UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INGENIERIA, INDUSTRIA Y CONSTRUCCION

CARRERA DE INGENIERIA ELECTRICA

DISEÑO DE UN SISTEMA DE PARKING AUTOMATICO MEDIANTE TECNICAS DE VISION ARTIFICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRICO

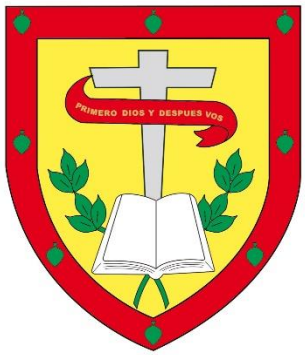
AUTOR: RONALD ALEXIS SEGARRA GUZMAN

DIRECTOR: ING. ELEC. JUAN CARLOS COBOS TORRES

CUENCA - ECUADOR

2022

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INGENIERÍA, INDUSTRIA Y CONSTRUCCIÓN

CARRERA DE INGENIERÍA ELÉCTRICA

DISEÑO DE UN SISTEMA DE PARKING AUTOMÁTICO MEDIANTE TÉCNICAS DE VISION ARTIFICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELÉCTRICO

AUTOR: RONALD ALEXIS SEGARRA GUZMÁN

DIRECTOR: ING. ELEC. JUAN CARLOS COBOS TORRES

CUENCA - ECUADOR

2022

DIOS, PATRIA, CULTURA Y DESARROLLO

Declaratoria de Autoría y Responsabilidad

Ronald Alexis Segarra Guzmán portador(a) de la cédula de ciudadanía Nº **0105079990**. Declaro ser el autor de la obra: "**Diseño de un sistema de parking automático mediante técnicas de visión artificial"**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, **15 de febrero de 2022**

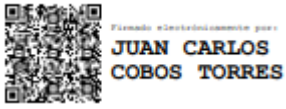
F: .....

Ronald Alexis Segarra Guzmán

0105079990

CERTIFICACIÓN

Certifico que el presente trabajo de titulación fue desarrollado por el estudiante Ronald Alexis Segarra Guzmán bajo mi supervisión.



……

Ing. Elec. Juan Carlos Cobos Torres, PhD.

DIRECTOR

DEDICATORIA

Esta tesis la quiero dedicar a mis padres, Euclides Segarra Solano y Estela Guzmán Calle, quienes me han dado la fortaleza para seguir adelante y cumplir mis sueños, entregándome todo su apoyo, ánimo y estando allí siempre para mí.

A mi esposa Jessica que siempre ha estado para mi animándome para lograr y cumplir mis sueños.

También quiero dedicar también esta tesis a mis hermanos Edison y Gissela, que han estado allí brindándome todo su apoyo, animándome para que no decaiga en todo este proceso.

Ronald Alexis Segarra Guzmán

AGRADECIMIENTO

Quiero agradecer primero a Dios, por su inmensa bendición a lo largo de mi vida; porque por el estoy cumpliendo todo lo que he soñado, con esfuerzo, sacrificio y mucha dedicación.

Quiero agradecer a mis padres Euclides Segarra Solano y Estela Guzmán Calle, quienes han sido siempre pilares fundamentales y mis consejeros diarios, sin ellos no sería la persona que ahora soy, gracias queridos padres por todo.

Agradezco también a cada uno de los profesionales docentes de la Universidad Católica de Cuenca y de manera muy especial al PhD. Juan Carlos Cobos Torres, quien ha tenido esa paciencia, amistad y sabiduría para ser mi guía y tutor para así poder llevar a cabo este proyecto.

Ronald Alexis Segarra Guzmán

Resumen

Una de las situaciones más complejas que se vive en la actualidad es la falta de tiempo, el problema más común de un conductor es el tiempo empleado en buscar una plaza para aparcar el vehículo. Esto provoca distintas consecuencias, tales como: atrasos, estrés y contaminación. Por lo mismo, la presente investigación mediante el uso de la visión artificial busca innovar tecnológicamente a los parqueaderos con la detección automática de plazas de parqueo. Funcionando como herramienta eficiente para mejorar la satisfacción del usuario, quien mediante un mensaje podrá conocer la disponibilidad de plazas; además, reduciendo costos para los administradores de parqueaderos al no tener que instalar sensores costosos, reduciendo la necesidad de personal, entre otros. Provocando un impacto económico, social y ambiental.

Este trabajo de titulación al ser un proyecto piloto tuvo como objetivo implementar un parking inteligente que cuenta con 5 plazas automatizadas; el mismo que será capaz de determinar los espacios de parqueo libres y ocupados mediante cámaras de seguridad. Es importante mencionar, la implementación de un chatbot que se comunica a través de Telegram con los usuarios, permitiendo la interacción con el usuario para informar el nivel de ocupación y disponibilidad del parqueadero. Tras pruebas realizadas al desarrollo se obtuvo un 94% de efectividad en la detección de vehículos. Lo importante es, que este prototipo se puede escalar e instalar por un precio reducido para todas las plazas de parqueo de Posgrados y de otras dependencias de la Universidad Católica de Cuenca.

Palabras clave: visión artificial, detección de objetos, parking inteligente, seguimiento de objetos

Abstract

One of the most complex situations that we live in nowadays is the lack of time, the most common problem of a driver is the time spent looking for a parking space. This causes different consequences, such as delays, stress, and pollution. For the same reason, the present research through the use of artificial vision seeks to technologically innovate parking lots with the automatic detection of parking spaces. Functioning as an efficient tool to improve user satisfaction, who through a message will be able to know the availability of parking spaces; in addition, reducing costs for parking lot managers by not having to install expensive sensors, reducing the need for personnel, among others. This will have an economic, social, and environmental impact. As a pilot project, the objective of this degree project was to implement an intelligent parking lot with 5 automated parking spaces, which will be able to determine free and occupied parking spaces using security cameras. It is important to mention, the implementation of a chatbot that communicates through Telegram with users, allowing interaction with the user to inform the level of occupancy and availability of the parking lot. After tests carried out on the development, 94% of effectiveness was obtained in the detection of vehicles. The important thing is that this prototype can be scaled and installed for a reduced price for all the parking spaces of Postgraduate and other facilities of the Catholic University of Cuenca.

Keywords: Artificial vision, object detection, intelligent parking, object tracking

Contenido

ÍNDICE DE IMÁGENES

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1

1. INTRODUCCIÓN

La visión artificial en la actualidad se viene utilizando con mayor énfasis dentro de la industria que ayuda a los procesos de calidad y seguridad; pues permite obtener información sin necesidad de contacto alguno con los procesos o maquinarias

Existen empresas como INFAIMON, a nivel internacional, teniendo claro que la visión artificial está presente de manera indispensable en la industria automotriz, aeronáutica, electrónica, alimenticia, farmacéutica, entre otros. Ellos colaboran mediante diversas librerías para trabajar, por ejemplo, con el reconocimiento de matrículas vehiculares. Logrando así garantizar la fiabilidad de reconocimiento mediante su librería de multiplataforma en lenguajes de programación distintos, como Visual Basic, C++, Borland, entre otros. Son múltiples las aplicaciones logradas, entre las cuales se puede enumerar aplicaciones de control de acceso en aparcamientos, supervisión de tráfico en autopistas y carreteras e inventario de vehículos utilizando captura de imagen.

Las Cámaras web de ordenadores o de celulares al igual que las tabletas electrónicas son accesibles en nuestro entorno social y económico, cumpliendo el mismo principio en comparación a cámaras LPR utilizadas en visión artificial. Para la aplicación de visión artificial existen diferentes métodos utilizados basados en la utilización de sensores específicamente colocados referentes a la necesidad del campo de acción, mediante el uso de software, se armoniza con el hardware para la coordinación y servicio con el usuario.

La estructura que rige un sistema de parqueo inteligente, se refleja en tres ámbitos importantes.

a. La detección de movimiento.
b. La comunicación con el sistema.
c. El software de aplicación.

La implementación de esta estructura en conjunto manifiesta un sistema de parqueo que se rige por los sensores colocados en cada espacio de parqueo, la comunicación se realiza mediante redes LAN implementadas en el sistema, la inversión de la estructura mencionada es deficiente al utilizar sensores para la detección y comunicación con el servidor del sistema equipara una inversión significativa en relación a la dimensión del parking.

Otra opción, es la utilización de sensores con comunicación WIFI como por ejemplo en la empresa iparkings la utilización de una estructura inalámbrica que utiliza internet para la comunicación con el servidor. Según (Nelson, 2016), la comunicación se basa en V21 (Vehicle-to-Infraestructure) definido en la comunicación del vehículo y una infraestructura RSU (Roadside Unit). La utilización del servidor SAE permite identificar los espacios libres y ocupados de parqueo mediante los sensores de movimiento NMP los mismos que se comunican inalámbricamente con SAE mediante internet y el SAE a su vez con el usuario, si bien la comunicación es de manera directa con el usuario la deficiencia de contar con sensores de movimiento en cada espacio de parqueo dificulta la inversión en parkings de menor dimensión, su comunicación es mediante internet sin ninguna forma de respaldo, así disminuye su garantía en la efectividad las interrupciones de ruido o datos en la red pueden causar problemas de comunicación de igual manera la empresa SENSIT utiliza una comunicación inalámbrica WIFI de los sensores con el servidor, en este caso la utilización de sensores de movimiento se comunican con una central de nodos de relés montados, se comunica con el servidor identificando espacios libres y ocupados, el depender de una red de Internet dificulta su funcionamiento por caídas de servicio o la red.

Un método de comunicación directa se ve reflejado en la implementación de sensores ultrasónicos SROFO2 que se ubican en un espacio de parking con una distancia de 2 metros, la comunicación con un microcontrolador Atmega328 permite identificar los espacios libres y ocupados al comunicarse con un Arduino Yun que analiza el estado de los sensores de movimiento, la detección se verifica con la programación de un software libre la misma que analiza los umbrales de los sensores SROF2 (Acosta Jazmin, Tintos Juan P, 2014) para una distancia que excede los dos metros identifica un espacio libre y para una distancia menor espacio ocupado mediante 0 y 1 digital, cabe recalcar que la comunicación entre los sensores es mediante cable UTP pero la interacción de una plataforma mediante internet con temboo permite mediante twitter controlar los espacios libres para el cliente , el problema se ve en la identificación con sensores que dependen de un objeto de aproximación y una distancia de 2 metros la que no garantiza que pueda ser otro objeto y no un automóvil.

El sistema que utiliza un respaldo en los sensores de aparcamiento al tener una red tipo malla y digimesh ante fallas del sistema la comunicación se realiza mediante dos sensores colocados en cada espacio libre, la comunicación inalámbrica en digimesh concede mayor garantía en la ubicación del vehículo al analizar el espacio acorde a las dimensiones de diferentes vehículos, a su vez se comunica inalámbricamente a la red mesh (malla) de los routers identificados en el sistema que se comunicaran con el

coordinador Gateway que analiza y se comunica con una emisora la que analiza los sensores y se comunica a un servidor que permite analizar los datos, el problema de ocupar sensores de precisión o aparcamiento no se ve reflejado en la inversión de un sistema de parqueo, es verdad que al utilizar dos sensores por espacio garantiza la eficiencia del sistema, pero no se representa económicamente al implementarlo en un parking estándar, un parking especifico de un local comercial o un domicilio puede funcionar el sistema con mayor garantía y seguridad con una inversión alta en la aplicación, no así en un centro comercial o universidad.

Al analizar de manera general los diferentes sistemas de parqueo inteligente, se puede observar que los sistemas integrados en la actualidad cuentan con sensores de precisión, ultrasónicos o de movimiento infrarrojos, o con radares que delimitan los espacios libres, en cuanto a la comunicación, esta varia, el usuario en general ha optado por una comunicación inalámbrica, de costo elevado, pero existen algunos de manera autónoma más rústicos y elementales, implementar sensores en cada espacio de estacionamiento dificulta el tráfico de datos y la inversión inicial del sistema evidenciando económicamente que no es viable.

Según (Carlos, 2016), la implementación de una red SWAP del controlador a los sensores debe tener un análisis más detallado ya que al comunicarlos con el nodo central Gateway se pueden encontrar diferentes Gateway en el campo de acción de los mismos sensores, por lo tanto se debe elegir un direccionamiento adecuado a los Gateway, ya que se puede conseguir tráfico de datos o duplicado de los mismos al servidor, por lo tanto la implementación es crear una red por Gateway que se asemeje con un grupo n de sensores, lo que identifica varias redes de acceso por Gateway, podemos evidenciar que la implementación de sensores inalámbricos hacia un servidor que dará como resultado un tráfico de datos o repetición de los mismos si no se logra direccionarlos, el momento que un Gateway deje de funcionar no se podrá colocar o redireccionar los sensores hacia otro Gateway, por lo que vamos a evidenciar un sistema obsoleto.

En este proyecto, se propone realizar un parking inteligente con el mismo principio de funcionamiento presentes en el mercado, la detección de espacios libres y la notificación de los espacios ocupados mediante una interacción con el propietario del parking, la diferencia que no se va a utilizar sensores de aproximación ni infrarrojos, en los parkings se utilizara cámaras web para la detección de movimiento, espacios libres y ocupados mediante bloques previamente programados en el software OpenCV, la utilización de cámaras no se efectuará por espacio de parking, por lo contrario se utilizará por sectores de parking evidenciando, mediante el posicionamiento correcto, así como el ángulo de

colocación adecuado de las cámaras para así lograr identificar varios espacios libres en el parking, la implementación de las cámaras con relación a los sistemas SWAP en cuanto al ámbito económico tiene una diferencia muy representativa ya que solo se necesita cámaras colocadas específicamente y la comunicación será mediante cable a un ordenador el mismo que podrá identificar los espacios libres mediante el software aplicado.

Actualmente la ciudad de Cuenca no cuenta con ningún parking con tecnología loT, de tal manera que resultaría el primero en que una institución cuencana tenga dicha tecnología, brindando así el confort, la comodidad y la seguridad del usuario.

La propuesta es diseñar y construir un parking automático mediante técnicas de visión artificial con un software libre y hardware DIY para mejorar la experiencia de los usuarios y gestores del parqueadero. Este parking estará implementado en el Departamento de Posgrados de la Universidad Católica de Cuenca.

1.1. Formulación del problema.

¿Qué tiempo hemos perdido en buscar un sitio de parqueo? ¿Quién no se ha atrasado a clases o hasta ha faltado por no encontrar un sitio donde dejar su vehículo? Este tipo de situaciones son a las que enfrentamos, tanto estudiantes o docentes todos los días a todas horas puesto que en nuestra ciudad tiene un alto índice de circulación vehicular generando así niveles de tráfico elevados.

Hay que considerar también que este tipo de situaciones no solo genera un estrés o malestar llevando así a un mal estado de salud, sino también repercute problemas económicos y hasta ambientales.

Según estudios realizados en Europa, el 9% de los conductores pasan por lo menos 15 minutos al día buscando sitios para parquear su vehículo, mientras que, en Norte América, la mayoría de conductores han sentido frustraciones al momento de buscar lugares de parqueo llegando al punto de perder sus citas por no encontrar estacionamiento.

Ante este tipo de problemas varios parkings han buscado opciones donde que se puede determinar un lugar sitio para parquear de manera más rápida, en nuestra ciudad se ve dos casos para dar solución a esta situación. En el primer caso, se necesita de 2 personas para efectuar estas acciones, la una persona es la que hace el ingreso de datos del vehículo y la otra persona es la que indica donde hay lugar, generando así más gastos de recursos y pérdida de tiempo. El segundo caso, es que mediante un sensor de color verde se determina si el sitio de parqueo está libre

o si no existe un sensor de color rojo que determina si está en uso, generando así un pasatiempo al usuario e inconformidad.

De tal manera que, si consideramos todas las situaciones mencionadas, se puede denotar que es necesario crear un sistema de parqueo automático para así mejorar la experiencia del usuario.

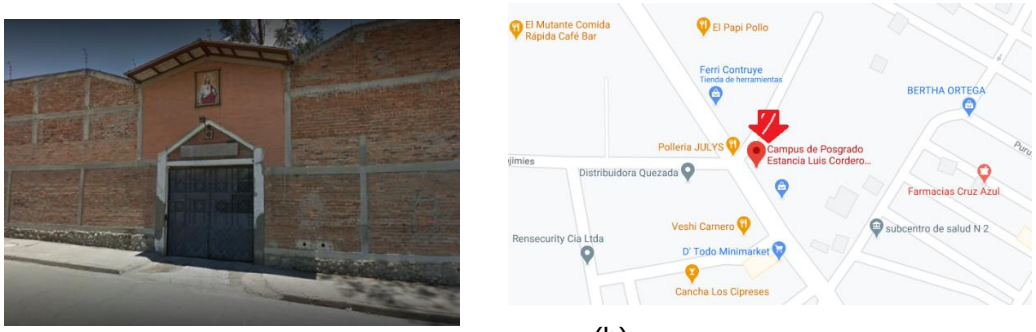
1.2. Delimitación del problema.

El Departamento de Posgrados de la Universidad Católica de Cuenca en la actualidad no cuenta con un sistema de parking automatizado, haciendo que los conserjes se encarguen de esta situación y dejando a un lado las actividades por las que fueron contratados.

Por esa esta situación, el presente proyecto busca adecuar un sistema de asignación de plazas de manera automática mediante el parking inteligente haciendo uso de la tecnología IoT, cuyo funcionamiento hace referencia al análisis de imágenes, denotando los lugares ya sean libres o en uso, para que así el usuario pueda acceder con mayor facilidad al lugar de estacionamiento.

1.3. Definición de la zona de estudio.

Las cámaras se instalarán en sitios estratégicos del parking del Campus de Postgrado Luis Cordero El Grande de la Universidad Católica de Cuenca (Ilustración 1), el mismo que se encuentra ubicado la parroquia Hermano Miguel en la Vía a Patamarca y calle Cojimies. Este Campus cuenta con un solo acceso el mismo que sirve tanto para peatones y/o para vehículos, este Campus al ser frecuentado por muchos estudiantes cuenta con 3 zonas amplias para el uso de parking, de tal manera que se realizará un levantamiento de estas zonas para elaborar un plano donde se distribuirá de manera correcta las plazas donde los usuarios podrán dejar su vehículo.



(a) (b)

Ilustración 1. Ubicación del Campus de Posgrados Luis Cordero El Grande de la Universidad Católica de Cuenca (a) Ingreso al campus (b) Ubicación en Google Maps. (Google Maps)

El parking inteligente que se implementará será uno de los más completos a nivel del país, puesto que se usará el hardware DIY (Do It Yourself), para mejorar la experiencia ya sea usuario o gestor del parking.

