# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

## DISEÑO E IMPLEMENTACIÓN DE MINIDRONES (UAVS)

## INTELIGENCIA ARTIFICIAL APLICADA EN MINIDRONES ENFOCADO AL CAMPO DE VISIÓN COMPUTACIONAL

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN**

**JOSÉ LUIS PILLAJO CORREA**

**jose.pillajo01@epn.edu.ec**

**DIRECTOR: ING, ANDRÉS ROSALES ACOSTA, PhD**

**andres.rosales@epn.edu.ec**

**Quito, Octubre 2022**

# CERTIFICACIONES

Yo, José Luis Pillajo Correa declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

**José Luis Pillajo Correa**

Certifico que el presente trabajo de integración curricular fue desarrollado por José Luis Pillajo Correa, bajo mi supervisión.

**Ing. Andrés Rosales Acosta, PhD**

**DIRECTOR**

# DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

José Luis Pillajo Correa

Ing. Jorge Andrés Rosales Acosta, PhD

# DEDICATORIA

A mis padres Edison y Carmen y a mis hermanos Vale y Danny, cuyo apoyo incondicional han permitido culminar mis estudios universitarios, uno de los muchos objetivos en mi vida.

A mi perrito Matías, por ser mi animalito de la suerte en las actividades que realizo y compañía durante las noches de desvelo al estudiar, hacer deberes e incluso en el desarrollo de este proyecto.

A mi abuelita Manuelita por su inmenso cariño, compañía física y espiritual, al tenerme presente en sus oraciones. A mi tía Gladys por sus consejos, apoyo y preocupación por mi familia en todo momento.

# AGRADECIMIENTO

A Dios, por regalarme una nueva oportunidad de vida en alguna etapa de mi vida, ser mi luz y fortaleza en las decisiones y actividades que realizó día a día.

A mis padres y hermanos, que aparte de ser las personas que más quiero, se han convertido en un ejemplo a seguir, por su constante lucha en búsqueda de superación personal y profesional, gracias por sus palabras y apoyo incondicional.

A mi tutor el Ing. Andrés Rosales PhD, por su apoyo y guía en el desarrollo de este proyecto, permitiendo culminar con éxito los objetivos planteados.

A mis amigos Hans, Alex, Jonathan y Fausto por su amistad incondicional que hicieron de la universidad un lugar ameno y muchas veces entretenido, por todas las experiencias que compartimos.

ÍNDICE DE CONTENIDO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# RESUMEN

Una subrama del campo de inteligencia artificial es la visión computacional, misma que provee ha robots móviles la capacidad de captar imágenes, procesarlas y tomar decisiones. Al implementarlas en UAVs se ha trabajado en distintas tareas como supervisión de productos en grandes bodegas, uso correcto de mascarillas en exteriores en épocas COVID, actividades militares, entre otras. El uso de técnicas de aprendizaje automático y profundo “machine learning – Deep learning”, se ha popularizado con la existencia de computadoras pequeñas que poseen altas capacidades de procesamiento como son las tarjetas raspherry, NVIDIA y otros.

En este trabajo, se propone la implementación de algoritmos autónomos de visión artificial para el desarrollo de dos aplicaciones: la primera consiste en el monitoreo de vehículos que circulan a través de una región de interés “ROI” trazada en alguna zona de una carretera concurrida, llevando así el conteo de carros y captura de estos para un posterior análisis, la segunda aplicación consiste en la detección y seguimiento de un objetivo particular el cual es una persona que lleva un saco color rojo. Para ambas aplicaciones se ha tomado como base, video e imágenes varias, simulando capturas realizadas por un mini – dron, a las cuales se ha realizado procesamientos adicionales, utilizando técnicas de algoritmos de detección de objetos siendo el pionero Yolo v5 en su versión Tiny y algoritmos de seguimiento o tracking robustos de múltiples objetos entre ellos “Deep Sort”, los cuales han sido elegidos considerando una futura ejecución en una SBC, evaluados actualmente en distintas aplicaciones reales.

**Palabras clave:** Yolov4, Deep Sort , Filtro de Color, Python, Open CV, Centroide Tracking, Haar Cascade, SBC – computadadora de placa reducida.

# ABSTRACT

A sub-branch of the field at artificial intelligence is computer vision, which provides mobile robots the ability to capture images, process them and take decisions. By implementing them in UAVs, different tasks have been worked on, such as supervision of products in large warehouses, correct use of masks outdoors in times of COVID, military activities, etc., The use of automatic and deep learning techniques "machine learning - Deep learning", has become popular with the existence of small computers that have high processing capabilities such as raspberry, NVIDIA and others.

In this work, the implementation of autonomous artificial vision algorithms is proposed for the development of two applications: the first consists of monitoring vehicles that circulate through a region of interest "ROI" drawing in some area of a busy road. Thus, counting cars and capturing them for later analysis, the second application consists of detecting and tracking a particular target, which is a person wearing a red jacket. For both applications, video and various images have been taken as a basis, simulating captures taking by a mini - drone, to which additional processing has been carried out, using object detection algorithm techniques, being the pioneer Yolo v5 in its Tiny version and algorithms of robust tracking for multiple objects, including "Deep Sort", which have been chosen considering a future execution in an SBC, currently evaluated in different real applications.



**Keywords:** Yolov4 y 5, Deep Sort, Color Filter, Python, Open CV, Centroid Tracking, Hear Cascade.

# 1 INTRODUCCIÓN

El uso de vehículos no tripulados utilizados para realizar actividades terrestres, aéreas, acuáticas, han experimentado importantes avances en los últimos años, lo cual ha llevado a realizar constates investigaciones en el campo de la robótica principalmente en los sistemas robóticos aéreos más conocidos como UAV's, para los cuales se ha trabajado en importantes aplicaciones industriales, mapeo y vigilancia, búsqueda y rescate, actividades en el sector agrícola, transporte, etc. [1]. En el artículo [2] se menciona que la siguiente gran revolución tecnológica de los UAVs corresponden a los UAVs inteligentes prometiendo brindar nuevas oportunidades en diferentes campos a menor costo, para lo cual el conocimiento de inteligencia artificial es importante, en este trabajo se realiza dos aplicaciones con la ayuda de uno de sus subcampos correspondiente a la visión computacional.

El dotar de visión artificial o computacional a mini - drones los convierte en equipos inteligentes, permitiéndolos tomar decisiones en base a información presente en imágenes y posteriormente tomar acciones de control determinadas.

El enfoque principal de este trabajo de investigación es el desarrollo de aplicaciones en el campo de seguridad: vigilancia y monitoreo, para lo cual actividades como detección de objetos específicos y seguimiento de estos son indispensables, actualmente estas actividades pueden realizarse sin problema alguno en áreas reducidas con la ayuda de cámaras fijas, sin embargo, el trabajo en áreas extensas requiere de mayores dispositivos de captura. La primera aplicación corresponde al monitoreo y control de tráfico de vehículos en una carretera concurrida, para lo cual se registra el número de carros que circulan por cierta región de interés "ROI" a la cual apuntará la cámara del mini - dron y posteriormente se realiza una captura de estos teniendo una imagen del vehículo como registro, adicionalmente se puede hacer uso de la selección de la ROI para apuntar al lugar donde el mini – dron puede aterrizar que necesariamente tiene que ser un área libre de circulación. La segunda aplicación consiste en la detección y seguimiento de una persona que lleva un distintivo particular, teniendo en cuenta factores importantes como es la oclusión del objeto o la perdida temporal del mismo en el cuadro de visualización.

El algoritmo de detección es el pilar fundamental en la implementación de ambas aplicaciones puesto que sin una previa detección, no existirá un posterior seguimiento, detectar objetos involucra el uso de redes neuronales, mismas que serán capaces de reconocer objetos particulares dentro de imágenes proporcionadas por la cámara de un

mini - dron para un posterior procesamiento, para lo cual hace uso de una serie de operaciones matriciales implicando recursos computacionales elevados, mientras que el algoritmo de tracking o seguimiento se alimenta de la información proporcionada por el algoritmo de detección, por lo que el objeto de interés obtenido en la etapa anterior y encerrado en un recuadro será buscado durante toda la secuencia de imágenes o frames de video. [3]

Los algoritmos seleccionados serán elegidos teniendo en cuenta una futura implementación física del proyecto por lo tanto limitaciones en cuanto a software del controlador seleccionado serán considerados.

## 1.1 Objetivo general

Desarrollar y simular aplicaciones que involucren detección y seguimiento de objetos (personas y autos) para ámbitos de seguridad, aplicando técnicas de inteligencia artificial enfocados en el campo de visión computacional, con imágenes y video en tiempo real captadas por la cámara de un mini - dron.

## 1.2 Objetivos específicos

- Realizar una investigación teórica sobre el funcionamiento de las redes neuronales convolucionales CNN (Deep Learning) y su aplicación en el campo de la visión computacional.
- Investigar y analizar diferentes algoritmos utilizados en la detección y tracking de objetos.
- Desarrollar aplicaciones enfocadas en el ámbito de seguridad que involucren los algoritmos seleccionados.
- Simular, comparar y evaluar los resultados obtenidos, teniendo en cuenta una futura implementación física, considerando las imágenes enviadas por un mini -drón y la implementación del algoritmo de visión computacional en una computadora de placa reducida “SBC”.

## 1.3 Alcance

- Se implementarán al menos dos de los diferentes algoritmos de detección y tracking de objetos consultados con el fin de seleccionar los adecuados, que se adapten a las aplicaciones a realizar.

- Se utilizará la programación en Python para la implementación de los algoritmos y principalmente la librería de Open CV para realizar el procesamiento de imágenes, típicas funciones en aplicaciones de visión computacional.
- Se realizará simulaciones en varios campos con el fin de evaluar el desempeño de los algoritmos implementados.
- Se realizará un análisis de los microcontroladores disponibles en el mercado que permitan una futura implementación física del proyecto diseñado.

## 1.4 Marco Teórico

En el presente capítulo se desarrollan los principales conceptos teóricos relacionados con la inteligencia artificial haciendo enfoque en una de sus ramas que es la visión computacional, para la cual se explican técnicas consideradas en el desarrollo del algoritmo planteado, abordando principalmente temas como: técnicas de algoritmos de detección y seguimiento o tracking de objetos que se pueden ejecutar en videos previamente grabados o imágenes en tiempo real.

### 1.4.1 Visión artificial o Computacional

Visión Computacional pertenece a una rama de la inteligencia artificial misma que basa su funcionamiento en emular la visión del ser humano con el fin de poder captar, procesar y analizar imágenes obtenidas a partir de dispositivos digitales, por ejemplo: cámaras, con el fin de obtener información numérica o simbólica que puede ser interpretada por una computadora [4]. La Visión Computacional se puede dividir en 4 grandes etapas:

- Primera etapa: Consiste básicamente en la adquisición de imágenes que pueden ser capturas o videos en tiempo real. Sin embargo, este proceso no es tan simple puesto que la computadora a la imagen en sí no la entiende, requiere de un procesamiento adicional, en la cual se trata a la imagen como una matriz de datos en la que cada elemento corresponde a un píxel, con un valor determinado (0 – 255 o en valores normalizados 0 – 1) que proporciona información como el color, saturación, brillo y en su conjunto conforman a una imagen. Una vez obtenida esta matriz se puede proseguir a las siguientes etapas en donde se podrá manipular o alterar dichos elementos. [5]
- Segunda etapa: Comprende netamente el procesamiento de imágenes con las matrices de pixeles antes obtenidas, cuya dificultad dependerá de la aplicación a implementar. Este proceso viene desde un simple redimensionamiento de la

imagen, aplicación de filtros con el fin de resaltar algún patrón importante o eliminarlo, extracción de colores, transposiciones, etc., todo lo mencionado con el fin de facilitar el trabajo en las siguientes etapas.

- Tercera etapa: Consiste en la segmentación de la imagen es decir encerrar las partes u objetos de interés de acuerdo con la aplicación a implementar.
- Cuarta etapa: Etapa final en la cual se realiza el proceso de reconocimiento y clasificación de los elementos u objetos previamente segmentados, tomando en cuenta características conocidas también como “features” que se obtienen anticipadamente en el periodo de entrenamiento, lo cual permitirá poder distinguir unos objetos de otros, es decir, poder caracterizar personas, carros, etc., de entre varios objetos detectados.[6]

### 1.4.2 Redes Neuronales

La inteligencia artificial emplea redes neuronales con el fin de emular las actividades que realizar el cerebro humano siendo una de las principales el aprendizaje, que se efectúa a través de recopilación de información o data [7].Si se lleva este concepto al campo de visión computacional, este está ligado a su cuarta etapa en donde se menciona el reconocimiento y clasificación de objetos, por lo que permitir que una máquina aprenda a realizar lo mencionado, requiere del uso redes neuronales que al implementarse en una computadora se habla de redes neuronales artificiales (ANN).

#### 1.4.2.1 Red Neuronal Artificial

Una red neuronal artificial está compuesta por un gran número de neuronas artificiales definidas matemáticamente en Eq (1), emulando de esta forma una neurona biológica, su estructura se muestra en

$$\boldsymbol{z = f\left(b + \sum_{i=1}^{n} (wi + xi)\right)} \quad \boldsymbol{(1)}$$

Donde:

$\boldsymbol{x_i}$: entradas (valores entre 0 a 1)

$\boldsymbol{w_i}$: pesos (valores entre 0 a 1)

$\boldsymbol{b}$: bias o sesgo

## $f$: Función de activación

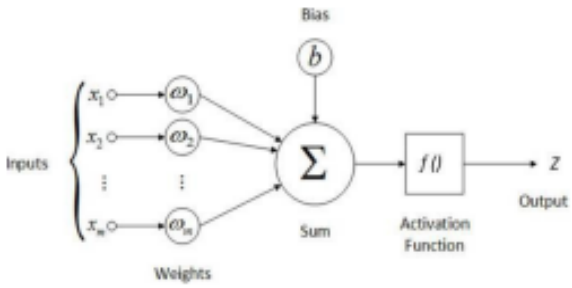


**Figura 1.1**. Estructura de una Neurona Artificial

Cada uno de los parámetros mencionados cumplen una función específica; el peso, aumenta o disminuye el valor de entrada, mismo que toma valores de acuerdo con el nivel de importancia enfocado en el aprendizaje del modelo a implementar, inicialmente estos valores son aleatorios posteriormente durante el entrenamiento “training” poco a poco se van ajustando hasta lograr la tarea para la cual ha sido entrenado; el bias o sesgo, permite ajustar el modelo a la función de activación que básicamente hará que la señal de salida sea o no propagada al resto de las neuronas. Una red neuronal artificial está conforma por n número de neuronas conectadas entre sí, dispuestas en distintas capas como se muestra en Figura 1.2.

- Capa de Entrada: Recibe directamente los datos de entrada, en primera instancia los datos de entrenamiento de la red y posteriormente datos de prueba.
- Capa Ocultas: Son responsables de manejar el desempeño de la red, tratando de mejorar cada vez su rendimiento y exactitud mejorando de esta forma el aprendizaje. Una red puede contar con n capas ocultas.
- Capa Salida: Enviar resultados para los cuales la red ha sido entrenada.[8]

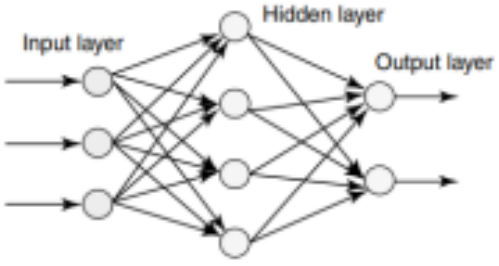


**Figura 1.2.** Estructura de una Red Neuronal Artificial

Una red neuronal convencional puede ser utilizada para implementar modelos de aprendizaje en cuanto a procesamiento de datos, modelización, predicciones numéricas etc., sin embargo, para trabajar con imágenes que prácticamente son tratadas como matrices se requiere de redes neuronales especiales como son las redes neuronales convolucionales.

#### 1.4.2.2 Redes Neuronales Convolucionales (CNN)

Las CNN trabajan con el tipo de aprendizaje supervisado, lo cual consiste básicamente en tener una data de imágenes de entrenamiento debidamente etiquetado que alimentará a la red, es decir a cada imagen de entrenamiento se le encierra en un recuadro al objeto que se desea aprender a identificar con la respectiva etiqueta utilizando programas especializados como Label Img [9], como se muestra en Figura 1.3, donde se requiere identificar personajes de legos. Hay que considerar que para entrenar una red se debe contar con una base de datos de imágenes muy grande, sin embargo, se cuenta con algoritmos previamente entrenados usados en este proyecto que serán de gran ayuda, como se explicarán posteriormente.

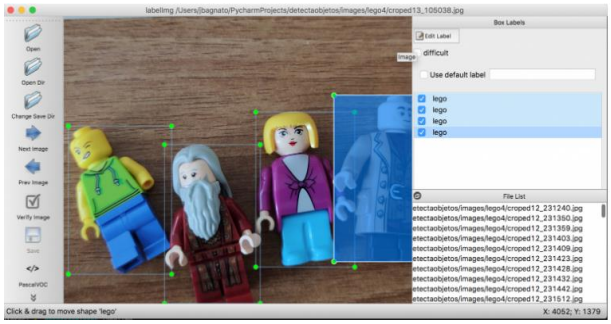


**Figura 1.3.** Label Img, Programa para etiquetado de imágenes para entrenamiento

Como se mencionaba una imagen puede ser tratada como una matriz cuyos elementos son pixeles con valores normalizados de 0 a 1 con los cuales trabaja la CNN. Existen cuatro operaciones básicas que se realizan dentro de una CNN: paso de convolución, RELU, Pooling y clasificación.

##### 1.4.2.2.1 Paso de Convolución + RLU

Consiste en multiplicar cada uno de los pixeles de la imagen por un filtro también conocido como “kernel - detectores de características”, el cual es una matriz S X S de menor tamaño que la imagen original, dicha matriz se desliza por toda la imagen en pasos determinados y finalmente se obtiene una matriz de menor tamaño también llamado “Mapa de

características" como resultado de la suma total entre cada píxel de la imagen original con el filtro en cada paso de convolución. Es evidente que para tener mayor exactitud en el objeto a detectar el proceso de convolución se lo realiza con muchos filtros en la etapa de entrenamiento.

El resultado, es un set de mapas de características obtenidos por los diferentes filtros, los cuales se comportan como una primera capa oculta donde básicamente se detectarán bordes, curvas, etc., indispensables en el entrenamiento de una CNN. Este proceso se puede observar en Figura 1.4.

