Conducción autónoma para tareas de logística basada en visión por computadora y Deep Learning

Resumen. En el presente trabajo se propone el desarrollo de un prototipo a escala de un vehículo autónomo para tareas de logística en un depósito industrial. Para ello, se combinan técnicas avanzadas de procesamiento de imágenes basadas en visión artificial y técnicas de Deep Learning para el reconocimiento de objetos. El vehículo tiene la capacidad de identificar paquetes mediante códigos QR y de llevarlos hasta el depósito correspondiente. Una vez concluida dicha tarea, retorna a la base de forma automática a la espera de un nuevo paquete. Para ensayar el correcto funcionamiento del vehículo, se diseñó una pista de prueba, en la cual el vehículo debe reconocer el paquete a transportar, y siguiendo una línea de guía, busca el depósito correspondiente al paquete recibido. Para realizar dicha búsqueda, identifica distintos carteles con códigos QR en su trayectoria, y carteles numéricos sobre las bocacalles principales mediante Deep Learning. Una vez que llega al depósito, se posiciona de forma alineada con la calle y se aproxima a la línea identificatoria. Cabe destacar que, además, el vehículo es capaz de reconocer un semáforo peatonal.

Palabras claves: Conducción autónoma, Deep learning, Computer Vision.

1 Introducción

Hoy en día, la industria a nivel global automatiza sus líneas de producción de forma continúa buscando un funcionamiento lo más autónomo posible utilizando técnicas de inteligencia artificial. Este concepto difundido actualmente se relaciona con la Industria 4.0, siendo un pilar de la misma en conjunto con la conectividad en la red y el internet de las cosas (IoT) [1]. Actualmente en las grandes industrias, el transporte de paquetes se realiza totalmente de forma automatizada [2]. Existe una amplia variedad de vehículos de guiado automático (AGV), pero a costos muy elevados para pequeñas empresas. Hay diferentes empresas que los producen actualmente, algunas de ellas son por ejemplo CEIT, Jungheinrich AG, Toyota, Egemin, entre otras. Algunos de los modelos disponibles actualmente en el mercado se muestran en la Figura 1.

A comienzos del año 2020 se logró desarrollar la conducción sin supervisión e interacción humana [3], sobre ciertas condiciones de circulación estrictamente definidas. De esta forma, se alcanzó el nivel 4 de automatización (alta automatización) según la

escala de automatización [4]. Hasta el momento, no hay tecnología capaz de lograr un nivel 5 de automatización (automatización completa), y algunos expertos afirman que este nivel de automatización nunca se podrá alcanzar. Uno de los desafíos más importantes para lograr una autonomía completa de nivel 5, es el ambiente en el cual se desenvuelve el vehículo. El mismo es de gran influencia en la toma de decisiones ya sea por las reglas de circulación en cada país o la cultura, clima, y estado de las calles.

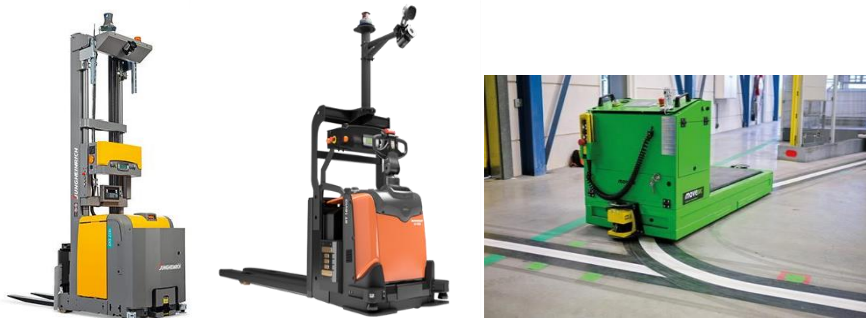


Fig. 1. Ejemplos de AGV utilizados actualmente en la industria.

Los vehículos autónomos personales más avanzados hasta el momento se ubican en el nivel 2 de automatización, donde se requiere que el conductor supervise la conducción autónoma y corrija en caso de ser necesario. Un ejemplo de lo antes mencionado, es el conocido “autopilot” desarrollado por la compañía Tesla.

Diferentes herramientas y técnicas se combinan para lograr una conducción autónoma completa: mapeo del entorno mientras se mantiene registro de la ubicación (SLAM), combinación de información de los diferentes sensores presentes en el vehículo, planeamiento de la ruta, control de movimiento del vehículo, etc. Las técnicas actualmente utilizadas incluyen: GPS diferencial (DGPS), comunicación entre vehículo y su entorno V2X (Vehicle-to-Everything, se distinguen comunicaciones de corto rango, DSRC, basadas en WiFi y de celular, C-V2X), algoritmos de Machine Learning tales como Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) y Deep Reinforcement Learning (DRL), etc.

En este trabajo se propone el diseño de un prototipo a escala de un vehículo autónomo, no tripulado y eléctrico, para el transporte de paquetes en un ambiente industrial, el cual se desplazará en un depósito predeterminado según el paquete que se le asigne, sin necesidad de control humano. Una vez en el depósito, se deberá descargar el paquete y luego automáticamente volverá a su posición inicial. El propósito principal es conseguir que en la industria se automatice el proceso de transporte de paquetes, ya que este suele ser inseguro y poco eficiente, permitiendo que el personal solo se encargue de descargar los paquetes del vehículo cuando finalice su recorrido. En la sección 2 se presenta una descripción general del trabajo. Luego, en la sección 3 se presenta el hardware utilizado en el diseño del prototipo y su interconexión. A continuación, en la sección 4 se presenta el software desarrollado para llevar a cabo las diferentes tareas. Por último, en la sección 5 se detallan los diferentes experimentos realizados y en la sección 6 se presentan algunas conclusiones.

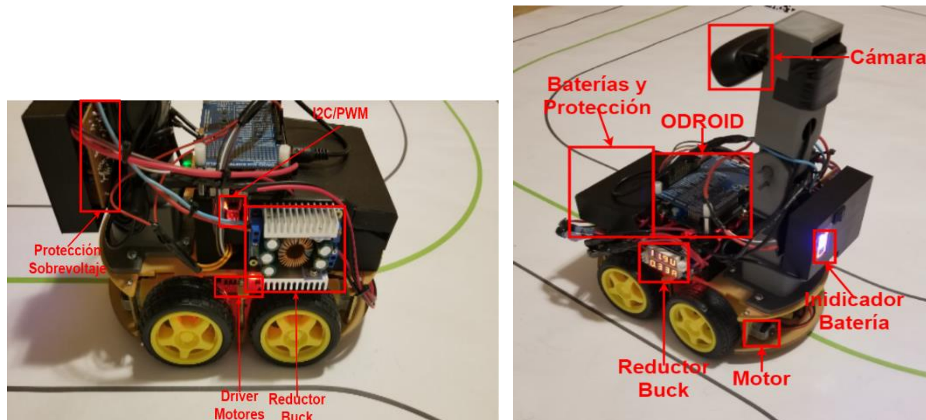


Fig. 2. Disposición de los componentes del vehículo desarrollado.

2 Descripción del sistema

El prototipo propuesto está compuesto por un auto de radio control, el cual originalmente poseía solamente las ruedas, motores y sus dirvers. Al mismo se le agregó una placa encargada del procesamiento y control "ODROID XU4", y una placa "I2C to PWM" para comandar el driver de los motores. Por otro lado, a la placa de control se le conectan dos dispositivos, un driver USB-wifi para conectarla a la red y una cámara web para la captura de imágenes. Además, posee una protección de sobrevoltaje para la placa ODROID XU4 y otra protección de voltaje para las baterías. También se diseñaron piezas impresas en 3D a medida para el debido posicionamiento óptimo de dichas placas y drivers, lo cual se encuentra detallado en el Anexo III. Esto a su vez nos permitió interconectar todos los dispositivos de una forma ordenada. Con respecto a la alimentación del vehículo, se optó por colocar 4 baterías de litio modelo 18650, en el frente del vehículo, debajo de la tapa cobertora, junto a su protección. Para regular la tensión de alimentación tanto del driver de los motores como de la placa principal, instalamos 2 reductores tipo Buck. En el anexo IV se encuentran más detalles sobre este tema. En la Figura 2 se puede ver la disposición de todos los componentes del vehículo.

La pista que se muestra en la Figura 3 es el "laboratorio de ensayo" más adecuado para demostrar todas las funciones del vehículo desarrollado, en un circuito representativo y a escala de lo que podría llegar a ser una situación real. Con respecto a las dimensiones de la pista, la calle tiene un ancho de 20 cm, la longitud de su tramo recto 40 cm y el radio interior de curvatura de giro es de 20 cm. El tamaño total de la pista resulta ser entonces de 220 cm x 320 cm. Como se detalla en la Figura 3, se identifica el área de inicio donde los paquetes serán cargados al vehículo. La misma está demarcada por su correspondiente senda peatonal de color rojo, ante la cual el vehículo detendrá su movimiento para cargar un nuevo paquete. Además, se identifican tres áreas correspondientes a los depósitos (D0, D1 y D2), también demarcadas por una senda peatonal de color rojo donde el vehículo se detendrá para descargar los paquetes. Previo a dichas áreas se encuentran carteles con un código QR identificatorio, que indicarán la

proximidad a un depósito determinado. La línea verde corresponde a línea central de la calle, con la cual el vehículo se guiará. Por otro lado, en las áreas identificadas como “Área 1” y “Área 2” se encuentran los carteles numéricos (en color celeste) y el semáforo (en color gris), donde los carteles numéricos cumplen la función de indicarle al vehículo qué depósito se encuentra al doblar en la bocacalle donde se encuentran.