1.4. Objetivos.

1.4.1. Objetivo General.

Construir un sistema inteligente de parking basado en loT mediante técnicas de visión artificial con software libre y hardware DIY para mejorar la experiencia para los gestores del parking y para los usuarios.

1.4.2. Objetivos Específicos.

- Revisar las fuentes de información científicas primarias y secundarias mediante la revisión de bases de datos especializadas para identificar las distintas tecnologías, algoritmos y herramientas a implementarse.
- Diseñar el sistema de circuito cerrado de grabación mediante el levantamiento del parking de posgrado para determinar la ubicación y cantidad de cámaras necesarias.
- Desarrollar el sistema de detección de plazas libres mediante software de visión artificial de código abierto para determinar las plazas libres y ocupadas.
- Integrar el sistema de grabación y detección de plazas libres mediante su instalación en el parking de posgrados para comprobar su funcionamiento.
- Generar los sistemas de información visual mediante hardware y software que permitan identificar el estado de las plazas libres y ocupadas.

CAPÍTULO 2.

2. ESTADO DEL ARTE

2.1. Introducción a la visión artificial.

La visión artificial se la relaciona con la inteligencia artificial, de tal manera que hace uso de técnicas precisas, para así poder obtener, procesar y a su vez analizar información adquirida por medio de imágenes digitales (Rojas Rafael, 2016).

2.1.1. Formación de las imágenes.

Las imágenes no son más que matrices representadas por m x n pixeles ver la Ilustración 2 , de tal manera que cada elemento de la matriz viene a ser un pixel. Para entender mejor el pixel será una unidad minúscula que puede conformar una imagen, este a su vez está compuesto por tres colores rojo verde y azul (Alvear Vanessa, Rosero Paul, Peluffo Diego, 2017).

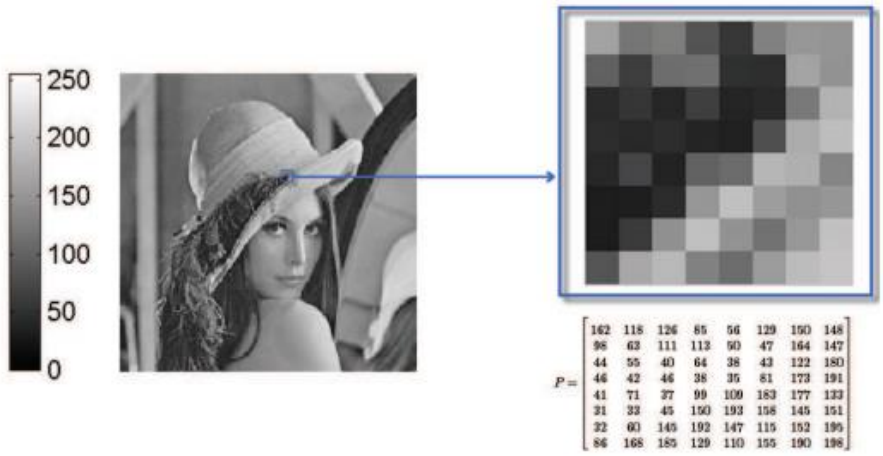


Ilustración 2. Imagen con 256 niveles de intensidad y representación numérica de un fragmento 8x8 (Nicolas, 2013)

2.1.2. Formación del sistema de visión artificial

Para que este sistema llegue a formarse, se debe aplicar varias operaciones matemáticas en cada imagen para obtener resultados. De tal manera, que la imagen será procesada por diferentes técnicas: reconocimiento de la imagen de entrada, el preprocesamiento cuyo fin es reparar la imagen para segmentarse de la mejor manera, luego pasa por la segmentación donde que a la imagen se le divide en diferentes partes para un análisis, luego de ello va a la extracción de las características que consiste en operaciones para poder adquirir las características

específicas, luego pasa por la clasificación donde usa un algoritmo para clasificar ya sea en kmeans o knm y finalmente va a la interpretación donde que se logra reconocer el objeto y se da paso a la asignación mediante una etiqueta (Moises, 2018).

2.2. Machine learning

Machine learning o aprendizaje automático es una rama de la inteligencia artificial, el cual hace referencia a la capacidad que tiene un sistema TI (tecnologías de la información) para poder hallar una solución a los distintos problemas o conflictos que se puedan presentar siendo ejecutado por el reconocimiento de patrones en las distintas bases de datos. Por lo tanto, el aprendizaje automático da la posibilidad de que los sistemas TI puedan reconocer los distintos patrones de la base de algoritmos, además de la cantidad de datos que puedan existir con el fin de ejecutar las soluciones adecuadas (Denniye, 2018).

Se debe tomar en cuenta que se necesita de alguna acción por parte de las personas o animales, para que el software logre desarrollar soluciones inmediatas. Por lo tanto, ya sean algoritmos y datos deberán ser inyectados al sistema de manera anticipada, estableciendo su análisis para así dar paso al reconocimiento de patrones en el centro de los datos, de manera consecuente el aprendizaje automático podrá ejecutar las tareas siguientes:

- Capacidad para encontrar y extraer datos sobresalientes.
- Crear predicciones que se basen en los datos que se hará el análisis.
- Realizar cálculos para obtener resultados específicos.
- Se puede ajustar según la necesidad
- Reduce procesos si estos ya tienen patrones reconocidos.

2.2.1. Funcionamiento de Machine Learning

El funcionamiento en sí, no es tan complejo ya que adquiere cierta cantidad de datos y diferentes líneas de comando, de tal manera que el computador adquiere esa habilidad de aprender para así poder identificar ya sea personas animales o cosas.

Siempre el software que se utiliza está adquiriendo información por parte de la persona que programe, de manera consecuente el algoritmo usara estas señales para poder retroalimentar, adaptar y optimizar el programa creado (Javier, 2017).

2.2.2. Ventajas de Machine learning

Provoca que el entorno de trabajo sea creativo, eficiente y eficaz. Ya que mediante el aprendizaje automático podemos efectuar tareas tan sencillas que van desde el escanear algún documento hasta llegar a editar imágenes.

Pero no solo esto, ya que se pueden tener tareas mucho complejas como en las industrias de manufactura; que su producción es continua y sin ningún tipo de error, ahí es donde machine learning entra en funcionamiento ya que este identifica el error de manera rápida, ahorrando tiempo y dinero. Siendo este un método efectivo en la automatización de control de calidad.

Otra ventaja es que se usan en el campo de la medicina, ya que existen aplicaciones donde al médico se le informa las distintas medicaciones (Alberto, 2019).

2.3. Procesamiento de imágenes.

Básicamente el procesamiento de imágenes (PI) es una subcategoría del tratamiento digital de todas las señales. Siendo esta la ciencia capaz de manipular imágenes haciendo uso de computadoras para poder efectuar técnicas según lo que pida el usuario o la especificación de la aplicación, entre ellos podemos encontrar: los filtros, los recortes, la segmentación y el reconocimiento (Vladislav, 2017).

2.3.1. Imagen.

A una imagen se la puede definir como una función bidimensional de la siguiente manera $f(x_1, x_2)$ en donde que $x = (x_1, x_2)$ siendo estas las coordenadas espaciales, en tanto que f para cualquier x será la intensidad que puede tener la imagen en el punto.

A una imagen se la puede considerar, ya sea como una función continua o discreta (analógica o digital), resultando de ambas maneras útiles para lograr un procesamiento de imágenes.

Para poder convertir una imagen de analógica a digital se deberá tomar en cuenta que se debe digitalizar ya sean las coordenadas y la intensidad. Por eso, a las coordenadas digitalizadas se las llama digitalizar en tanto que a la intensidad digitalizada se le llama cuantizar. De tal manera que, si absolutamente todas las cantidades son discretas, se le denomina imagen digital. Pero si vamos de lo contrario desde lo digital a lo analógico, se lo llama interpolación (Vladislav, 2017).

2.3.1.1. Tipos de imágenes.

- **Imagen binaria. –** esta imagen consiste de 1bit/pixel, y solo puede tener 2 colores ya sea blanco o negro, su representación es 1 blanco y 0 negro tal como se puede observar en la Ilustración 3.

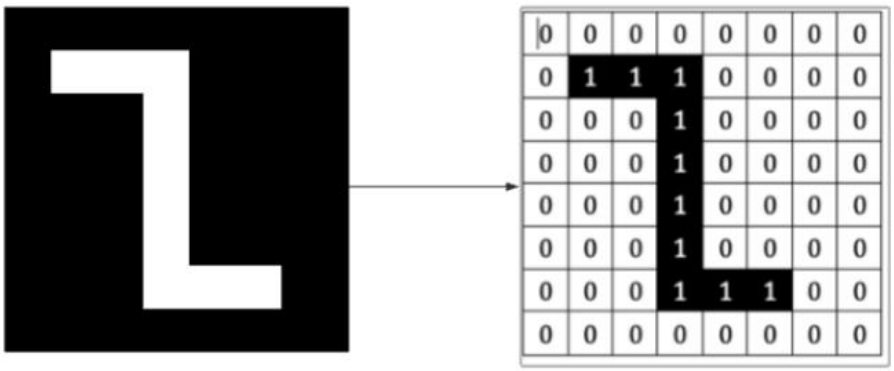


Ilustración 3. Representación de negro y blanco de una imagen binaria (Rafael, 2020)

- **Imagen a escala de grises. –** esta imagen consiste en 8bit/pixel, y puede llegar a tener 256 sombras distintas, representada por 0 pixel el color negro y 255 al blanco como bien se ve en la Ilustración 4.

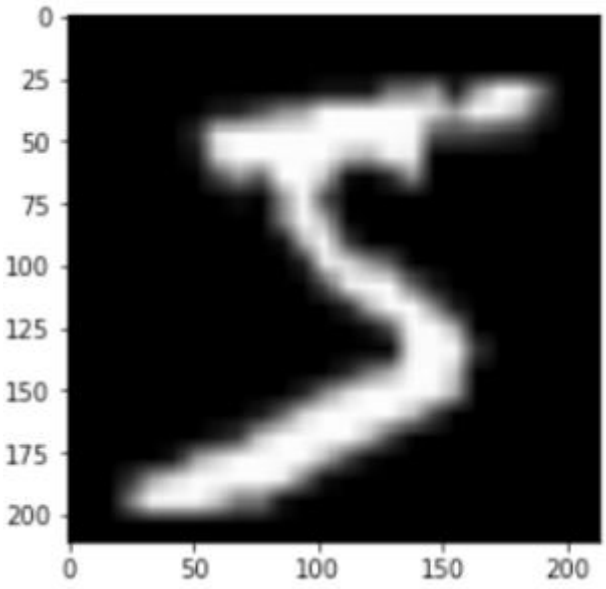


Ilustración 4. Representación de negro y blanco de una imagen a escala de grises (Rafael, 2020)

- **Imagen en color. -** esta imagen se asemeja a la imagen de intensidad (ver Ilustración 5) con la diferencia que tiene tres canales, o sea, en vez de llegar un solo valor de intensidad a cada pixel le llegan tres RGB tal como se puede.

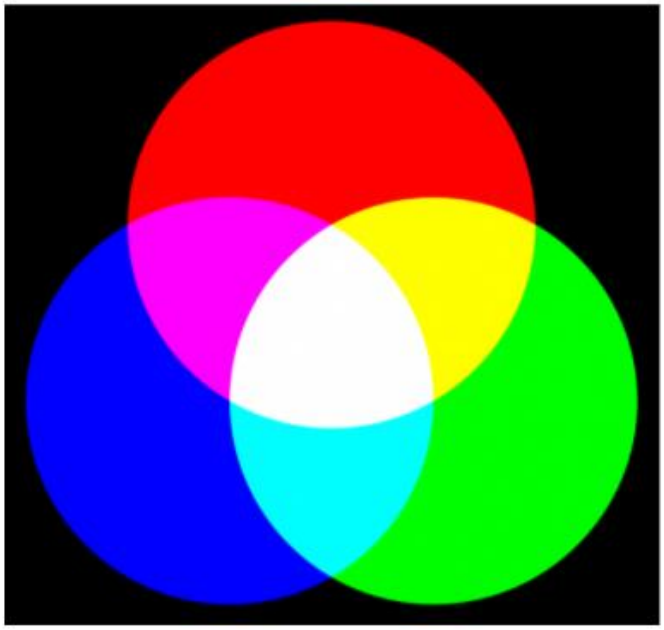


Ilustración 5. Imagen en color (Rafael, 2020)

2.3.2. Espacio de color.

El espacio de color es una herramienta muy útil para poder definir las capacidades de color de algún dispositivo, el mismo que logra producir imágenes o datos. Cualquier espacio de color llega a relacionar números, vectores, nombres, etc. Al estar basado en características matemáticas, permite reproducir representaciones de color entre varios formatos y a su vez varios dispositivos.

Los espacios de color llegaran a depender del dispositivo y el color con relación a algún dispositivo, llegando a ser expresado este color en términos absolutos contra cualquier color que sea estándar (Maria, 2016).

A continuación, se explicarán los espacios más comunes con aplicaciones para el procesamiento de imágenes:

RGB (Red Green and Blue) este modelo es de color aditivo, donde que se mezclan los colores rojo, azul y verde, en el que se describe el volumen de la tercera dimensión (tridimensional) ver Ilustración 6, ajustando los valores de los componentes en coordenadas cartesianas. (Cruz Henry, Meneses Juan, Eckter Martina, 2015).

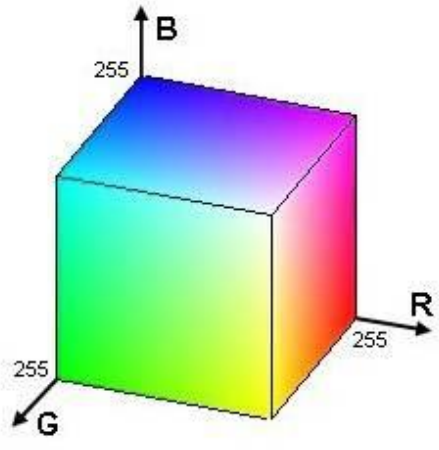


Ilustración 6. Modelo RGB (Angie, 2014)

HSV sus siglas hacen relación al tono, a la saturación y al brillo, de tal manera que se maneja en un espacio de coordenadas cilíndricas. Como se puede ver en la Ilustración 7, el que define el tono es el ángulo que va alrededor del eje del cilindro; que no es más que la longitud de onda que existe dentro del espectro de luz visible donde que la salida de energía de la fuente siempre será mayor ver en la Ilustración 7. ***Espacio de color (Vladislav, 2017).***

. En tanto que la saturación será el ancho de banda relativo de una fuente de luz y viene expresada por la distancia desde el centro del cilindro y finamente el brillo será la amplitud en la longitud de onda donde que la intensidad de onda refleja superioridad es decir mucho mayor (Vladislav, 2017).

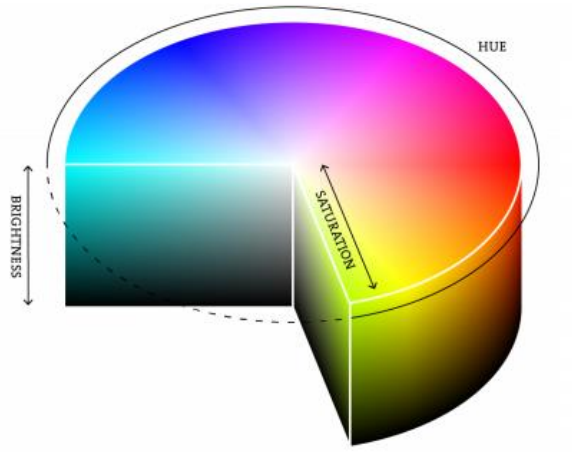


Ilustración 7. Espacio de color (Vladislav, 2017).

YUV en este modelo, la Y se encarga de determinar el brillo ver Ilustración 8. Toda la información del color se divide en 2 canales: en donde que la U y V llegan a ser luminancia azul y rojo (Cesar, 2020).

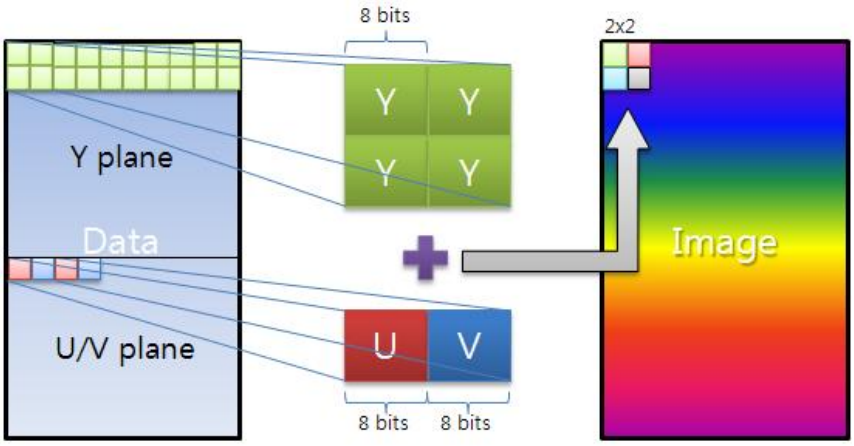


Ilustración 8. Espacio de color (Mohammad, 2016)

2.3.3. El seguimiento de objetos.

El seguimiento de objetos tiene por objetivo principal calcular la trayectoria del objeto en una sucesión de cuadros de video. La apariencia del objeto en el primer fotograma será la entrada del rastreador y la salida será la sucesión de coordenadas del objeto en varios fotogramas de manera consecutiva.

Por lo general, los rastreadores suelen adjudicarse que el objeto llega a ser visible en toda la secuencia. En caso que el rastreador permitiese la desaparición momentánea del objeto, se lo llama seguimiento de objeto a largo plazo (Ameijeiras David, González Hector, 2020).

2.3.4. La detección de objetos.

Su deber es detectar absolutamente todos los objetos que sean de interés en un solo cuadro de video, por lo general los detectores necesitan ser entrenados, puesto que no pueden detectar a objetos que no sean conocidos. Su salida es prácticamente el conjunto de coordenadas de los objetos detectados (Rosas Leonel, Vallejo Jair de Jesus, 2017).