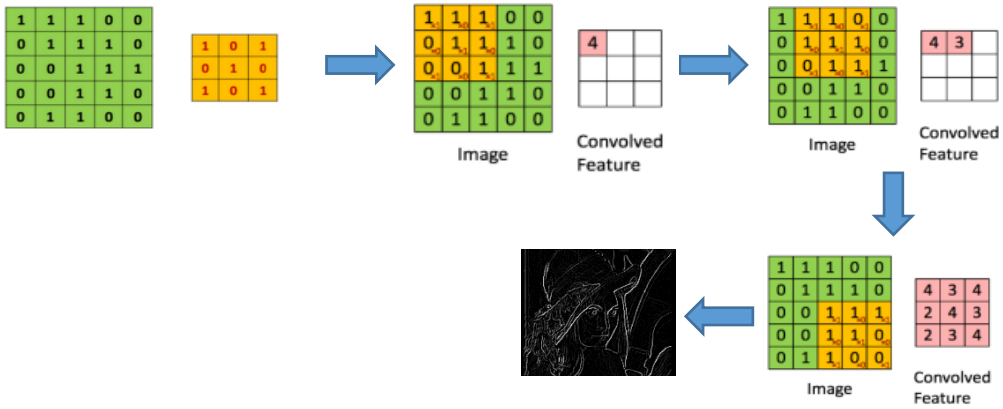


**Figura 1.4.** Operación de Convolución y obtención del Mapa de Características

Teniendo en cuenta que un filtro o kernel es una matriz S x S con números (0, 1, -1) se puede llegar a tener valores negativos en un mapa de características, por lo cual se usa RELU, mismo que es una función de activación que remplaza dichos valores por 0 ayudando de esta manera a introducir la no linealidad al sistema puesto que no siempre se contará con imágenes de entrada totalmente claras, ayudando de esta manera a que el entrenamiento tome en cuenta estos factores, al momento de detectar objetos.

#### 1.4.2.2.2 Paso de Agrupación o Pooling

Permite una reducción en dimensionamiento de los mapas de características obtenidos en el proceso anterior conservando las características importantes detectadas por cada uno de los filtros, con el objetivo de reducir el proceso de cálculos en el interior de la red, el paso de agrupación más utilizado es el máx -pooling donde se toma el valor mayor del mapa de características, con lo cual se añade a la red, robustez ante transformaciones, distorsiones, traslaciones puesto que se está tomando el mayor número de todos los vecinos cercanos por ende la característica más relevante.

#### 1.4.2.2.3 Clasificación capa totalmente conectada

El objetivo de esta última capa con neuronas totalmente conectadas es determinar la clase de o los objetos detectados en la imagen de entrada considerando las características extraídas en los procesos anteriores. Finalmente, la estructura de una red neuronal convolucional se muestra en Figura 1.5. La explicación previamente realizada ha sido simplificada con el fin de una mejor comprensión, sin embargo, si se requiere de un mayor detalle en cuanto a la matemática y complejidad del entrenamiento de una CNN puede consultar en [10][11].

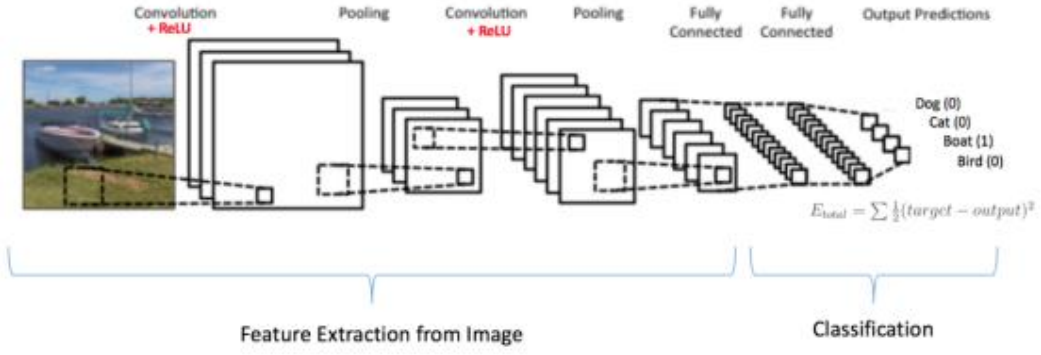


**Figura 1.5.** Estructura de Una Red Neuronal Convolucional

### 1.4.3 Algoritmo de Detección de Objetos

Consiste en identificar y localizar a un objeto que se encuentra presente dentro de una imagen, al mencionar identificar se refiere a clasificar al objeto entre las distintas clases para lo cual un modelo ha sido entrenado como son: personas, animales, vehículos, rostros, etc. La interpretación de localización se refiere a entregar la posición del objeto encerrándolo en un cuadro delimitador conocido también como “Bounding Box”, con sus respectivas coordenadas dentro de la imagen [12].

#### 1.4.3.1 Algoritmo Haar Cascade

Este algoritmo basa su funcionamiento en el uso de ventanas deslizantes como extractores de características “features” donde una ventana denominada también “kernel” de n x n dimensiones recorre toda la imagen de arriba hacia debajo de izquierda a derecha, entregando como respuesta un valor numérico en cada step o paso, producto de sumas y restas realizadas en cada píxel tomado por el kernel como se observa en Figura **1.6**, el proceso se lo detalló en la sección 1.4.2.2.1. Para lo cual, usa para su entrenamiento un dataset de imágenes positivas y negativas, es decir, imágenes que contiene o no el objeto a detectar respectivamente, en su lanzamiento fue entrenado con set de imágenes de rostros, sin embargo, puede ser entrenado para otra clase de objetos, consiguiendo de esta

forma valores umbrales que especifiquen el cuadro, dentro de la imagen original donde se ha encontrado el objeto de interés.

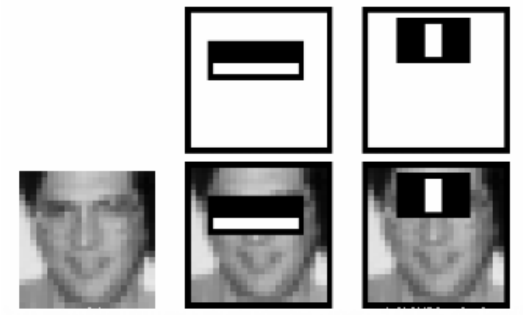

**Figura 1.6.** Extracción de Características Haar Cascade

El kernel recorre toda la imagen realizando cálculos matriciales y evidentemente usando recursos computacionales bastantes altos, motivo por el cual los creadores Paul Viola y Michael Jones [13], usaron un algoritmo adicional nominado AdaBoost, el cual permite realizar una serie de pruebas en cada paso del kernel por la imagen y al momento de detectar una falla, descartar inmediatamente la ventana o cuadro analizado y continuar con el resto, con el objetivo de detectar la ventana que contiene el objeto y descartar zonas irrelevantes, he allí el nombre de clasificador en cascada, trayendo un gran beneficio en cuanto a la extracción de características importantes con una alta velocidad de respuesta.

**Figura 1.7**. Respuesta Algoritmo Haar Cascade

Como se puede observar en Figura **1.7**, el kernel al recorrer la imagen presenta diversas ventanas con el objeto de interés en este caso rostros, para lo cual para obtener un solo cuadro se requiere la configuración de parámetros como MinNeighbors al momento de realizar la programación.

#### 1.4.3.2 Yolo

A diferencia de otros algoritmos basados en análisis de regiones como son R-CNN, F-RCNN, ventanas deslizantes, los cuales dividen en áreas determinadas a la imagen y aplican una CNN en cada una de ellas para detectar el objeto de interés, YOLO es un algoritmos más avanzado, siendo el pionero en el campo de la visión computacional debido

a que una única CNN es aplicada a toda la imagen de allí su nombre “You only look once”, obteniendo como resultado múltiples cuadros o cajas delimitadores que encierran al objeto detectado y además muestran las probabilidades con respecto a la pertenencia del mismo a las clases para las cuales el modelo han sido entrenados.

Básicamente su funcionamiento se basa en dividir a la imagen en cuadros de SXS dimensiones, y en cada uno de ellos dibujar n cuadros delimitadores, muchos de ellos con una probabilidad muy baja de pertenecer a una clase determinada, motivo por el cual valores de probabilidad que se encuentre por debajo de un valor umbral serán descartados. Los cuadros con el objeto detectado que lleguen a superar la primera etapa ingresan a un proceso de “non-max-supression”, en donde cuadros duplicados del mismo objeto con la menor probabilidad serán eliminados. El proceso antes mencionado que aplica dicho algoritmo se puede resumir en las imágenes de Figura **1.8**. [14]

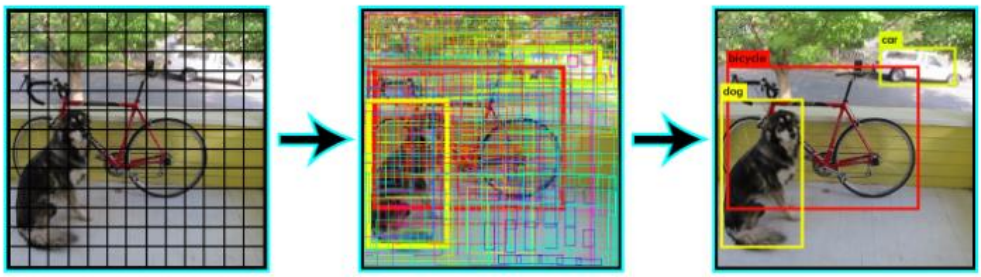


**Figura 1.8.** Respuesta Algoritmo de Detección de Objetos Yolo

### 1.4.4 Algoritmos de seguimiento de Objetos

El proceso de seguimiento o tracking de objetos consiste en estimar la trayectoria que un objeto puede seguir a medida que va pasando los frames o secuencia de video, otorgando al mismo un valor de identificación única que se mantendrá hasta que el objeto desaparezca del video, un algoritmo de seguimiento puede presentar problemas ante diversas circunstancias:

- Calidad de imágenes deficiente.
- Oclusión de objetos
- Velocidad de movimiento considerada de los objetos detectados.
- Iluminación del área de seguimiento.
- Perdida del objeto, ante una salida momentánea del mismo en el área de tracking.

### 1.4.4.1 Centroide Tracking

Como cualquier algoritmo de seguimiento de objetos requiere de una fase previa que es la detección de objetos, es decir identificación y localización del objeto de interés encerrado en un recuadro conocido como “Bounding Box” cuyos parámetros son (x, y, h, w).

$$(\boldsymbol{x}, \boldsymbol{y}): \boldsymbol{punto\ esquina\ superior}$$

$$(\boldsymbol{h}, \boldsymbol{w}): \boldsymbol{largo\ y\ ancho}$$

Al algoritmo de seguimiento Centro Tracking se alimenta con los parámetros enviados por el detector para que este realice su trabajo en dos pasos:

- Calcula el centroide de los objetos encontrados y asigna una identificación única, en el siguiente “frame” de acuerdo con el movimiento de los objetos, volverá a obtener la posición del centroide.
- Calcula la distancia Euclidiana entre los centroides encontrados, si esta es muy pequeña se podría considerar que se trata del mismo objeto, pues su movimiento ha sido mínimo, caso contrario se habla de otro objeto, por ende, el mismo algoritmo asigna una nueva identificación o mantiene la identificación anterior respectivamente. Su funcionamiento se lo podrá observar en Figura 1.9. [15]

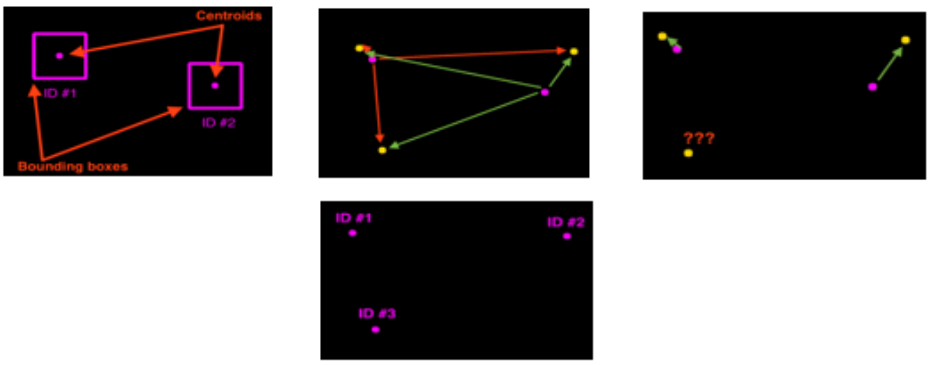


**Figura 1.9.** Descripción Gráfica Algoritmo de Seguimiento del Centroide

### 1.4.4.2 Deep Sort Tracking

Antes de explicar el algoritmo mencionado es necesario entender cómo trabaja su modelo previo de seguimiento conocido como “Sort – Simple Online Realtime Tracking”. Este algoritmo utiliza tres parámetros importantes “Bounding Box Prediction”, Filtro Kalman y técnicas de coincidencia IoU:

- Bounding Box Prediction: Consiste en generar marcos o cuadros delimitadores del objetivo a seguir, para lo cual se utiliza cualquiera de los algoritmos de detección

de objetos explicados anteriormente por ejemplo Yolo, cuyo resultado es mostrar en la imagen los objetos detectados con la etiqueta correspondiente, en donde resalte la clase de este con su valor de probabilidad de detección.

- Filtro Kalman: Maneja modelos de velocidad constante y distribución gaussiana cuya función es enviar un conjunto de ubicaciones probables a las cuales el objeto detectado tendrá a moverse “predicciones”. Al ser un algoritmo iterativo, dichas predicciones se actualizan de acuerdo con la información de entrada “detecciones realizadas” y poco a poco van mejorando, si se toma la probabilidad de ubicación con mayor valor se puede de alguna forma aproximarse a la ubicación que el objeto tendrá a moverse. Información matemática y detallada del filtro Kalman se la pueden encontrar en [16].

- Técnicas de coincidencia IoU: en español intersección sobre unión, la clasificación de objetos presenta información cuantitativa si retomamos el ejemplo del modelo de detección Yolo, el mismo genera varios cuadros delimitadores para un mismo objeto y cuya puntuación mayor encerrará perfectamente al mismo, en el caso del tracking se tiene un vector con los objetos captados en la imagen y otro con los identificadores o id ’s correspondientes, los cuales se irán emparejando de acuerdo a los objetos que se están rastreando, a cada id le corresponde una detección con una puntación IoU que debe ser maximizada para lo cual se utiliza un algoritmo conocido como “Hungarian algoritm”, resolviendo de esta manera el problema de asignación lineal.

De esta manera funciona el algoritmo de seguimiento “Sort”, si se establece en una secuencia de video prácticamente los tres pasos mencionados entran en un bucle de forma repetitiva dando lugar al tracking de objetos “Sort”. Ahora, el algoritmo Deep Sort es una extensión al algoritmo explicado anteriormente, en donde su nombre proviene de dos pasos extras llamados “Deep Appearance Descriptor” y “Cascade Matching”:

- Deep Appearance Descriptor: Es una CNN que está entrenada para detectar un objeto similar en diferentes imágenes, lo que quiere decir que dada una cantidad de imágenes “frames – hablando de un video” de un mismo objeto en diferentes vistas, determinar si es o no el mismo. Como entrada recibe el objeto recortado proporcionado por el detector de objetos y como salida proporciona un vector que codifica la información que está presentada en la imagen recortada, permitiendo de esta forma la comparación de esté con diferentes objetos. La puntuación del descriptor de apariencias la da una métrica conocida como “cosine distance metric”

y la del filtro Kalman la métrica llamada “Mahalanobis distance”, combinando dichas métricas se obtiene un puntaje escalar general que puede ser usado para asignar las detecciones a sus identificadores en el proceso anterior usando el “Hungarian algoritm”.

- Cascade Matching: es una extensión del IoU matching para el problema de asignación, misma que intenta y hace coincidir las ultimas detecciones con las últimas identificaciones y las detenciones anteriores con las identificaciones anteriores.

De igual forma los pasos extras junto con los anteriores entran en un bucle repetitivo al implementarlo en un video en tiempo real dando lugar al algoritmo detección Deep Sort, su estructura se puede observar en Figura 1.10 [17].

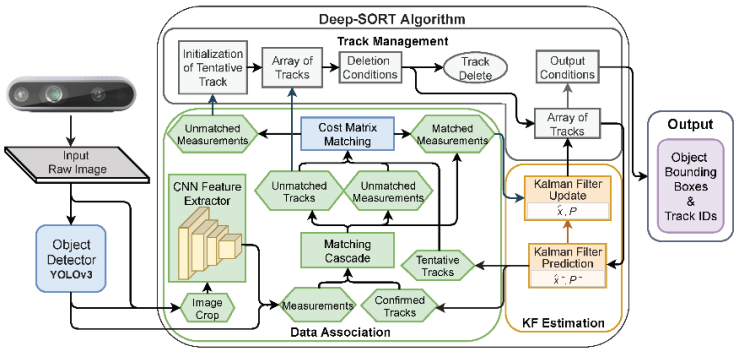


**Figura 1.10**. Estructura interna Algoritmo de Seguimiento Deep Sort

### 1.4.5 Hardware Adicional

En el desarrollo de proyectos de IA enfocados en el campo de visión computacional a implementarse en robots aéreos móviles “UAV’s”, se requiere de una tarjeta de control o miniordenador conocida como “SBC - Single Board Computer”, en la cual se ejecutará los algoritmos de procesamiento de imágenes (“algoritmos de detección y tracking de objetos, filtrado, transposiciones, etc.) y las respectivas acciones de control (“movimiento del mini-dron, movimiento cámara, etc.). En el mercado existe un sin número de dispositivos especializados en IA con la capacidad de ejecutar modelos de machine learning y Deep learning en tiempo real, entre las más populares se incluyen las tarjetas “Raspberry Pi, Jetson de NVIDIA, BeagleBone”.

Basándose en la literatura revisada con un enfoque en proyectos de visión artificial ejecutados en UAV's [18] [19] , se propone el uso de la tarjeta Jetson Nano J1020 , utilizada

anteriormente en proyectos de vigilancia en exteriores y actividades de registro de inventario de productos en interiores de grandes bodegas en EE. UU con alta precisión [20], la SBC descrita tiene las siguientes características.