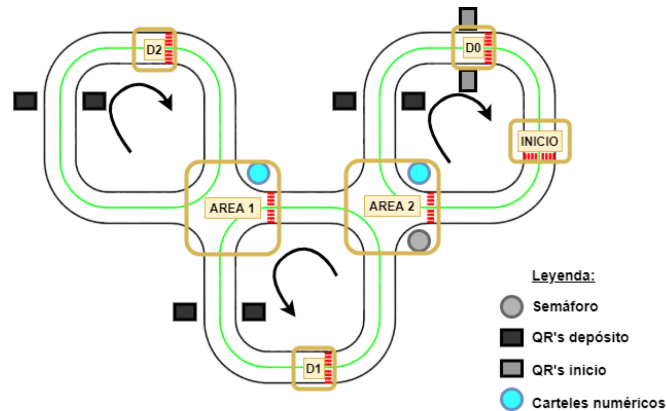


Fig. 3. Diseño de la pista utilizada para realizar los experimentos con el vehículo desarrollado.

El funcionamiento del sistema se resume de la siguiente forma. En un principio, el vehículo situado en el inicio del trayecto (frente a la senda peatonal roja) espera a detectar el código QR de un paquete. Una vez que detecte el código del paquete, el cual contiene el depósito al cual pertenece, este empezará a seguir la línea principal verde, en busca de la primera senda roja. Una vez encuentre la primera senda roja, el vehículo se detendrá y se activará la detección de objetos. El vehículo entonces, se encontrará en la primera bocacalle buscando el cartel y el semáforo. Si detecta solo el semáforo, el mismo girará en su lugar en dirección hacia el cartel hasta encontrarlo. Si detecta solo el cartel, repetirá este proceso, pero girando en dirección al semáforo. Una vez detectado el cartel y el semáforo se ponga en verde, el mismo seguirá otra vez la línea verde hasta detectar la primera bocacalle. Una vez aquí, el vehículo decidirá qué hacer, si el número del cartel representa el depósito deseado, entonces seguirá la línea verde. Por otro lado, si el cartel no representa el depósito deseado, éste cruzará la bocacalle. Entonces, se nos presentan dos posibles caminos.

En el primero de ellos, si el vehículo decidió seguir por la línea verde, este continuará así hasta detectar el código QR del depósito deseado, lo cual le da pie para continuar siguiendo la línea hasta encontrar una senda peatonal roja. Cuando detecte la senda, procederá a alinearse contra la misma para luego detenerse frente a ella. Esto dicta el fin del recorrido del paquete, con lo cual espera un cierto tiempo a que se lo descarguen para luego retornar al inicio. Para esto, el mismo sigue la línea verde hasta detectar el QR de inicio, para luego buscar una última senda roja. Una vez que la detecte, se alineará con la misma y se detendrá frente a esta, dando final al trayecto y poniéndose listo para recibir paquetes.

Mientras que, para el segundo camino, donde el vehículo decidió cruzar la bocacalle, se lanza un hilo en el programa que durante 2 segundos le permitirá al vehículo cruzar la bocacalle de forma orientada, mediante la detección de la línea verde en la calle a tomar. Es decir que, durante estos dos segundos, el vehículo seguirá la línea verde que vea en la parte superior de la imagen (la calle a tomar). Luego de haber transcurrido 2 segundos, si se sigue detectando que estamos en condición de bocacalle, se vuelve a la lanzar dicho hilo, y así sucesivamente hasta dejar de detectar la bocacalle. Una vez que el vehículo deje de detectar la bocacalle, continuará guiándose por la línea verde en la parte superior de la imagen y una vez transcurrida una cierta cantidad de frames, en los cuales se detecte que efectivamente el vehículo detecta la línea verde en la parte inferior de la imagen, se procederá a seguir dicha línea verde en la sección inferior de la imagen. Cabe destacar que se optó por usar un hilo y no un método interno dentro de la clase utilizada en el programa ya que de esta forma logramos asegurar que dicha función para cruzar de forma orientada dure exactamente dos segundos y que no sea interrumpida por alguna otra función de la clase. Luego, seguirá por la misma hasta, una vez más, encontrar una senda roja, en donde activará nuevamente la detección de objetos y buscará el cartel con información pertinente al depósito deseado (girando en el lugar si no lo encuentra). Una vez que lo encuentre, seguirá avanzando por la línea verde y se encontrará con una segunda bocacalle. Luego, comparará el valor del cartel detectado con el valor del depósito al cual el paquete debe dirigirse. Si estos coinciden, entonces el vehículo continuará siguiendo la línea verde. En caso contrario, cruzará nuevamente la bocacalle. En ambos casos el vehículo se topará con los QRs de depósito y sus sendas rojas, alineándose con estas y deteniéndose, finalizando así el recorrido del paquete. Además, para volver nuevamente al inicio el vehículo deberá cruzar cualquier bocacalle que se le presente (sin leer ningún código o detenerse) hasta encontrar el QR de inicio, para luego buscar una senda roja y alinearse con la misma, dándose así el final del trayecto del vehículo y volviendo al estado inicial de esperando paquetes.

Cabe aclarar que el vehículo no sabe dónde está cada depósito por el solo hecho de estar en orden consecutivo. Se podría alterar el orden de los depósitos y va a seguir funcionando correctamente. Por ello se agregó la detección de objetos mediante el uso de inteligencia artificial. Si la búsqueda de depósitos fuera de forma secuencial, no necesitaríamos los carteles porque sabríamos que al doblar en la primera bocacalle iríamos siempre al depósito 1, al doblar en la segunda al depósito 2, y así sucesivamente. Para el desarrollo del programa implementamos diversas técnicas de procesamiento de imágenes basadas en visión artificial, tales como: filtros autoajustables de color HSV (Hue Saturation Value) y HSL (Hue Saturation Luminescence) según luz ambiente, análisis estadístico de los puntos detectados mediante técnicas de binarización para la estimación de diferentes parámetros (distancia a la senda peatonal, alineamiento respecto a la pista, etc.), operaciones morfológicas en imagen (dilatación, erosión, etc.), diversos filtros y técnicas para disminución de ruido presente en imágenes y corrección del efecto de perspectiva en la imagen. Sumado a ello, utilizamos una red convolucional (CNN), basada en Deep Learning para realizar la detección de 4 objetos, en particular: semáforo en verde, semáforo en rojo, cartel “1” y cartel “0”. En el anexo I puede encontrarse un GRAFCET (Graphe Fonctionnel de Commande Etape Transition) del sistema desarrollado.

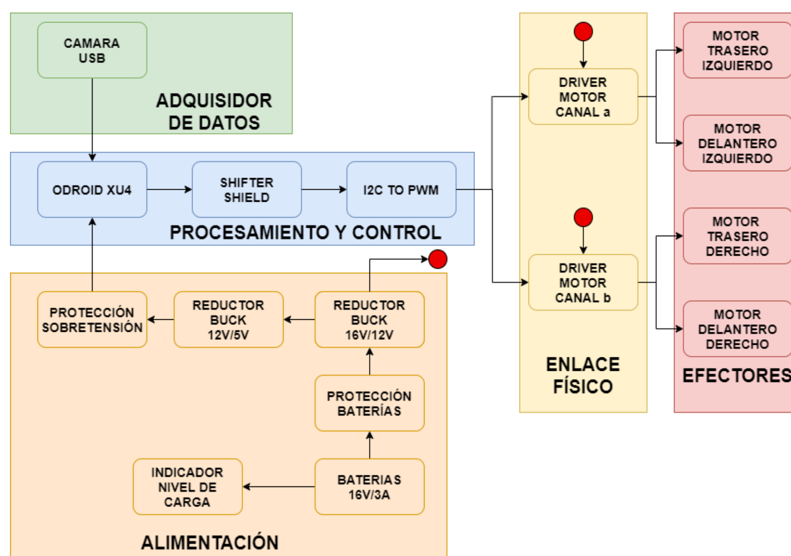


Fig. 4. Diagrama en bloques de los componentes del vehículo.

3 Hardware utilizado

Como se observa en la Figura 4, los componentes del vehículo pueden dividirse en cinco bloques principales. En el bloque de Alimentación se agrupan las baterías, protecciones y circuitos de regulación de tensión. En el bloque de procesamiento y control, se encuentran las placas principales encargadas del procesamiento. Ligado a este último bloque vemos el de adquisición de datos, el cual se encarga de capturar imágenes para luego ser procesadas por la unidad de procesamiento y control. El mismo está compuesto por una cámara web con una resolución máxima de 720p, tipo de enfoque de foco fijo, un campo visual de 60° y definición de 3 megapíxeles. Luego, en el enlace físico se encuentran los drivers de los motores, los cuales convierten la señal PWM en una señal eléctrica capaz de alimentar los motores. Por último, se encuentra el bloque de efectores, donde se encuentran los motores del vehículo.