2.3.4.1. Haar Cascade

El método Haar Cascade hasta hace poco era uno de los métodos más populares para detección de objetos, ya que su eficiencia era considerada eficiente. Para que este algoritmo funcione correctamente es necesario entrenarlo con miles de imágenes enfocando a un objeto en particular, estas imágenes se las llama imágenes positivas, ya que están dentro del rango aceptado como por ejemplo 12x12 pixeles. Pero también existe otro tipo de imagen la negativa, siendo las imágenes de las regiones donde se ubicaba el objeto a detectar. Para ello se adquieren características propias, usando las denominadas características Haar.

Se puede considerar una característica Haar a regiones de forma rectangular adyacente en algún punto de detección, a esto se le suma las intensidades de cada pixel y calculando la diferencia que existe entre ellas, para así poder clasificar la subregión en alguna categoría.

En la Ilustración 9 se puede observar un conjunto de características Haar para 3 tipos de características: siendo A y B dos rectángulos, C y D tres rectángulos y la última que es E cuatro rectángulos. Para el caso de los dos rectángulos su valor calculado será la diferencia entre la sumatoria de los pixeles que están dentro de las dos regiones rectangulares, para ello deberán tener mismo tamaño y forma. En tanto que para el de tres rectángulos su cálculo será la suma dentro de los dos rectángulos que están en el exterior menos el que se encuentra en el centro. Y finalmente para el caso de los cuatro rectángulos su valor será la diferencia existente entre los pares diagonales de rectángulos (Karen, 2017).

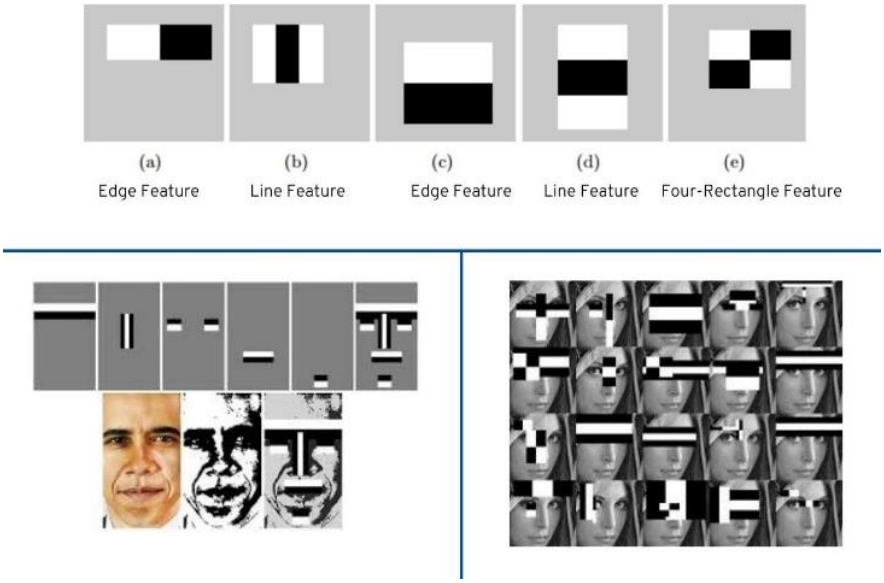


Ilustración 9. Haar Features (Rashmi, 2020).

2.3.4.2. YOLOv3

You Only Look Once es un detector de objetos muy sofisticado, de tal manera que reconoce cualquier tipo de objeto y su posición dentro de un video, imagen o hasta en tiempo real.

En la Ilustración 10 se puede ver que este detector de objetos capta de manera precisa los objetos con tan solo mirar una vez el video, de tal manera que solo necesita que la red neuronal actúe sobre el video para poder así hacer determinar qué tipo de objeto es según su predicción (Joseph Redmon, 2018).

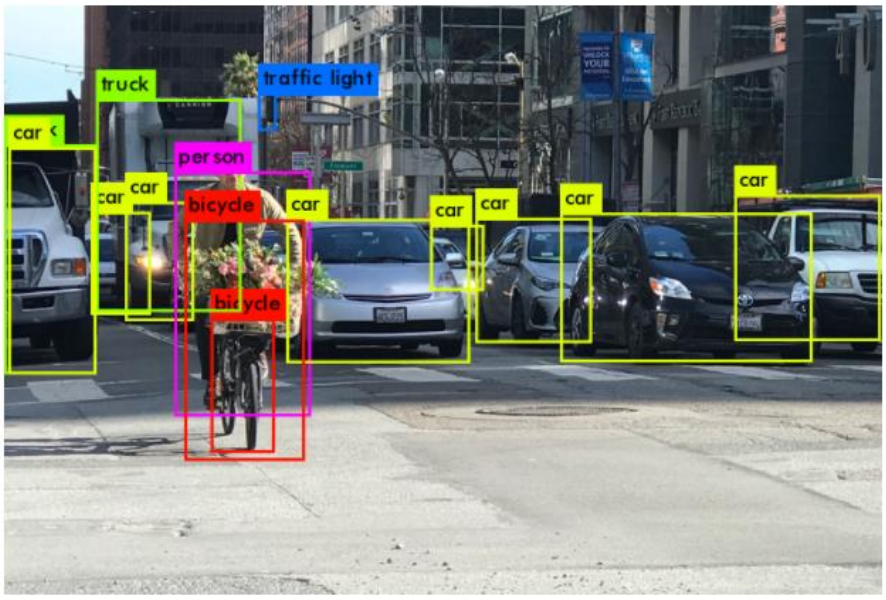


Ilustración 10. Visión por computadora YOLOV3 (Archi,2020)

2.3.4.2.1. Funcionamiento de YOLOv3

No es más que una red neuronal convolucional (CNN). Por lo tanto, son sistemas clasificadores que procesan las imágenes a la entrada como una matriz estructurada de datos logrando así identificar los distintos patrones que previamente fueron entrenados. Algo que hay que destacar de YOLOv3 es que detecta de manera rápida y es precisa en la Ilustración 11 se puede ver su funcionamiento. (David, 2020)

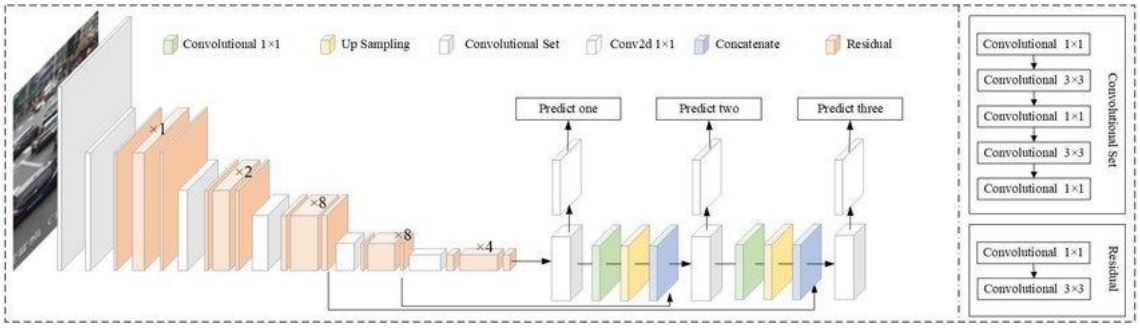


Ilustración 11. Funcionamiento de YOLOv3 (Qi-Chao Moo, Mei Sun, Yan-Bo Liu, 2019)

YOLOv3 hace que la red neuronal convolucional puntúe las diferentes regiones en función de las similitudes existentes. Por ello, la región de detección mayor es señalada como una detección positiva.

2.3.4.2.2. Arquitectura de YOLOv3

Lo primero que hace YOLO es separar una imagen en una cuadricula, de tal manera que cada cuadricula posee una celda prediciendo así distintas cantidades de cuadros

de límites en el objeto que tiene una puntuación elevada con las clases predefinidas. Hay que tomar en cuenta, cada cuadro delimitador posee una puntuación con respecto a la precisión asumiendo que la predicción detectada por un cuadro delimitador. El cuadro delimitador es generado por las agrupaciones de las dimensiones de los cuadros de verdad que no es más que datos para encontrar formas y tamaños (Kevin, 2018)

2.3.5. La segmentación de imagen.

La segmentación de imagen se encarga en dividir la imagen en diferentes regiones lógicas, a continuación, tenemos las distintas propiedades de segmentar imágenes: propiedad de color que es por la umbralización, la propiedad espacial y del dominio de degradado (hace referencia a la detección de las esquinas) (Vladislav, 2017).

2.3.6. Histograma de procesamiento de imágenes.

En el histograma se puede observar la frecuencia de los valores de intensidad en pixeles, de tal manera que el histograma llega a ser la función de densidad de probabilidad de la luminancia esto en pixeles. Mostrando que el eje x da el rango de valores de la característica, en tanto que el eje y muestra la frecuencia existente de estos valores. (Vladislav, 2017)

2.4. Procesamiento de datos.

El procesamiento de datos sirve para poder descubrir patrones y dependencias, para poder así extraer información útil de datos que no sean estructurados ni complejos. Estos datos se pueden analizar desde diferentes perspectivas generando información útil. Los problemas más comunes de este procesamiento son a clasificación y la regresión (Alvear Vanessa, Rosero Paul, Peluffo Diego, 2017)

2.5. Lenguaje de programación para el uso de la inteligencia artificial.

2.5.1. Python

Actualmente la inteligencia artificial crece de una manera exponencial, de tal manera que Python se ha convertido en un lenguaje de programación único, ya sea por su versatilidad o legibilidad, siendo este muy útil y comprensible al momento de ser usado (Pérez Ivet, Diaz Yanet, 2014)

2.5.1.1. Ventajas de Python

En si la inteligencia artificial y el machine learning se han desarrollado a tal magnitud que en el mercado es necesario contar con un grupo de programadores que tengan la capacidad de dar soluciones rápidas a inconvenientes, por ello la forma más efectiva de logarlo es haciendo uso de Python. Por ello se enlista las siguientes ventajas (Asler, 2020)

- **Sencillo y de fácil aprendizaje.**

Cualquier persona que sepa programar independientemente de la experiencia que este pueda poseer, Python ha demostrado ser un lenguaje de programación accesible. De tal manera que se puede aprenden de manera sencilla y rápida.

- **Open source**

Este lenguaje de programación es de código abierto, esto quiere decir que es netamente transparente para que cualquier persona pueda hacer uso del mismo. No obstante, a ello tenemos también que Python permite elaborar mejoras o también corregir cualquier tipo de error.

- **Engloba a una comunidad extensa**

Esta es una de las ventajas más relevantes ya que este lenguaje de programación tiene una comunidad la cual siempre está dispuesta a ayudar sobre cualquier duda que se tenga, todo esto se debe a su popularidad. Siendo métodos de ayuda para pedir consejos basados y resolver cualquier tipo de problema con mayor seguridad.

- **Variedad en librerías**

Actualmente Python posee una gran variedad de librerías, esto gracias a ser un programa de código abierto, de tal manera que estas librerías se las puede encontrar en línea de manera gratuita.

- **Flexible y versátil**

Básicamente esta ventaja se concentra en:

Combinación. Se puede combinar con cualquier otro tipo de lenguaje de programación.

Elección. Se puede elegir ya sea programación orientada a objetos o scripting.

Visualización. Se puede verificar los cambios que se hayan desarrollado en ese instante, sin que exista una combinación con el código fuente

Personalización. Se puede tener diferentes tipos de programación, siendo eficaz al resolver alguna situación de problema.

- **Presentación de datos**

La capacidad de Python para dar respuesta a los problemas es ajustable a cualquier proyecto, puesto que sus herramientas donde se visualiza proponen un análisis de datos amplio determinado así los diferentes tipos de errores (Asler, 2020).

2.5.2. Librerías de Python

Para entender un poco más sobre Python debemos entender que son las librerías de programación, de tal manera que está definida como el conjunto de implementaciones con distinta funcionalidad. Entonces se puede decir que tiene una finalidad especifica pudiendo así ser llamada.

2.5.2.1. OpenCV

En el mundo de la visión artificial se está dando un giro muy importante al análisis y tratamiento de imágenes, de tal manera que Python ha creado una librería muy importante que sirve para la detección de objetos y a su vez detección de rostros como se puede ver en la Ilustración 12 (Beyeler Michael, 2017).

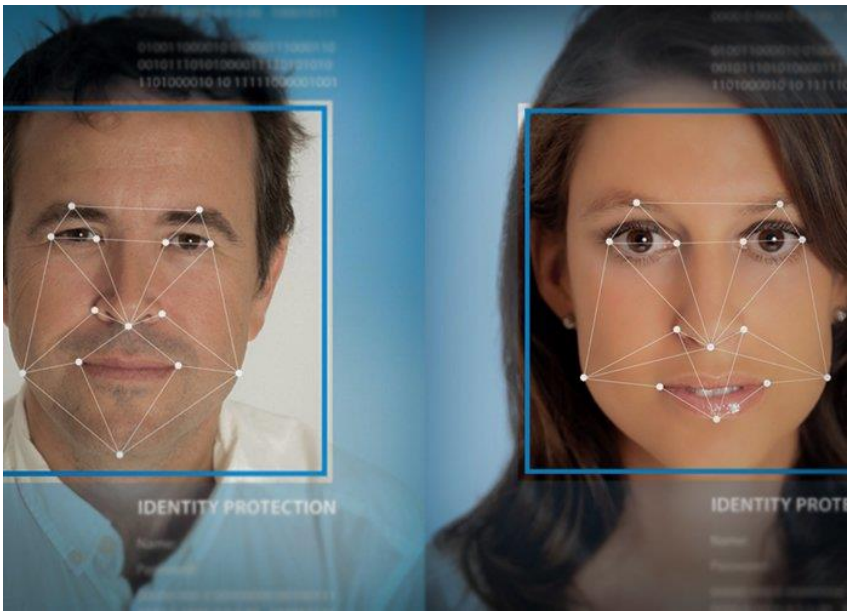


Ilustración 12. Reconocimiento facial de imágenes usando Open-Cv Python (Eduanix, 2018)

Es necesario mencionar que OpenCv para Python es de código abierto y se usa para distintas actividades:

- Para la lectura y escritura de imágenes
- Detectar las características de los rostros
- Detectar formas
- Cambiar la calidad y el color de la imagen

Es importante saber que esta librería es fácil de aprender por lo existen muchos tutoriales, además de que funciona con todo idioma y es gratuito

2.5.2.1.1. Imágenes como una matriz

Como se mencionó anteriormente una imagen es una matriz que tiene pixeles de puntos de datos, de tal forma que mientras más números de pixeles contenga una imagen, su resolución será mejor.

Los pixeles son bloques pequeños de información que viene definido por 2 dimensiones en tanto que su profundidad está definida por el color que esta tenga.

Para que una imagen sea procesada, esta deberá convertirse a la forma binaria. A continuación, se presenta la ecuación para calcular el color de la imagen:

$$numero\ de\ color\ /sombra = 2^{bpp}$$

En donde que Bpp es los bits por los pixeles, entonces mientras más bits/pixel se tenga más color de imagen se obtendrá (Kaehler Adrian, 2017).

2.5.2.2. NumPy

Esta librería se enfoca en el cálculo numérico y sirve también para el análisis de datos.

La característica de esta librería es que integra una clase de objetos denominados array, siendo los que permiten representar una infinidad de colecciones de datos del mismo tipo en distintas dimensiones (Higuera Clara, 2015).

2.5.2.2.1. Clase de objetos array

El array es una estructura de datos que está organizado ya sea en forma de cuadriculas de distintas dimensiones o también en forma de una tabla como se ve en la Ilustración 13. Un array en cambio por sus dimensiones se le puede conocer como eje.

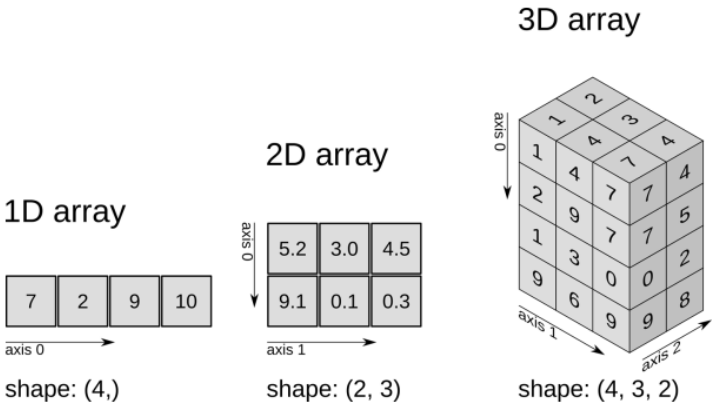


Ilustración 13. Clase de objeto array (Alfredo, 2020)

Se debe tomar en cuenta que el vector no es más que una lista de valores creados en un array de una sola dimensión, en tanto que la matriz no es más que una lista de valores creados en un array que tiene dos dimensiones.

2.5.2.3. Pandas

Esta librería se enfoca en el manejo de datos y al análisis de sus estructuras. Las características de esta librería son:

- Determina estructuras nuevas de datos que están en los arrays de NumPy pero toman nuevas funcionalidades.
- Lee y escribe ficheros en varios formatos como Excel o CSV.
- Aprueba el acceso a los distintos datos por medio de índices o bien sea nombre puede ser por la fila o por columna.
- Brinda métodos para poder ordenar, combinar y dividir los conjuntos de datos.
- Nos da la facilidad de trabajar con series temporales.
- Ejecuta toda operación de manera eficiente (Meher, 2017).

2.5.2.3.1. Tipos de datos pandas

Series. - son de una sola dimensión que tienen arrays, se generan a partir de listas.
DataFrame. – son datos que forman una estructura de dos dimensiones.
Panel. – hace que podamos trabajar con más de dos dimensiones.

2.5.2.4. TensorFlow

TensorFlow es una biblioteca enfocada para machine intelligence, como bien se sabe TensorFlow es de código abierto que hace uso de gráficos de flujo de datos. Las operaciones matemáticas están representadas por nodos, en tanto que el borde representa la matriz de datos que son multidimensionales.

Su característica principal es que está destinado para entrenar y construir redes neuronales, de tal manera que sean capaces de detectar, lograr descifrar patrones y además de correlaciones.

Esta biblioteca en el campo de procesamiento de imágenes, usa una red convolucional de tal manera que sea capaz de descubrir y enriquecer los patrones en imágenes (David, 2020).

CAPÍTULO 3

3. DESARROLLO

3.1. Levantamiento del parking de Posgrados UCACUE

El departamento de posgrados de la Universidad Católica de Cuenca está constituido por diferentes departamentos, salón de eventos, garajes, entre otros. De tal manera, que en este proyecto se tomó el garaje principal como referencia para determinación de posición de cámaras, toma de puntos de conectividad y tomas de energía.