**Tabla 1.1.** Características Jetson Nano J1020, NVIDIA

| Características | Descripción |
|---|---|
| GPU (Unidad de Tarjeta Gráfica): | NVIDIA Maxwell ™ de 128 núcleos |
| CPU: | ARM® A57 Quad-core |
| ALIMENTACIÓN: | Micro-USB 5V 2A o Adaptador de corriente DC 5V 4ª |
| CODIFICACIÓN DE VIDEOS: | 4K a 30 cuadros (H.264/H.265) |
| DECODIFICACIÓN DE VIDEOS: | 4K a 60 cuadros (H.264/H.265) |
| PUERTO MÓDULO WIFI: | Tarjeta WiFi de doble banda de 2,4 GHz/5 GHz y Bluetooth 4.2. |
| CONECTIVIDAD: | GPIO, I2 C, I2 S, SPI, UART |
| DIMENSIONES: | 100mmx80mm x29 mm |
| CONEXIÓN CÁMARA: | 12 vías (3 x 4 o 4 x 2) MIPI CSI-2 DPHY 1.1 (18 Gbps) |

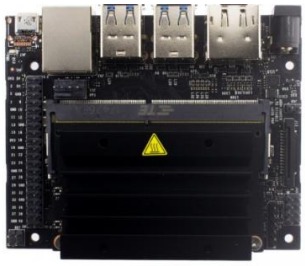

**Figura 1.11** Módulo Jetson Nano Marca NVIDIA

# 2 METODOLOGÍA

El objetivo del presente trabajo de investigación es desarrollar dos aplicaciones enfocadas en el campo de la visión computacional que posteriormente podrán ser implementadas en una SBC “Single Board Computer”, misma que alimentada con imágenes o video en tiempo real captadas por un mini - dron, permitirán que el mismo pueda realizar tareas varias como se mencionará posteriormente. La sección teórica con respecto al detalle de los algoritmos a utilizar fue descrita en el capítulo 1, por lo que en esta sección se procederá a un enfoque en cuanto a su diseño, comparación y experimentación con el fin de determinar la mejor opción de implementación del algoritmo en cuanto Software, para una futura ejecución física.

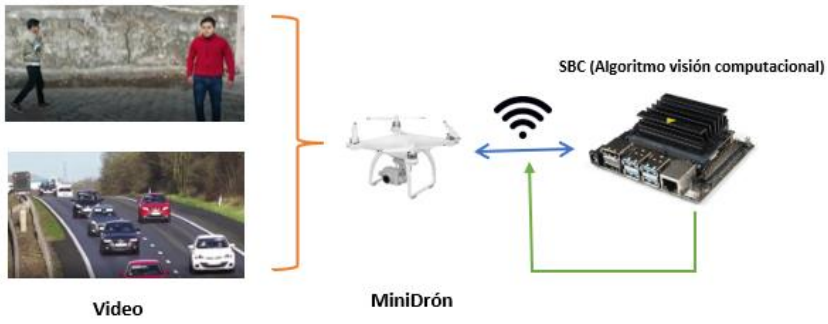


**Figura 2.1.** Diagrama General de Funcionamiento, Implementación Física

En la Figura 2.1 se puede observar a breves rasgos el funcionamiento del sistema a implementar en la cual, como primera aplicación se tiene el control de tráfico en una carretera concurrida en la cual por medio del sensor o cámara instalada en un mini - dron, se obtendrá imágenes o video en tiempo real a la cual se le fijara una región de interés (ROI) que tendrá que apuntar el mini -dron con el fin de realizar un conteo de vehículos y el registro en imagen de cada uno de ellos al pasar por dicha región, adicionalmente se podrá establecer una zona de aterrizaje, evidentemente en un área libre de circulación. Como segunda aplicación se realizar el seguimiento en particular de un objeto de interés en este caso una persona con un distintivo resaltante, que es un saco de color rojo.

El lenguaje de programación utilizado es Python, debido a que maneja un gran número de librerías de código abierto, además que ha sido adaptado por la mayoría de SBC's utilizadas en tareas de visión computacional y control, y sobre todo es el lenguaje por excelencia aplicado en tareas de IA.

Es importante mencionar que la primera aplicación tendrá un sentido más estático del mini – dron, puesto que este apuntará a una región fija y realizará lo mencionado, mientras que la segunda aplicación es más dinámica ya que el mini - dron tendrá que seguir a un objeto en particular. Es indispensable tener en cuenta que en ambos casos el movimiento del dron y la cámara no son enfoque en la presente investigación, sin embargo, se envía datos importantes para poder realizar dichas actividades.

## 2.1 Algoritmos de Visión Computacional o Artificial

En las dos aplicaciones antes mencionadas el objetivo primordial es detectar un objeto u objetos particulares en un "video", definido también como una secuencia de imágenes conocidos también como "frames", realizar el proceso de identificación de estos y seguirlos, además de obtener información necesaria para poder realizar las aplicaciones particulares.

Se puede tener una mejor visualización de lo escrito observando los diagramas de flujo planteados en Figura 2.2

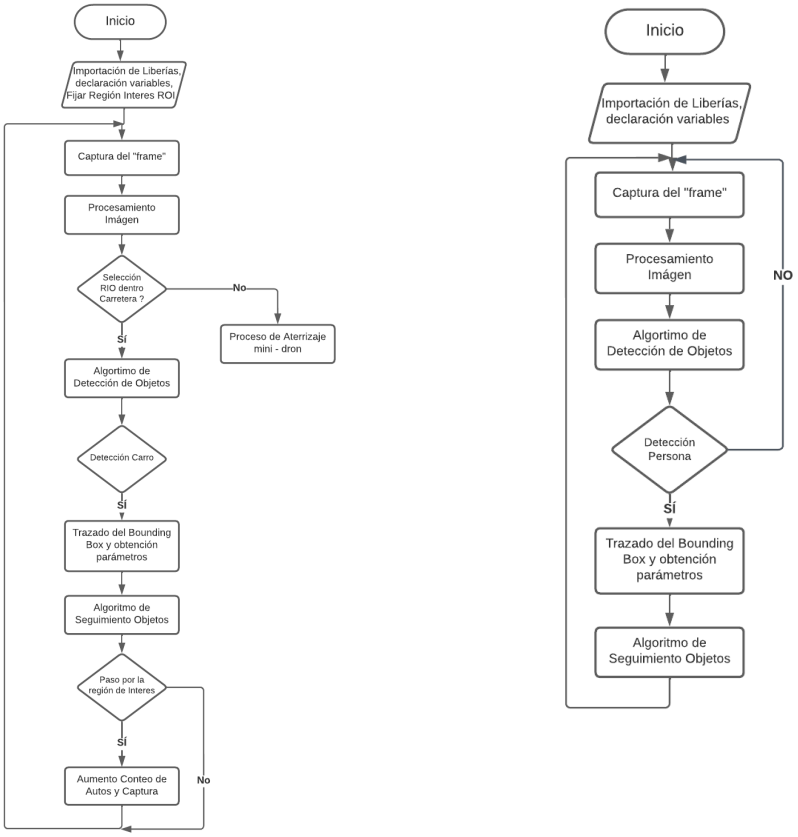


**Figura 2.2. a)** Diagrama general Aplicación 1 **b)** Diagrama General Aplicación 2

Para una mejor compresión cada una de las etapas mencionadas en los diagramas de flujo se explicarán a detalle en las siguientes secciones con el fin de comprender el funcionamiento de los algoritmos de visión computacional implementados.

### 2.1.1 Requerimientos Importantes

#### 2.1.1.1 Instalación Python, IDE, Librerías

Antes de iniciar con la programación es importante que se tenga Python instalado en el computador por lo cual se requiere descargar el mismo desde su página oficial Python.org [21], posteriormente es indispensable instalar un IDE “entorno de desarrollo integrado” de programación que puede ser jupyter, spyder, Pycharm entre otros, ambientes donde se desarrollará el código fuente. Una vez realizado lo mencionado se procede a la instalación las librerías desde el CMD “comand prompt” de Windows para lo cual se requiere visitar la dependencia pypi.org [22], repositorio donde se encuentran todas las librerías disponibles para Python y sus requerimientos. El proceso se muestra en Figura 2.3.

El comando *pip install "librería a utilizar"* es importante tenerlo presente para la instalación de cualquier librería que se requiera.

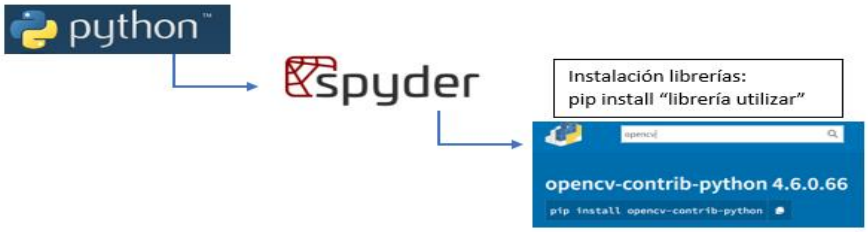


**Figura 2.3.** Requerimientos Iniciales, Python, IDE, Instalación Librerías

#### 2.1.1.2 Librería de Open CV

Open CV es una librería de código abierto desarrollado por Intel cuyo primer lanzamiento se realizó en el año 2000 y actualmente continua en constante desarrollo, esta librería es utilizado por estudiantes e investigadores que trabajan en proyectos enfocados en visión computacional, tales como proyectos de medicina; análisis de enfermedades cancerígenas utilizando imágenes de radiografías, seguridad; análisis y detección de objetos, robótica; evasión de obstáculos, seguimiento de trayectorias etc. [23]

La ventaja de esta librería es que permite trabajar con el lenguaje de programación Python, actualmente cuenta con alrededor de 2500 algoritmos que permiten la adquisición, análisis y tratamiento o procesamiento de imágenes utilizando algoritmos de inteligencia artificial, siendo la librería principal para el desarrollo del presente proyecto.[24]

Para la instalación de la librería de OpenCV se procede a ejecutar en el CMD el comando *pip install opencv-contrib-python,* con el fin de instalar el paquete completo con todos los algoritmos disponibles en OpenCV.

### 2.1.2 Adquisición de Video en Tiempo Real

Si se recuerda el diagrama de flujo Figura 2.2 , luego de inicializar librerías en ambas aplicaciones, comienza la etapa la adquisición de video o "captura de cada frame" proporcionada por la cámara del mini - dron, siendo el punto de partida de los algoritmos a implementar.

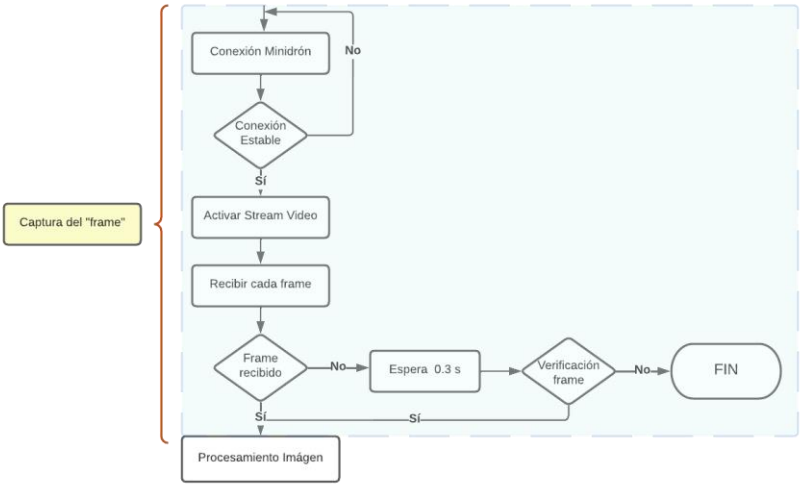


**Figura 2.4.** Diagrama General Proceso Adquisición Video

Por lo tanto, es indispensable verificar una conexión segura con el mini - dron desde la cámara a la tarjeta de control SBC utilizada, las imágenes pueden ser enviadas vía wifi, mecanismo que utilizan la mayoría de UAV's. Para la verificación del envío correcto de imagen se ha programado un bucle que determinará si llegó o no la imagen, en el caso de no tener respuesta de video o verificarse un "empty frame" el algoritmo no se ejecutará, caso contrario se procederá a la siguiente etapa perteneciente al procesamiento de la imagen. Lo mencionado se pude entender de mejor manera observando el diagrama presentado en Figura 2.4.

### 2.1.3 Procesamiento de Imagen - Video

Esta etapa consiste en redimensionar el tamaño del video de entrada, es decir, de cada frame captado por la cámara del mini - dron, a una resolución que acepte cualquiera de los algoritmos de detección de objetos a utilizar, teniendo en cuenta adicionalmente, la resolución de la ventana de visualización que podrá tener el usuario final. Básicamente un cambio de resolución consiste en disminuir o aumentar la cantidad de pixeles tanto en ancho como en alto de la imagen original.

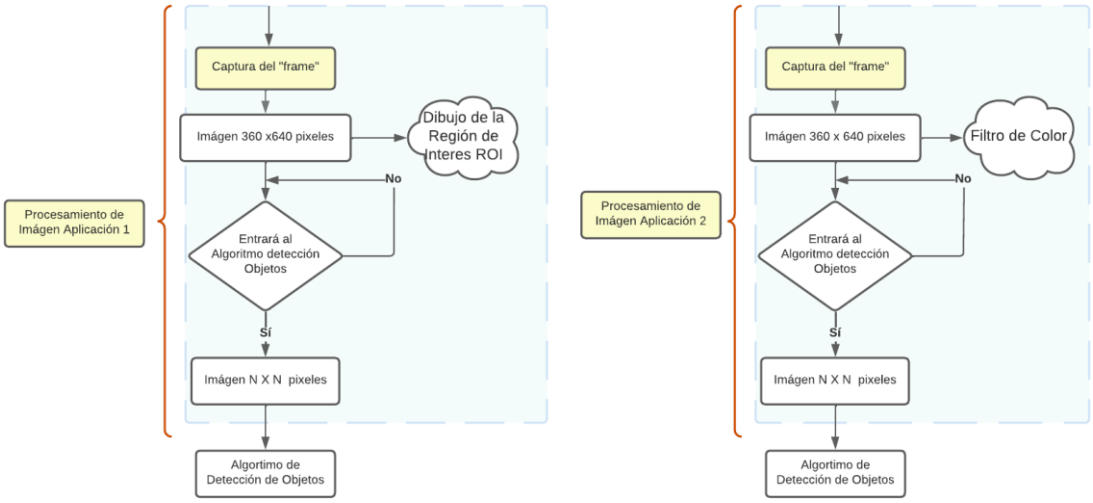


**Figura 2.5.** Diagrama General Procesamiento Imagen – Video

Como se puede observar en el diagrama de flujo Figura 2.5, la imagen “Captura del frame” obtenida en la etapa anterior tendrá una determinada resolución dependiendo de la cámara montada en el mini - dron, en el mercado existen UAV’s cuyas cámaras proporcionan imágenes en resolución de HD “1080p”, 720p, etc. Ofreciendo una alta calidad de imágenes en cuanto a nitidez.

Sin embargo, existe un problema al trabajar con resoluciones grandes, como se mencionaba una imagen en Open CV se la trata como una matriz de pixeles, en la cual se debe encontrar a un objeto de interés que en muchos casos ocupa un lugar pequeño en el área total de la imagen, si consideramos una imagen con una resolución de 1080p (1920 x 1080 pixeles) la imagen tendrá un área de 2073600 pixeles de trabajo, donde el algoritmo de detección de objetos a utilizar tendrá de acoplarse implicando una mayor carga computacional considerando una gran cantidad de operaciones matriciales que tendrá que realizar. Es por este motivo que cada algoritmo de detección acopla la imagen captada a una resolución más pequeña N X N antes de entrar a su red, si se toma en cuenta el algoritmo de Yolo V4, el mismo redimensiona la imagen a una resolución de (416 x 416 pixeles), como se pude observar en Figura 2.6, el comando para redimensionar una imagen es *cv2.resize(img, (pixeles ancho, pixeles alto)).*

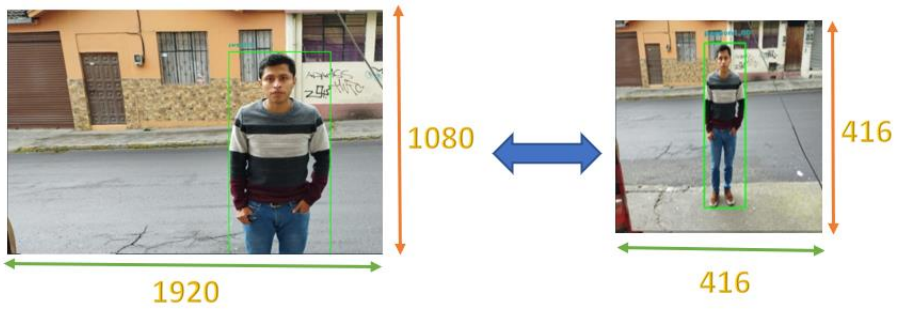


**Figura 2.6.** Redimensionamiento Imagen de Entrada adecuada para Algoritmo Yolo a) Visualización usuario final b) Redimensionamiento usado por Yolo

Yolo dentro de su red redimensiona la imagen a una resolución cuadrada perdiendo un poco de perspectiva en la imagen, sin embargo, el entrenamiento al que fue sometido le permitirá tranquilamente realizar la detección de objetos sin problema, la desventaja es la presentación de esta al usuario final, como se muestra en la Figura 2.6, ya que, no podrá tener una visualización adecuada del resultado del algoritmo, se puede evidenciar adicionalmente, que la resolución de la imagen de entrada es superior a la resolución de la pantalla de visualización del usuario " Pantalla PC" , motivo por el cual, la imagen original no se muestra completa , sin embargo, el algoritmo de detección es capaz de identificar el objeto.

Para solucionar el inconveniente de visualización al usuario final, se requiere redimensionar a la imagen desde el momento inicial antes de ingresar a la red a un formato de 640 X 360 pixeles, como se podrá observar en Figura 2.7. Cabe aclarar que la velocidad de respuesta del algoritmo de detección depende del área de trabajo de la imagen de entrada es decir su resolución, si esta es pequeña, entonces, la respuesta será más rápida caso contrario será más lenta.