Con respecto al interconexionado de los bloques, el bloque de alimentación alimenta únicamente a los drivers de los motores en el enlace físico y a la placa principal en el bloque de procesamiento y control. A su vez, la placa principal ODROID XU4, proporciona tanto la alimentación como el enlace de transferencia de datos a la cámara y el Shifter Shield. Y finalmente, la capa de enlace físico es la encargada de energizar los motores. Dentro del bloque de procesamiento y control se distingue la placa principal de procesamiento ODROID XU4, la cual es una PC de placa única o “Single Board Computer” (SBC) y sus características principales son: CPUs ARM A15 Quad Core @2.0GHz y ARM A7 Quad Core @1.4GHz, RAM 2GB DDR3, permite el uso de memorias tanto micro SD como eMMC y alimentación de 5V 4A máx.

Además, para establecer la comunicación I2C mediante la placa ODROID XU4 y la placa I2C a PWM, es necesario que la tensión de los pines TX y RX del protocolo I2C

de ambas placas se encuentren al mismo nivel de tensión (5V), para lo cual se utiliza el Shifter Shield. Finalmente, mediante la placa “Adafruit PCA9685”, se generan señales de tipo PWM ante una señal de control en sus pines SCL y SDA, mediante el protocolo de comunicación serie I2C. Su función en el presente trabajo es generar cuatro señales de tipo PWM para controlar la excitación al driver de los motores, y de esta forma controlar su velocidad y tiempo de movimiento de forma precisa.

4 Software

El sistema operativo utilizado en la placa de procesamiento es Ubuntu 18.04 LTS Bionic Beaver de 32bits [5]. El primer paso luego de instalar el sistema operativo, fue establecer un medio de comunicación con la misma mediante WiFi. Por ello, se instaló el software “VNC Remote Desktop” en la misma [6]. Para ello, nos conectamos mediante ventana de comandos SSH a la ODROID XU4, y desde allí instalamos “Tiger vnc-viewer”, junto con los paquetes requeridos. Finalmente se hicieron las configuraciones necesarias para que VNC viewer inicie con Ubuntu, y de esta forma permitirnos conectarnos sin ingresar antes mediante SSH para habilitar dicho servicio.

El lenguaje de programación utilizado en el presente trabajo fue Python, el cual es un lenguaje de alto nivel, interpretado y dinámico de propósito general que se focaliza en la legibilidad del código. El mismo ayuda a los programadores a programar en pocos pasos en comparación con otros lenguajes de programación orientados a objetos como Java y C++. Por lo tanto, se consideró el indicado para realizar el presente trabajo, al disponer, además, de una gran cantidad de librerías disponibles y suficiente documentación respecto a técnicas de procesamiento digital de imágenes y deep learning.

Para realizar el procesamiento digital de imágenes y reconocimiento de objetos mediante un modelo pre-entrenado de deep learning se utilizó la librería OpenCV [7]. OpenCV se encuentra disponible en PIP, sin embargo, para versiones de Ubuntu de 32 bits, la versión disponible de OpenCV es antigua. En el presente proyecto fue necesaria una versión más actualizada de OpenCV, donde se encuentre incluido el módulo DNN (Deep Neural Network) [8]. Por ello, fue necesario compilar e instalar OpenCV desde el código fuente. De esta forma, mediante el módulo DNN de OpenCV, es posible correr modelos diseñados y entrenados en distintos frameworks, generalmente utilizando una GPU de alto rendimiento, con simplemente el uso de una CPU. Esto es de mucha utilidad en el presente trabajo, ya que si bien la placa de procesamiento tiene una GPU ARM Mali-T628, la misma posee únicamente 6 núcleos de 600MHz, mientras que la ODROID XU4 presenta dos CPU de 4 núcleos cada una, en la primera de ellas a 1.4GHz y en la segunda 2.0GHz.

Para mejorar significativamente la velocidad de ejecución del programa, se utilizó las librerías de Python: multithreading (multi hilos) y multiprocessing (multi procesos). Por un lado, utilizamos multiprocessing para independizar procesos que puedan correr simultáneamente. En particular, en el presente trabajo se utilizaron dos procesos. En el primero de ellos se encuentra el código “principal”, en donde se realiza el procesamiento de las imágenes y la toma de decisiones. Y el segundo de ellos incluye la función encargada de, mediante el protocolo I2C, realizar la comunicación con el driver de los

motores. De esta forma, se logra independizar el control de movimiento del vehículo, para que éste sea más preciso. Al ser dos procesos diferentes los antes mencionados, no comparten el uso de la memoria, sino que cada uno tiene un espacio de memoria independiente al otro. Por ello, junto con la librería multiprocessing, se utilizan los llamados "Pipe" (tuberías) que permiten la transferencia de datos entre dos procesos distintos. Los "Pipe", se interpretan como una lista de Python de tipo FIFO (First In - First Out). Por otro lado, se implementó multi hilos dentro del proceso principal mediante la librería multithreading. La utilidad de éstos es paralelizar, mediante el uso de los distintos núcleos que presenta la placa ODROID XU4, tareas no correlativas (es decir, que la siguiente tarea no depende de la tarea anterior) y finitas. El resultado de todo esto fue finalmente, aprovechar al máximo los ocho núcleos de los que dispone la ODROID XU4 y aumentar considerablemente la cantidad de frames procesados por segundo. En nuestro caso, se logró aumentar los FPS de 7 a 50 aproximadamente.

Otras librerías de utilidad fueron Pyzbar (para decodificar códigos QR presentes en una imagen), WiringPi (librería de Python de alto nivel [9], utilizada para acceder a los pines GPIO de la ODROID XU4) y Statistics (brinda funciones básicas de probabilidad y estadística).

4.1 Detección de objetos mediante deep learning

Para la detección de las señales y los semáforos, se re-entrenó un conocido modelo para detección de objetos, optimizado para funcionar con bajos recursos. Dicho modelo, Tiny YOLOv3 [10], es una versión compacta del modelo YOLOv3. YOLO (You Only Look Once) [11] es un sistema de código abierto para detección de objetos en tiempo real. Este mismo consiste en una CNN (Convolutional Neural Network) para detectar objetos. Como se mencionó, para correr el modelo ya entrenado de YOLOv3 Tiny únicamente mediante CPU, utilizamos el módulo DNN de OpenCV [12]. Los resultados obtenidos se observan en la Figura 5, donde en la imagen de la izquierda, se detecta el semáforo verde + cartel cero, y en la imagen de la derecha semáforo rojo + cartel uno. Se observa además que la precisión es mayor al 90% en el caso de los carteles y mayor a 98% en el caso de los semáforos, por lo cual podemos afirmar que el modelo de detección de objetos funciona correctamente. En el anexo II se encuentra un análisis más extenso sobre este tema, junto a detalles de implementación.

Sin embargo, y como era de esperar al correr un modelo de Deep Learning en una placa de estas características sin GPU, el tiempo que demora para procesar cada imagen es alto. En el presente trabajo, la placa ODROID XU4 es capaz de procesar una imagen por segundo (1 FPS) cuando se activa la detección de objetos. Debido a ello, implementamos algunas técnicas para mejorar los tiempos de procesamiento en general, las cuales son: activar la detección de objetos únicamente cuando se detecte la senda peatonal de color rojo y no se haya identificado aún un cartel con código QR. Además, una vez activada la detección de objetos, se evalúan únicamente una de cada 5 imágenes capturadas por la cámara. De esta forma, si bien perdemos información al no buscar objetos en 4 de las 5 imágenes adquiridas, logramos como resultado incrementar la capacidad de procesamiento de 1 FPS A 5 FPS aproximadamente.

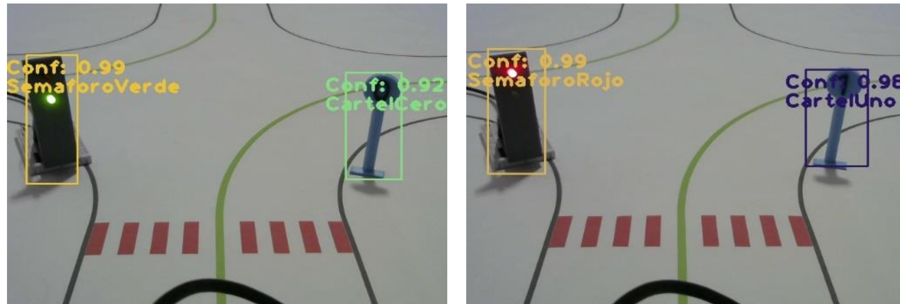


Fig. 5. Detección de objetos utilizando el modelo YOLOv3 Tiny re-entrenado.