La distancia existente del parking en cuanto al largo es de un valor de 39,46 metros y con referencia al ancho del parking es 25,84 metros como se puede ver en la Ilustración 14 realizada desde un punto de vista superior.

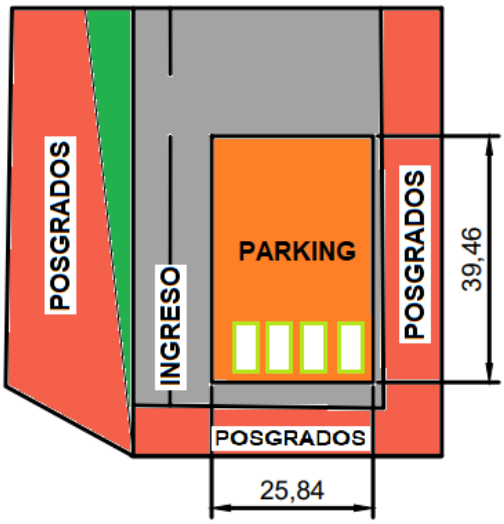


Ilustración 14. Distancia de largo y ancho del parking (Autor)

3.1.1. Arquitectura del sistema de parking

La arquitectura del sistema de parking ha sido diseñada para que sea sencilla y de bajo costo, con materiales de fácil adquisición en el medio local. Este consta de una cámara y un grabador DVR, este último conectado a un punto de acceso; el mismo que permite el acceso remoto a la cámara, con lo cual con estas imágenes se puede detectar las distintas plazas seleccionadas del parking tal como se puede ver en la Ilustración 15.

Ilustración 15. Arquitectura del sistema de parking (Autor)

A continuación, se detalla cada uno de los equipos utilizados:

Cámara tipo tubo hikvision turbo hd 720p.- este tipo de cámaras ha revolucionado el mercado, puesto que su nivel de resolución es muy alto, de tal manera que se usan en diversos ámbitos, ya sea para seguridad o para la detección de objetos; esto por las diferentes funciones inteligentes que posee, en la Ilustración 16 se puede ver su parte física.

A continuación, se detalla las diferentes características:

- Con tecnología HD-TVI/AHD/CVI.
- Posee una tecnología Smart IR detectando hasta 20 m.
- Tiene una resolución en alta definición HD 720P (1280 píxeles de ancho por 720 píxeles de alto).
- Tiene un sistema de codificación y transmisión NTSC: 1280 (H) x 720 (V) pixeles.
- Lente con una distancia focal de 2.8 mm.
- Tiene un ángulo de visión de 99.72º.
- Funciona con una alimentación de 12Vdc.
- Consume una potencia de 3.5watts (HIKVISION, 2019).

Ilustración 16. Cámara tipo tubo hikvision turbo hd 720p (WORD COMPUTERS , 2019)

DVR de 4 canales HD720p hikvision. - estos sistemas de administración de señales de video y grabación es un grabador con capacidad para conectar hasta 4 cámaras analógicas HD. Cuenta con tecnología para manejo simultaneo de audio y video; además, cuenta con capacidad de almacenamiento como se puede ver en la Ilustración 17. Uno de los beneficios de este grabador es que puede generar un sistema integral.

A continuación, se detalla las características de este grabador:

- Tiene una entrada de hasta 4 canales, con lo cual sirve para 4 cámaras como máximo.
- Posee una resolución en HD720P (1280x720).
- La capacidad del disco duro es de 6 TB.
- Cuenta con conexión VGA y HDMI.
- Incluye 2 puertos USB 2.0, RJ45 y audio.
- Esta alimentada a 12Vcd.
- El consumo sin disco duro es de 8W.
- Puede activar la grabación mediante el movimiento (HIKVISION, 2019).

Ilustración 17. DVR de 4 canales HD720p Hikvision (WORD COMPUTERS , 2019)

3.2. Ubicación e instalación de los equipos

3.2.1. Ubicación de la cámara tipo tubo Hikvision Turbo HD 720p.

Para la ubicación de las cámaras se tomó en cuenta la posición de las líneas de parqueo del parking, para así tener una referencia y ubicar de la mejor manera la cámara.

Si bien es cierto el parking tiene una forma rectangular bien amplia, de tal manera que para experimentación del proyecto se tomó como referencia una sección determinada.

Siendo así la cámara a utilizar fue instalada en la Jefatura de Posgrados. En la parte superior de las clínicas Odontológicas, a una distancia media del ancho del parking de 12,70 m, aproximadamente y a una altura aproximada de 9 m como se puede ver en la Imagen 1.

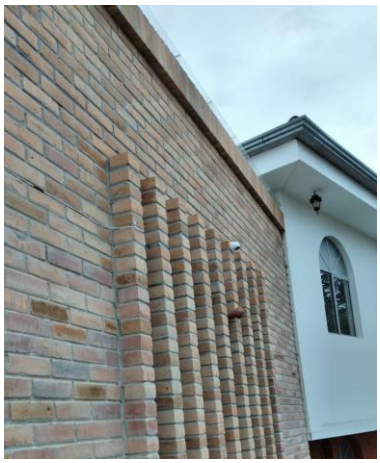


Imagen 1. Ubicación de la cámara tipo tubo hikvision turbo hd 720p (Autor)

3.2.2. Instalación del DVR de 4 canales HD720p hikvision

Para la ubicación e instalación del DVR se tomó en cuenta un lugar donde se pueda obtener un punto de acceso a internet y un punto de energía (tomacorriente) para su alimentación. De tal manera, que el DVR quedo a una distancia de 4 metros de la cámara tal como se puede ver en la Imagen 2.

(a) (b)

Imagen 2. (a) Ubicación (b) instalación del DVR de 4 canales HD720p hikvision (autor)

3.3. Algoritmos de detección

A continuación, se especificará tres distintos algoritmos de detección que se tomaron en cuenta para la ejecución el proyecto.

3.3.1. Aggregate Channel Features

Este algoritmo se encarga de ejecutar una búsqueda profunda, haciendo uso de un modelo de objeto holístico, para ello se calculan distintos canales $C = \mathbf{\Omega}(\mathbf{I})$, donde I será la entrada de una imagen, la misma que se usará para elaborar un sub muestreado de la sumatoria de todos los bloques de los pixeles de los canales. De tal manera que, por el impulso, este se entrena y realiza una combinación para así distinguir en el fondo el

objeto existente, generalmente se utiliza cuando existen ventanas deslizantes multi escala.

3.3.2. Deformable Parts Model

Es un sistema de detección de objetos que usa mezclas multi escala de distintos modelos de partes deformables. Entonces este tipo de modelos serán entrenados realizando un proceso discriminante haciendo uso de cajas que rodeen a los objetos que se encuentren en las imágenes. Una de sus debilidades son su bajo rendimiento.

3.3.3. FASTER R-CNN

Este detector se basa en las redes convolucionales, de tal manera que realizan un aprendizaje para extraer y poder seleccionar las características con las que se puede detectar los objetos.

Fundamentalmente este algoritmo se maneja por 4 etapas; la primera capta la imagen, la segunda determina las regiones de interés, la tercera adquiere el vector de características que tiene longitud fija para cada región, finalmente la última etapa consiste en un conjunto de clases lineales.

Si bien es cierto, el tamaño del vector siempre será fijo, por ello los vectores se intercalarán uno después de otro creando así una secuencia para la detección.

Este algoritmo de detección se compone de dos módulos; la red convolucional pura la misma que procesa las distintas regiones y el segundo que es un detector R-CNN.

3.4. Transferencia de información

Para la transferencia de información, se la realiza con la obtención de video mediante la cámara esta a su vez adquiere la información de la cantidad de sitios de parking existentes, ya sea ocupados o vacíos. Una vez que esta cámara adquiere dicha información (ver Ilustración 18), esta pasa a ser procesada en el algoritmo y comienza a ejecutar la detección de los sitios de parking.

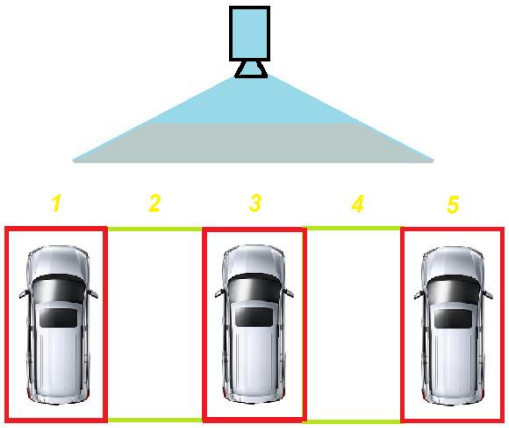


Ilustración 18. Transferencia de información

3.5. Desarrollo Experimental

3.5.1. Prueba 1. Detección de objetos haciendo uso de Raspberry Pi y TensorFlow 2 (YOLO v3)

La detección de objetos haciendo uso de la Raspberrry Pi y TensorFlow2 realiza una clasificación de los objetos visualizados en un video en tiempo real. Por ende, el primer paso a realizar una actualización y una configuración para asi obtener una detección eficiente.

La Raspberry a utilizar es un modelo de mini ordenador de placa única y creada con el propósito de promover y asistir en la tanto en la informática como en la programación. El sistema operativo base de este miniordenador es Raspbian, una derivación de Linux. Esto genera accesos ilimitados a cualquier tipo de descargas y manejar softwares libres.

3.5.1.1. Diagrama de bloques de la detección de objetos haciendo uso de Raspberry Pi y TensorFlow

Para la detección de objetos mediante la Raspberry Pi y TensorFlow se ha creado un diagrama de bloques dividido en 4 fases como se puede ver en la Ilustración 19, cada una de estas fases cumple una función específica. Se debe considerar que cada fase contiene subprocesos, los mismos que ejercen un papel fundamental para la detección.

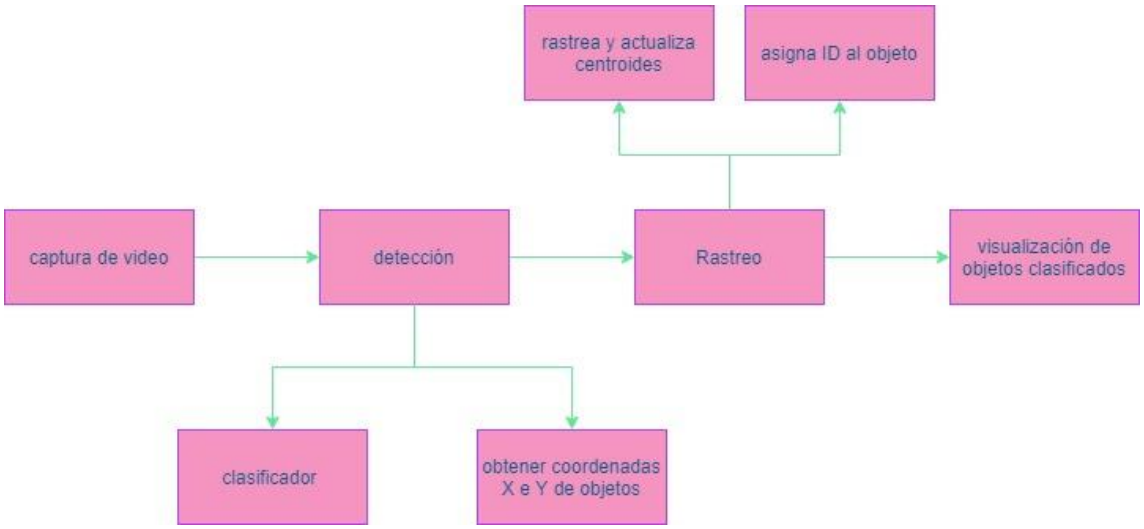


Ilustración 19.Diagrama de bloques de la detección de objetos haciendo uso de Raspberry Pi y TensorFlow (Autor)

3.5.1.2. Captura de imágenes de la detección de objetos haciendo uso de Raspberry Pi y TensorFlow

La función principal de la captura de imágenes es obtener el flujo del video, para eso se utiliza un video captado en tiempo real que se obtiene desde la, esta cámara deberá estar conectada a la Raspberry Pi.

La librería que se utiliza para capturar el video será la de OpenCv con una línea de código con modulo VideoCapture. El objetivo principal de este módulo es procesar y decodificar en forma secuencial; esto quiere decir que captura la imagen, luego se procesa y finalmente se decodifica.

YOLOv3 para su lectura y decodificación realiza los siguientes pasos: redimensionar el video en su tamaño y transponer de manera interna la matriz a cada imagen de HWC (sus siglas hacen referencia a: H=altura (Height), W=ancho (Width) y C= color channel), es importante mencionar que el color del canal viene dado en RGB.

3.5.1.3. Detección y clasificación de la detección de objetos haciendo uso de Raspberry Pi y TensorFlow

Para la detección y clasificación de los objetos, una vez que pasa por la captura de las imágenes previamente indicado, se procede hacia la red neuronal, obteniendo así la clase de objeto detectado.

3.5.1.3.1. Detalle de la detección

Para efectuar la detección de objetos con la Raspberry se siguió los siguientes pasos:

a. **Configuración de la Raspberry**. - para ello se debe instalar el sistema operativo de la Raspberry en una micro SD para este proyecto se usa una SD con capacidad de 8 Gigabytes tal como se puede ver en la Ilustración 20.

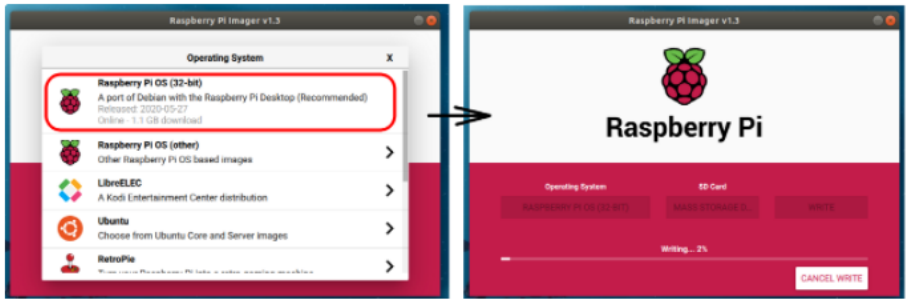


Ilustración 20. Configuración de la Raspberry Pi (Autor)

b. **Actualización de la Raspberry**. –Una vez instalado el sistema operativo Raspbian se da paso a la actualización, la misma que se logra desde una terminal ingresando el comando apt-get, como se observa en la Ilustración 21:

```
ronald@ronald-VirtualBox:~$ sudo apt-get dist-upgrade
ronald@ronald-VirtualBox:~$ sudo apt-get update
[sudo] contraseña para ronald:
```

Ilustración 21. Actualización de la Raspberry (Autor)

c. **Instalación de paquetes y bibliotecas**. - En este caso, se necesita Python 3, para la instalación del mismo solamente se ingresa mediante el comando apt, como se puede observar en la Ilustración 22.

```
ronald@ronald-VirtualBox:~$ sudo apt install python3-pip
```

Ilustración 22. Instalación de Python 3 (Autor)

Cabe mencionar que se debe instalar una variadad de paquetes, por lo tanto, en la terminal se ingresa varias líneas de código especificado en la Ilustración 23 .

```
ronald@ronald-VirtualBox:~$ sudo apt-get install -y libhdf5-dev libc-ares-dev li
beigen3-dev gcc gfortran python-dev libgfortran5 \
>                         libatlas3-base libatlas-base-dev libopenblas-dev lib
openblas-base libblas-dev \
>                         liblapack-dev cython libatlas-base-dev openmpi-bin l
ibopenmpi-dev python3-dev python3-venv
```

Ilustración 23 Instalación de paquetes (Autor)

Instalación de OpenCv. – para la instalación de la librería OpenCv solamente se debe ingresar el siguiente código, como se ve en la Ilustración 24.

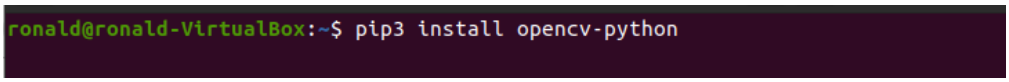


Ilustración 24. Instalación de Opencv (Autor)

Existen muchas dependencias para que funcione OpenCv con Python de tal manera que se debe ejecutar lo siguiente (ver en la Ilustración 25) para que las dependencias se instalen.

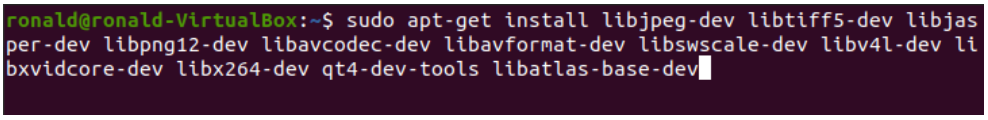


Ilustración 25. Dependencias de Opencv (Autor)

Instalación de TensorFlow2. – Para la instalación de esta librería, se debe tomar en cuenta la actualización y sus dependencias, puesto que si solo se instala con el comando pip3 se tiene una serie de problemas. Entonces, es recomendable instalar como se ve en la Ilustración 26.

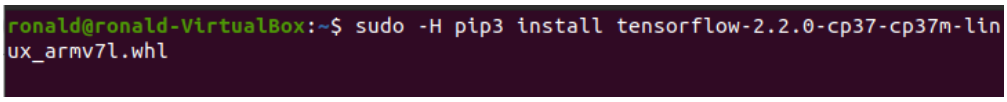


Ilustración 26. Instalación de TensorFlow 2 (Autor)

d. **Configuración de YOLOv3**. – YOLOv3 es un sistema de reconocimiento de patrones que permite entrenar un modelo, si bien es cierto es un entrenador que esta creado de tal manera que se lo debe clonar, para ello se debe hacer desde una terminal introduciendo el siguiente código como se ve en la Ilustración 27.