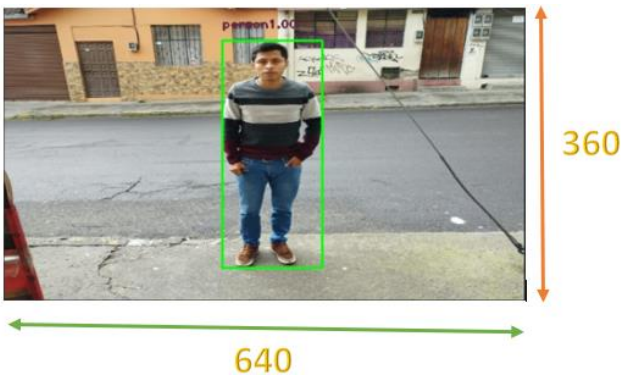


**Figura 2.7.** Redimensionamiento realizado - Visualización usuario final

Como se mencionaba en el desarrollo teórico el procesamiento de imagen no solo consiste en redimensionar la misma, sino realizar un respectivo tratamiento de esta, cuya dificultad

dependerá de la aplicación a implementar. En la primera aplicación por ejemplo se requiere establecer y dibujar una región de interés, mientras que en el segundo caso un filtrado de color es necesario, el tratamiento de la imagen se realizará en aquella ventana con la resolución que será mostrada al usuario (360 x 640), no a la que entrará a la red de detección de objetos, por motivos que será posteriormente explicados.

#### 2.1.3.1 Selección ROI (Región de Interés)

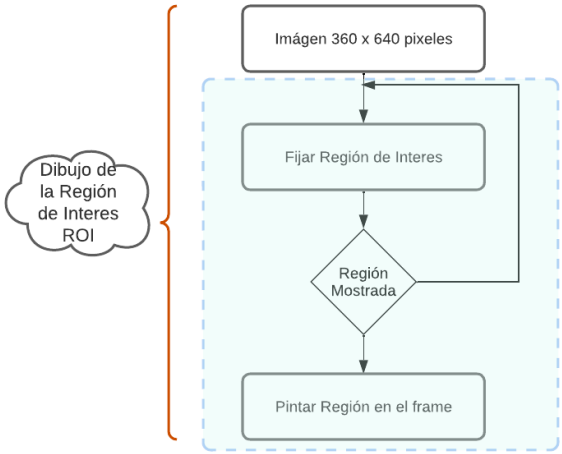


**Figura 2.8.** Diagrama selección de una región de interés "ROI"

El dibujo de la región de interés consiste básicamente en fijar un área dentro de la ventana de visualización del usuario con el fin de establecer una región donde se va a realizar alguna acción determinada, en el caso de la primera aplicación, el conteo y captura de autos que circularán por dicha área para el control de tráfico o también fijar un área de aterrizaje para el mini - dron. Establecer un ROI se puede extender a muchas otras aplicaciones como a temas de vigilancia, en donde se podrá captar personas que han cruzado por alguna área restringida o en el campo de la agricultura para determinar el número de plantas existentes en un área determinada dentro de un terreno etc.; ejemplares que se pude apreciar en las imágenes Figura 2.9.

El comando utilizado para el trazado de regiones de interés es *cv2.polylines(img, np. array [ROI], Figura_Cerrada=True, color, grosor línea).*

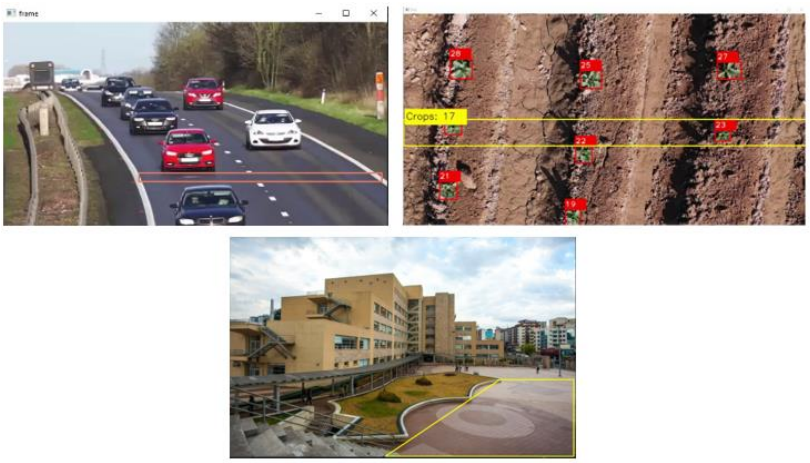


**Figura 2.9.** Aplicaciones varias, utilizando la selección de una región de interés "ROI"

Es importante aclarar que la Región de interés debe ser un parámetro declarado de forma de vector, cuyos componentes deben ser los vértices de la figura que se desea dibujar. Estos pueden ser declarados manualmente al inicio del código, si se considera un video previamente grabado pues se conoce con anterioridad la resolución de imagen que se va a trabajar y posteriormente utilizando algún programa de edición gráfica como Paint se puede hallar los vértices de la ROI como se muestra en Figura 2.10. De otro modo al trabajar con imágenes en tiempo real se puede trazar la ROI a cualquier instante directamente en la ventana de visualización del usuario, ambos casos han sido desarrollados he implementados.

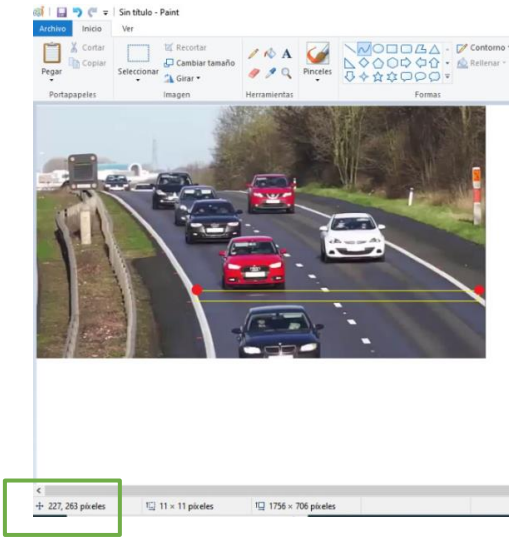


**Figura 2.10.** Trazo manual ROI, utilizando Paint (Video Pregrabado)

En el primer caso para trazar la ROI de forma manual se ha utilizado Paint donde, en el recuadro verde inferior, se pueden observar las coordenadas en pixeles de la imagen mostrada, efectivamente esta debe estar en la resolución de la ventana que se mostrará al

usuario, en este caso la imagen tiene una resolución de (360 x 640) pixeles, por lo tanto, todos los vértices de la figura a trazar deben estar dentro de este rango. Para hallar los vértices, se ha colocado el cursor sobre puntos específicos dentro de la imagen original, dando como resultado el vector (2), de igual forma para un mejor entendimiento se ha graficado el diagrama de flujo Figura 2.8.

$$ROI = [(227,263), (630,263), (630,278), (227,278)] \ [2]$$

En el segundo caso se traza la ROI con el mouse, directamente en la ventana de visualización del usuario como se muestra en Figura 2.11. Una vez trazada se puede continuar con el desarrollo del algoritmo.

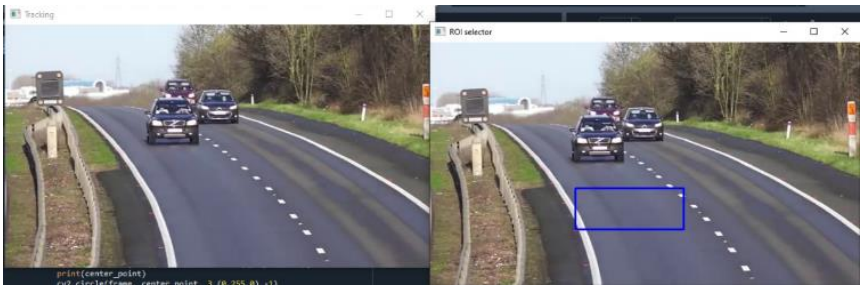


**Figura 2.11.** Trazo ROI, Video en tiempo Real

### 2.1.3.2 Filtrado de Color (HSV)

Dependiendo del proyecto a implementar, se realizar el tratamiento de imagen correspondiente, por lo tanto, para la segunda aplicación se requiere identificar una persona entre varias, misma que lleva un distintivo particular en este caso un saco de color rojo, el diagrama de flujo de Figura 2.12, muestra una idea del proceso a realizar.

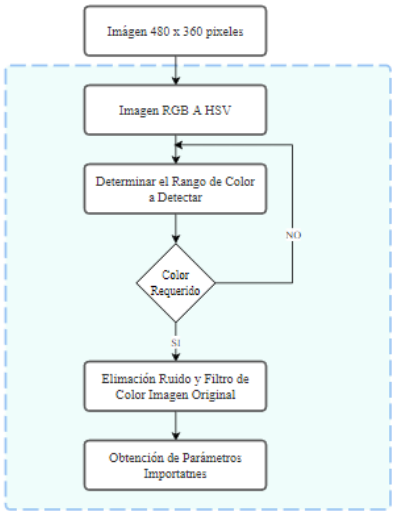


**Figura 2.12**. Diagrama General Filtrado de Color

Como se puede observar se inicia el proceso con la imagen redimensionada “ventana de visualización del usuario (360 x 640), ya que, el trabajo de detección de objetos funciona de forma independiente del filtrado.

Una imagen puede representarse en varios formatos RGB, BGR, CMYK, entre otros, sin embargo, Open CV maneja las imágenes en formato BGR (azul, verde, rojo), teniendo en cuenta que la mezcla de estos tres colores principales en diferente proporción da lugar a cualquier color. Sin embargo, para el proyecto se utilizará el espacio de color HSV con el fin de poder detectar de manera más sencilla el rango de color deseado.

HSV representa a la imagen de forma análoga a RGB, basando en tres componentes “H: Matiz, S: Saturación y V: Brillo”, su representación puede ser entendida tomando en cuenta la Figura 2.13.

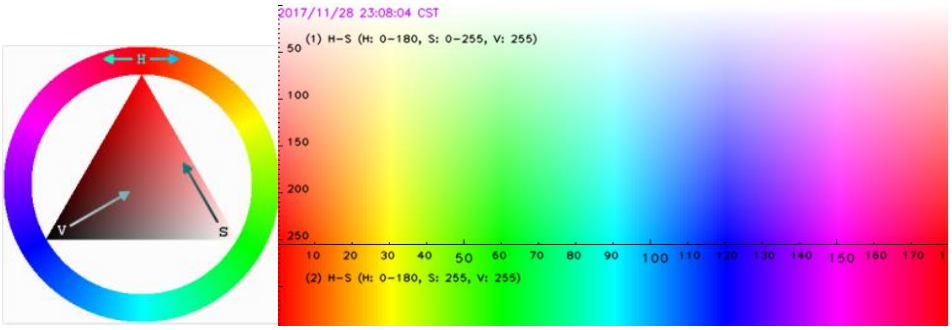


**Figura 2.13.** Representación espacio de color HSV

- H (Tinte o Matiz): este parámetro es muy importante, pues de este depende el color a filtrar, como se pueden notar el mismo comienza en color rojo y en forma circular recorre toda la gama de colores, HSV maneja esta representación en valores numéricos representados en un rango de 0 a 180.
- S (Saturación): representa la pureza del color o más claro la intensidad del color seleccionado, HSV maneja esta representación en valores con un rango de 0 a 255, siendo 0 un nivel de intensidad nula es decir prácticamente blanco y 255 la representación más intensa del color.
- V (brillo): básicamente representa el nivel de luminosidad del color, haciendo un símil este parámetro representará a la luminosidad del ambiente, HSV maneja esta variable en valores de 0 a 255, siendo 0 luminosidad nula “opaca” y 255 luminosidad “clara”.

Teniendo en cuenta estos tres parámetros se puede filtrar cualquier color considerando niveles de intensidad y luminosidad que van de la mano con las condiciones ambientales en donde el mini - dron estará captando imágenes.

El proceso de filtrado se efectúa cuatro etapas descritas a continuación:

- Transformación BGR A HSV: proceso de transformación de un espacio de color a otro como se puede observar en la Figura 18, Open CV lo realiza ejecutando simplemente el comando *cv2.cvtColor(img, cv2COLOR_BGR2HSV)*

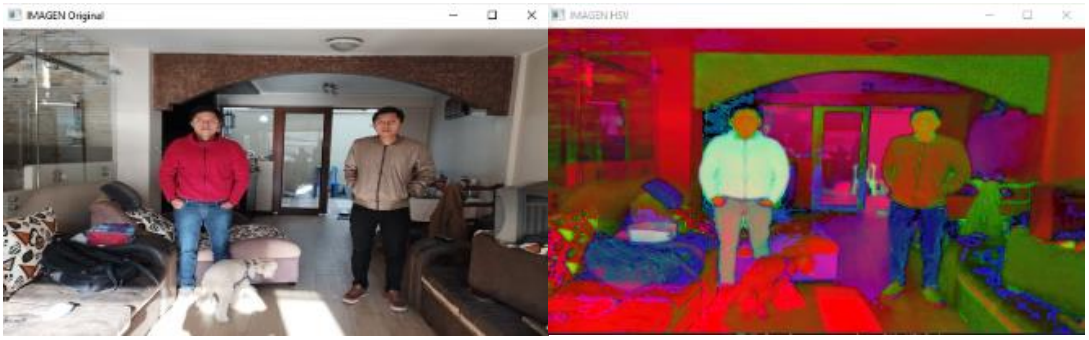


**Figura 2.14.** Conversión, espacio de color RGB – HSV

- Creación Mascará: Realizando una variación de los parámetros HSV se puede obtener cualquier color a filtrar, su respuesta dará valores de 0 o 1 por lo tanto, cada píxel de la imagen original se comparará con el rango de color seleccionado y en el caso de encontrarse dentro del rango, tomará el valor de 1, prácticamente el pixel se pintará de blanco o de lo contrario será 0 lo que significa que se pintará de negro como se muestra en la Figura 2.15. Para mayor facilidad se ha desarrollado un programa externo, para encontrar el rango de cualquier color deseado, mismo que se encuentra disponible en Anexo1. El comando de Open CV a ejecutar para realizar lo mencionado es *cv2.inRange(img, color_rango_low, color_rango_high).* Para la aplicación a implementar el rango de color rojo en su nivel bajo y alto aceptable es el siguiente:

*rojo_low= [156,92,0]*

*rojo_high = [179,255,255]*


**Figura 2.15.** Creación Máscara de la Figura 2.14

Como se ve en la imagen resultante existen pequeñas áreas de color blanco o ruido que posteriormente será eliminado. Para verificar un correcto filtrado, se procede a realizar un and lógico entre la máscara hallada y la imagen original dando como resultado lo mostrado en la Figura 2.16. El comando en Open Cv es el siguiente, *maskRedvis = cv2.bitwise_and (img, img, mask = maskRed).*


**Figura 2.16.** Imagen con ruido, filtro de color

- Se procede a eliminar el ruido para evitar falsos filtros de color por ejemplo si se tiene en el ambiente otros objetos que tengan colores pertenecientes al rango de color a filtrar y estos no son representativos serán eliminados, considerando el área de color encontrada, lo que es lo mismo decir, si el área encontrada es demasiado pequeña pues no será tomada en cuenta caso contrario sí. En una imagen en donde se encuentre presente la persona con el saco de color rojo, efectivamente el algoritmo planteado resaltará el área mayor, que en cuyo caso, representa es el color filtrado y se eliminará el resto “ruido”, lo mencionado se puede observar en la Figura 2.17, en la cual se ha dibujado un contorno en la imagen original donde resalte el color filtrado y se elimine el resto de la imagen con colores similares.

Para dibujar el contorno se ejecutó el comando *cv2.findContours(mask, cv2.RETR_EXTERNAL,CV2.CHAIN_APROX_SIMPLE)* y para suavizarlo *cv2.convexHull(contorno).*


**Figura 2.17.** Dibujo contorno, correspondiente al color filtrado

- Finalmente se procede a encontrar un parámetro importante que es el punto céntrico del contorno encontrado, mismo que será útil en el proceso de identificación de objetos y filtrado de clases sección 2.1.4.2, el cual será explicado a su tiempo, como se muestra en la Figura 2.18. [25][26]


**Figura 2.18.** Obtención del punto céntrico del contorno encontrado

### 2.1.4 Identificación – Detección de Objetos

En este apartado se desarrollan la implementación de los algoritmos de detección de objetos planteados en la sección 1.4.3, donde se explicaban brevemente su funcionamiento, con la finalidad de determinar la mejor opción en cuanto a su aplicación. Los algoritmos fueron desarrollados y ejecutados en una PC portátil con el fin de observar su comportamiento para una posterior implementación en alguna tarjeta de control SBC. Justamente esta etapa del proceso de diseño del algoritmo es la más importante, pues si no se conoce al objeto de interés a detectar las etapas posteriores de tracking y aplicación

general no tendrían sentido, además de ser una etapa de mayor consumo computacional, puesto que la computadora tendrá que estar realizando un sin número de operaciones matriciales con las imágenes en cada frame, propio en una red neuronal convolucional.

Los 2 algoritmos de detección de objetos a considerar y sobre todo utilizado en grandes aplicaciones de visión computacional son: Haar Cascade y YOLOv4, 5.