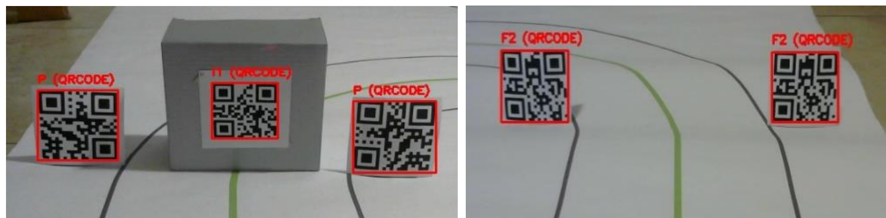


Fig. 6. Detección de códigos QR.

4.2 Detección de códigos QR

Como se mencionó anteriormente, utilizamos la librería Pyzbar para la identificación de códigos QR en la imagen. En la Figura 6 se observan diferentes capturas realizadas desde el vehículo. En el presente proyecto se utilizan códigos QR para 3 funciones diferentes según su valor:

- Si la lectura es “P”: una vez que el vehículo identifique dicho código, comenzará a buscar la próxima senda peatonal de color rojo que indicará el inicio, donde debe detenerse a la espera de un nuevo paquete.
- Si la lectura comienza con “I”: una vez que el vehículo identifique dicho código, almacenará el número seguido del carácter “I”, el cual será el depósito a buscar para dejar el paquete. Luego de procesar y almacenar dicho valor, comienza su movimiento habiendo estado parado en el inicio.
- Si la lectura comienza con “F”: una vez que el vehículo identifique dicho código, comenzará a buscar la próxima senda peatonal de color rojo que indicará el depósito buscado, donde debe detenerse durante un tiempo determinado para permitir la descarga del paquete y luego, continuar su movimiento.

4.3 Detección de línea central

Para la detección de línea verde, se procedió a realizar distintas operaciones de procesamiento de imágenes. Primero, se aplica un filtro pasa bajos tipo Gaussiano a cada plano RGB (Red, Green y Blue) de la imagen original obtenida con la cámara, para

reducir el ruido presente en esta. Luego se recorta la parte inferior, en particular los últimos 160 píxeles, de manera tal que el seguimiento de la línea solo se aplique al tramo más cercano de la misma (respecto al vehículo). De esta manera se asegura que el movimiento del vehículo siga al tramo de la línea verde más próxima al vehículo y se evita seguir un tramo más alejado que podría ser por ejemplo una curva. Por último, se aplica un filtro de color verde, definiendo un umbral máximo y un umbral mínimo en el plano HSV, para dejar únicamente este color en la imagen y obtener así una máscara binaria donde solamente estaría representada la línea central verde.

Luego, para guiar al vehículo en base a la línea detectada, en primer lugar, se cuenta la cantidad de puntos detectados en la máscara. Si dicha cantidad es menor a cierto umbral, se descarta la imagen (se asume que hay ruido presente). En caso contrario, se obtienen las coordenadas X e Y (ejes presentes en la Figura 7 (a)) de los puntos detectados y se calcula la mediana de la coordenada X de todos los puntos. En la Figura 7 (b) pueden observarse en color blanco los puntos que representan a la línea central, en verde el valor de la mediana obtenida y en rojo el centro de la imagen. La línea verde es una línea vertical y su coordenada X es la mediana previamente calculada. Por último, se obtiene la distancia d en el eje X entre el centro de la cámara y la mediana previamente calculada. El valor de d es esencial para el seguimiento de la línea verde. Primero, supongamos un tramo recto de la línea verde, como se ve en la Figura 7 (a). Si d es positiva, esto quiere decir que la línea se encuentra a la izquierda (respecto al centro de la cámara, y por lo tanto del vehículo). Por lo tanto, el vehículo debe doblar hacia la izquierda. Del mismo modo, si d es negativa esto quiere decir que la línea se encuentra a la derecha (respecto al centro de la cámara, y por lo tanto del vehículo). Por lo tanto, el vehículo debe doblar hacia la derecha. En el caso de la Figura 7 (a), la distancia d es levemente negativa, por lo tanto, debe doblar hacia la derecha.

Como se puede ver, el objetivo es alinear el centro de la cámara con el valor de la mediana, para lograr así que el vehículo quede alineado con la línea verde. Observando la definición de d, se puede discernir que esto se logra cuando la distancia tiende a 0. Por lo tanto, es deseable que esta distancia sea cercana a cero. Esto en principio es sólo válido para tramos rectos de línea verde, pero a su vez, tomando como ejemplo un tramo curvo cuya máscara obtenida se muestra en la Figura 7 (d), se observa que la línea que representa a la mediana se encuentra a la derecha respecto del centro de la cámara, y por ende, si utilizamos el mismo procedimiento que para las líneas rectas, obtendremos que el vehículo debe doblar hacia la derecha, el cual es el comportamiento que queremos lograr. Debido a esto, se puede considerar que esta técnica es un buen indicativo de hacia dónde debe girar el vehículo en caso de toparse con un tramo curvo de línea de referencia del vehículo (línea verde mostrada en Figura 7 (c)). Por lo tanto, vale también el mismo razonamiento con líneas curvas.

Cabe destacar que los valores de umbral del filtro de color verde, varían según la luminosidad presente en la imagen mediante una función que realiza los siguientes pasos. Primero, se define un umbral mínimo en el plano RGB con valores característicos del frente de la imagen original. Luego se le aplica a la misma un primer filtro de color, a cada uno de sus planos RGB, utilizando dicho umbral. Como resultado se obtiene una imagen binaria que sólo contiene los elementos más visibles en la imagen original. Luego, se multiplica esta imagen binaria con la imagen original para obtener una nueva

imagen en la cual solo se verán los objetos más visibles de la imagen original, y de esta manera se elimina el fondo de la misma. A esta nueva imagen sin fondo se la transforma al espacio HLS de manera de obtener los valores de luminosidad de cada pixel de la imagen. Con estos valores de luminosidad, calculamos el promedio de los mismos para obtener el valor de luminosidad ambiente. Utilizando dicho valor como entrada de una función lineal, obtenemos como resultado un valor de saturación. Dicho valor de saturación obtenido se utiliza en la máscara HSV de la detección de línea central verde para detectar los puntos verdes en la imagen. De esta manera el filtro de línea verde puede reaccionar a cambios (ligeros) en la luminosidad ambiente. Esto es necesario ya que, aunque la luminosidad ambiente es controlada, en algunos lugares de la pista la luz afecta de manera distinta que en otras (ángulo de la luz, sombras, etc.).

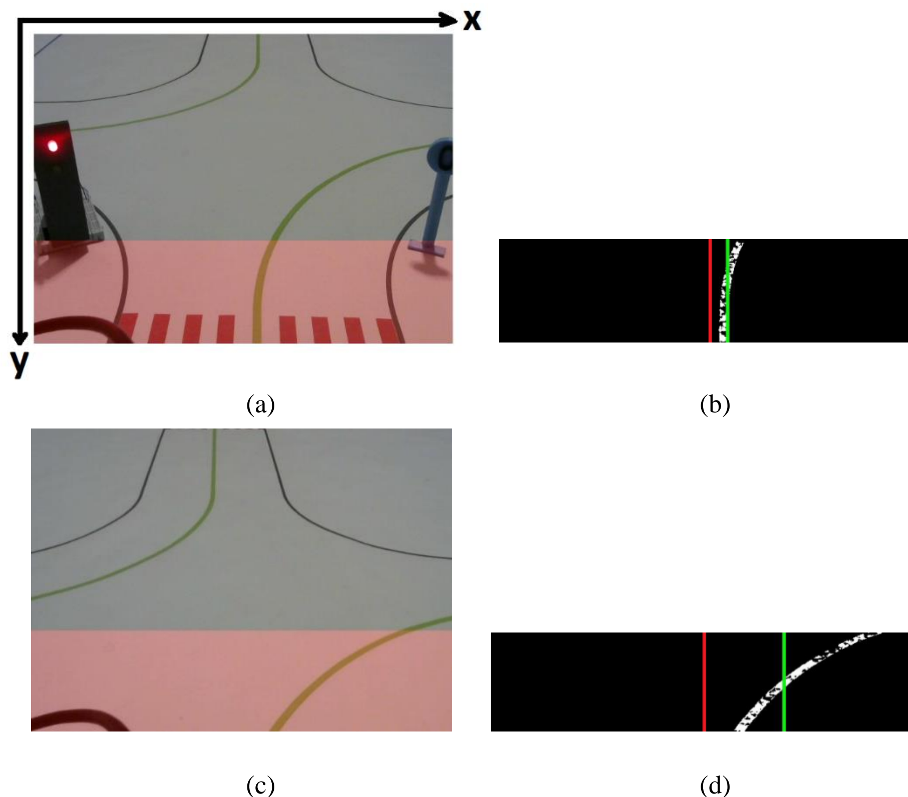
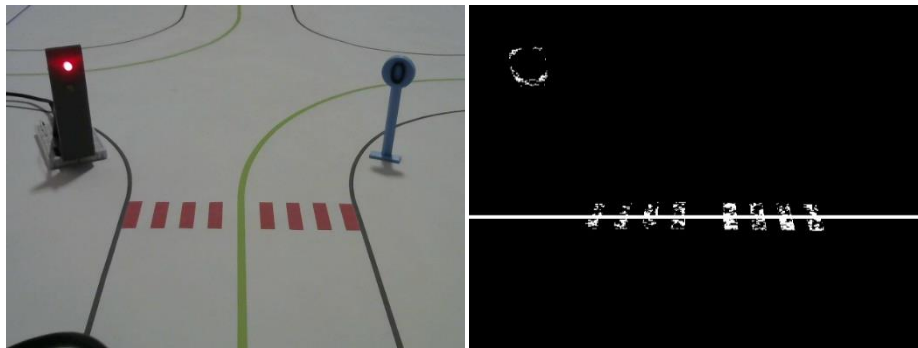


Fig. 7. Detección y alineamiento de línea verde central para los casos de tramo recto ((a) y (b)) y, casos tramo curvo ((c) y (d)). En (a) y (c) se muestra la imagen original con la zona de recorte indicada. En (b) y (c) se muestra la máscara binaria de la línea central y sobre ella se dibujan la línea verde correspondiente a la mediana y la línea roja correspondiente al centro de la cámara.