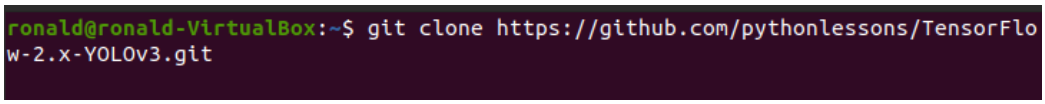
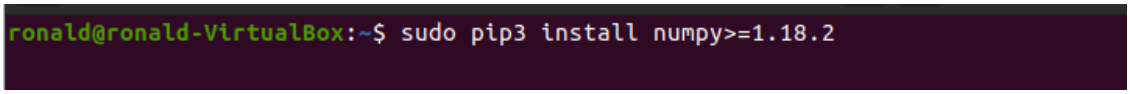
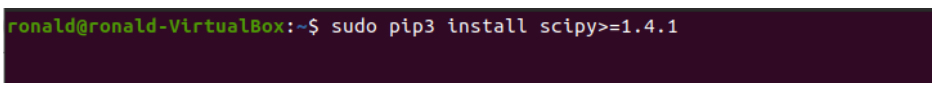
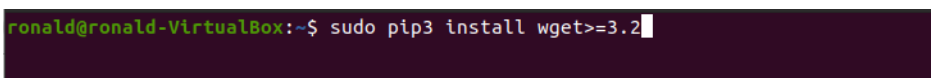


Ilustración 27. Clonación de YOLOv3 (Autor)

Al estar clonando la carpeta YOLOv3 debemos instalar unos requisitos para que este pueda funcionar, estos requisitos son varias bibliotecas (ver Ilustración 28) necesarias para que YOLOv3 funcione.







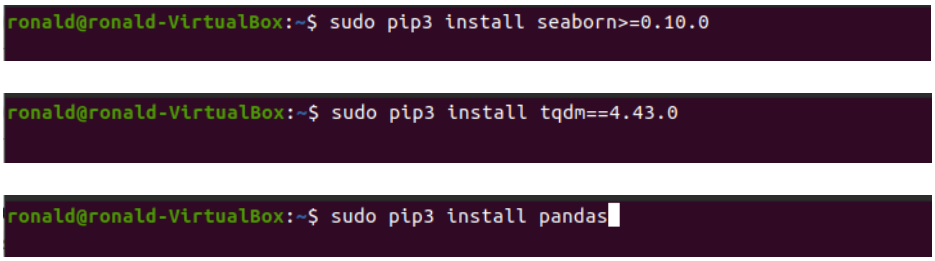


Ilustración 28. Bibliotecas necesarias para que YOLOv3 ejecute la detección (Autor)

Un punto importante que se debe tomar en cuenta es que al usar la Raspberry Pi esta tiene un procesador bajo, por eso se debe ejecutar un Tiny, que es una versión ligera del entrenador, para ello se debe ingresar el código de línea reflejado en la Ilustración 29.

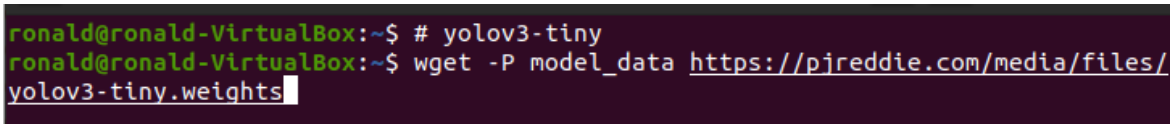


Ilustración 29. YOLOv3 Tiny (Autor)

e. **Configuración de la cámara (Raspberry).** – para nuestro caso se utiliza una cámara externa, una cámara conectada al puerto USB, pero si hacemos uso de la Raspberry se debe considerar que es necesario habilitar dentro de la configuración de la BIOS, ya que por default viene deshabilitada.

3.5.1.4. Rastreo

La parte de rastreo es una parte fundamental del proyecto como tal, ya que permite identificar tanto a los vehículos como a las personas.

3.5.1.5. Visualización de objetos clasificados

Esta fase es la encargada de imprimir en pantalla los resultados, es decir al video captado detectara que clase de objeto es, pero en pantalla. No cabe duda que pueden existir tanto falsos positivos como falsos negativos, o también puede ser posible que funcione de mejor manera con la captura de la cámara desde una mejor posición. En realdad cada detalle va a influenciar a la eficiencia de la detección de los objetos.

3.5.1.6. Algoritmo

Algo muy importante que se debe tener presente es que este algoritmo es previamente entrenado. Ahora si bien es cierto, anteriormente se habló sobre las 4 fases para realizar la detección de objetos, pero se debe tomar en cuenta que esas fases contienen subprocesos; estos subprocesos se encuentran detallados a continuación.

Como se ve en la Ilustración 30 lo primero que se hace es importar las librerías a utilizar ya sean librerías de programación abierta y personalizadas.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, LeakyReLU, ZeroPadding2D, BatchNormalization, MaxPool2D
from tensorflow.keras.regularizers import l2
from yolov3.utils import read_class_names
from yolov3.configs import *
```

Ilustración 30. Librería de uso para detección de objetos (Autor)

La siguiente parte de la programación es la construcción de capas mediante normalización por lotes (ver Ilustración 31), en realidad esto no es tan común, pero si se debe utilizar la API dada por TensorFlow. Momento después de una imagen de colores ingrese a la estructura de red convolucional, la forma que tiene esta cambiara, en definitiva, se supone que utiliza un aproximado de 32 núcleos convolucionales.

```
class BatchNormalization(BatchNormalization):
    # "Estado congelado" y "modo de inferencia" son dos conceptos separados.
    # `layer.trainable = False` es congelar la capa, por lo que la capa usará
    # almacenados moviendo `var` y` mean` en el "modo de inferencia", y ambos `gama`
    # y `beta` no se actualizarán !
    def call(self, x, training=False):
        if not training:
            training = tf.constant(False)
        training = tf.logical_and(training, self.trainable)
        return super().call(x, training)

def convolutional(input_layer, filters_shape, downsample=False, activate=True, bn=True):
    if downsample:
        input_layer = ZeroPadding2D(((1, 0), (1, 0)))(input_layer)
        padding = 'valid'
        strides = 2
    else:
        strides = 1
        padding = 'same'

    conv = Conv2D(filters=filters_shape[-1], kernel_size = filters_shape[0], strides=strides,
                  padding=padding, use_bias=not bn, kernel_regularizer=l2(0.0005),
                  kernel_initializer=tf.random_normal_initializer(stddev=0.01),
                  bias_initializer=tf.constant_initializer(0.))(input_layer)
    if bn:
        conv = BatchNormalization()(conv)
    if activate == True:
        conv = LeakyReLU(alpha=0.1)(conv)

    return conv
```

Ilustración 31. Normalización por lotes (Autor)

Lo siguiente del algoritmo es el módulo residual (ver Ilustración 32), este a su vez tiende a hacer uso de un mecanismo para conseguir un atajo con el fin de moderar el problema causante por la desaparición del gradiente provocado por el incremento de la profundidad de la red neuronal. El método que se usa es el mapeo de identidad, estableciendo así un canal el cual se encargue de correlacionar la entrada y la salida, de tal forma que la red neuronal adquiera información restante entre la entrada y la salida.

```
def residual_block(input_layer, input_channel, filter_num1, filter_num2):
    short_cut = input_layer
    conv = convolutional(input_layer, filters_shape=(1, 1, input_channel, filter_num1))
    conv = convolutional(conv       , filters_shape=(3, 3, filter_num1,   filter_num2))

    residual_output = short_cut + conv
    return residual_output
```

Ilustración 32. Modulo residual (Autor)

A continuación, en la Ilustración 33 se puede observar que el algoritmo hace uso de un muestreo ascendente de los dos mapas de características. La operación que realizan es multiplicar el ancho y la altura del mapa de características por dos, obteniendo así un mapa de característica totalmente nuevo.

```
def upsample(input_layer):
    return tf.image.resize(input_layer, (input_layer.shape[1] * 2, input_layer.shape[2] * 2), method='nearest')
```

Ilustración 33. Muestreo ascendente (Autor)

Parte muy importante del algoritmo es la estructura darknet53; que no es más que una librería de redes neuronales ya creada; a continuación, en la Ilustración 34 se ve la estructura.

```
def darknet53(input_data):
    input_data = convolutional(input_data, (3, 3,  3,  32))
    input_data = convolutional(input_data, (3, 3, 32,  64), downsample=True)

    for i in range(1):
        input_data = residual_block(input_data,  64,  32, 64)

    input_data = convolutional(input_data, (3, 3,  64, 128), downsample=True)

    for i in range(2):
        input_data = residual_block(input_data, 128,  64, 128)

    input_data = convolutional(input_data, (3, 3, 128, 256), downsample=True)

    for i in range(8):
        input_data = residual_block(input_data, 256, 128, 256)

    route_1 = input_data
    input_data = convolutional(input_data, (3, 3, 256, 512), downsample=True)

    for i in range(8):
        input_data = residual_block(input_data, 512, 256, 512)

    route_2 = input_data
    input_data = convolutional(input_data, (3, 3, 512, 1024), downsample=True)

    for i in range(4):
        input_data = residual_block(input_data, 1024, 512, 1024)

    return route_1, route_2, input_data
```

Ilustración 34. Estructura darnet53 (Autor)

Para la detección de objetos con YOLOv3, si bien es cierto usa capas convolucionales, de tal manera que se logra convertir en una red netamente convolucional. En la Ilustración 35, se puede ver las líneas de código utilizadas.

```
def YOLOv3(input_layer, NUM_CLASS):
    # Después de que la capa de entrada ingresa a la red Darknet-53, obtenemos tres ramas
    route_1, route_2, conv = darknet53(input_layer)
    # Vea el módulo naranja (DBL) en la figura anterior, un total de 5 operaciones de subconvolución
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv = convolutional(conv, (3, 3,  512, 1024))
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv = convolutional(conv, (3, 3,  512, 1024))
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv_lobj_branch = convolutional(conv, (3, 3, 512, 1024))

   # conv_lbbox se usa para predecir objetos de gran tamaño, Shape = [None, 13, 13, 255]
    conv_lbbox = convolutional(conv_lobj_branch, (1, 1, 1024, 3*(NUM_CLASS + 5)), activate=False, bn=False)

    conv = convolutional(conv, (1, 1,  512,  256))
    # upsample aquí usa el método de interpolación de vecino más cercano, que tiene la ventaja de que
    # El proceso de muestreo superior no necesita aprender, lo que reduce el parámetro de red
    conv = upsample(conv)

    conv = tf.concat([conv, route_2], axis=-1)
    conv = convolutional(conv, (1, 1, 768, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv_mobj_branch = convolutional(conv, (3, 3, 256, 512))

    # conv_mbbox se usa para predecir objetos de tamaño mediano, shape = [None, 26, 26, 255]
    conv_mbbox = convolutional(conv_mobj_branch, (1, 1, 512, 3*(NUM_CLASS + 5)), activate=False, bn=False)

    conv = convolutional(conv, (1, 1, 256, 128))
    conv = upsample(conv)

    conv = tf.concat([conv, route_1], axis=-1)
    conv = convolutional(conv, (1, 1, 384, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv_sobj_branch = convolutional(conv, (3, 3, 128, 256))

    # conv_sbbox se usa para predecir objetos de tamaño pequeño, shape = [None, 52, 52, 255]
    conv_sbbox = convolutional(conv_sobj_branch, (1, 1, 256, 3*(NUM_CLASS +5)), activate=False, bn=False)

    return [conv_sbbox, conv_mbbox, conv_lbbox]
```

Ilustración 35. Uso de capas convolucionales (Autor)

YOLOv3 Tiny se encarga de detectar los objetos en tiempo real, esta red utiliza el muestreo superior para así poder extraer las distintas características. Una vez que se adquiere la imagen, YOLOv3 Tiny logra dividir las imágenes en cuadriculas de N x N además de predecir los cuadros delimitadores que se encuentran dentro de cada celda de la cuadricula, generando así la detección del objeto. En la Ilustración 36, se puede observar las líneas de código utilizadas para logar lo mencionado.

```
def YOLOv3_tiny(input_layer, NUM_CLASS):
   # Después de que la capa de entrada ingresa a la red Darknet-53, obtenemos tres ramas
    route_1, conv = darknet19_tiny(input_layer)

    conv = convolutional(conv, (1, 1, 1024, 256))
    conv_lobj_branch = convolutional(conv, (3, 3, 256, 512))

    # conv_lbbox se usa para predecir objetos de gran tamaño, Shape = [None, 26, 26, 255]
    conv_lbbox = convolutional(conv_lobj_branch, (1, 1, 512, 3*(NUM_CLASS + 5)), activate=False, bn=False)

    conv = convolutional(conv, (1, 1, 256, 128))
    # upsample aquí usa el método de interpolación de vecino más cercano, que tiene la ventaja de que
    # El proceso de muestreo superior no necesita aprender, lo que reduce el parámetro de red
    conv = upsample(conv)

    conv = tf.concat([conv, route_1], axis=-1)
    conv_mobj_branch = convolutional(conv, (3, 3, 128, 256))

    # conv_mbbox se usa para predecir objetos de tamaño mediano, forma = [None, 13, 13, 255]
    conv_mbbox = convolutional(conv_mobj_branch, (1, 1, 256, 3 * (NUM_CLASS + 5)), activate=False, bn=False)

    return [conv_mbbox, conv_lbbox]
```

```
def Create_Yolov3(input_size=416, channels=3, training=False, CLASSES=YOLO_COCO_CLASSES):
    NUM_CLASS = len(read_class_names(CLASSES))
    input_layer  = Input([input_size, input_size, channels])

    if TRAIN_YOLO_TINY:
        conv_tensors = YOLOv3_tiny(input_layer, NUM_CLASS)
    else:
        conv_tensors = YOLOv3(input_layer, NUM_CLASS)

    output_tensors = []
    for i, conv_tensor in enumerate(conv_tensors):
        pred_tensor = decode(conv_tensor, NUM_CLASS, i)
        if training: output_tensors.append(conv_tensor)
        output_tensors.append(pred_tensor)

    YoloV3 = tf.keras.Model(input_layer, output_tensors)
    return YoloV3
```

Ilustración 36. Red neuronal YOLOv3 Tiny (Autor)

La siguiente parte del algoritmo es el procesamiento de decodificación, para ello se debe tener presente que las capas de salida van a pronosticar las coordenadas del cuadro delimitador y las dimensiones, por ello estos atributos serán la respuesta de varias capas de la convolución de la red. A continuación, en la Ilustración 37 se presenta las líneas de código a utilizar.

```
def decode(conv_output, NUM_CLASS, i=0):
    # donde i = 0, 1 o 2 para corresponder a las tres escalas de la cuadrícula
    conv_shape       = tf.shape(conv_output)
    batch_size       = conv_shape[0]
    output_size      = conv_shape[1]

    conv_output = tf.reshape(conv_output, (batch_size, output_size, output_size, 3, 5 + NUM_CLASS))
    conv_raw_dxdy = conv_output[:, :, :, :, 0:2] # desplazamiento de la posición central
    conv_raw_dwdh = conv_output[:, :, :, :, 2:4] # Desplazamiento de la longitud y el ancho del cuadro de predicción
    conv_raw_conf = conv_output[:, :, :, :, 4:5] # confianza del cuadro de predicción
    conv_raw_prob = conv_output[:, :, :, :, 5: ]# categoría de probabilidad del cuadro de predicción


   # siguiente necesidad Dibuja la cuadrícula. Donde output_size es igual a 13, 26 o 52
    y = tf.range(output_size, dtype=tf.int32)
    y = tf.expand_dims(y, -1)
    y = tf.tile(y, [1, output_size])
    x = tf.range(output_size,dtype=tf.int32)
    x = tf.expand_dims(x, 0)
    x = tf.tile(x, [output_size, 1])
    xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :, tf.newaxis]], axis=-1)
    xy_grid = tf.tile(xy_grid[tf.newaxis, :, :, tf.newaxis, :], [batch_size, 1, 1, 3, 1])
    xy_grid = tf.cast(xy_grid, tf.float32)

   # Calcule la posición central del cuadro de predicción:
    pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) * STRIDES[i]

    # Calcule la longitud y el ancho del cuadro de predicción:
    pred_wh = (tf.exp(conv_raw_dwdh) * ANCHORS[i]) * STRIDES[i]
    pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
    pred_conf = tf.sigmoid(conv_raw_conf) # el cuadro de objeto calcula la confianza prevista
    pred_prob = tf.sigmoid(conv_raw_prob) # cálculo del objeto de cuadro de categoría de probabilidad predicha
   # cálculo del objeto de cuadro de categoría de probabilidad predicha
    return tf.concat([pred_xywh, pred_conf, pred_prob], axis=-1)
```

Ilustración 37. Procesamiento de decodificación (Autor)

A continuación, como se puede ver en la Ilustración 38 en el algoritmo se implementa las medidas de precisión en la detección de objetos (IOU), de tal manera que esta función se utiliza para poder determinar los verdaderos falsos positivos y negativos. Esta función IOU al ser utilizada deberá evaluar toda la métrica eligiendo así el umbral de precisión, un punto a tomar en cuenta es que cualquier precisión de localización que sobrepase a la del lumbral siempre se tratara por igual.

```
def bbox_iou(boxes1, boxes2):
    boxes1_area = boxes1[..., 2] * boxes1[..., 3]
    boxes2_area = boxes2[..., 2] * boxes2[..., 3]

    boxes1 = tf.concat([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                        boxes1[..., :2] + boxes1[..., 2:] * 0.5], axis=-1)
    boxes2 = tf.concat([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                        boxes2[..., :2] + boxes2[..., 2:] * 0.5], axis=-1)

    left_up = tf.maximum(boxes1[..., :2], boxes2[..., :2])
    right_down = tf.minimum(boxes1[..., 2:], boxes2[..., 2:])

    inter_section = tf.maximum(right_down - left_up, 0.0)
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area

    return 1.0 * inter_area / union_area
```

Ilustración 38. Precisión en la detección (Autor)

Como parte del proceso del algoritmo una función que se considera es GIoU cuya función es sencilla, lo único que hace es comparar las coordenadas de cada vértice haciendo uso de los máximos y mínimos entre los cuadros delimitadores. En la Ilustración 39, se puede ver las líneas de código utilizadas para esta función.