#### 2.1.4.1 Implementación Algoritmo Detección Haar – Cascade

Inicialmente se usa un algoritmo propio de la librería de Open CV, el cual es capaz de detectar un objeto con características muy similares a las que fue sometido en su etapa de entrenamiento, pues su funcionamiento se basa en extraer características de una imagen, compararlas y clasificarlas, teniendo en cuenta que para cada objeto existe una cascada de características específicas. Para las aplicaciones a realizar se requiere clasificar personas y carros, por lo cual, para comprobar el funcionamiento del algoritmo, inicialmente se procede a elegir el conjunto o cascada de características correspondiente a cuerpos enteros, mismos que ya fueron pre – entrenado por la librería de Open CV aplicando diversas técnicas de machine Learning.[27]

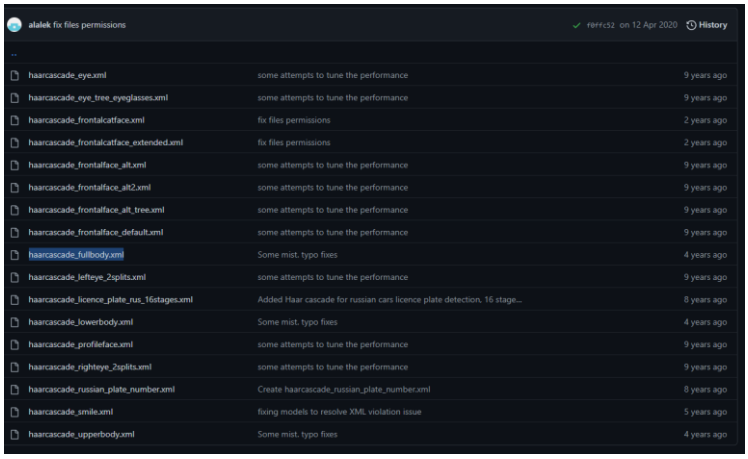


**Figura 2.19.** Cascadas existentes por defecto en la librería de Open CV

Para lo mencionado se requiere el archivo de detección de cuerpos que se lo puede descargar de la página oficial de Open CV Haar Cascade [28], mismo que se encuentra en formato XML. La ventana de las distintas cascadas disponibles se puede observar en la Figura 2.19. La ventaja de este método es que no requiere de un framework para su funcionamiento, simplemente se lo descarga, se lo incluye en el código con su respectiva configuración de parámetros y finalmente se lo ejecuta en una imagen o video en tiempo real, usando el comando: *cv2.CascadeClassifier ("haarcascade_fullbody.xml").*

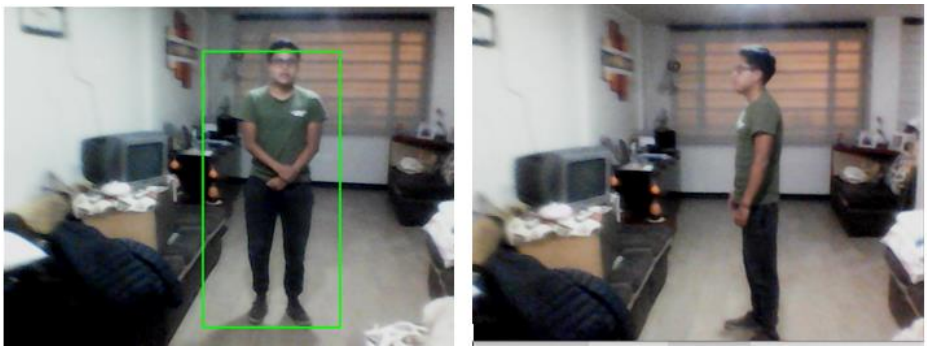

**Figura 2.20.** Respuesta del algoritmo Haar Cascade al detectar un cuerpo entero a) Detección Correcta y dibujo del Boundig Box b) Detección Incorrecta al menor movimiento del objetivo

Como se puede observar en la Figura 2.20, los resultados de este algoritmo, en cuanto a detección de personas, es bastante susceptible a fallas, muchas de las veces no son capaces de detectar cuando la persona cambia de postura o realiza algún movimiento; tiende a realizar buenas detecciones únicamente cuando el individuo se encuentra frente a cámara, lo cual para la aplicación de seguimiento a una persona particular no es adecuada. Además, es indispensable conocer que, si no se cuenta con la cascada clasificadora del objeto deseado; que en el caso de una de las aplicaciones son carros, se debe entrenar el modelo con una data de imágenes muy grande, acompañado de un proceso bastante tedioso que no es complicado, sin embargo, los resultados en cuanto a funcionamiento y respuesta del algoritmo van a ser muy similares a los obtenidos con personas.

#### 2.1.4.2 Implementación algoritmo de detección Yolo

Yolo es uno de los algoritmos de detección y clasificación objetos más utilizados en aplicaciones de visión computacional, pues utiliza redes neuronales convolucionales para identificar una gran variedad de objetos para los que fue entrenado como se observa en la Figura 2.21. Existen varias versiones que han ido en continua mejor en cuanto a su capacidad de exactitud, nivel de confianza en la detección y velocidad de respuesta, existe su versión completa Yolov1 ,v2 , v3, v4 con las respectivas versiones más ligeras conocidas como “tiny”, su diferencia radica en el menor número de capas convolucionales que utiliza en su red, lo cual evidentemente aumentan la velocidad de respuesta, sin embargo, decrementa la confianza en cuanta a la detección y clasificación de objetos [29]. En el proyecto se ha experimentado con versiones de Yolov4 en su versión completa y Yolo v5 en su versión tiny con el fin de observar la capacidad que tiene esta red, sin embargo, los ejemplos mostrados a continuación se han realizado con yoloV4.

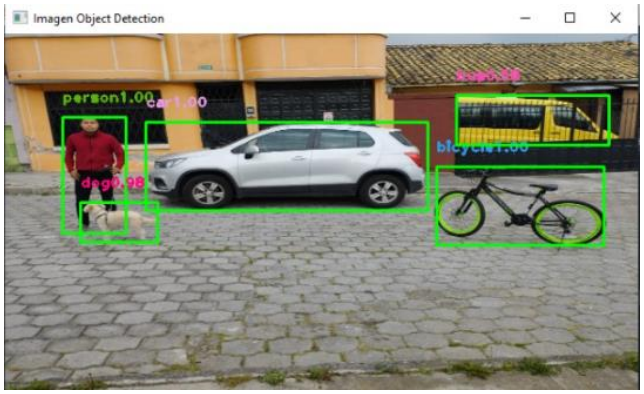


**Figura 2.21.** Detección de distinta clase de objetos utilizando Algoritmo Yolo v4

De igual forma que el algoritmo de detección de objetos anterior es una red pre – entrenada, para su ejecución requiere de un framework como Tensorflow que permite el trabajo con redes neuronales multicapas, un ejemplo que muestra la capacidad de esta red se observa en la Figura **2.22**, donde se puede notar la identificación de varios objetos en imágenes tomadas desde alturas significativas, importante para el desarrollo del proyecto, ya que estas serán tomadas de la cámara de un mini - dron. Las ejemplos mostrados fueron captados desde una altura aproximada de 9 metros, donde claramente se han identificado carros y personas con un porcentaje de confianza bastante alto, cabe resaltar el área pequeña que ocupan las personas en comparación al área total de pixeles, y aun así la red ha identificado correctamente.


**Figura 2.22.** Respuesta del Algoritmo de detección Yolo v4, en imágenes tomadas a alturas significativas

Los resultados arrojados por esta red básicamente es un tensor que se compone de tres parámetros importantes en forma de vector, datos vitales para poder realizar una posterior filtrado de clases, y delimitación de objetos.

*[[class_ids], [scores], [boxes]]*

Es importante mencionar que cualquier algoritmo de detección de objetos que emplee redes neuronales convolucionales multicapa, requiere de descarga de importantes archivos que se los puede encontrar en el repositorio de GitHub [30] , lugar donde se ha cargado los archivos necesarios y requisitos indispensables “librerías adicionales” proporcionado por el creador de cada algoritmo, mismo que sin conocimiento previo no deben ser modificados puesto que influyen directamente en su funcionamiento como son:

- Archivo de origen de Configuración: corresponde a un archivo de tipo .cfg, el cual contiene tota la información de la red neuronal, como capas de convolución, iteraciones que va a realizar, funciones de activación de cada neurona, etc; en sí es la fuente del algoritmo de detección.
- Clases: archivo tipo .txt, con los nombres de las clases que es capaz de detectar el algoritmo.
- Pesos: archivo tipo. weigths, el cual como se explicó en la parte teórica, toda red neuronal requiere de pesos “weights”, que provienen del entrenamiento previo de la red.
- Requeriments: es un archivo tipo.txt donde se encuentran todas las librerías usadas por la red que previamente deben ser instaladas, con sus respectivas versiones. Figura 2.23.

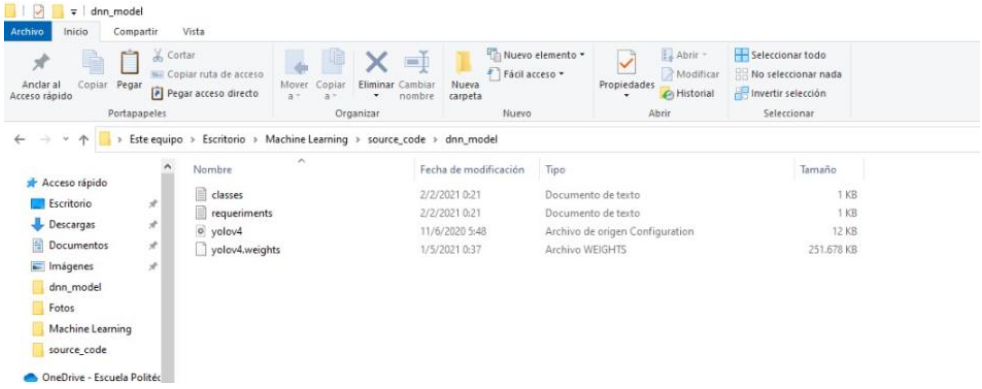


**Figura 2.23.** Archivos importantes dependiendo del algoritmo de detección de objetos a utilizar

### 2.1.5 Filtrado de Clases Específicas

Descartando directamente el algoritmo Haar cascade por su ineficiencia, se procede a realiza un filtrado de clases considerando las detecciones del algoritmo Yolo, de acuerdo con la aplicación a implementar, en el proyecto se requiere en primer caso filtrar carros y en el segundo caso personas o pueden ser ambas en una aplicación en conjunto.

YOLO fue entrenada para detectar 80 diferentes clases de objetos en una imagen con un dataset de aproximadamente 330k de imágenes “coco.dataset” [31], obteniendo los pesos mencionados anteriormente. Sin embargo, esta red por sí sola no permite realizar un filtrado de clases, puesto que, entrega la imagen de entrada con todas las etiquetas de objetos detectados al mismo tiempo, lo cual no es adecuado para las aplicaciones a implementar.

Motivo por el cual, aprovechando el vector de class_ids dentro del tensor arrojado como resultado del algoritmo YOLO, en el cual se guardan con un identificador de 0 a 80 correspondiente a las clases detectadas, se procede a compararlo con un vector inicialmente declarado en el código, correspondiente a las clases permitidas “allowed_objects”. En la Figura 2.25a, se tiene el vector class_ids, elemento del tensor arrojado por YOLO, con los índices correspondientes a los objetos encontrados Figura 2.25b, donde se puede evidenciar 5 valores numéricos, los cuales si se compara con los elementos del bloc de notas pertenecientes a las clases que detecta el algoritmo YOLO, corresponde exactamente a los índices de los objetos detectados “enumerados de arriba hacia abajo”, un vector similar se declara inicialmente en “allowed_objects” que en este caso tendrá los índices 0 y 2 o [“person” , “car”]

*Tensor = [**[class_ids],** [scores], [boxes]]*

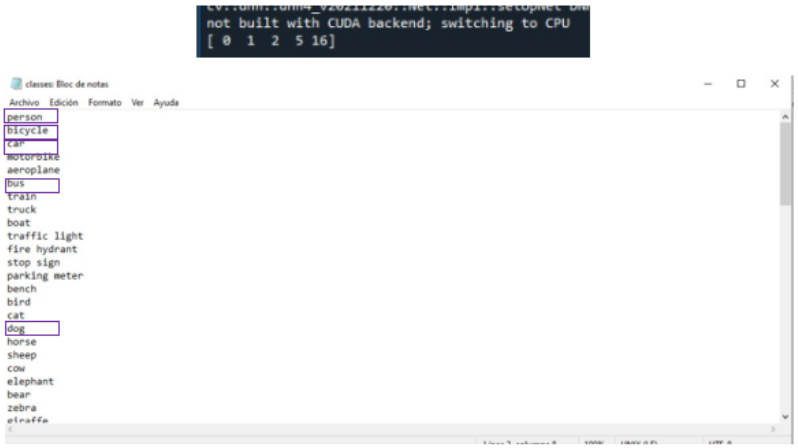


**Figura 2.24.** a) Tensor Respuesta del Algoritmo Yolo Figura 2.25b b) Archivo txt, Coco - datase, encerrados los índices respectivos de los objetos encontrados Figura 2.25b

Una vez realizada la comparación se obtiene el Bounding Box correspondiente y las etiquetas resultantes con los objetos filtrados, como se pueden evidenciar en la Figura 2.25

**Figura 2.25**. a) Filtrado de clases personas b) Filtrado de Clases carros

Como se pueden observar en las imágenes anteriores se ha detectado distintas clases de objetos (personas, bicicleta, perro, micro – bus), sin embargo, al ser comparadas con el vector de objetos permitidos que en el primer caso es [“personas”] y en el segundo caso [“carro”], se ha podido filtrar los objetos deseados.

### 2.1.6 Dibujo y delimitación de los objetos encontrados

Consiste básicamente en trazar el cuadro delimitador que encierre el objeto detectado con su respectiva etiqueta y nivel de confianza conocido también como “Bounding Box”, para lo cual es indispensable conocer ciertas coordenadas entregadas en forma directa en uno de los vectores del tensor proporcionado por el algoritmo de detección de objetos YOLO.

*[[class_ids], [scores],* ***[boxes]]***

[boxes], es vector o “array” de 4 elementos [x, y, w, h] que contiene inicialmente los parámetros (x, y) correspondiente a la coordenada en pixeles de la esquina superior izquierda del cuadro donde se encontrará el objeto detectado, mientras que (w, h) corresponde al ancho y largo de dicho recuadro. Con datos arrojados por el algoritmo YOLO se puede dibujar el cuadro delimitador con el comando *cv2.rectangle (img, cor_sup_izq, cor_inf_der, color,grosor),* y finalmente el texto correspondiente a la clase o “etiqueta” y nivel de confianza con el comando *cv2.putText (img, text, origen, fuente, relleno_letra, color, tamaño).*

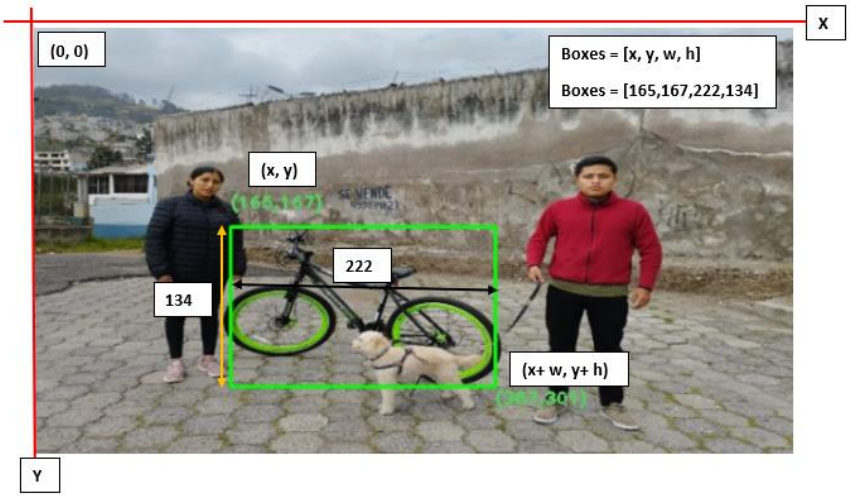


**Figura 2.26.** Coordenadas Bounding Box, objeto detectado (Bicicleta) [x, y, w, h]

El comando para trazar el cuadro delimitador requiere de dos parámetros importantes, la coordenada superior izquierda y la coordenada inferior derecha, YOLO ya entrega la primera coordenada en los dos primeros elementos del vector [boxes] = (x, y), los cuales en el caso del objeto filtrado bicicleta corresponde a (166,167) y es una coordenada aceptable puesto que la imagen al tener una resolución de 640 pixeles de largo y 360 pixeles de ancho, cualquier par ordenado localizado en el interior del mismo debe variar en un rango de (0-640, 0,360) siendo la coordenada de origen (0,0) el punto superior izquierdo de la imagen, con lo mencionado el otro punto del Bounding box, simplemente se obtiene sumando a la coordenada (x ,y) los otros dos parámetros restantes del vector [boxes] correspondientes a (w , h), por lo tanto en este caso resulta (x + w, y+ h) = (387,301), se podrá entender de mejor manera observando cada uno de los valores mostradas en la Figura 2.26.

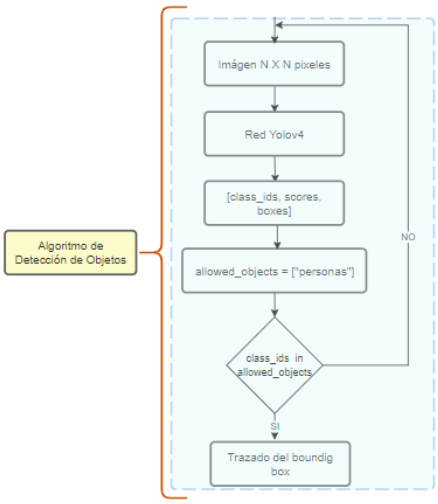


**Figura 2.27.** Diagrama General Algoritmo de Detección de Objetos - Filtrado de Clases

Para una mejor compresión del algoritmo de detección de objetos juntamente con el filtrado de clases y el trazado de Bounding Box, se propone el diagrama de flujo mostrado en la Figura 2.27, donde como resultado se tendrá la imagen o frame del video con los objetos deseados, en la primera aplicación se requiere filtrar carros, mientras que en la segunda personas. Cabe aclarar que, para la segunda aplicación se requiere de un proceso posterior y en la primera es suficiente con una simple detección de objetos, puesto que el segundo caso aparte de filtrar personas se debe delimitar a únicamente individuos con un distintivo particular en este caso se requiere realizar el tracking a una persona que lleve en particular un saco de color rojo, por lo tanto, en este punto es de vital importancia un segundo filtrado usando el filtro de color descrito en la sección 2.1.3.2, y el punto céntrico hallado del contorno de color deseado.

### 2.1.7 Procedimientos Adicionales

Para un posterior tracking de un objetivo en particular, en la sección 2.1.3.2,, se realizó un procesamiento de imagen centrado en el filtrado de color cuyo resultado fue el punto céntrico dentro de un contorno cerrado que delimitaba en la imagen la mayor concentración de color rojo característico de la prenda distintiva que llevará la persona a seguir como se observa en la Figura 2.28.


**Figura 2.28.** Contorno cerrado que encierra el área de mayor concentración de color rojo (Prenda Distintiva)

El punto céntrico hallado es un dato de importancia relevante, pues será tomado en cuenta para filtrar aún más las detecciones realizadas por la red YOLO en donde ya se realizó un filtrado de clases previo correspondiente a personas. Se propone el diagrama de la Figura 2.29, donde se combina el proceso de filtrado de color y el filtrado de clase en la aplicación que se requiera.

