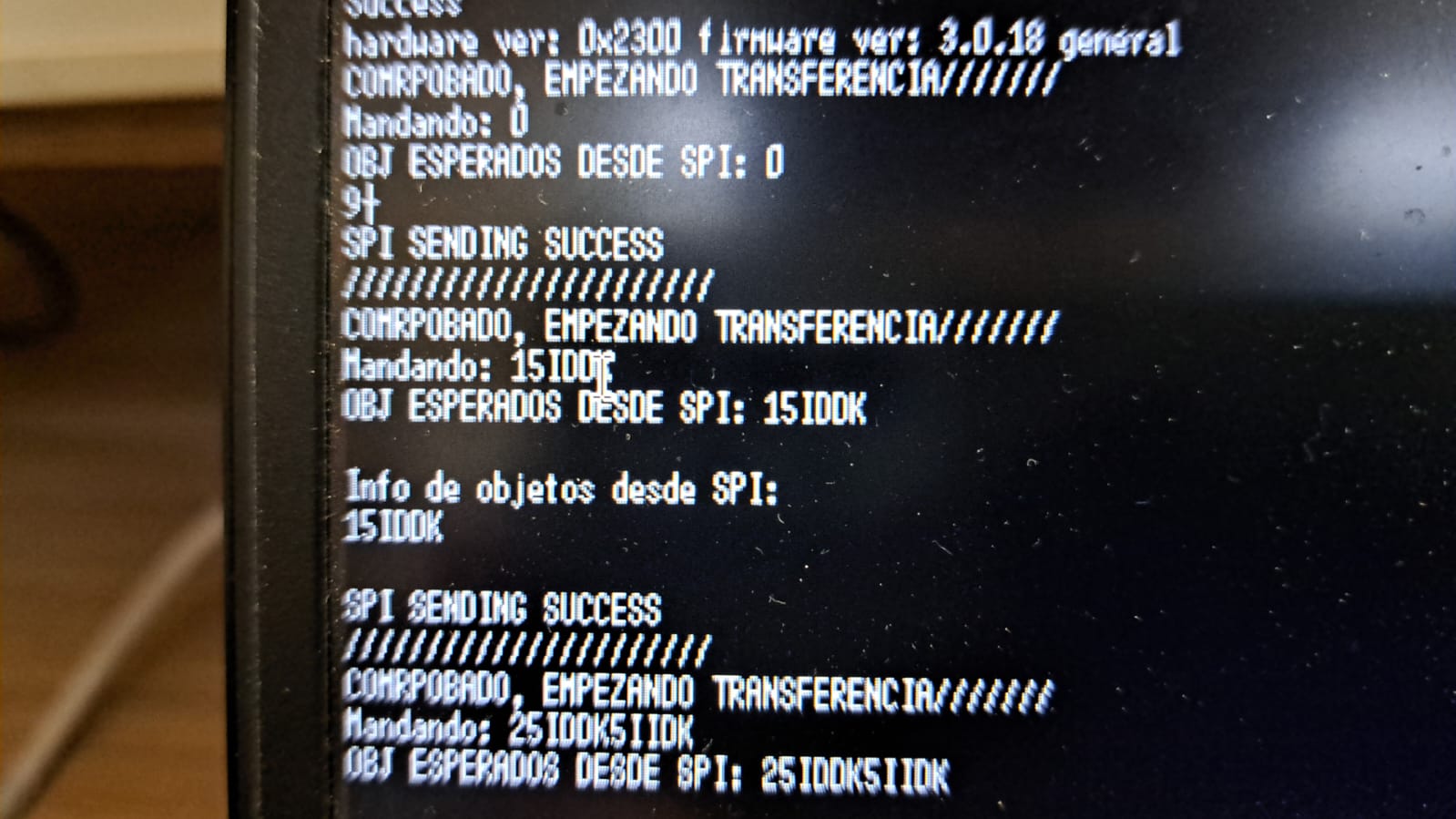


Ilustración 20 Impresión de confirmación de transferencias exitosas.

El manejo de estas transferencias fue utilizando la *IMX* como dispositivo principal y la K66 como dispositivo solamente de recepción, para mantener la sincronía entre transferencias, se limitó la salida de los tensores para evitar overruns del buffer de *SPI*, dando tiempo a que la *K66* recabe los datos y los procese antes de que nuevas transferencias

Ilustración 21 Trama de transmisión por SPI desde IMX.

Por la parte de los periféricos de la *K66*, la comunicación con los periféricos se realizó dependiente a las necesidades de cada uno de ellos:

* *GPS:* Se utilizó *UART* para obtener las tramas completas, aislar y procesar los datos de *GPRMC* y *GPGGA*, separándolos en un task asignado a la decodicifación.
* *Brújula*: Se utilizó el protocolo *I2C* para obtener el punto cardenal al que apunta la flecha del magnetómetro en el momento en el que se toma la medición.
* *Ultrasónicos*: Se utilizaron pines de *GPIO* de la *K66* para lanzar el *trigger* y recibir el *echo* de la señal de cada ultrasónico.

**Pathing**

Para el pathing nos basamos en la metodología usada en la materia de Sistemas de Control Automático impartida por el profesor Luis Enrique González Jiménez, utilizada para el control de un sistema en retroalimentación de estados. Esta metodología se centra en identificar si primeramente es necesario trabajar o hacer alguna modificación al sistema natural, después ver si es posible realizar ese trabajo o esa modificación sobre el sistema para poder alcanzar un objetivo y finalmente, empezar a realizar esa acción.

Para nuestro proyecto esto se tradujo principalmente en evaluar si es necesario mover el carrito en base al objetivo interpretado como coordenadas GPS en latitud y longitud en comparación con las obtenidas en tiempo real con el GPS, en caso de ser necesario mover el carrito, evaluaríamos si el carrito se encuentra mirando a una dirección que le pueda favorecer para poder llegar al destino esperado, para este paso también evaluamos si era posible moverse en la dirección que le conviene a partir del análisis de los objetos obtenidos por el procesamiento de la red neuronal para determinar si había un objeto cerca que le impida el paso por esa dirección, en caso de que esté obstruida esa dirección se guiará por la siguiente dirección que le conviene para nuevamente verificar si es posible ir por ese camino. Si una dirección se identificó obstruida, esta se bloqueará para futuras evaluaciones a menos que el carrito efectúe un cambio de dirección valido. En caso de estar libre comenzará la tarea del movimiento del carrito.

La tarea relacionada para el movimiento del carrito es la encargada en avanzar y poder hacer correcciones mínimas cuando este se encuentre con objetos en su camino, en esta función evaluará constantemente los objetos obtenidos por la red neuronal para entonces así como los cuadrantes en los que se ubiquen su xmin, xprom y xmax, para a partir de estos valores identificar si es posible esquivar el objeto hacia una dirección o simplemente detenerse si no hubo ningún cuadrante libre y solicitar una nueva evaluación para un cambio de ruta.

1. Función de Torriceli en Geógebra, hecha por José Manuel Arranz Recuperado de: <https://www.geogebra.org/m/MvjnbP53> [↑](#footnote-ref-2)