4.4 Detección de senda peatonal

Para la detección de senda peatonal, se procedió a realizar distintas operaciones de procesamiento de imágenes y matemáticas. En primer lugar, se aplica un filtro de color rojo, definiendo un umbral máximo y un umbral mínimo en el plano HSV, para dejar únicamente este color en la imagen y obtener una máscara binaria. Luego, obtenemos de la máscara todos los puntos de valor 1, con sus coordenadas X e Y (ejes presentes en la Figura 7 (a)). Por último, se calcula el valor de la mediana de la coordenada Y de todos los puntos rojos presentes en la imagen. Sin embargo, debido a que como se observa en la Figura 8 (b), el LED rojo del semáforo representa lógicamente un valor verdadero en la máscara obtenida en el paso anterior, y por lo tanto se hace necesario utilizar alguna función que ignore dichos puntos. Para ello, se utilizó la función "median_low" de la librería Statistics, la cual calcula en principio el valor medio de la entrada (el promedio) y para el cálculo de la mediana sólo tiene en cuenta los valores que estén por debajo de dicho promedio. El resultado de este proceso se puede ver en la Figura 8 (b) (línea blanca cortando la senda roja).

Usando el valor de la mediana baja podemos fijar un valor de distancia (en píxeles) en el cual el vehículo se pueda detener. Además, se puede usar la cantidad de puntos rojos presentes en la máscara para asegurar que lo que aparece en la misma sea la senda peatonal roja y no un ruido u objeto indeseado. Por lo tanto, se puede definir dos umbrales. El primero se cumple si la el valor de la mediana baja supera una distancia en píxeles determinada. Mientras más grande sea esta distancia en píxeles, más cercano se detendrá el vehículo de la senda peatonal roja. El segundo umbral consiste en que la cantidad de puntos vistos por la máscara binaria de color sea mayor a cierto valor, para el cual se puede asegurar que el objeto de la máscara es la senda roja y no otra cosa.



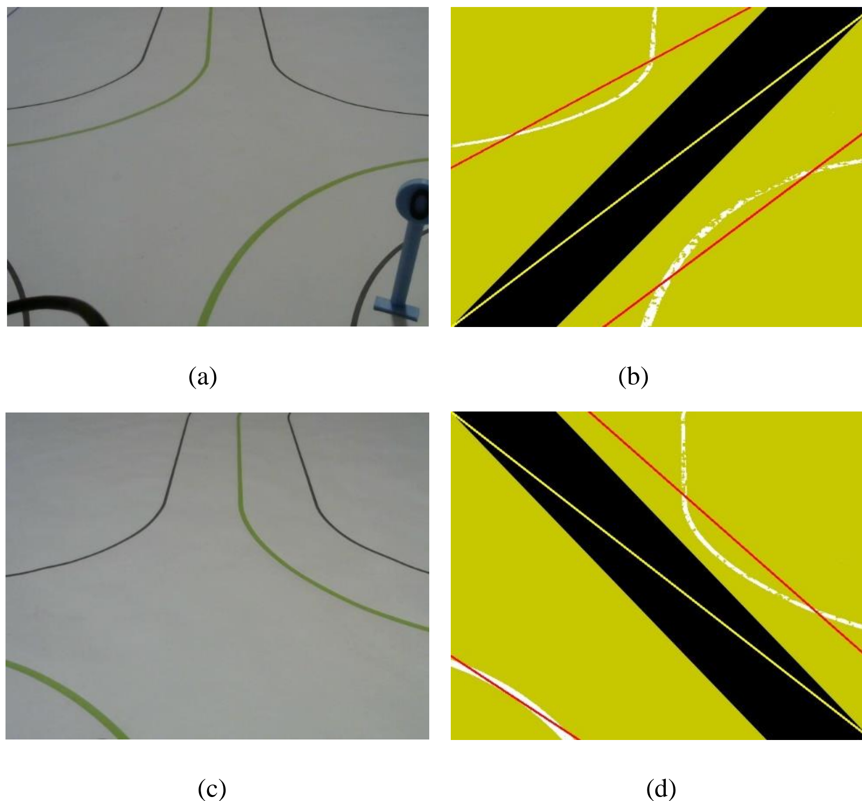
(a) (b)

Fig. 8. Detección de senda peatonal de color rojo. (a) Imagen original. (b) Máscara binaria de color rojo, donde se observa la línea blanca horizontal que representa la mediana de la coordenada Y de los puntos de valor 1.

4.5 Detección de bocacalles

Se diseñó una función capaz de detectar el instante en el cual el vehículo se encuentra frente a una bocacalle. Esto le permite al vehículo decidir si debe cruzar la bocacalle o debe seguir circulando por la línea verde y consecuentemente doblar. Además, permite que al estar el vehículo frente a una bocacalle, no confunda la línea de guía verde inferior con la superior. El método propuesto se puede subdividir en dos condiciones que deben cumplirse. En primer lugar, se evalúa la situación de bocacalle para la cual la curva inferior es hacia la derecha. Si alguna de las dos condiciones no se cumple, se procede a realizar el mismo proceso considerando que estamos en la segunda situación de bocacalle, es decir, cuando la curva inferior es hacia la izquierda. Esta segunda situación corresponde a la versión espejada respecto del eje vertical de la primera.

Las dos condiciones a cumplir son: la suma de la cantidad de puntos que representan las líneas verdes en cada área resaltada en color amarillo mostrada en las Figuras 9 (b) y (c) debe superar cierto umbral, y las líneas rojas, que representan las rectas de interpolación lineal de los puntos de color verde en cada área resaltada en color amarillo, no intersecan a la diagonal central representada en color amarillo en algún punto dentro de la imagen.



(a) (b) (c) (d)

Fig. 9. Identificación de bocacalle. (a) y (b) Curva hacia la derecha. (c) y (d) Curva hacia la izquierda. En (a) y (c) se muestra la imagen original. En (b) y (d) las máscaras binarias utilizadas.

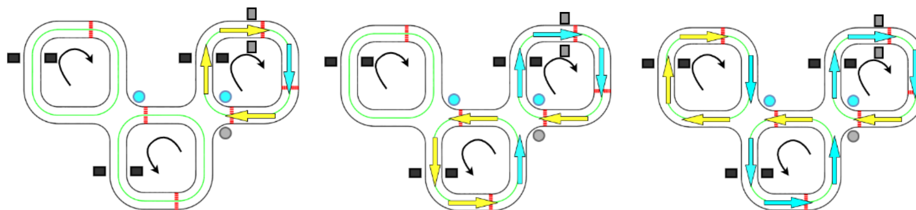


Fig. 10. Esquemas de las pruebas de funcionamiento completo realizadas.

5 Experimentos

Para verificar el correcto funcionamiento del vehículo, se realizaron pruebas de funcionamiento completo mostradas en la Figura 10[1]. En dicha Figura, se han agregado flechas de color amarillo en la pista que indican el trayecto que debe realizar el vehículo hasta el depósito a buscar. Luego, las flechas de color celeste, indican el trayecto por el cual debe regresar el vehículo para volver al inicio, una vez descargado el paquete.

En la primera de ellas, se prueba la capacidad del vehículo de llevar un paquete hasta el primer depósito junto al inicio, donde se evalúa principalmente la capacidad del vehículo de seguir la línea verde y además, la correcta detección de las sendas peatonales de color rojo, los objetos (semáforo y carteles) y la detección de los códigos QR. La segunda prueba, en donde el vehículo lleva un paquete hasta el segundo depósito, complementa a la anterior en demostrar la capacidad de detectar la bocacalle correctamente, cruzarla y retomar el camino sin perderse una vez que la cruza. También se prueba la capacidad del vehículo de retornar a la base cruzando cada bocacalle que se le presente, una vez que haya dejado el paquete en el segundo depósito. La última prueba de funcionamiento completo, donde el vehículo lleva un paquete al último depósito presente en la pista, permite evaluar que, al recorrer toda la extensión de la pista con distintos niveles de luminosidad, la función que permite adaptar continuamente el filtro de color verde, según el nivel de luminosidad presente en la imagen funciona correctamente.