```
def bbox_giou(boxes1, boxes2):
    boxes1 = tf.concat([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                        boxes1[..., :2] + boxes1[..., 2:] * 0.5], axis=-1)
    boxes2 = tf.concat([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                        boxes2[..., :2] + boxes2[..., 2:] * 0.5], axis=-1)

    boxes1 = tf.concat([tf.minimum(boxes1[..., :2], boxes1[..., 2:]),
                        tf.maximum(boxes1[..., :2], boxes1[..., 2:])], axis=-1)
    boxes2 = tf.concat([tf.minimum(boxes2[..., :2], boxes2[..., 2:]),
                        tf.maximum(boxes2[..., :2], boxes2[..., 2:])], axis=-1)

    boxes1_area = (boxes1[..., 2] - boxes1[..., 0]) * (boxes1[..., 3] - boxes1[..., 1])
    boxes2_area = (boxes2[..., 2] - boxes2[..., 0]) * (boxes2[..., 3] - boxes2[..., 1])

    left_up = tf.maximum(boxes1[..., :2], boxes2[..., :2])
    right_down = tf.minimum(boxes1[..., 2:], boxes2[..., 2:])

    inter_section = tf.maximum(right_down - left_up, 0.0)
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area

   # Calcule el valor de iou entre los dos cuadros delimitadores
    iou = inter_area / union_area
    # Calcule las coordenadas de la esquina superior izquierda y
    # la esquina inferior derecha de la superficie convexa cerrada más pequeña
    enclose_left_up = tf.minimum(boxes1[..., :2], boxes2[..., :2])
    enclose_right_down = tf.maximum(boxes1[..., 2:], boxes2[..., 2:])
    enclose = tf.maximum(enclose_right_down - enclose_left_up, 0.0)
    # Calcule el área de la superficie convexa cerrada más pequeña C
    enclose_area = enclose[..., 0] * enclose[..., 1]
    # Calcule el valor de GIoU de acuerdo con la fórmula de GIoU
    giou = iou - 1.0 * (enclose_area - union_area) / enclose_area

    return giou
```

Ilustración 39. Función GIoU (Autor)

Se debe tener presente que siempre existe perdida, para ello el algoritmo debe tener presente diferentes perdidas; de tal manera que se tiene la perdida de la confianza la misma que se utiliza para poder realizar una distinción entre el fondo y el área del primer plano. A continuación, se ve en la Ilustración 40 las líneas de código a utilizar para esta función.

```python
def compute_loss(pred, conv, label, bboxes, i=0, CLASSES=YOLO_COCO_CLASSES):
    NUM_CLASS = len(read_class_names(CLASSES))
    conv_shape  = tf.shape(conv)
    batch_size  = conv_shape[0]
    output_size = conv_shape[1]
    input_size  = STRIDES[i] * output_size
    conv = tf.reshape(conv, (batch_size, output_size, output_size, 3, 5 + NUM_CLASS))

    conv_raw_conf = conv[:, :, :, :, 4:5]
    conv_raw_prob = conv[:, :, :, :, 5:]

    pred_xywh     = pred[:, :, :, :, 0:4]
    pred_conf     = pred[:, :, :, :, 4:5]

    label_xywh    = label[:, :, :, :, 0:4]
    respond_bbox  = label[:, :, :, :, 4:5]
    label_prob    = label[:, :, :, :, 5:]

    giou = tf.expand_dims(bbox_giou(pred_xywh, label_xywh), axis=-1)
    input_size = tf.cast(input_size, tf.float32)

    bbox_loss_scale = 2.0 - 1.0 * label_xywh[:, :, :, :, 2:3] * label_xywh[:, :, :, :, 3:4] / (input_size ** 2)
    giou_loss = respond_bbox * bbox_loss_scale * (1 - giou)

    iou = bbox_iou(pred_xywh[:, :, :, :, np.newaxis, :], bboxes[:, np.newaxis, np.newaxis, np.newaxis, :, :])
    # Encuentre el valor de IoU con el cuadro real El cuadro de predicción más grande
    max_iou = tf.expand_dims(tf.reduce_max(iou, axis=-1), axis=-1)
    # Si el iou más grande es menor que el umbral, se considera que el cuadro de predicción no contiene objetos,
    # entonces el cuadro de fondo
    respond_bgd = (1.0 - respond_bbox) * tf.cast( max_iou < YOLO_IOU_LOSS_THRESH, tf.float32 )

    conf_focal = tf.pow(respond_bbox - pred_conf, 2)

    # Calcula la pérdida de confianza
    # Esperamos que si la cuadrícula contiene objetos,
    # entonces el cuadro de predicción de salida de la red tenga una confianza de 1 y 0 cuando no haya ningún objeto.
    conf_loss = conf_focal * (
            respond_bbox * tf.nn.sigmoid_cross_entropy_with_logits(labels=respond_bbox, logits=conv_raw_conf)
            +
            respond_bgd * tf.nn.sigmoid_cross_entropy_with_logits(labels=respond_bbox, logits=conv_raw_conf)
    )
    prob_loss = respond_bbox * tf.nn.sigmoid_cross_entropy_with_logits(labels=label_prob, logits=conv_raw_prob)
    giou_loss = tf.reduce_mean(tf.reduce_sum(giou_loss, axis=[1,2,3,4]))
    conf_loss = tf.reduce_mean(tf.reduce_sum(conf_loss, axis=[1,2,3,4]))
    prob_loss = tf.reduce_mean(tf.reduce_sum(prob_loss, axis=[1,2,3,4]))
    return giou_loss, conf_loss, prob_loss
```

Ilustración 40. Perdidas (Autor)

Para comprobación del funcionamiento de algoritmo se hizo una prueba en tiempo real utilizando una cámara externa conectada a la entrada USB de la Raspberry Pi, tal como se puede ver en la Ilustración 41 se tiene la detección de una persona.

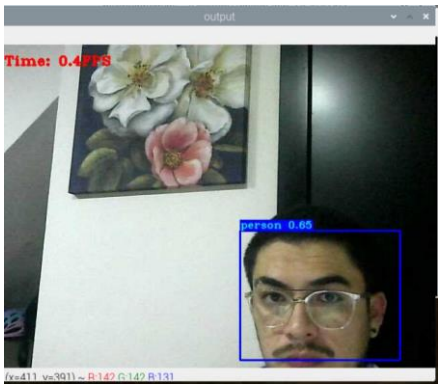


Ilustración 41. Pruebas de funcionamiento (Autor)

3.5.2. Prueba 2. Detección de plazas mediante el análisis de imágenes

Para la implementación de este programa de detección se realiza en Jupyter Notebook, esta es una aplicación que permite crear y compartir proyectos de Python siguiendo un esquema ordenado. Esta aplicación es una aplicación web y puede funcionar en cualquier navegador.

Para la detección de plazas de parking se crea dos algoritmos, el un algoritmo será el ejecutable y el otro algoritmo será auxiliar el cual simplemente realizara comparaciones para que el ejecutable realice la detección.

3.5.2.1. Diagrama de bloques

El diagrama de bloques de la Ilustración 42. establece las diferentes fases del diseño del algoritmo, este diagrama está formado por 5 bloques principales, estos bloques abarcan distintos subprocesos. Cada bloque está planteado según la función a desarrollar por el algoritmo.

Ilustración 42. Diagrama de bloques de la detección de plazas mediante el análisis de imágenes (Autor)

El bloque 1 es el inicio del algoritmo, este será el encargado de que la cámara entre en funcionamiento e inicie con la capturar las imágenes del video. El bloque 2 es el encargado de realizar todo el trabajo de preprocesamientos de imágenes, en este bloque se utilizan las distintas librerías importadas, es importante mencionar que en este bloque se determinan las diferentes regiones donde el algoritmo se enfocará en diferenciar los lugares de estacionamiento para así determinar si algún vehículo está ocupando este lugar. El bloque 3 es el encargado de realizar el análisis que permite determinar si la plaza de parking esta libre u ocupada. El bloque 4 se encarga de realizar una visualización en pantalla de los vehículos que están ingresando a las plazas de parking. En tanto, el bloque 5 se utiliza para tener una información mediante un chatbot de las plazas libres u ocupadas, este chatbot es creado para dialogar mediante la aplicación Telegram.

3.5.2.2. Captura de imágenes

OpenCv es la librería encargada en realizar la captura y trasmisión de imágenes, esto se logra gracias a una línea de código sencilla. Se necesita de una función que se llama

VideoCapture para que esta pueda ejecutar la función de captura, para nuestro proyecto se utilizara el parámetro IP de la cámara.

Como ya se estableció los límites de referencia del parqueadero anteriormente, para la delimitación de las plazas de parking se realizará una vez que se capture la imagen de video por parte del algoritmo, entonces se tomara distintos puntos de referencia los mismos que están pintados en el suelo del parqueadero para así introducir al algoritmo las coordenadas de las mismas.

3.5.2.3. Procesamiento de imágenes

Esta parte hace la determinación si un sitio de parking se encuentra libre o no mediante el uso de una cámara para identificar los lugares de parqueo, y para estar continuamente monitoreando si los mismos están o no ocupados. La forma en la que esto se logra es haciendo una primera captura de las plazas y delimitando el área que ocupa cada sitio de parking. Luego se hace un escaneo continuo de las imágenes para verificar si ha ocurrido algún cambio de estado y, si lo hubo, se actualiza el mismo en la lista (la que lee el chatbot). Para hacer esta comprobación, se hace el uso de procesamiento de imágenes y las librerías correspondientes a estas funciones. Al recibir un nuevo frame (imagen), el programa lo divide en 5 subframes recortando las imágenes en las coordenadas especificadas por los límites establecidos, estos subframes se transforman en imágenes de escala a grises. Esto se realiza con la finalidad de reducir el gasto computacional. Posteriormente, se aplica filtro denominada desenfoque Gaussiano, para la reducción de ruido en las imágenes. A continuación, se compara con su respectiva captura inicial mediante una diferencia de imágenes. La operación de umbralización se utiliza para conseguir una imagen solo con dos valores posibles, o sea si el valor de los pixeles es mayor al valor del umbral, se le asigna el valor de color blanco, de otro modo se le asigna el valor de color negro. Finalmente se utiliza la operación de dilación para aumentar o expandir las formas contenidas en la imagen resultante de la imagen umbral izada. Con este tratamiento de las imágenes ya se puede identificar las plazas libres u ocupadas, pues si el valor de contorno (valor de sensibilidad de detección) es menor a 5000, se consideraría que el sitio de parking está libre y caso contrario se lo considera como ocupado. Esto se usa para continuamente actualizar la lista y para que el chatbot lo lea cuando requiera y se genere una respuesta efectiva a los usuarios cuando requieran la información de cuales plazas están disponibles.

3.5.2.3.1. Procesamiento detallado

Cuando el programa inicia se hace una primera captura donde se adquiere un frame con todas las plazas de parqueo disponibles, tal como se puede ver en la Imagen 3.

Imagen 3 Primera captura de video (Autor)

Luego, se delimita cada uno de las plazas a gestionar, tal como se puede ver en la Imagen 4.

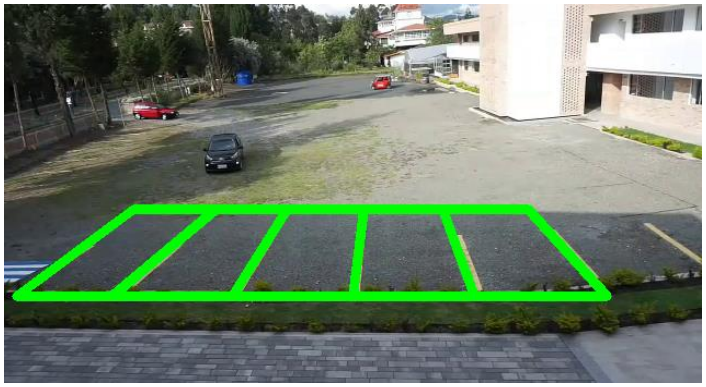
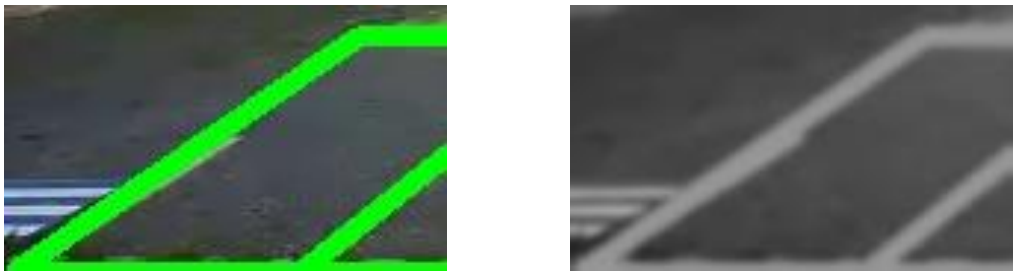


Imagen 4. Delimitación de las plazas de parking (Autor)

En la Imagen 5, se puede ver que después de la delimitación se hace el recorte de cada sitio de parking y se lo maneja como subframes independientes, cada uno. De tal manera que cada Subframe es transformado a escala de gris



(a) (b)

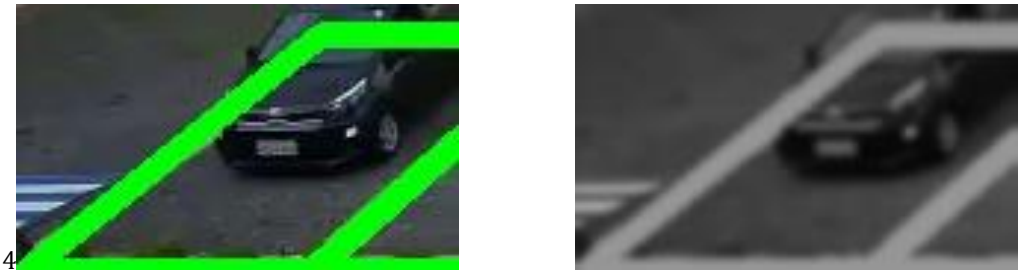
Imagen 5. (a) Subframe de cada sitio de parking (b) Subframe transformado a blanco y negro (Autor)

Estos a su vez, se guardan como las imágenes de referencias de un sitio de parking disponible; de tal forma, que al transcurso del día se verifica si están o no ocupados mediante una comparación entre esta imagen y los frames tomados en el transcurso del tiempo, esta comparación es detallada a continuación.

Se realiza una supervisión continua mediante la cámara. Se toma cada frame y se va comparando mediante un procesamiento de cada subframe por sitios de parking. En la Imagen 6, se explica como el programa detecta que efectivamente un sitio de parking fue ocupado.

Imagen 6.Supervisión de la cámara (Autor)

En este caso, el sitio de parking 1 fue ocupado por un vehículo negro ver Imagen 7 (a). En un inicio se procede tal cual los pasos anteriormente descritos. Se delimita el área de los sitios de parking, se recorta los subframe y se pasa a blanco y negro ver Imagen 7 (b).



4

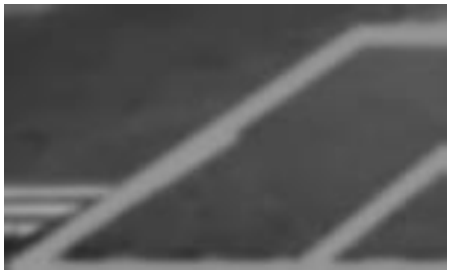
(a) (b)

Imagen 7. (a) Parking 1 ocupado (b) Transformación a blanco y negro (Autor)

A partir de esta última imagen, se hace un procesamiento para efectuar una comparación. Lo que se hace, es una diferencia entre la imagen actual ver Imagen 8. (a) Parking ocupado (comparaciones) (b) parking libre (comparaciones) (c) imagen resultante de las comparaciones entre las imágenes 20 y 21 (Autor) y la obtenida en un inicio ver **¡Error! No se encuentra el origen de la referencia.**(b). Y se obtiene la Imagen 8. (a) Parking ocupado

(comparaciones) (b) parking libre (comparaciones) (c) imagen resultante de las comparaciones entre las imágenes 20 y 21 (Autor) que es la resultante entre las comparaciones.

(a)



(b)

(c)

Imagen 8. (a) Parking ocupado (comparaciones) (b) parking libre (comparaciones) (c) imagen resultante de las comparaciones entre las imágenes 20 y 21 (Autor)

Siendo la Imagen 8 (c) la resultante comparativa, se puede notar claramente la sombra de un vehículo, el mismo que es un diferenciante de toda el área captada. Esta sombra deberá pasar por un siguiente proceso que es el aclarado, para ello se debe aplicar thereshold en cascada, que no es más que el aclaramiento de imágenes. A continuación, en la Imagen 9 se puede observar el aclaramiento de la captura de la primera plaza ocupada.

Imagen 9.(a) Threshold 1 (b) Threshold 2 (Autor)

Al ser un algoritmo que realiza comparaciones de imágenes, lo que determina si una plaza de parking está o no ocupada es la comparación de pixeles, es decir si el área de

una plaza de parking tiene la equivalencia de 100 pixeles, entonces cuando un vehículo ocupa esta área se procede a generar pixeles blancos para ello todo el proceso anterior de aclaración de imágenes con thershold, de tal manera que comparara el área conocida con el área actual como se puede ver en la Imagen 9 (b) determinado si la plaza está o no ocupada.

3.5.2.4. Algoritmo de detección de plazas

El proyecto como tal consta de dos algoritmos, el algoritmo principal será el ejecutable y el algoritmo secundario será utilizado como una librería personalizada, a continuación, se detalla cada algoritmo.

3.5.2.4.1. Algoritmo principal

El algoritmo principal es la ejecutable, si bien es cierto consta de varias líneas de código, estas líneas de código tendrán varios funcionamientos los cuales se les ha dividido en varias secciones; en la Ilustración 43, se puede ver la importación de las librerías que maneja Python para procesamiento de imágenes, manejo de vectores, hilos y manejo del chatbot. Además, se puede evidenciar la importación del algoritmo auxiliar llamado librería.

```
#Programa de Python para detectar slots de parqueo

#Se importa las librerias necesarias:
import cv2 #opencv, para el procesamiento de imagenes
import numpy as np #para el manejo de los vectores
import libreria as lb #una libreria custom creada para funciones secundarias
from threading import Thread #para manejar el hilo del bot de telegram
import logging #para el inicio se sesion del bot de telegram
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters #para el envio de los msjs del bot
```

Ilustración 43. Importación de librerías (autor)

La siguiente sección del algoritmo principal es: la configuración de inicio del chatbot y las variables que se crea para los contornos de las plazas de parking en la Ilustración 44 se puede ver las líneas de comando que se utilizan.

```
#Configuraciones iniciales para iniciar el bot de Telegram
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)

#Variables para los contornos de cada uno de los slots de parqueo
cnts=0
cnts2=0
cnts3=0
cnts4=0
cnts5=0
```

Ilustración 44. Configuraciones iniciales para el chatbot y creación de variables para las plazas (autor)

En la sección principal de nuestro algoritmo principal, se debe crear una línea de código la cual guarde los estados de disponibilidad de las plazas de parqueo tal como se ve en la Ilustración 45.

```
#Lista para el guardado de estados de disponibilidad de los slots de parqueo
parqueosdisponibles=["disponible","disponible","disponible","disponible","disponible"]
```

Ilustración 45. Estados de disponibilidad para las plazas de parking (Autor)

Otro punto importante para la detección de una plaza, es la creación de variables para guardar fondos estáticos, para ello se debe crear las líneas de código para cada uno de las plazas (ver en Ilustración 46). En nuestro caso 5 plazas de parking.