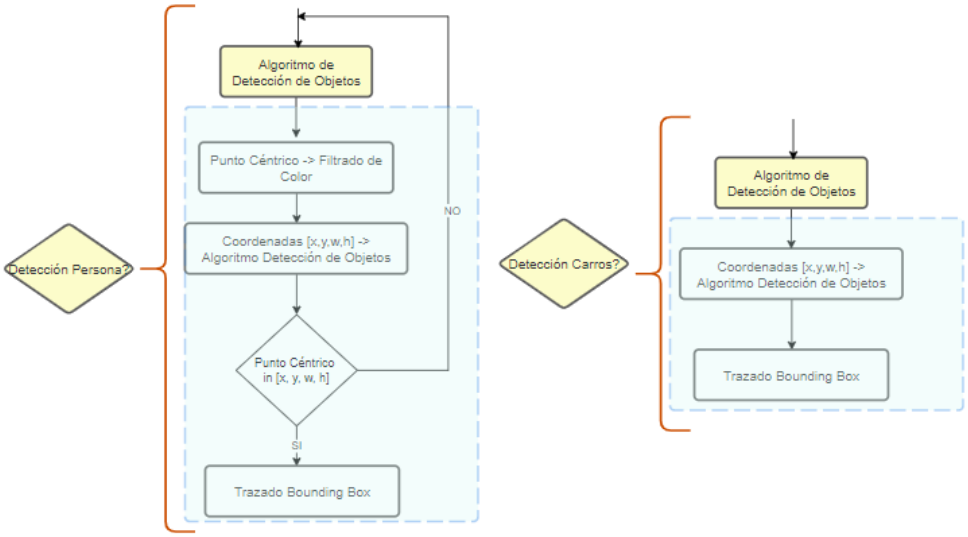


**Figura 2.29**. a) Diagrama que combina Algoritmo de detección, filtrado de clases y color (Aplicación 1) b) Diagrama correspondiente solo a Algoritmo de detección (Aplicación 2)

La red YOLO una vez realizado el filtrado de clases mostrará la imagen o frame de video con todas las personas detectadas encerrados en su respectivo Boulding Box, de las cuales previamente se conocen todas las coordenadas de sus vértices, por lo tanto, se plantea la idea de filtrado específico, utilizando el punto céntrico obtenido anteriormente, creando un bucle en donde se ejecute un condicional, en el cual se pregunte en que Bounding Box de todas las personas detectadas se encuentra el punto céntrico. Una vez terminado de recorrer el bucle se tendrá como resultado un único Bounding Box con la persona con la característica específica en este caso la que lleva el saco de color rojo como se muestra en la Figura 2.30.

**Figura 2.30.** a) Filtro de clase (Personas) b) Filtro de color (área con mayor concentración de color rojo) c) Combinación de los filtros a y b

Para la aplicación de control de tráfico no se requiere un segundo filtrado, por lo tanto, con los datos obtenidos “coordenadas Bounding Box” de la ejecución del algoritmo de detección de objetos YOLO en la cual se filtraron solamente los carros es suficiente.Figura 2.31**.**

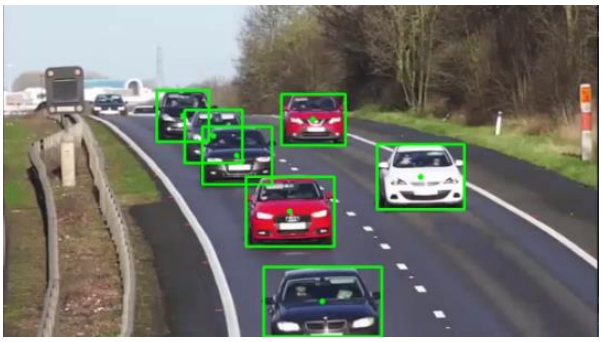

**Figura 2.31** Filtrado de Clase (Carros)

Finalmente es importante mencionar que las coordenadas de los objetos detectados en cualquiera de las aplicaciones servirán como entrada para el algoritmo de seguimiento de objetos.

### 2.1.8 Tracking o Seguimiento de Objetos

En este apartado se desarrollan la implementación de los algoritmos de tracking o seguimiento de objetos planteados en la sección 1.4.4, mismos que requieren del algoritmo de detección de objetos planteado en la anterior sección para su funcionamiento. Básicamente estos algoritmos son los encargados de seguir a un objeto previamente identificado y encerrado en un Bounding box durante una secuencia de frames de video.

Los 2 algoritmos de tracking de objetos a considerar y sobre todo utilizado en aplicaciones de visión computacional son: Centroide Tracking y Deep Sort Tracking, de los cuales se rescatará sus principales ventajas y desventajas para ser considerados en las aplicaciones a implementar.

#### 2.1.8.1 Implementación Algoritmo Centroide Tracking:

El funcionamiento del algoritmo de tracking del Centroide puede verse resumido en el siguiente diagrama mostrada en la Figura **2.32**, la explicación a detalle puede encontrarse en la sección 1.4.4.1.

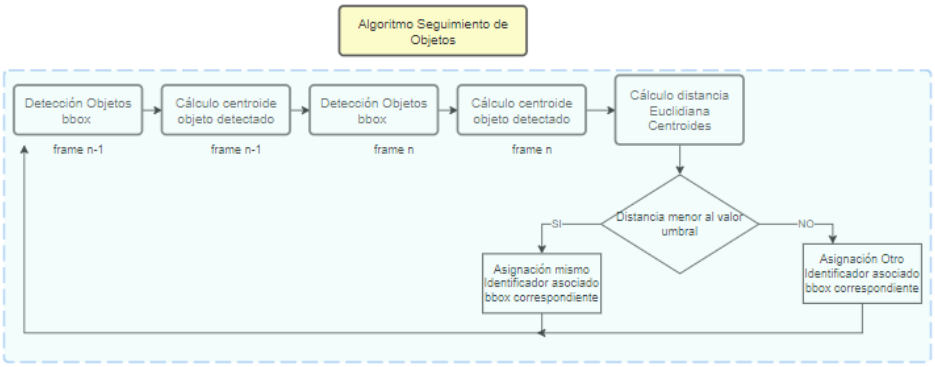


**Figura 2.32.** Diagrama funcionamiento algoritmo Centroide Tracking (Resumido)

Tabla 2.1. Ventajas y desventajas uso del Algoritmo de seguimiento Centroide Tracking

| Ventajas | Desventajas |
|---|---|
| - Al ser un algoritmo de respuesta rápida puede trabajar con algoritmos de detección de objetos que no consuman recursos computacionales altos como Yolo o SSD.<br><br>- Puede ejecutarse en aplicaciones con video en tiempo Real.<br><br>- Proporciona un número de identificación única para cada objeto detectado.<br><br>- Utiliza el cálculo de la distancia euclidiana para mejorar su velocidad de respuesta. | - Presenta problemas ante la oclusión es decir pérdida de visión de un objeto previamente identificado.<br><br>- Presenta cambios de identificación cuando dos objetos similares se cruzan.<br><br>- Inmediatamente el Bounding Box del objeto detectado salga del cuadro del video el algoritmo asigna otra identificación si reaparece<br><br>- No muestra en su etiqueta la clase de objeto detectado, solo números. |

### 2.1.8.2 Implementación Algoritmo Deep Sort Tracking:

El funcionamiento del algoritmo de tracking Deep Sort puede verse resumido en el siguiente diagrama mostrada en la Figura **2.33**, la explicación a detalle puede encontrarse en la sección 1.4.4.2.

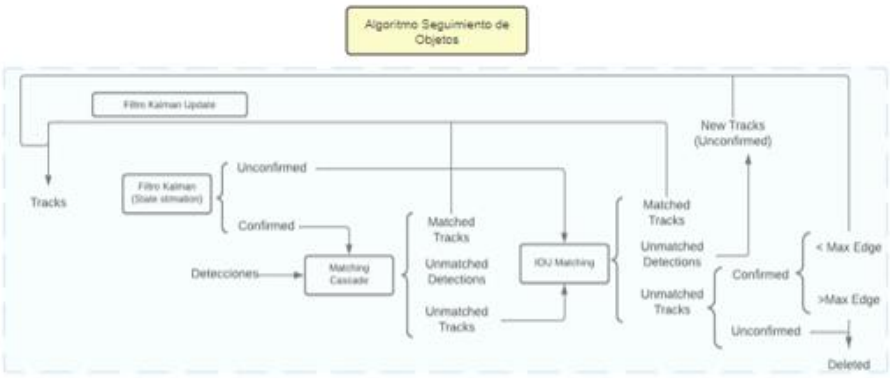


**Figura 2.33.** Diagrama funcionamiento algoritmo Deep Sort (Resumido)

**Tabla 2.2.** Ventajas y desventajas uso del Algoritmo de seguimiento Deep Sort

| Ventajas | Desventajas |
|---|---|
| - Es un algoritmo utilizado en el seguimiento de un único o múltiples objetos, a diferencia de algoritmos propios de Open CV, como TLD o KCF.<br><br>- Puede ejecutarse en aplicaciones con video en tiempo Real, motivo por el cual es uno de los algoritmos más utilizados en aplicaciones de visión computacional.<br><br>- Proporciona un identificador más claro que el algoritmo anterior, es decir muestra además del identificador, la clase del objeto que se está siguiendo (persona, carro) ,etvc<br><br>- Soluciona el problema existente de oclusión y pérdida parcial del objeto al salir del cuadro de visualización. | - Requiere de un código externo para su implementación, básicamente se puede considerar una librería externa a Open CV.<br><br>- Requiere de mayor configuración en sus parámetros, para poder realizar un correcto seguimiento, sin embargo, si estos parámetros son bien calibrados, se tiene una gran ventaja ante otros algoritmos de tracking, pues soluciona problemas de oclusión parcial o total del objeto detectado.<br><br>- Consume un poco más de recursos computacionales, sin embargo, no deja de ser útil en aplicaciones de visión artificial. |

### 2.1.9 Descripción completa de los algoritmos implementados

En las anteriores secciones se detallaron los algoritmos de detección y tracking de objetos, así como también los procedimientos adicionales que permitirán realizar el procesamiento de imágenes que engloban básicamente tres importantes puntos, el filtrado de clases, el filtrado de color y la selección de una región de interés “ROI”, mismas que permitirán el desarrollo completo de los sistemas a implementar en las aplicaciones objetivo, las cuales se pueden resumir en los siguientes puntos:

#### 2.1.9.1 Aplicación de Control y Monitoreo

1. Se verifica conexión entre la cámara del mini – dron y la tarjeta de control SBC, para poder recibir las imágenes en tiempo real.

2. Una vez estable la conexión, el algoritmo de detección de objetos se encarga de encerrar en un Bounding Box todos los objetos encontrados en un frame con su punto cétrico , posteriormente se realiza un filtrado de clases con el fin de identificar una única clase de objetos, que en el caso de esta aplicación son carros que circularán en una carretera concurrida.

3. En la ventana de visualización del usuario se selecciona una región de interés. La cual se ha divido en dos casos: si se selecciona una región dentro de la carretera, el dron apuntará al punto céntrico del mismo y se dará inicio al proceso de control y monitoreo, mientras que al seleccionar una región fuera de la carrera y libre de circulación vehicular, el dron apuntará al punto céntrico de dicha región e iniciará el proceso de aterrizaje.

4. Antes de iniciar el proceso de control y monitoreo, ya se tiene previamente la detección de cada uno de los carros en un frame inicial, y a partir aquí comienza el seguimiento múltiple de los vehículos detectados utilizando el algoritmo de tracking en los posteriores frames, con el fin de realizar un seguimiento continuo de los carros hasta que estos salgan de la ventana de visualización.

5. Al circular los carros por la carretera y al atravesar la región de interés trazada, se incrementará un contador, mismo que llevará la cuenta de los vehículos que han cruzado dicha región, cabe aclarar que en este punto el algoritmo de seguimiento tiene sentido utilizarlo, puesto que un algoritmo de detección de objetos simplemente detecta la presencia del objeto en un frame determinado, sin embargo, no indica que se trata del mismo objeto a largo todos los frames que componen el video, por lo que si no se utiliza un algoritmo adicional de seguimiento, el carro al cruzar cada instante por la ROI provocará el incremento del contador sin control, proporcionando información errónea ya que asume que se trata de un carro diferente que pasa por la región de interés a cada momento.

6. Para el registro de los carros en imagen de cada uno, se utiliza el borde superior de la región de interés y cada vez que el punto céntrico del cuadro que encierra el carro detectado cruce dicha línea, se realizará una captura que será almacenada en un repositorio externo.

#### 2.1.9.2 Aplicación de seguimiento de un Objetivo Particular

1. Se verifica conexión entre la cámara del mini – dron y la tarjeta de control SBC, para poder recibir las imágenes en tiempo real.

2. Una vez estable la conexión, el algoritmo de detección de objetos se encarga de encerrar en un Bounding Box los objetos encontrados en un frame, posteriormente se realiza un filtrado de clases con el fin de identificar una única clase de objetos, que en el caso de esta aplicación son personas.

3. En esta aplicación se requiere un filtrado de clases adicional, puesto que de un grupo de personas previamente detectado se requiere identificar al individuo que lleve una preda particular, en este caso un saco de color rojo, por ende, se realiza un segundo proceso de identificación de objetos considerando el filtrado de color.

4. En el filtrado de color se encierra en un contorno cerrado, el área de mayor valor que contenga al color filtrado dentro del frame, evidentemente esta pertenecerá al saco de color rojo, se halla su punto céntrico y el mismo se evalúa en cada uno de los recuadros que encierren a las personas detectadas. En el caso de que el punto céntrico se halle dentro de algún recuadro de alguna persona detectada, se podrá obtener al individuo con la prenda distintiva.

5. Se inicial el proceso de seguimiento único de dicha persona, aplicando algún algoritmo de tracking durante todos los frames de video, considerando factores como oclusión o desaparición temporal de la persona dentro del cuadro de visualización del usuario.

# 3 Pruebas y análisis de resultados

En el presente capítulo se detallan los resultados obtenidos al realizar diversas pruebas con los distintos algoritmos que se llevaron a cabo para la implementación de las dos aplicaciones descritas, comenzando con pruebas realizas a los algoritmos de detección, posteriormente los algoritmos de seguimiento y finalmente la combinación de ambas dando como resultado el desarrollo y funcionamiento de las aplicaciones como producto final entregable.

Cabe aclarar que el desarrollo y ejecución de las pruebas implementadas fueron realizadas en un computador portátil de las siguientes características:

**Tabla 3.1.** Características PC

| | |
|---|---|
| Procesador | Intel R Core I7-7500U CPU @ 1.30 GHz |
| RAM Instalada | 8.00 GB |

| | |
|---|---|
| Sistema Operativo | Windows 10 Pro-64 |
| Tarjeta Gráfica | NVIDIA Gforce MX130 |

Debido a que no se cuenta físicamente con alguna de las tarjetas de desarrollo “SBC”, y adicionalmente el tiempo de procesamiento de cada uno de los algoritmos sería bastante extenso ya que conlleva una amplia capacidad computacional requerida, especialmente en la etapa de entrenamiento de estos, sin embargo, los resultados servirán como punto de partida para una posterior implementación física.

## 3.1 Selección Algoritmo de Detección de Objetos

### 3.1.1 Parámetros de Evaluación

Con el fin de evaluar la capacidad de respuesta de los algoritmos de detección de objetos implementados, se ha decidido recolectar imágenes, la mayoría de ellas tomadas desde alturas que superan los tres metros y con distintos niveles de luminosidad simulando de esta manera imágenes captadas por el mini - dron en campo. Con el resultado de las detecciones realizadas por cada uno de los sistemas en cada imagen, se pretende a través de fórmulas estadísticas medir la capacidad de respuesta de los algoritmos utilizados para detectar objetos de interés (personas, carros). Dichas fórmulas involucran parámetros como Miss Rate (tasa de pérdidas) y Recall (Tasa de aciertos). [32]

**Tabla 3.2.** Evaluación Detecciones Correctas vs Incorrectas (Positivas - Negativas)

| | | Detecciones Algoritmos | |
|---|---|---|---|
| | | Personas o carros | No personas o carros |
| Imágenes Captadas en Tiempo Real | Personas o carros | TP | FN |
| | No personas o carros | FP | TN |

El término Miss Rate (RT) se puede definir como la relación existente entre los false negative (FN) sobre el número de objetos de interés existentes en la imagen (N), mismo que han sido contados previamente. Los FN básicamente son objetos que el algoritmo no fue capaz de detectar aún estos estando presentes en la imagen original, como se expone en la **¡Error! No se encuentra el origen de la referencia.**. La fórmula para realizar su c álculo es la siguiente Eq (2):

$$\boldsymbol{MR = \frac{FN}{N}} \ [\mathbf{2}]$$

Donde:

FN: False Negative – Falsos Negativos

N: Número total de objetos de interés presentes en la imagen (personas, carros)

El término Recall, es un parámetro que muestra las detecciones correctas realizadas por el algoritmo de detección, es decir, los objetos de interés correctamente etiquetados, su expresión matemática queda descrita de la siguiente manera Eq (3):

$$\boldsymbol{Recall = 1 - MR} \ [\mathbf{3}]$$

Según se menciona en [33] un valor de Recall superior a 0.85 se considera apropiado para que el algoritmo de detección de objetos sea usado en una futura implementación física, es decir se pueda ejecutar en una tarjeta de desarrollo cumpliendo correctamente su función.

El primer algoritmo de detección de objetos planteados es el de Haar Cascade, sin embargo como se demostró en la sección 1.4.3.1, su precisión en cuanto a detección de objetos es muy imprecisa, se comprobó inicialmente que el sistema detectaba a la persona

Figura **3.1** Detecciones Incorrectas al menor movimiento, Algoritmo Haar cascade

, sin embargo al menor movimiento de este, el algoritmo ya no realizaba la detección, por ende se puede decir que es bastante sensible, y además para la segunda aplicación que tiene que ver con vehículos, no se cuenta con una cascada de características de este objeto (imágenes que contengan y no contengan carros) para entrenar el modelo, aproximadamente se requieren más de 10000 imágenes, lo cual entrenar desde cero este modelo no es una buena opción.

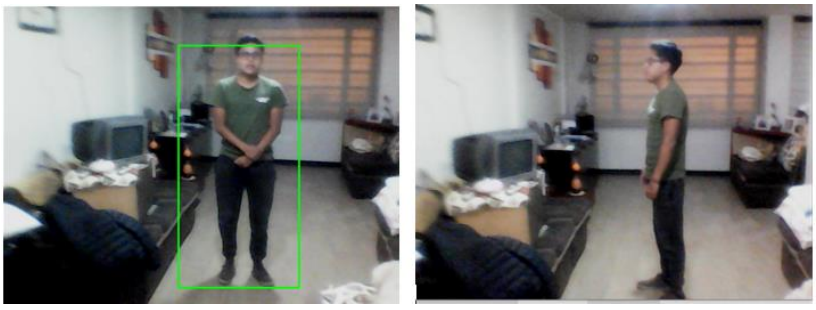

**Figura 3.1** Detecciones Incorrectas al menor movimiento, Algoritmo Haar cascade

Por este motivo en el presente trabajo se ha descartado definitivamente este modelo y se han realizado las diferentes pruebas considerando los algoritmos de detección vigentes en el campo de la investigación de visión computacional siendo los pioneros los algoritmos de detección de objetos YOLO en sus diferentes versiones.