Finalmente, se realizaron diferentes pruebas adicionales para verificar que no haya errores en el sistema. Las mismas incluyen la verificación de no pérdida de frames, revisión minuciosa de la función de toma de decisión durante la detección de objetos, aumento de la velocidad del vehículo, comprobación de alineación y proximidad correctas a la senda peatonal en depósitos e inicio, y por último, intercambiar el orden de los depósitos. En todos lo escenarios se obtuvo un buen desempeño del sistema.

6 Conclusiones

Se logró desarrollar un sistema que cumple con las condiciones mínimas para realizar tareas de logística en un almacén industrial. Es decir que, el vehículo es capaz de seguir con exactitud la línea central y corregir su dirección y/o posición en caso de perderla.

[1] Pueden encontrarse videos demostrativo en: https://tinyurl.com/ConduccionAutonomLogist1 y https://tinyurl.com/ConduccionAutonomLogist2

Además, debido a que el mismo es capaz de detectar distintos objetos tales como semáforos o carteles, esto permite adaptar fácilmente el funcionamiento del vehículo ante cualquier disposición de pista. Destacamos la dificultad que conlleva integrar técnicas de procesamiento digital de imágenes clásicas proporcionadas por la librería OpenCV, con un modelo de Deep Learning que le proporciona a la visión clásica por computadora propiamente dicha, la capacidad de detección de objetos de manera robusta. En relación a esto, nos gustaría mencionar que muchas de las tareas de procesamiento de imágenes podrían haber sido implementadas con modelos más complejos de Deep Learning para intentar obtener mejores resultados en situaciones donde, por ejemplo, la iluminación no sea controlada. Pero, esto conlleva una mayor carga de procesamiento a la placa y, debido a esto, precisaríamos el uso de una GPU de mayor jerarquía, de la cual no dispone la ODROID XU4, y por lo tanto resulta imposible en nuestro caso implementar más tareas con Deep Learning. Es importante mencionar los problemas que se producen al no trabajar con una iluminación controlada, ya que, por ejemplo, un cambio en la iluminación implica un cambio en la cantidad de puntos que detecta un filtro de color y esto puede traer problemas en los umbrales utilizados.

Referencias

1. Lee, J., Kao, H. and S. Yang: Service innovation and smart analytics for industry 4.0 and big data environment. Procedia Cirp, vol. 16, pp. 3-8 (2014).
2. Iris, F. A. Vis: Survey of Research in the Design and Control of Automated Guided Vehicle Systems. European Journal of Operational Research 170(3), pp. 677-709 (2006).
3. Reporte Tecnológico de Vehículo Autónomo 2020, https://www.wevolver.com/article/2020.autonomous.vehicle.technology.report, última visita 2020/06/18.
4. Conducción Autónoma, Niveles y tecnología, https://www.km77.com/reportajes/varios/conduccion-autonoma-niveles, última visita 2020/06/18.
5. Sistema Operativo Ubuntu Mate 18.04 ODROID, https://www.hardkernel.com/blog-2/ubuntu-18-04-for-odroid-xu4/, última visita 2020/06/18.
6. Usar escritorio remoto con ODROID, https://wiki.odroid.com/odroid-xu4/application_note/software/headless_setup, última visita 2020/06/18.
7. Instalación de OpenCV en ODROID, https://medium.com/analytics-vidhya/iot-opencv-4-1-on-odroid-xu4-8a14d395f191, última visita 2020/06/18.
8. Utilizado de Deep Learning con OpenCV, https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV, última visita 2020/06/18.
9. Librería WiringPi, http://wiringpi.com/, última visita 2020/06/18.
10. Redmon, J. and Farhadi, A.: Yolov3: An incremental improvement, en arXiv:1804.02767, Disponible en: https://arxiv.org/abs/1804.02767, (2018).
11. Redmon, J., Divvala, S., Girshick R. and Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788 (2016).
12. Detección de objetos con YOLO y OpenCV, https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/, última visita 2020/06/18.
13. YOLO web oficial, https://pjreddie.com/darknet/yolo/, última visita 2020/06/18.
14. Reentrenando Tiny YOLOv3, https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/, última visita 2020/06/18.
15. Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg.

ANEXO I: GRAFCET de funcionamiento

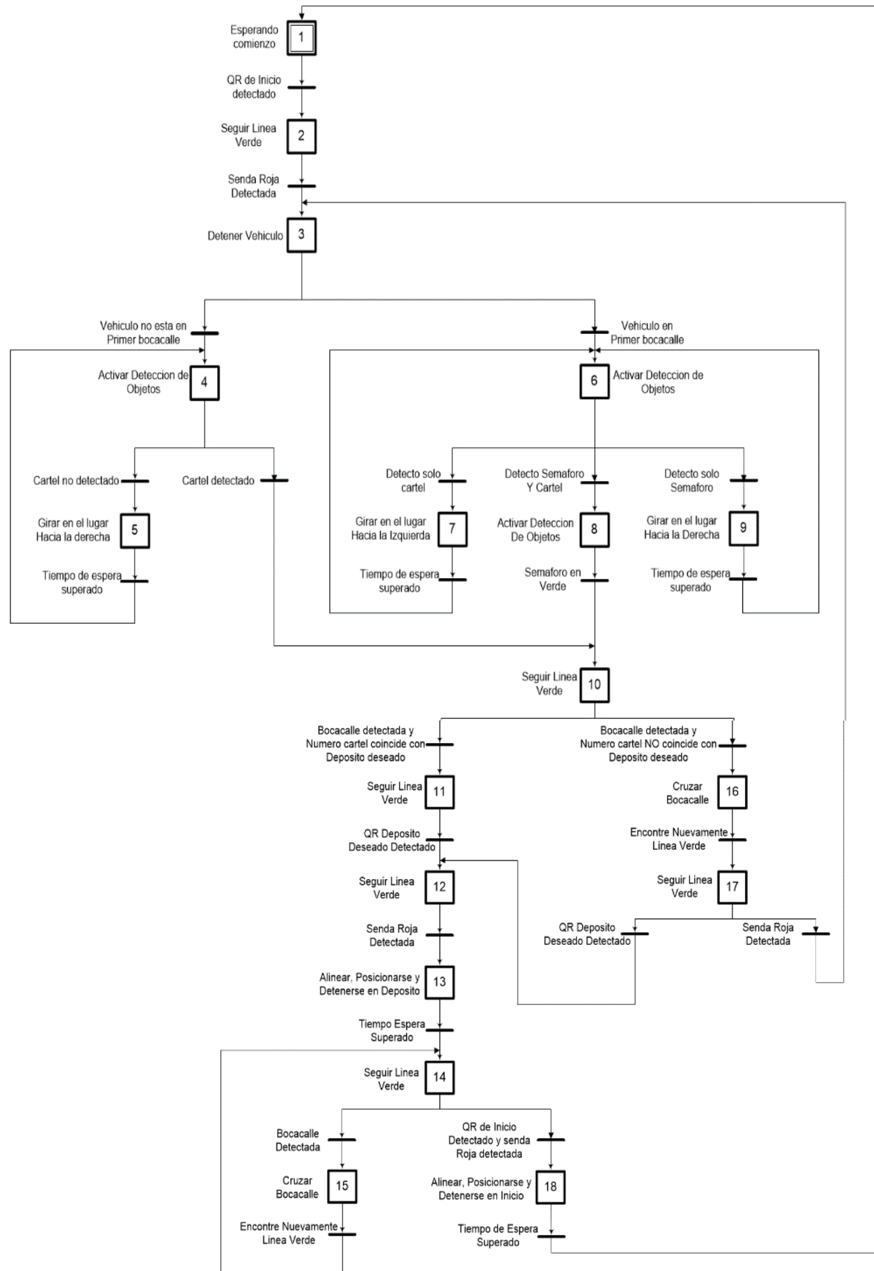


Fig. 11. GRAFCET de funcionamiento del sistema desarrollado.

ANEXO II: Tiny YOLOv3, características y re-entrenamiento

Tiny YOLOv3 [10], es una versión compacta del modelo YOLOv3. YOLO [11] (You Only Look Once) es un sistema de código abierto para detección de objetos en tiempo real. Este mismo consiste en una CNN (Convolutional Neural Network) para detectar objetos. Dentro de Deep Learning, una CNN es un tipo de red profunda, más comúnmente utilizada para analizar imágenes visuales. El objetivo de la CNN es aprender características de orden superior utilizando la operación de convolución. Las redes neuronales convolucionales se forman utilizando principalmente dos tipos de capas: convolucionales y pooling. Estas capas aprenden progresivamente las características de orden superior de la entrada sin procesar. Este proceso para aprender características automáticas es la característica principal del modelo de Deep Learning, llamado descubrimiento de características.

YOLOv3 se distingue de sus "competidores" porque, como lo indica su nombre requiere de "ver" la imagen una sola vez, lo que le permite ser el más rápido de todos (aunque sacrifica un poco de exactitud a cambio). Esta rapidez le permite fácilmente detectar objetos en tiempo real en videos (hasta 30 FPS).