```
#Variables para guardar los fondos estáticos para cada uno de los slots de parqueo.
static_back = None
static_back2 = None
static_back3 = None
static_back4 = None
static_back5 = None
```

Ilustración 46. Variables para guardar fondos estáticos de las plazas de parking (autor)

Parte fundamental del algoritmo es la captura del video, para ello se utiliza una línea de código sencilla, para nuestro proyecto el video se reproducirá en tiempo real, de tal manera que este será llamado desde una cámara que contenga dirección IP.

En esta sección, se tendrá la línea de código que será la que ejecute el chatbot de Telegram, es decir las respuestas a las preguntas generadas desde la APP, tal cual se puede ver en la Ilustración 47.

```
#Variable para la captura de video que se usará para la identificacion de los slots de pàrqueo
video = cv2.VideoCapture(0)

#Funcion que se ejecutara cuando el bot de telegram recibe el msj 'hola'
def start(update, context):
    update.message.reply_text('Bienvenido al Parking UCACUE')
```

Ilustración 47. Variable para capturar video en tiempo real y función para ejecutar el chatbot (autor)

Al hablar del chatbot se debe tomar en cuenta los mensajes de interactuación ya sea entre usuario y chatbot, por ende, en las siguientes líneas de código (ver Ilustración 48) se debe especificar la cantidad de plazas y su disponibilidad tomando en cuenta que siempre caerá en un bucle repetitivo ya que este siempre estará preguntando la disponibilidad de las plazas.

```
#Funcion que se ejecutara cuando el bot de telegram recibe el msj 'disponibilidad'
def availability(update, context):
    #Se inicia contador y string para armar el msj de plazas disponibles
    ind=1
    plazas='Tenemos estas plazas '
    #Se interactua con cada uno de los elementos de la lista
    for element in parqueosdisponibles:
        #Se comprueba si cada slot de parqueo esta disponible y se agrega al msj a enviar
        if element=='disponible':
            plazas=plazas+str(ind)+' '
            print(plazas)
        ind=ind+1
```

Ilustración 48. Función que ejecutara el chatbot al recibir mensajes e interactuación (autor)

En la Ilustración 49, se puede ver la interactuación que desarrolla el programa al tener un mensaje no aceptable, es decir cualquier otra palabra que no haya sido entrenada.

```
    #Se envia el msj con los slots de parqueo disponibles
    update.message.reply_text(plazas)

#Funcion que se ejecutara cuando el bot de telegram recibe un msj no programado
def help(update, context):
    update.message.reply_text('Comando no aceptable, reintente por favor.')

#Funcion echo que se ejecutara cuando al bot se le pide retornar al mismo msj recibido
def echo(update, context):
    update.message.reply_text(update.message.text)

#Funcion que se ejecutara cuando al bot sufrio un error
def error(update, context):
    logger.warning('Comando "%s" causo el siguiente error "%s"', update, context.error)
```

Ilustración 49.Interactuación entre usuario y programa (autor)

La siguiente sección (ver Ilustración 50) es la ejecución al iniciar el hilo del chatbot en la APP de Telegram, para ello Telegram brinda el Updater que no es más que una credencial, es necesario mencionar que este hilo tendrá un bucle infinito ya que se enfocará en la interactuación entre usuario y app.

```
#Funcion que se ejecutara al iniciar el hilo del bot de telegram
def threadbot(threadname):
    #Se crea el Updater y se le pasa al bot con las credenciales unicas
    updater = Updater("1844521532:AAHXNH4jrSEuZsTHhODPCkYEU--ANCHJ6kE", use_context=True)

    #Se obtiene el dispatcher para poder registrar los handlers de cada accion del bot
    dp = updater.dispatcher

    #Se registran handlers o manejadores para cuando el bot recibe un msj especifico o comando
    dp.add_handler(CommandHandler("hola", start)) #al recibir 'hola', ejecutar la funcion start
    dp.add_handler(CommandHandler("disponibilidad", availability)) #al recibir 'disponibilidad', ejecutar la funcion sta

    #Se registra un handler para cuando sucede un error
    dp.add_error_handler(error)
```

Ilustración 50.Función para ejecutar el hilo del chatbot de Telegram (autor)

Para dar inicio y ejecutar el chatbot se necesita de dos líneas de código, sin embargo, se debe tomar en cuenta del procesamiento en paralelo encargado por el hilo, tal como se ve en la Ilustración 51.

```
    #Comenzar a ejecutarse el boot
    updater.start_polling()

    #Se ejecuta el bot hasta que se pare el programa main
    updater.idle()

#Bloque de definicion y correr el hilo para el manejo del bot mediante un procesamiento en paralelo
threadbot = Thread( target=threadbot, args=("Thread-bot", ) )
threadbot.start()
threadbot.join()
```

Ilustración 51.Función para ejecutar el chatbot y función para un procesamiento paralelo (autor)

La siguiente sección del algoritmo hace referencia al área que se manejara para las diferentes plazas de parqueo, para ello se hace uso de los arrays como se puede ver en la Ilustración 52.

```
#Loop infinito para la lectura del video en tiempo real
while True:

    # Lectura del frame del video
    check, frame = video.read()
    if check == False: break

    #Area para los rectángulos del estacionamiento
    area_pts = np.array([[160, 180], [85, 180], [10, 270], [80, 270]], np.int32)
    area_pts1 = np.array([[238, 200], [163, 200], [120, 250], [200, 250]], np.int32)
    area_pts2 = np.array([[332, 200], [253, 200], [225, 250], [325, 250]], np.int32)
    area_pts3 = np.array([[418, 190], [353, 190], [340, 260], [425, 260]], np.int32)
    area_pts4 = np.array([[518, 190], [428, 190], [440, 260], [535, 260]], np.int32)
```

Ilustración 52. Lecturas del frame y áreas de las plazas de parking (autor)

Al tener creado las áreas de las plazas el siguiente proceso es crear cuadros para delimitar los mismos dentro del frame, como se puede ver en la Ilustración 53 estas líneas de código constaran del frame, de las áreas ya determinadas, siempre de un True, de una sección para el color de la línea (para nuestro caso 0,255,0) y finalmente la determinación del grosor de la línea (para nuestro caso 3).

```
    #Creación de cuadros para delitimar los estacionamientos de los vehículos dentro del frame
    estacionamiento1 = cv2.polylines(frame, [area_pts], True, (0, 255, 0),3)
    estacionamiento2 = cv2.polylines(frame, [area_pts1], True, (0, 255, 0), 3)
    estacionamiento3 = cv2.polylines(frame, [area_pts2], True, (0, 255, 0), 3)
    estacionamiento4 = cv2.polylines(frame, [area_pts3], True, (0, 255, 0), 3)
    estacionamiento5 = cv2.polylines(frame, [area_pts4], True, (0, 255, 0), 3)
```

Ilustración 53. Creación de cuadros para delimitar los estacionamientos dentro del frame (autor)

Un punto muy importante para nuestro algoritmo principal es la creación de los Subframe, para ello se creará un Subframe para cada plaza de parqueo, este tendrá varios subframes tal como se puede ver en la Ilustración 54.

```
    # Creación de subframe para analizar las diferentes áreas
    subframe=frame[180:270,10:160]
    subframe2=frame[200:250,120:238]
    subframe3=frame[200:250,225:332]
    subframe4=frame[190:260,340:425]
    subframe5=frame[190:260,428:535]
```

Ilustración 54. Creación de Subframe para analizar las diferentes áreas (autor)

En cuanto al desarrollo como tal del algoritmo para su detección, es importante realizar la conversión de Subframe de cada uno de las plazas a una escala de grises siendo

este un procesamiento previo para la detección de cada plaza ya sea disponible u ocupado (ver Ilustración 55).

```
# Conversión de subframes a grises y preprocesamiento previo
gray = lb.preprocesamiento_frame(subframe, (7,7))
gray2 = lb.preprocesamiento_frame(subframe2,(7,7))
gray3 = lb.preprocesamiento_frame(subframe3,(7,7))
gray4 = lb.preprocesamiento_frame(subframe4,(7,7))
gray5 = lb.preprocesamiento_frame(subframe5,(7,7))
gray_frame = lb.preprocesamiento_frame(frame, (23, 23))
```

Ilustración 55. Conversión de Subframe a grises y procesamiento previo (autor)

La siguiente sección del algoritmo hace referencia a la condición para la primera iteración comparando los fondos estáticos de cada Subframe y convertirle la imagen en gris, esta condición siempre estará en un bucle repetitivo (ver Ilustración 56).

```
# Condición para en primera iteración comparar los fondos estáticos de cada subframe y hacerlo la imagen en gris
if static_back is None:
    static_back = gray
    static_back2 = gray2
    static_back3 = gray3
    static_back4 = gray4
    static_back5 = gray5
    static_back6 = gray_frame
    continue
```

Ilustración 56. Condición para comparar los fondos estáticos (autor)

Esta sección del algoritmo es muy importante, ya que es donde se crea los contornos de los vehículos en movimiento, aquí se hace uso del algoritmo secundario ya que realiza procesamientos de comparaciones, de igual manera se realiza para cada plaza de parqueo tal como se ve en la Ilustración 57.

```
# Cración de contornos de vehículos en movimiento
cnts = lb.procesamiento_subframes(gray,static_back,8)
cnts2 = lb.procesamiento_subframes(gray2,static_back2,8)
cnts3 = lb.procesamiento_subframes(gray3, static_back3,8)
cnts4 = lb.procesamiento_subframes(gray4, static_back4,8)
cnts5 = lb.procesamiento_subframes(gray5, static_back5,8)
cnts6= lb.procesamiento_subframes(gray_frame, static_back6,57)
```

Ilustración 57. Creación de contornos para los vehículos en movimiento (autor)

Al realizar las comparaciones previas, el siguiente proceso a realizar es la especificación de las plazas de parqueo ocupados para ello se utiliza el algoritmo secundario más lo realizado anteriormente, tal como se ve en la Ilustración 58.

```
# Especificación de parkings ocupados
lb.dibujo_rectangulo(cnts,frame,area_pts)
lb.dibujo_rectangulo(cnts2,frame,area_pts1)
lb.dibujo_rectangulo(cnts3,frame,area_pts2)
lb.dibujo_rectangulo(cnts4,frame,area_pts3)
lb.dibujo_rectangulo(cnts5,frame,area_pts4)
```

Ilustración 58. Especificación de plazas ocupados (autor)

Si bien es cierto la comprobación tiene que ser realizada de cada uno de las plazas de parqueo para ello siempre tendrá que estar sujeto a un bucle infinito, este estará detectando si está o no ocupado de manera constante, es importante mencionar que este tendrá un If y un Else; If servirá para el caso de que esté disponible en tanto que Else determinará si está ocupado (ver la Ilustración 59).

```
#Comprobacion de la disponibilidad del slot de parqueo 1
for contour in cnts:
    if cv2.contourArea(contour) < 5000:
        parqueosdisponibles [0] = "disponible"
        continue
    else:
        parqueosdisponibles [0] = "ocupado"
    estacionamiento = cv2.polylines(frame, [area_pts], True, (0, 0, 255), 3)

#Comprobacion de la disponibilidad del slot de parqueo 2
for contour in cnts2:
    if cv2.contourArea(contour) < 5000:
        parqueosdisponibles [1] = "disponible"
        continue
    else:
        parqueosdisponibles [1] = "ocupado"
    estacionamiento = cv2.polylines(frame, [area_pts1], True, (0, 0, 255), 3)

#Comprobacion de la disponibilidad del slot de parqueo 3
for contour in cnts3:
    if cv2.contourArea(contour) < 5000:
        parqueosdisponibles [2] = "disponible"
        continue
    else:
        parqueosdisponibles [2] = "ocupado"
    estacionamiento = cv2.polylines(frame, [area_pts2], True, (0, 0, 255), 3)
```

Ilustración 59. Comprobaciones de la disponibilidad de cada uno de las plazas (autor)

Como detalle del programa se introduce varias líneas de código, con el fin de generar título y números en cada plaza de parqueo tal como vemos en la Ilustración 60.

Finalmente se introduce varias líneas de código para que el algoritmo pueda cerrar el programa ejecutado.

```
    # Agregación de título en color azul
    txtTitulo = "           SISTEMA DE PARKING         "
    cv2.putText(frame, txtTitulo, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (225, 0, 0), 2)

    # Agregación de numeros de parking
    txtTitulo = "     1     2     3     4    5         "
    cv2.putText(frame, txtTitulo, (0, 230), cv2.FONT_HERSHEY_SIMPLEX, 1, (200, 200, 0), 2)
    # Mostrar frame con el cambio en los parkings
    cv2.imshow("Frame de parking", frame)

    # Si se ingresa la tecla q se para el programa
    key = cv2.waitKey(1)
    if key == 27:
        break

#Se libera la camara
video.release()

#Se cierra todas las ventanas
cv2.destroyAllWindows()
```

Ilustración 60. Creación de textos y cierre del programa (autor)

3.5.2.4.2. Algoritmo secundario

Este algoritmo está diseñado específicamente para hacer diferentes tipos de conversiones a las imágenes haciendo uso de threshold y dilatación tal como se detallará a continuación (ver Ilustración 61).

```
#libreria con procesamiento de imágenes
import cv2
import numpy as np

#Conversión de subframe a gris y aplicación de un Blur Gausiano
def preprocesamiento_frame(subframe,kernel):
    gray = cv2.cvtColor(subframe, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, kernel, 0)
    return gray

#Procesamiento de subframes a través de una resta, treshold y dilación, obtención de objetos en movimiento en subframes
def procesamiento_subframes(gray,static_back,thresh):

    diff_frame = cv2.absdiff(static_back, gray)
    thresh_frame = cv2.threshold(diff_frame, thresh, 255, cv2.THRESH_BINARY)[1]
    thresh_frame = cv2.dilate(thresh_frame, None, iterations=2)
    cnts, _ = cv2.findContours(thresh_frame.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    return cnts