### 3.1.2 Prueba Yolo v4

En esta prueba se ha considerado un total de 10 imágenes Tabla **3.3**, muchas de ellas tomadas desde alturas que superan los tres metros, adicionalmente se ha escogido fotos con baja y alta resolución, poco y mucha luminosidad con el fin de simular diversos escenarios donde el mini - dron realizará las capturas. La base de datos de las imágenes se las puede encontrar en Anexo1.

**Tabla 3.3.** Imágenes capturadas para prueba algoritmo Yolov4

| IMÁGENES | FN | N | MR | RECALL |
|---|---|---|---|---|
| IMG1 | 1 | 16 | 0,06 | 0,94 |
| IMG2 | 0 | 6 | 0,00 | 1,00 |
| IMG3 | 0 | 27 | 0,00 | 1,00 |
| IMG4 | 3 | 20 | 0,15 | 0,85 |
| IMG5 | 0 | 7 | 0,00 | 1,00 |
| IMG6 | 0 | 4 | 0,00 | 1,00 |
| IMG7 | 0 | 7 | 0,00 | 1,00 |
| IMG8 | 1 | 6 | 0,17 | 0,83 |
| IMG9 | 0 | 7 | 0,00 | 1,00 |
| IMG10 | 0 | 7 | 0,00 | 1,00 |
| **PROMEDIO** | 0,50 | 10,70 | 0,04 | 0,96 |

Como se puede notar el modelo YoloV4 alcanza un valor promedio de 96 % correspondiente a la taza de aciertos en los objetos de interés detectados por el algoritmo, sin embargo, como se mencionó inicialmente las pruebas fueron realizadas en una computadora portátil con características considerables, misma que procesaba las imágenes a 29.85 fps – 30 fps, factores para los cuales el algoritmo de detección Yolov4 si está preparado para trabajar, sin embargo, para una futura implementación física, se requiere implementar el código en una tarjeta SBC, que evidentemente no tendrá una velocidad de procesamiento de imágenes que una laptop pues no posee las mismas prestaciones, principalmente el prescindir de una tarjeta gráfica, por ende se busca una versión YOLO más liviana.

Por este motivo se pone a prueba la versión de Yolo V5 en su versión Tiny, misma que trabaja con mayor facilidad a velocidad de procesamiento reducidas pues cuenta en su estructura con un menor número de capas de convolución, considerando que, una tarjeta de control SBC se tendrá hasta un máximo de velocidad de procesamiento de imágenes de 5 a 8 fps [34], para imágenes captadas por el mini - dron en tiempo real.

### 3.1.3 Pruebas Yolo v5, Tiny

En esta etapa se ha considerado las mismas cantidades de imágenes que en la anterior prueba realizada sin embargo se ha escogido la versión más liviana de los algoritmos de detección Yolo de su última versión actualizada V5, Tabla 3.4.

**Tabla 3.4.** Imágenes capturadas para prueba algoritmo Yolov5, Tiny

| IMÁGENES | FN | N | MR | RECALL |
|---|---|---|---|---|
| IMG1 | 0 | 16 | 0,00 | 1,00 |
| IMG2 | 0 | 6 | 0,00 | 1,00 |
| IMG3 | 3 | 27 | 0,11 | 0,89 |
| IMG4 | 1 | 20 | 0,05 | 0,95 |
| IMG5 | 1 | 7 | 0,14 | 0,86 |
| IMG6 | 1 | 4 | 0,25 | 0,75 |
| IMG7 | 1 | 7 | 0,14 | 0,86 |
| IMG8 | 0 | 6 | 0,00 | 1,00 |
| IMG9 | 0 | 7 | 0,00 | 1,00 |
| IMG10 | 0 | 7 | 0,00 | 1,00 |
| **PROMEDIO** | 0,70 | 10,70 | 0,07 | 0,93 |

Como se puede observar los resultados de la Tabla 3.4, el uso de una versión más liviana implica una reducción en cuanto a capacidad de precisión en la detección de objetos con un decremento en la tasa de aciertos del 3%. La diferencia no es considerable, por lo tanto, cualquiera de los algoritmos puede ser implementado, según las prestaciones de la tarjeta SBC en la cual se ejecute el algoritmo. Cabe aclarar que la elección de un algoritmo de detección correcto influirá directamente en la respuesta del sistema en conjunto, puesto que, las demás etapas dependen directamente de esta.

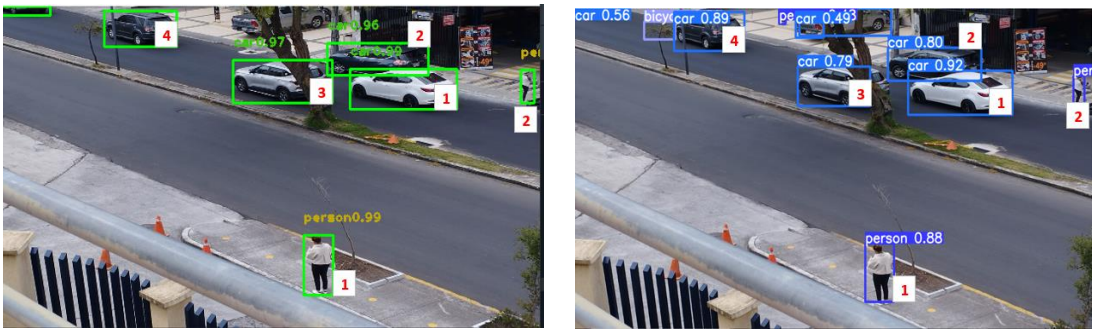


**Figura 3.2.** Resultados a) Yolo v4 vs b) Yolo v5 Tiny, detecciones realizadas a personas y carros

Una de las imágenes prueba (Img9) se puede observar en la Figura 3.2, donde se evidencia el resultado de las detecciones con las diferentes versiones de YOLO, efectivamente se puede observar diferencias en cuanto a nivel de confianza en la detección en cada una de las etiquetas anexadas a cada objetivo de interés “personas, carros” y la insignificanante cantidad de Falsos Negativos que se presentan, Tabla 3.5

**Tabla 3.5.** Comparación detecciones realizadas Yolo v4 vs Yolov5 Tiny, ejemplo Figura 3.2

| | | **Yolo V4** | **Yolo v5 Tiny** |
|---|---|---|---|
| **Carros** | 1 | 0,99 | 0,92 |
| | 2 | 0,96 | 0,80 |
| | 3 | 0,97 | 0,79 |
| | 4 | 0,95 | 0,89 |
| **Personas** | 1 | 0,99 | 0,88 |
| | 2 | 0,97 | 0,79 |
| **FN** | **Carros/personas** | 1 | 0 |

## 3.2 Pruebas Filtro de Color

En una de las aplicaciones implementadas requiere realizar el filtrado de color proceso que se explica detalladamente en la sección 2.1.3.2**,** donde se utiliza el filtrado mediante técnicas de HSV, en este caso se ha tomado como base distintos frames de videos previamente grabados a distintas horas del día (mañana, tarde, noche) , evidentemente con la persona llevando el distintivo particular que es un saco “rojo” , el color se ha configurado previamente con sus parámetros en un rango de valores altos y bajos, correspondientes al color a filtrar, cabe aclarar que**,** como salida de esta etapa se tiene el área en pixeles, y el contorno que encierra el distintivo particular, de la cual se puede obtener su punto céntrico, mismo que  es útil al momento del segundo filtrado de clase como se explica en la sección 2.1.5.Sin embargo, es importante tener en cuenta algunas consideraciones:

- Mientras más alto sea el vuelo del mini – dron el contorno que encierra el área en pixeles del color filtrado va a ser más pequeño, caso contrario si el mini – dron vuela a alturas bajas el área será grande. Lo mencionado anteriormente es muy importante, puesto que en un frame de video el área más grande generalmente aislará el color filtrado del resto de la imagen, pues el saco de color rojo en relación con su alrededor ocupa mayor área de pixeles. Por lo tanto, se requiere especificar un valor de área fija como punto de partida de comparación, con el fin de que al momento que se detecte en un frame alguna área de color rojo y esta sea superior al valor fijado sean tomados en cuenta, caso contrario serán consideradas como ruido y se descartarán.

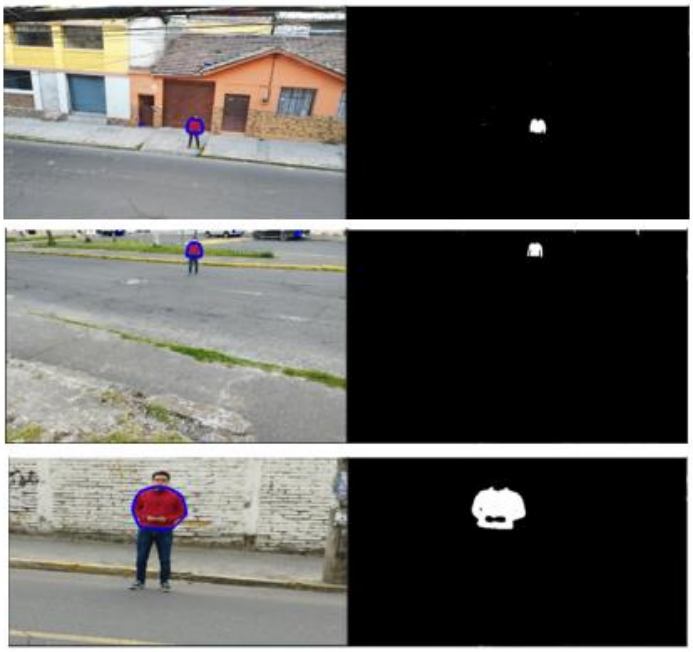

**Figura 3.3.** Resultados filtro de color con imágenes tomadas a distintas alturas a) 6m b) 4m c) 2m, para determinar el área que comprende el color deseado.

- Con lo mencionado anteriormente se procede a realizar muestras de áreas en pixeles que se pueden encontrar en distintos frames de video (40 muestras) para determinar el valor de partida, guardando cada valor de área de contorno encerrada en un vector para luego proceder a sacar un promedio con los valores almacenados, descartando siempre áreas pequeñas como por ejemplo 20 - 60 pixeles (ruido) y considerando el resto que prácticamente pertenece al área que contiene el contorno de color filtrado. En la Figura 3.3, se tiene un ejemplo de frames tomados a distintas alturas y efectivamente se puede evidenciar que el área de mayor valor, encierra al color filtrado “rojo”.

**Tabla 3.6**. Valores de áreas para setear en la sección del código correspondiente al filtrado de color, tomando en cuanto la altura de vuelo del minidrón.

| | Distintas Alturas de Captura | | |
|---|---|---|---|
| | 2m | 4m | 6m |
| **Promedio (Área) 40 Muestras realizadas** | 1000 | 500 | 450 |

Con las muestras realizadas se procede a establecer un valor de área fija de 1000 para vuelos a bajas alturas y un área de 450 para vuelos a alturas elevadas**,**Tabla 3.6**.**

- Para poder obtener los valores pertenecientes a los rangos altos y bajos del color a filtrar se ha trabajado en un código externo disponible en Anexo1, en el cual, por medio de un video en tiempo real, obtenido directamente de la cámara de la PC, se puede ir variando los parámetros de H – S – V por medio de barras de seguimiento, hasta obtener los valores que se ha de colocar como rangos de color a filtrar, como se muestra en la Figura 3.4

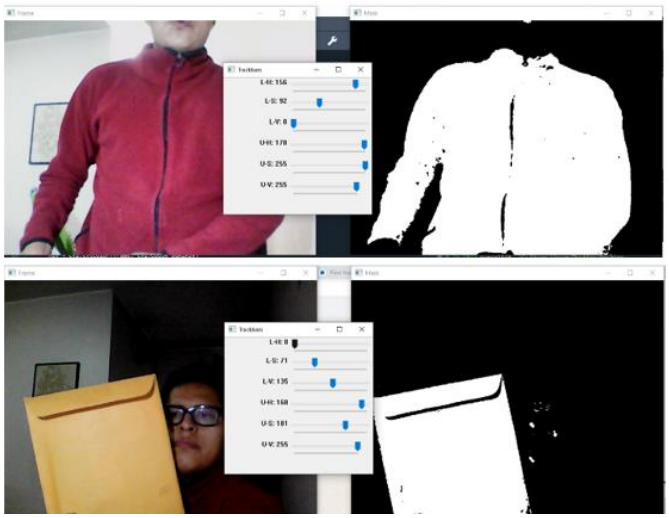

**Figura 3.4.** Proceso de selección de valores correspondientes al Rango de color a filtrar a) Rojo b) tomate

Para la aplicación implementada se requiere reconocer la persona que lleve un saco de color rojo, sin embargo, el código puedo utilizarse para poder encontrar cualquier color, como se observa en la Tabla 3.7.

**Tabla 3.7**. Valores correspondientes al rango de color alto y bajo para filtrar los colores rojo y tomate

| Filtrado Color Rojo |
|---|

| Rango Bajo | Rango Alto |
|---|---|
| [156,92,0] | [179,255,255] |
| **Filtrado Color Tomate** | |
| **Rango Bajo** | **Rango Alto** |
| [0,71,135] | [168,181,255] |

Es importante mencionar que el rango de color elegido puede variar, de acuerdo con factores de iluminación, por lo que, se ha realizado una prueba adicional con los rangos de valores color rojo encontrados previamente y comprobar el funcionamiento del filtro de color en un frame capturado en la noche, como se observa en la Figura 3.5.

**Figura 3.5.** Comprobación filtro de color rojo para captura tomada en la noche

Queda demostrado que, en el caso de que el rango de color sea correctamente asignado no se tendrá problema alguno en el filtrado de color con distintos niveles de iluminación.

## 3.3 Selección algoritmos de seguimiento

Para las pruebas a realizar se ha considerado dos algoritmos de seguimiento o tracking utilizados ampliamente en aplicaciones de visión computacional, el primero corresponde a uno de los algoritmos disponibles directamente en la librería de Open CV conocido como Centroide Tracking y el segundo un algoritmo externo llamado Deep Sort, para los cuales su funcionamiento tanto teórico y práctico fue descrito detalladamente en las secciones 1.4.4.1 y 1.4.4.2 respectivamente. Es importante mencionar que los algoritmos seleccionados pueden ser utilizados como seguimiento de un único objeto o de múltiples objetos, lo cual es una ventaja con respecto a otros algoritmos disponibles en la librería de Open CV como el KCF o el TLD.[35]

### 3.3.1 Pruebas Centroide Tracking

Se dispone de un total de 10 video con duración aproximada de 15 a 20 segundos disponibles en Anexo1, en la cual se presenta diversos objetos (personas y carros) el

objetivo de esta prueba es obtener una cantidad cuantitativa que relacione el tiempo que el algoritmo es capaz de seguir al objeto con el tiempo de duración total del video, considerando la presencia del objeto, obteniendo de esta forma un valor porcentual característico de la efectividad de este algoritmo, se presentaran escenarios con oclusión parcial del objeto y distintas niveles de iluminación.

Tabla 3.8. Pruebas con distintos videos, comprobando efectividad del Algoritmo Centroide Tracking

| | **# objetos a Seguir (personas / carros)** | **Tiempo Seguimiento** | **Tiempo Duración Video** | **Efectividad** |
|---|---|---|---|---|
| **Video 1** | 1/1 | 20/20 | 20 | 100% |
| **Video 2** | 0/1 | 13 | 15 | 86,66% |
| **Video 3** | 2/0 | 15 , 15 | 17 | 88,23% |
| **Video 4** | 1/1 | 18/18 | 18 | 100% |
| **Video 5** | 0/1 | 20/20 | 20 | 100% |
| **Video 6** | 2/0 | 13,12 | 15 | 83,33% |
| **Video 7** | 1/0 | 16 | 18 | 88,88% |
| **Video 8** | 0/1 | 16 | 18 | 88,88% |
| **Video 9** | 1/0 | 10 | 18 | 55,55% |
| **Video 10** | 0/1 | 10 | 17 | 58,82% |
| **Promedio** | | | | 85% |

Consideraciones importantes:

- Los resultados son bastante favorables obteniendo un 85% de efectividad, sin embargo, se ha incluido dos videos importantes donde se tiene oclusión del objeto (videos 9 y 10) en un determinado instante del video, consiguiendo una respuesta no satisfactoria del algoritmo, puesto que el objeto al pasar el periodo de oclusión es identificado como uno nuevo “fijarse en las etiquetas”, lo cual es erróneo pues se continúa siguiendo a un mismo objeto, siendo uno de los principales problemas de este tipo de algoritmo. Figura 3.6, fila a.

- Al ser un algoritmo de seguimiento de objetos, dependerá directamente de una detección previa, realizada efectivamente por el algoritmo de detección de objetos, por lo tanto, si en algún instante de video el algoritmo de detección dejo de funcionar consecuentemente el algoritmo de seguimiento no realizará correctamente su trabajo, por lo que al recuperarse del daño dará otra etiqueta al mismo objeto detectado previamente, lo cual es erróneo, como se aprecia en la Figura 3.6, fila b.

- Este algoritmo no requiere de configuración previa de parámetros que ayuden a mejorar el seguimiento de objetos, simplemente al ser parte de la librería de Open CV, solo se necesita importar la función que realizar el seguimiento.