Para llevar a cabo la detección, primero divide la imagen en una cuadrícula de SxS (imagen de la izquierda en la Figura 12). En cada una de las celdas predice N posibles objetos (imagen del centro), los cuales encierra mediante "bounding boxes" y calcula para cada objeto detectado un puntaje que indica la certeza de que dicha predicción es correcta. Es decir, se calculan SxSxN diferentes cuadros, y en cada uno de ellos se anota un valor entre 0 y 1, donde por ejemplo 0.1 significa que el modelo solo puede asegurar en un 10% que la predicción para el objeto detectado es correcta y 1 significa que la predicción para el objeto detectado se puede asegurar en un 100%. Después de obtener estas predicciones, lógicamente la gran mayoría de ellas posee un puntaje muy bajo, por lo tanto, se procede a eliminar los cuadros que posean un puntaje por debajo de un límite determinado (umbral). A los cuadros restantes se les aplica un proceso de "non-max suppression", que sirve para eliminar posibles objetos que fueron detectados por duplicado y así dejar únicamente el más preciso de ellos (imagen de la derecha).

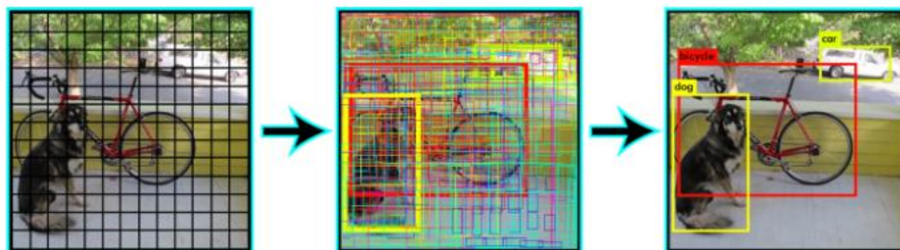


Fig. 12. Cuadricula y "bounding boxes" utilizando YOLO.

En la actualidad hay 3 versiones principales de YOLO, éstas son YOLOv1, YOLOv2 y YOLOv3, y sus correspondientes versiones compactas YOLOv2 Tiny y YOLOv3

Tiny. La primera versión propone la arquitectura general, mientras que en la segunda versión se refina el diseño y se hace uso de cuadros de anclaje para mejorar la calidad del cuadro delimitador (bounding box). La tercera versión mejora aún más la arquitectura y el proceso de entrenamiento.

En particular, YOLOv3 Tiny, posee diferentes capas [10] como se observa en la Figura 13. Entre ellas se distinguen capas convolucionales, de max-pooling y de upsampling. Además, se observa que la primera capa tiene una dimensión de entrada de 416 x 416 x 3, las cuales se corresponden con el tamaño de las imágenes a analizar de dimensión 416 x 416 y 3 colores, pertenecientes al espacio de colores RGB. Cabe aclarar también que la estructura de YOLOv3 Tiny no es la de un modelo secuencial, por lo tanto, las dimensiones de entrada de la capa siguiente, no siempre son equivalentes a la de salida de la capa anterior, según se muestra en la Figura 13.

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Fig. 13. Capas de YOLOv3 Tiny.

Para re-entrenar el modelo YOLOv3 Tiny, es necesario utilizar una computadora con GPU. Dicha GPU debe tener como mínimo 8GB de memoria de video (VRAM) para re-entrenar el modelo. En la misma, utilizando un sistema operativo Ubuntu 18.04

LTS (de 64 bits), realizamos los pasos necesarios para instalar el framework darknet [13], necesario para re-entrenar YOLOv3 Tiny.

Para instalar darknet, descargamos la carpeta del mismo, lo compilamos e instalamos. Sin embargo, se requiere la instalación de CUDA y OpenCV para correrlo. CUDA son las siglas de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo) que hace referencia a una plataforma de computación en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVIDIA. CUDA intenta explotar las ventajas de las GPU frente a las CPU de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos.

Una vez instalado darknet en la computadora virtual, seguimos un tutorial muy completo y detallado [14] para realizar el proceso completo de re-entrenamiento del modelo. Como primer paso, preparamos los datos de entrenamiento. Para ello, sacamos 500 fotografías de cada objeto a detectar con diferentes niveles de luminosidad, distancia a la cámara, fondo y posición. De esta forma, al ser cuatro los objetos a detectar (cartel uno, cartel cero, semáforo verde y semáforo rojo), sacamos 2000 fotografías en total. Cabe destacar que todas estas fotografías fueron tomadas con la misma cámara utilizada en nuestro prototipo, de manera de obtener la mejor precisión posible al momento de detectar objetos. Luego, etiquetamos cada fotografía como se indica en la Figura 14, para entrenar el modelo correctamente. Finalmente, dividimos el total de los datos en dos sets: entrenamiento y prueba (80% entrenamiento y 20% prueba).

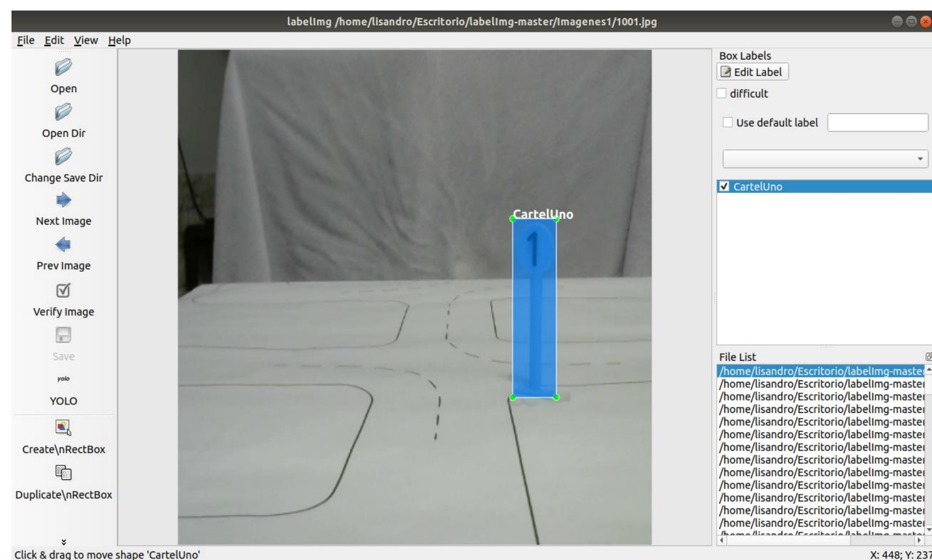


Fig. 14. Etiquetado de imágenes mediante labelimg [15].

Una vez preparados los datos de entrenamiento, descargamos los pesos pre-entrenados del modelo YOLOv3 Tiny y su archivo de configuración (.cfg). Posteriormente, en el archivo “darknet.data” indicamos los directorios donde el modelo buscará los datos de entrenamiento, la cantidad de clases a detectar, las etiquetas asignadas a cada una de dichas clases y el directorio donde se desea guardar los pesos del modelo una vez entrenado.

Finalmente, el último paso previo al entrenamiento es configurar algunos parámetros en el archivo “darknet-yolov3_tiny.cfg”, tales como: tamaño de baches (batch), cantidad de subdivisiones (subdivisions), dimensiones de las imágenes de entrada (width, height, channels), tasa de aprendizaje (learning rate) y finalmente el número máximo de iteraciones (max_batches).

Durante el entrenamiento, se puede ir consultado la evolución del error de predicción. En la Figura 15 se observan dos etapas del entrenamiento, donde en la primera estando casi a la mitad del proceso de entrenamiento, luego de 6.000 iteraciones (primera columna) y de haber procesado aproximadamente 384.320 imágenes (última columna), observamos que el error es de aproximadamente un 7% (segunda columna). En cambio, en la segunda captura se observa que, al finalizar el proceso de entrenamiento, luego de 16.000 iteraciones y de haber procesado 1.024.000 imágenes, el error disminuye a un 4,5%.

```
6005: 0.059494, 0.068552 avg, 0.001000 rate, 1.547830 seconds, 384320 images
6006: 0.072616, 0.068959 avg, 0.001000 rate, 1.505332 seconds, 384384 images
6007: 0.093679, 0.071431 avg, 0.001000 rate, 1.499358 seconds, 384448 images
6008: 0.068683, 0.071156 avg, 0.001000 rate, 1.475360 seconds, 384512 images
...
15997: 0.041607, 0.046756 avg, 0.001000 rate, 3.017903 seconds, 1023808 images
15998: 0.029164, 0.044997 avg, 0.001000 rate, 3.069274 seconds, 1023872 images
15999: 0.036506, 0.044148 avg, 0.001000 rate, 2.993831 seconds, 1023936 images
16000: 0.065248, 0.046258 avg, 0.001000 rate, 3.038292 seconds, 1024000 images
lisandro@lisandro-PC:~/Escritorio$
```

Fig. 15. Información de salida obtenida durante el proceso de entrenamiento del modelo YOLOv3 Tiny utilizando darknet.

ANEXO III: Piezas diseñadas e impresas en 3D

Debido al reducido rango de visión de la cámara utilizada, se optó por diseñar una torre, la cual se muestra en la Figura 16, para elevar su campo de visión. Además, se buscó que con dicho diseño sea posible variar la inclinación de la cámara, en caso de ser necesario. Por otro lado, esta torre cumple una segunda función, la cual es darle un soporte físico a la placa de protección de sobrevoltaje, para aprovechar al máximo el espacio disponible. El interior de la torre fue diseñado para que se puedan almacenar en él, cables y conectores, logrando así ocultar la cantidad de cables visibles en el vehículo. Además, la ranura en la parte superior a ambos lados de la torre, permite mantener la cámara fija.