#Escritura en frame general de rectángulo en rojo de estacionamiento previo
def dibujo_rectangulo(cnts,frame,area_pts):
    for contour in cnts:
        if cv2.contourArea(contour) < 5000:
            continue
        estacionamiento = cv2.polylines(frame, [area_pts], True, (0, 0, 255), 3)
```

Ilustración 61.Conversiones y procesamientos de Subframe y frames (autor)

CAPÍTULO 4

4. RESULTADOS

4.1. Resultados de la prueba 1. Detección de objetos haciendo uso de Raspberry Pi y TensorFlow

Como se vio anteriormente el algoritmo de detección de objetos mediante el uso de Raspberry Pi y TensorFlow es un tanto complejo, ya que es un sistema previamente entrenando, es importante saber y tener presente que el algoritmo al momento de su ejecución actúa ante la detección.

Entonces al tener un programa ejecutable, el siguiente paso para efectuar la detección es mandar a correr tal como se ve en la Ilustración 62, para ello se debe ingresar a una terminal de la Raspberry Pi e ingresar en la pantalla la línea de código del demo, tal como se puede ver en la imagen los mismos procesos.

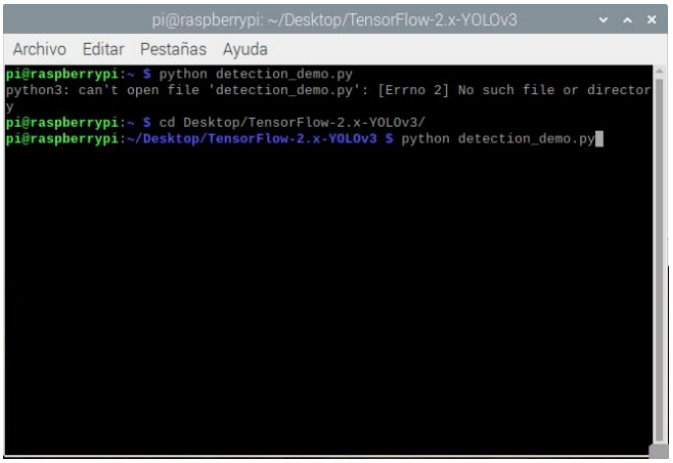


Ilustración 62. Ejecución del programa (Autor)

Una vez que este comience a ser ejecutado rebotará una pantalla donde que se vera la captura del video logrado desde la cámara, este video comenzará a rodar y dará inicio a la detección de los objetos, tal como se ve en la Ilustración 63 .

Ilustración 63. Detección de objetos (Autor)

No cabe duda que se realiza la detección, pero tiene un problema, que el video captado sufre retrasos por el costo computacional de las librerías Yolo y Tensorflow, aproximadamente con una pérdida de reacción del video de unos 6 segundos. Por ende, se descarta este algoritmo de detección, ya que para el proyecto sería no deseado tener un retardo que ocasione problemas de dobles ocupaciones de plazas.

4.2. Resultados de la prueba 2. Detección de plazas mediante el análisis de imágenes

Para la ejecución de la detección de plazas mediante el análisis de imágenes se tiene que dar inicio al programa (ver Imagen 10Imagen 1) para ello se tiene que dar click en Run, para que el algoritmo comience a desarrollar toda su función tal como se ve en la imagen

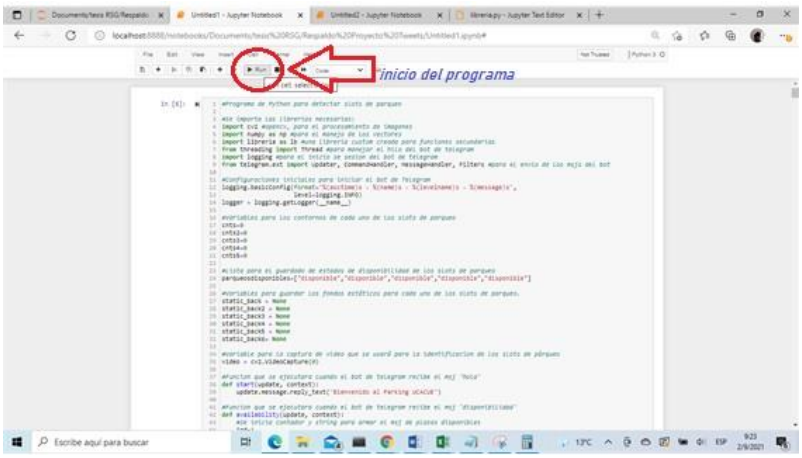


Imagen 10. Inicio del programa (Autor)

Una vez que se dio inicio al algoritmo de detección este procederá a mostrar una pantalla donde que se tendrá la captura del video del parking de la Universidad, por lo tanto, el algoritmo elaborado comenzara a realizar su funcionamiento dando como resultado la

detección de las plazas ya sean ocupadas o vacías, cabe mencionar que su detección es eficiente tal como se puede ver en la Imagen 11.

Imagen 11. Plazas del parking (Autor)

El plus de este algoritmo es el uso del chatbot en donde, desde la aplicación de Telegram s puede revisar la disponibilidad de las plazas. A continuación, en la Imagen 12 se puede observar el demo del parking utilizado para el proyecto.

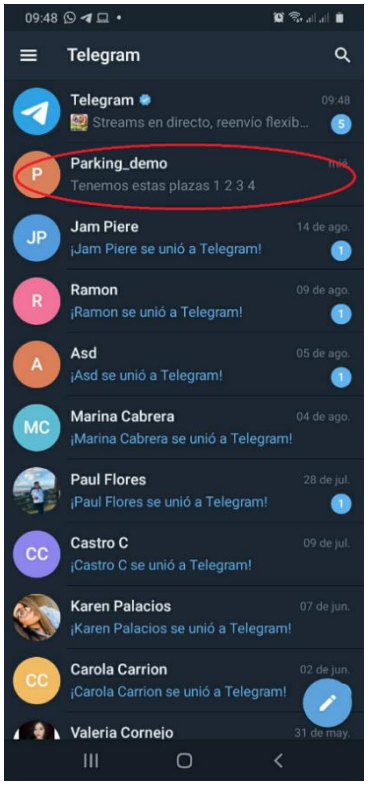


Imagen 12.Imagen del chatbot en Telegram (Autor)

Al ingresar al demo del parking para tener la información de la disponibilidad de las plazas de parqueo, se tiene que ingresar dos mensajes; el primer mensaje (Imagen 13 (a)) como inicio de la información para ver si existe parkings disponibles es “/hola” el cual devolverá una respuesta inmediata de “Bienvenido al Parking UCACUE”. El

segundo mensaje (Imagen 13 (b)) será “/disponibilidad” este devolverá de manera inmediata que plaza de parkings está disponible.

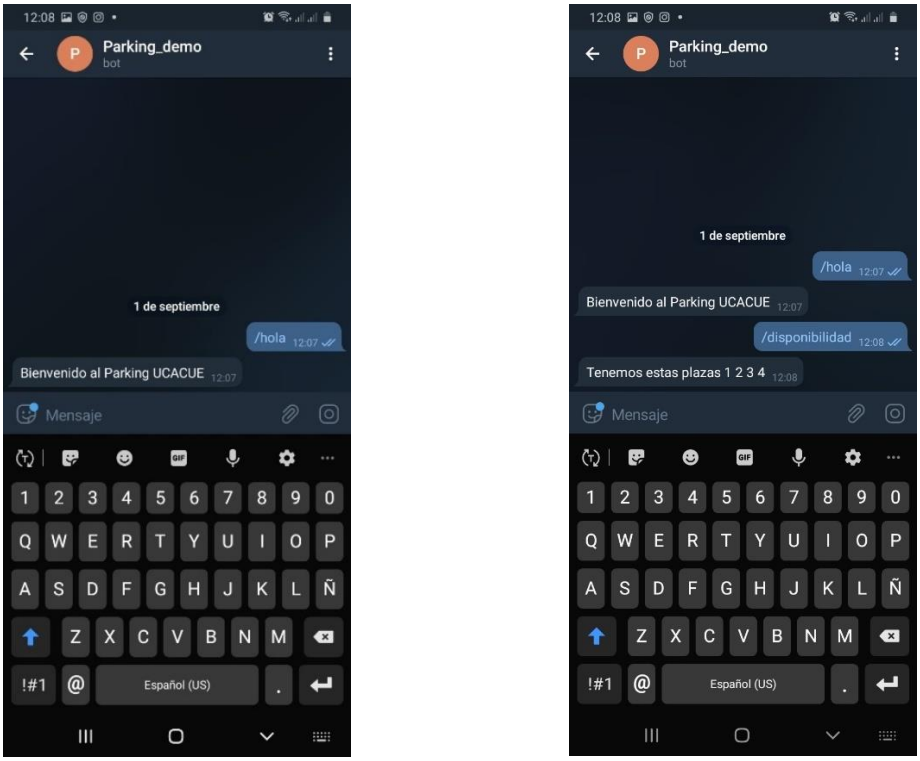


Imagen 13. (a) Mensaje de inicio (b) mensaje de disponibilidad (Autor)

En la Imagen 14 se puede observar una tabla de resultados de la detección de vehículos en las plazas de parking en el transcurso de una semana, como se puede ver se tiene n números de vehículos diarios versus la detección que realiza el algoritmo, en los días lunes, miércoles y viernes se tiene una efectividad del 100% realizando la detección de manera eficiente, en tanto que los días martes y jueves realizan una detección efectiva del 90% y 80% respectivamente. Esto puede ser provocado por varias anomalías como, por ejemplo, el clima (lluvia) o sombras. De tal manera, que al realizar un promedio basados en los resultados de toda la semana tenemos que la detección tiene una efectividad del 94%. Como consecuencia el algoritmo de la detección de plazas mediante el análisis de imágenes es el apropiado para nuestro proyecto, ya que cumple con todas las necesidades planteadas desde el inicio.

	LUNES	MARTES	MIERCOLES	JUEVES	VIERNES
NUMERO DE VEHICULOS	4	10	6	5	9
DETECCION	4	9	6	4	9
EFECTIVIDAD %	100%	90%	100%	80%	100%

Imagen 14. Tabla de resultados (Autor)

5. CONCLUSIONES

Este proyecto de tesis presenta un programa elaborado en Jupyter Notebook de Python que sirve para determinar plazas vacías u ocupadas del parking de Posgrados de la Universidad Católica de Cuenca, a través de un procesamiento de imágenes en tiempo real con la ayuda de un código abierto llamado OpenCV contando con un chatBot el cual sirve para verificar la disponibilidad de las plazas.

En definitiva, el uso de las técnicas de visión artificial se está inmiscuyendo en nuestra sociedad, pretendiendo así ser la base para la detección de plazas libres u ocupados mediante cámaras instaladas en lugares estratégicos de parkings. En tanto, que esta solución gracias al chatbot servirá para disminuir el tiempo que se toma en buscar una plaza, además de reducir la contaminación y bajar costos de parqueo.

El diseño de este programa al ser realizado por líneas de código abierto, da la posibilidad de agregar muchos otros tipos de funciones para así innovar y brindar nuevos requerimientos o servicios.

El costo del proyecto es realmente un factor influyente, puesto que entre equipos, materiales y programación se lo realizo por un costo de $750 dólares, de tal manera que su implementación sería rentable para cualquier institución que forme parte de la era tecnológica.

El programa desarrollado para la detección de plazas depende de varios factores como la posición y ubicación de las cámaras para hacer la detección, otro factor como el medio ambiente y la iluminación puesto que para eso se debe hacer una calibración en los Subframe, otro factor es el clima y las sombras que se pueden producir.

El sistema Chatbot implementado con Telegram motivará al usuario para garantizar la visibilidad de la disponibilidad de las plazas y así dirigirse de manera inmediata al lugar de parqueo.

Este programa de detección presenta una efectividad del 94%, ya que los errores son por generación de falsos positivos y falsos negativos a causa de la calibración de la sensibilidad.

Al hacer las pruebas en tiempo real, se confirma que el programa para realizar la detección de plazas en el parking de Posgrados de la Universidad Católica de Cuenca depende de los distintos tipos de obstáculos que se puedan presentar, ya sea peatones o sombras en las distintas áreas de las plazas creando así falsos positivos y negativos. En tanto, que para lograr una solución a estos errores se modifica la sensibilidad de detección en el programa.

Como se mencionó anteriormente, este proyecto de tesis se desarrolló en base a la visión artificial y este aporte podrá ser utilizado por parte de la Universidad Católica de Cuenca, de tal manera que se podrán consultar información de la cantidad de plazas libres desde la App Telegram distribuyendo así de mejor manera a los vehículos en el parking de la Jefatura de Posgrados.

6. RECOMENDACIONES

Una de las recomendaciones para este proyecto de tesis es que cuando se instalen las cámaras se elija un lugar donde esta pueda optimizar las distintas plazas de parqueo de tal manera que esta tenga una visión especifica de las mismas, sin embargo, se debe tomar en cuenta también cada obstáculo existente para poder hacer la detección y ejecute su perfecto funcionamiento.

Otro punto a tomar en cuenta es verificar el funcionamiento óptimo de las cámaras, estas a su vez tienen que verificar la recepción de las imágenes además de su procesamiento tomando en cuenta que su funcionamiento será por direccionamiento IP.

Algo muy importante para la ejecución de este proyecto es verificar que el programa o algoritmo este corriendo con normalidad para que así pueda guardar datos para realizar las comparaciones.

En tanto que con el uso de la App Telegram tomar en cuenta que el nombre del demo sea el correcto, ejecutar las palabras adecuadas para que el mismo pueda reconocer y dar una respuesta óptima.

En un futuro si bien es cierto se puede hacer mejoras a este programa, donde que se pueda reservar plazas de parqueo de tal manera redireccione de manera inmediata a la plaza, creando un flujo vehicular rápido y a su vez dejando a un lado la contaminación.

7. Bibliografía

costa Jazmin, Tintos Juan P, G. J. (2014). *i-PARKING: Sistema Inteligente para Control de Plazas de Estacionamiento en Vías Públicas de Zonas Urbanas*. https://rcs.cic.ipn.mx/2014_76/i-PARKING_ Sistema Inteligente para Control de Plazas de Estacionamiento.pdf

Alberto, M. (2019). *UTILIZACIÓN DEL MACHINE LEARNING EN LA INDUSTRIA 4.0* [UNIVERSIDAD DE VALLADOLID]. https://core.ac.uk/download/pdf/228074134.pdf

Alfredo, S. (2020). *La librería Numpy*. https://aprendeconalf.es/docencia/python/manual/numpy/#la-clase-de-objetos-array

Alvear Vanessa, Rosero Paul, Peluffo Diego, P. J. (2017). *Internet de las Cosas y Visión Artificial, Funcionamiento y Aplicaciones: Revisión de Literatura.* http://scielo.senescyt.gob.ec/pdf/enfoqueute/v8s1/1390-6542-enfoqueute-8-s1-00244.pdf

Ameijeiras David, González Hector, H. Y. (2020). *Revisión de algoritmos de detección y seguimiento de objetos con redes profundas para videovigilancia inteligente*. http://scielo.sld.cu/pdf/rcci/v14n3/2227-1899-rcci-14-03-165.pdf

Angie, A. (2014). *MODELOS DE COLOR: RGB Y CMYK*. http://ilustracionypresentaciondeproyectos.blogspot.com/2014/09/modelos-de-color-rgb-y-cmyk.html

Archie, S. (2020). *Overview of YOLO*. https://medium.datadriveninvestor.com/overview-of-yolo-673f5019d0f2

Asler, C. (2020). *Ventajas y desventajas de usar Python en la programación web*.

Beyeler Michael. (2017). *Machine Learning for OpenCV* (Packt (ed.)). https://books.google.es/books?hl=es&lr=&id=2eZDDwAAQBAJ&oi=fnd&pg=PP1&dq=open+cv+&ots=PP-ybIsaxv&sig=zuMHw8LeKmdpV6VGKADcfvwdOB4#v=onepage&q=open cv&f=false

Carlos, S. (2016). *Diseño e implmentacion de un prototipo de sistema para parqueo utilizando una red de sensores inalambricos. Quito-Ecuador: Escuela Politecnica Nacional.* [Escuela Politecnica Nacional]. https://bibdigital.epn.edu.ec/bitstream/15000/16507/1/CD-7182.pdf

Cesar, E. (2020). *ALGORITMOS DE IDENTIFICACIÓN DE PIEL HUMANA Y SU RELACIÓN CON LOS SISTEMAS DE COLOR. SU APLICACIÓN A LA SEGMENTACIÓN DE PIEL BASADA EN PÍXELES* [Universidad Nacional de La Plata]. http://sedici.unlp.edu.ar/bitstream/handle/10915/100550/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Cruz Henry, Meneses Juan, Eckter Martina, M. J. (2015). *Seguimiento Automático para RGB y Detección de Objetos en Color*. https://core.ac.uk/download/pdf/148682547.pdf

David, T. (2020). *Rendimiento de Raspberry Pi 3B+ con Intel NCS ejecutando métodos de aprendizaje profundo neuronal en clasificación de tipo de vehículos en tiempo continuo.* UNIVERSIDAD DEL AZUAY.

Denniye, H. (2018). *EL MACHINE LEARNING A TRAVÉS DE LOS TIEMPOS, Y LOS APORTES A LA HUMANIDAD* [UNIVERSIDAD LIBRE SECCIONAL PEREIRA]. https://repository.unilibre.edu.co/bitstream/handle/10901/17289/EL MACHINE LEARNING.pdf?sequence=1&isAllowed=y

Eduonix. (2018). *Image Facial Recognition using Open-CV Python*. https://blog.eduonix.com/wp-content/uploads/2018/09/Facial-Recognition-using-Open-CV-Python.jpg

Higuera Clara. (2015). *INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON*. https://www.cartagena99.com/recursos/alumnos/apuntes/Presentacion_Python1.pdf

Javier, C. (2017). *Máquinas de aprendizaje y aplicaciones* [Universitat de les Illes Balears]. https://dspace.uib.es/xmlui/bitstream/handle/11201/147224/Cantero_Javier.pdf?sequence=1&isAllowed=y

Joseph Redmon, F. A. (2018). *YOLOv3: An Incremental Improvement*.

Kaehler Adrian, B. G. (2017). *Learning OpenCV 3* (D. Schanafelt (ed.)). https://books.google.es/books?hl=es&lr=&id=LPm3DQAAQBAJ&oi=fnd&pg=PP1&dq=matrices+en+opencv&ots=2wIqRcchxg&sig=Exy_S9PV8hjQJDdeToqowOxRlDU#v=onepage&q=matrices en opencv&f=false

Karen, R. (2017). *ANALISIS E IMPLEMENTACI ´ON DEL ALGORITMO DE ´ DETECCION FACIAL DE ´ VIOLA-JONES* [UNIVERSIDAD TECNICA DEL NORTE]. http://repositorio.utn.edu.ec/bitstream/123456789/7315/1/04 MEC 204 TRABAJO DE GRADO.pdf

Kevin, G. (2018). *Sistema de Detección de Objetos para Reconocimiento Gestual mediante Redes Neuronales Convolucionales* [Universitat Politècnica de València]. https://riunet.upv.es/bitstream/handle/10251/129789/García - Sistema de Detección de Objetos para Reconocimiento Gestual mediante Redes Neuronales Co....pdf?sequence=1&isAllowed=y

Maria, V. (2016). *Vision Artificial Aplicada a la Clasificacion basada en Color* [Universidad de Burgos]. https://riubu.ubu.es/bitstream/handle/10259/4176/Viyuela_Fernández.pdf?sequence=1&isAllowed=y

Meher, K. (2017). *Pandas Guide*. https://buildmedia.readthedocs.org/media/pdf/pandasguide/latest/pandasguide.pdf

Mohammad, H. (2016). *Read YUV Videos and Extract the Frames*. https://www.mathworks.com/matlabcentral/fileexchange/59497-read-yuv-videos-and-extract-the-frames

Moises, B. (2018). *Internet de las Cosas* (Reus). Ulzama Digital. https://books.google.es/books?hl=es&lr=&id=jF-LDwAAQBAJ&oi=fnd&pg=PA9&dq=internet+de+las+cosas+visión+artificial&ots=3LK5bw5jxN&sig=ufxnnYWcCDpdxGBe7J63uCVlk4g#v=onepage&q=internet de las cosas visión artificial&f=false

Nelson, R. (2016). *Implementación de un parking inteligente utilizando arduino basado en loT.* [Universidad Politecnica Salesiana.]. https://dspace.ups.edu.ec/bitstream/123456789/13461/1/UPS-GT001798.pdf

Nicolas, A. (2013). *IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE SEÑALES DE TRÁFICO MEDIANTE VISIÓN ARTIFICIAL BASADO EN FPGA* [Universidad de Sevilla]. http://bibing.us.es/proyectos/abreproy/12112/fichero/Documento_completo%252FProyecto+Fin+de+Carrera-Nicolás+Aguirre+Dobernack.pdf

Pérez Ivet, Diaz Yanet, B. R. (2014). *El lenguaje de programación Python/The programming language Python*. https://www.redalyc.org/pdf/1815/181531232001.pdf

Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, R.-S. J. (2019). *Mini-YOLOv3: Real-Time Object Detector for Embedded Applications*. https://ieeexplore.ieee.org/document/8839032?denied=

Rafael, M. (2020). ¿Qué es OpenCV? Instalación en Python y ejemplos básicos. *Revistadigital INESEM*. https://revistadigital.inesem.es/informatica-y-tics/opencv/

Rashmi, R. (2020). *Terminologies used In Face Detection with Haar Cascade Classifier: Open CV*. https://ai.plainenglish.io/terminologies-used-in-face-detection-with-haar-cascade-classifier-open-cv-6346c5c926c

Rojas Rafael. (2016). *Visión Artificial Unidad de Competencia I Introducción a la Visión Artificial*. http://ri.uaemex.mx/bitstream/handle/20.500.11799/63809/secme-35716.pdf?sequence=1&isAllowed=y

Rosas Leonel, Vallejo Jair de Jesus, P. W. R. D. (2017). *Robot clasificador de objetos de color utilizando técnicas de filtrado RGB*. *3*. https://ecorfan.org/spain/researchjournals/Prototipos_Tecnologicos/vol3num10/Revista_de_Prototipos_Tecnologicos_V3_N10.pdf#page=57

Vladislav, J. (2017). *Detection of a license plate position from camera records of moving cars* [CZECH TECHNICAL UNIVERSITY IN PRAGUE].

https://dspace.cvut.cz/bitstream/handle/10467/69140/F8-DP-2017-Jasek-Vladislav-thesis.pdf?sequence=1

AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL

Yo, **Ronald Alexis Segarra Guzman** portador de la cédula de ciudadanía N.º 0105079990. En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación **"Diseño de un sistema de parking automático mediante técnicas de visión artificial"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos, Así mismo; autorizo a la Universidad para que realice la publicación de este trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, **15 de febrero de 2022**

F: .....

Ronald Alexis Segarra Guzman

0105079990