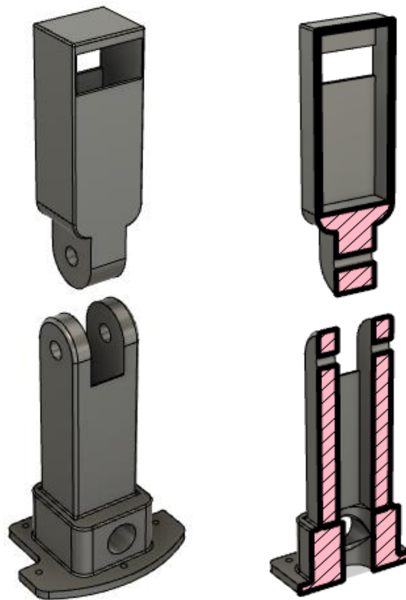


Fig. 16. Diseño en 3D de la torre de soporte para la cámara.

Por otro lado, el lugar en la parte superior del vehículo disponible para colocar el hardware necesario no era suficiente. Por ello, se diseñó el soporte mostrado en la Figura 17. Dicho soporte permite ubicar tanto la placa principal de procesamiento, el conversor I2C a PWM, ambos reductores de tensión Buck, y las baterías junto a su placa de protección.

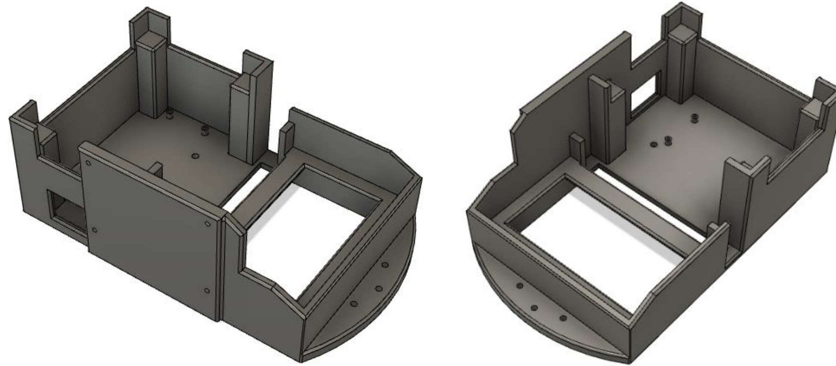
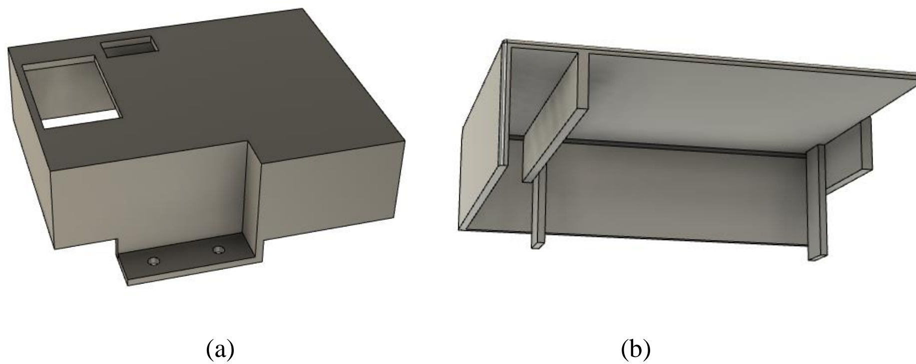


Fig. 17. Diseño en 3D del soporte principal de hardware.

A fin de proteger y ocultar visualmente la placa de protección de sobrevoltaje que se creó para el presente proyecto, colocamos la tapa mostrada en la Figura 18 sobre ella. La misma a su vez, tiene una segunda utilidad, la cual es darle un lugar físico y visualmente agradable al indicador de nivel de carga de las baterías y su tecla de encendido/apagado. Dicha tapa va atornillada sobre la torre para la cámara, donde sobre un costado de la misma, como se observa en la Figura 18, se monta la protección de sobrevoltaje mencionada.



(a) (b)

Fig. 18. (a) Diseño 3D de la tapa de la placa de protección montada sobre la torre. (b) Diseño de la tapa de las baterías en el frente del vehículo.

ANEXO IV: Alimentación y protecciones del vehículo

Para alimentar los módulos del vehículo se optó por usar baterías de litio, modelo 18650. Las mismas tienen 3.7V-4V cargadas, y a medida que se descargan llegan hasta 3.2V. Las mismas ofrecen una corriente máxima de descarga de 4A por lo general, en sus versiones más económicas. Considerando que la corriente máxima, en funcionamiento, que consumen las placas de procesamiento es de aproximadamente 3,2A (referenciado a 5V) y que el consumo máximo del conjunto driver y motores es de aproximadamente 2A (referenciado a 12V), da como resultado 3,34A (referenciado a 12V), se puede asegurar que estas baterías pueden entregar esta corriente máxima necesaria.

En principio se pensó alimentar al driver de los motores directamente con el pack de baterías a 12V de salida. Sin embargo, al no ser constante el nivel de tensión entregado a la salida de las baterías, el torque de los motores no era constante y, por lo tanto, era imposible ajustar los parámetros para controlar con exactitud el torque y velocidad de respuesta de los motores. Por lo tanto, luego de detectar que el problema era la falta de regulación de las baterías, se decidió agregar un reductor Buck para fijar la tensión de alimentación de los drivers de los motores a 12V. Sin embargo, debido a que dicho reductor precisa de por lo menos (según el fabricante) 1V de diferencia de tensión entre la salida y la entrada, se necesita una tensión mínima de entrada igual a 13V para asegurar los 12V a la salida del mismo. Por lo tanto, se decidió agregar una batería extra de manera tal de aumentar la tensión de alimentación en un rango de 13,2V (siendo el valor mínimo de cada batería igual a 3,3V antes de que actué la protección) a 16V (cuando todas las baterías tienen su máxima carga). De este modo podemos asegurar que el reductor Buck funcionará correctamente ya que el valor mínimo de alimentación supera al valor requerido (13,2V > 13V). De esta manera obtenemos 12V constantes reduciendo los 16V de las baterías con un primer reductor Buck. Además, para alimentar la placa de procesamiento necesitamos 5V constantes, por lo tanto, implementamos un segundo reductor Buck, reduciendo los 12V a 5V.

Un problema común que se presenta en las baterías comerciales es que al descargarse por debajo de un cierto nivel de tensión (2,7V aproximado), estas se vuelven inutilizables. Además, un problema típico de los packs de baterías sin regulación es que la descarga entre baterías no suele ser equitativa y por lo tanto una se descarga mucho más que las otras. La combinación de estos problemas implica que el utilizar un pack de baterías sin protección conlleva a que una o más celdas queden inutilizable. Por lo tanto, se optó por agregar un circuito de protección BMS (modelo 4S40A) de sobrevoltaje y sobredescarga para baterías de litio (incluyendo 18650, 26650, batería de polímero de litio) con un voltaje nominal entre 3,6V y 3,7V. El circuito de protección tiene dos funcionalidades principales. Una es asegurarse de que la distribución de carga de las baterías sea equitativa, de manera que todas se descarguen. Además, para reutilizar las baterías una vez descargadas, usamos esta misma protección para asegurarnos que los niveles de tensión de todas estas sean iguales.

Todos los dispositivos de alimentación utilizados son de una calidad aceptable y durante todo el desarrollo del trabajo no hubo ninguna situación de sobretensión. Sin embargo, con que uno de ellos falle puede significar la destrucción total de la placa de control y procesamiento. Por esto se optó por agregar una protección ante sobrevoltaje

que aísle a la placa de control de la alimentación en caso de que esta posea un pico de tensión. El circuito utilizado para esto se puede ver en la Figura 19.

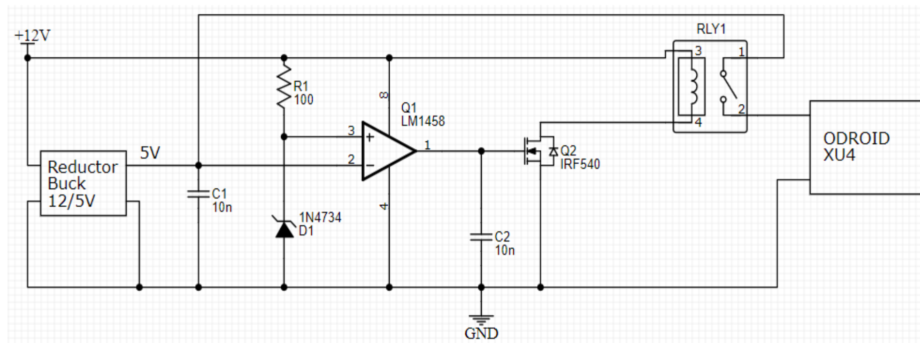


Fig. 19. Circuito de protección ante sobrevoltaje.