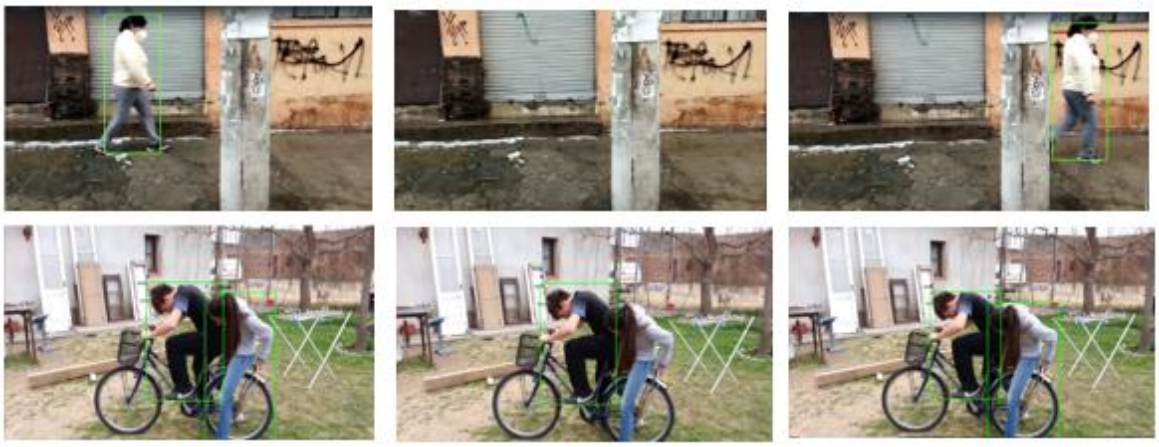

**Figura 3.6.** Resultados Algoritmo de seguimiento Centroide Tracking, Fila a) Problemas de oclusión Fila b) Pérdida temporal del algoritmo de detección

### 3.3.2 Pruebas Algoritmo Deep Sort

A diferencia del algoritmo de seguimiento anterior, este requiere de parámetros de configuración que influirán directamente en la respuesta del tracking de objetos, los cuales son configurados directamente en la programación y son:

- Max Age: En este parámetro se establece el número de frames que tiene que trascurrir, para que el objeto previamente detectado, pueda ser nuevamente identificado con la misma etiqueta una vez que este sale de la ventana de visualización por un tiempo determinado. En el caso de que el número de frames supere el valor seteado y el objeto no reaparezca nuevamente, simplemente será eliminado.

  **Ejemplo:**

  Max_Age = 40: El algoritmo recuperará al objeto si lo pierde 40 veces.

  Max_Age = 3: El algoritmo no recuperará al objeto después de perderlo 3 veces.

- Overlap: Define el porcentaje de solapamiento en el Tracking, lo que quiere decir que, si un objeto previamente identificado es ocluido total o parcialmente, este puede ser nuevamente identificado una vez supere este inconveniente. Puede tomar valores de 0 a 1

**Ejemplo:**

nms_overlap = 0.2: El algoritmo diferenciará objetos que estén solapados solo 20%.

nms_overlap = 1: El algoritmo diferenciará objetos que estén solapados totalmente.

- Matching – Threhold: Este parámetro define la similitud para que se considere un mismo objeto, es decir, consiste en identificar si este es el mismo objeto con otro que se supone que es el mismo en un frame diferente. Puede tomar valores de 0 a 1.

  **Ejemplo:**

  Max_cosine_distance = 0.1: El algoritmo asigna una nueva etiqueta si el objeto no es similar.

  Max_cosine_distance = 0.9: El algoritmo mantiene la etiqueta, aunque el objeto cambie en su aspecto.

Con los parámetros anteriormente explicados se procede a realizar la calibración del algoritmo de seguimiento con cualquier objeto y con la cámara de la PC en tiempo real, en este caso se ha utilizado dos botellas y un celular para realizar distintas pruebas y observar diferencias al modificar cada uno de los parámetros, como se observa en la Tabla 3.9

**Tabla 3.9.** Configuración de parámetros Deep Sort con dos objetos (botellas y celular)

| ANTES | DURANTE | DESPUÉS |
|---|---|---|
|  |  | **MAX_AGE = 50**<br>  <br>**MAX_AGE = 3**<br>  |

| | | |
|---|---|---|
|  | **OVERLEAP = 1**<br>  <br>**OVERLEAP = 0.2**<br>  |  <br>  |
|  | **MAX_COSINE = 0.9**<br>  <br>**MAX_COSINE = 0.1**<br>  |  <br>  |

Resultados de las pruebas realizadas:

- Considerando la Tabla 3.9, al variar el parámetro de Max_Age se puede observar, por un lado, que al retirar una de las botellas del cuadro de visualización y setear como primer valor Max_Age = 50, la botella al reaparecer conserva su etiqueta, mientras que en el segundo caso Max_Age = 3, inmediatamente una de las botellas sale del cuadro, al retornar esta cambia su etiqueta, aun siendo la misma botella

del frame anterior, pues ha pasado un número de frames suficientes como para que esta sea considerada una nueva botella.

- En el segundo caso se puede observar que al establecer el valor de 0.20 como Overleap, la botella de la parte posterior pierde su etiquetado cuando esta se encuentra ocluida un 20% de la misma, mientras que en el otro caso cuando se setea el mismo parámetro a 1, la botella posterior pierde su etiqueta cuando esta se encuentra completamente ocluida. Sin embargo, en ambos casos al pasar dicho inconveniente los objetos retoman su misma etiqueta, lo cual es de gran ayuda para solucionar el problema de oclusión total visto como desventaja del anterior algoritmo.

- En el tercer caso al establecer el parámetro de max_cosine en 0.9, inicialmente el celular se encuentra en una determinada posición exactamente con la pantalla encendida en la parte frontal, posteriormente al girar el celular, la misma se va a la parte trasera sin embargo su etiqueta se conserva, mientras que al setear el parámetro en 0.1, ocurre que al realizar el giro el celular cambia de etiqueta pues el algoritmo ha considerado que su aspecto ha sido modificado considerablemente.

Los parámetros de calibración han sido obtenidos utilizando como objetos dos botellas y un celular, sin embargo, su aplicación puede ser extendida a otros objetos como personas y carros importantes en el desarrollo de las aplicaciones a implementar. Cabe aclarar que antes de ejecutar el algoritmo de tracking Deep Sort únicamente se filtró los objetos necesarios “botella y celular” siguiendo el proceso descrito en la sección 2.1.5, y posteriormente se procedió a variar los parámetros de tracking antes mencionados obteniendo finalmente los valores descritos en la Tabla 3.10.

**Tabla 3.10.** Valores de los parámetros obtenidos después del proceso de calibración del algoritmo Deep Sort

| **Parámetros Deep Sort** | | |
|---|---|---|
| **Max _Age** | **Overleap** | **Max_cosine** |
| 50 | 1 | 0.9 |

Como segunda prueba, se realiza un filtrado de personas y carros, y se configura los parámetros Deep Sort con los valores antes calibrados, obteniendo los resultados que se muestran en la Figura 3.7.


**Figura 3.7.** Repuesta Algoritmo de seguimiento Deep Sort Fila a) Seguimiento carros en carretera concurrida Fila b) Oclusión total del objeto de interés

Se puede evidenciar la solución ante los dos principales problemas presentados con el algoritmo anterior, en la Figura 3.7, fila a se puede notar que al presentarse fallas parciales del algoritmo de detección al identificar los carros, los mismos vuelven a recuperar su etiqueta correspondiente tiempo despúes, mientras que en la Figura fila b, se ha solucionado el problema de oclusión, de tal forma que la etiqueta de la persona antes y despúes del periodo de oclusión es la misma, debido a que se trata del mismo individuo.

## 3.4 Pruebas Aplicaciones implementadas

Una vez realizadas las pruebas de funcionamiento de los algoritmos de detección de objetos, tracking, filtrado de color, se procede a realizar las pruebas en conjunto, considerando la particularidad expuesta en cada aplicación.

### 3.4.1 Prueba Aplicación Control de Tráfico:

En la primera aplicación se realiza un control de tráfico vehicular en una carretera concurrida, donde el usuario inicialmente seleccionará la región de interés “ROI” en cualquier instante del video proporcionado por la cámara del mini - dron , el objetivo de esta prueba es determinar la precisión del sistema diseñado, para lo cual se llevará a cabo un registro del número de vehículos que cruzan dicha región de forma manual, y se comparará con los vehículos que algoritmo en conjunto ha logrado detectar, contar y capturar.

**Figura 3.8.** Selección de una región de interés “ROI” (Trazo azul) en una carretera concurrida, video en tiempo Real.

En la Figura 3.8, se puede observar que el usuario ha trazado como ROI el sector correspondiente al carril izquierdo de la autopista, lo cual da inicio al conteo de vehículos, como se puede observar en la Figura 3.9.

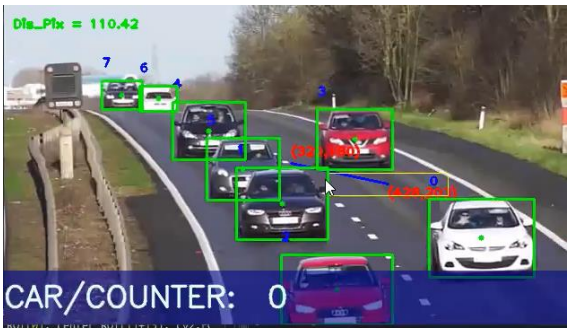


**Figura 3.9.** Inicio del proceso de conteo de vehículos que cruzan la región trazada

La Tabla 3.11 mostrada a continuación, permite realizar una comparación con los vehículos contados previamente y con los registrados por el algoritmo, en un video con duración de 40 sec. Al ser una aplicación de dificulta no compleja, se ha utilizado como algoritmo de detección Yolo V4 y algoritmo básico de seguimiento Centroide Tracking.

**Tabla 3.11.** Comparación manual y automática de los carros que cruzan la ROI en distintos tiempos de duración del video

| Tiempo Video | Color Vehículo | Logró Captura | Logró Detección |
|---|---|---|---|
| **4sec** | Rojo | SÍ | SÍ |
| **8sec** | Blanco | SÍ | SÍ |
| **11sec** | Negro | SÍ | SÍ |
| **15sec** | Negro | NO | SÍ |
| **21sec** | Plomo | SÍ | SÍ |
| **26sec** | Azul | SÍ | SÍ |
| **29sec** | Plomo | SÍ | SÍ |
| **31sec** | Rojo | SÍ | SÍ |
| **33sec** | Plomo | SÍ | SÍ |

| | | | |
|---|---|---|---|
| **38sec** | Rojo Vino | SÍ | SÍ |

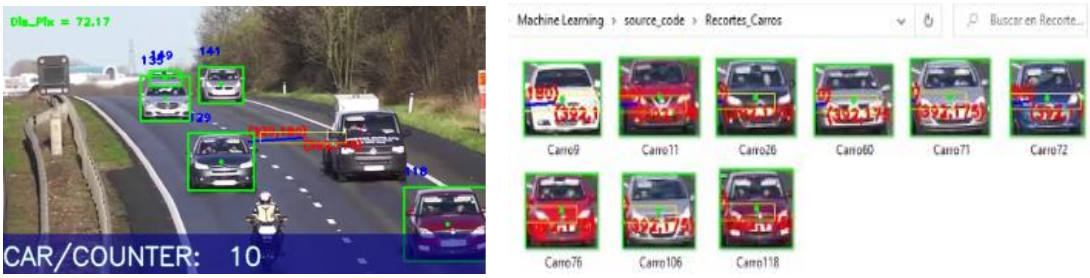


**Figura 3.10**. Resultados obtenidos Primera Aplicación (Monitoreo de Autos en una carretera concurrida)

Los resultados son aceptables, debido a que el algoritmo solamente no logró capturar en imagen 1 de los 10 vehículos que atravesaron la zona, sin embargo, si lo contabilizó, Figura 3.10. El problema se tendrá al tener áreas mayores donde el mismo vehículo u otro objeto que puede ser una persona, cruce alguna ROI trazada una y otra vez, realice movimientos bruscos dentro de ella, o exista oclusión parcial o total del objeto, debido a que, al utilizar el algoritmo de seguimiento del Centroide, éste es propenso a fallos pues no tiene forma alguna de configurar sus parámetros para mejorar el tracking, por lo tanto puede dar distintas identificaciones o etiquetas a un mismo objeto, lo cual es erróneo, pero para aplicaciones básicas donde la cámara del mini – dron apunte a una región fija y se tenga que hacer algo en particular, el algoritmo de seguimiento mencionado junto con el algoritmo de detección yolov4 o yolov5 tiny puede ser usado, un ejemplo es la aplicación realizada.

### 3.4.2 Pruebas aplicación seguimiento Particular

Para aplicaciones más complejas donde el objeto de interés detectado tenga que ser seguido por el mini – dron, mientras se mueve o realiza movimientos bruscos en un área determinada, algoritmos de seguimiento como el Centroide no podrían ser usados. Por este motivo se decide implementar una nueva aplicación donde, se realice un seguimiento a un objeto particular, en este caso un individuo con saco de color rojo. Anteriormente se realizaron ensayos teniendo en cuenta un nuevo algoritmo de tracking llamado Deep Sort mismo que será puesto a prueba juntamente con el algoritmo de detección yolov4 y el proceso de filtrado de color para localizar a la persona en particular dentro de un video, teniendo en cuenta factores extras como oclusión temporal.

En la primera prueba mostrada Figura 3.11, se experimenta el primer algoritmo de seguimiento Centroide Tracking, en donde la persona objetivo será obstaculizada por otra en un instante determinado.


**Figura 3.11.** Resultado Algoritmo Centroide Tracking - Error en identificación de un mismo objetivo en diferentes frames de video (Problema de Oclusión)

En los resultados se puede observar, que la detección de la persona objetivo se ha perdido, debido a que el algoritmo, no está preparado para superar factores de oclusión, en la etiqueta de esta se puede evidenciar que la identificación se ha cambiado, inicialmente el objetivo tenía como etiqueta persona 0 y al superar la oclusión se ha cambiado por persona 2, lo cual es erróneo ya que se trata de la misma, adicionalmente la línea de seguimiento que une el punto central del cuadro de visualización del usuario con el punto central del objeto detectado ha desaparecido. Por ende, se procede a ejecutar algoritmos más avanzados de seguimiento como es Deep Sort obteniendo los resultados observados en la Figura 3.12.


**Figura 3.12.** Resultado prueba 2, Algoritmo de Tracking Deep Sort, seguimiento del objetivo en distintos frames de video correctamente ejecutado

Evidentemente se tiene una mejora considerable, el objeto de interés conserva su etiqueta inicial, por lo tanto, se puede decir que no se ha perdido, adicionalmente la línea de seguimiento continua.

Se procede a realizar una tercera prueba en un área más extensa y con el objetivo moviéndose en diferentes direcciones, en una calle donde circulan carros por ende existe oclusiones constantes del objeto de interés y además el video fue captado desde una altura de 4 metros.

**Figura 3.13**. Resultados Algoritmo Deep Sort, sujeto moviendo en distintas direcciones y problemas de oclusión constante persistentes

Los resultados son eficientes, el seguimiento del objetivo fue continuo durante todo el video a pesar de las oclusiones existentes, de igual forma se puede notar que la etiqueta característica del individuo se mantiene constante todo el tiempo, Figura 3.13. Como última prueba se simula el movimiento de la cámara del mini - dron siguiendo al individuo montado en bicicleta, obteniendo los resultados mostrados en la Figura 3.14


**Figura 3.14.** Resultados Algoritmo Deep Sort, individuo montado en bicicleta con seguimiento de cámara constante

De igual forma, se puede observar un correcto desempeño del algoritmo implementado, la etiqueta no se ha perdido y el individuo ha sido seguido durante todos los frames de video, demostrado de esta manera la eficiencia y precisión del algoritmo Deep Sort al implementarlo con cualquier otro algoritmo de detección y etapas adicionales de procesamiento de imágenes como es el filtro de color.

# 4 CONCLUSIONES Y RECOMENDACIONES

## 4.1 Conclusiones

- Se ha desarrollado una revisión teórica en cuanto a la descripción del funcionamiento de 2 algoritmos de detección utilizados en visión artificial “Yolo, Haar Cascade” y 2 algoritmos de seguimiento “Centroide Tracking y Deep Sort”, los cuales han sido utilizados en la implementación de distintas aplicaciones, rescatando las ventajas y desventajas de cada uno de ellos en cuanto a su precisión, velocidad de respuesta, considerando una futura ejecución en una SBC.

- Se ha diseñado dos aplicaciones, la primera involucra el conteo, monitoreo y captura de carros que circulan dentro de una ROI trazada por el usuario y la segunda de un seguimiento particular a una persona con una prenda distintiva “saco color rojo”, aplicaciones que pueden servir como base para muchas otras, realizando modificaciones en el código elaborado, teniendo un conocimiento previo de programación en Python y visión computacional.

- De la correcta selección del algoritmo de detección de objetos dependerá el desempeño de los sistemas implementados, debido a que, posteriores etapas se alimentan de la está para enviar una respuesta satisfactoria. Mediante distintas pruebas, se comprobó que el algoritmo YoloV4 y su versión Tiny alcanzaron una eficiencia del 96% y 93% respectivamente en las detecciones realizadas, al modificar el parámetro IoU = 0.5 en el código, motivo por el cual, se usaron en ambas aplicaciones.

- Mediante distintas pruebas realizadas se puede concluir que, al trabajar en un ambiente controlado, es decir, una zona libre de obstáculos donde el mini – dron tendrá que realizar una aplicación particular de seguimiento de un objeto sin perderlo de vista, el algoritmo de tracking del Centroide es buena opción, caso contrario se podrá hacer uso del algoritmo Deep Sort, el cual ayuda a solucionar problemas de oclusiones parciales o totales del objeto, salida momentánea del mismo del área de visualización del usuario, para lo cual, se requiere una configuración correcta de sus parámetros, en la práctica se ha usado como max_adge=50, overleap = 1 y Max_cosine = 0.9, para superar los inconvenientes mencionados.

- En la primera aplicación, el algoritmo diseñado fue capaz de captar, seguir, contabilizar y capturar a los carros que circulaban por una región selecciona por el

usuario en varias tomas realizadas en carreteras concurridas, evaluando de esta forma el desempeño de este alcanzando una eficiencia del 90%.

- En la segunda aplicación se realizaron varias pruebas del individuo llevando el distintivo particular, con varios videos tomados a distintas alturas (2 – 8m) y niveles de luminosidad variada “día, tarde, noche”, observando que el algoritmo implementado fue capaz de seguirlo sin perderlo de vista. Considerando que, inicialmente se realizó un correcto procesamiento de imagen en cuanto a filtrado de color, ligado directamente a los niveles de luminosidad donde el mini – dron realizará las capturas.

- El procesamiento y manipulación de imágenes es importantes en aplicaciones de visión artificial, por lo cual se usó Open CV como librería base para la elaboración de ambas aplicaciones en tareas de redimensionamiento, transposiciones, dibujos de bounding box, operaciones con pixeles, filtrado de color, etc.

- Los algoritmos de seguimiento Deep Sort y Centroide pueden ser usados para seguir uno o múltiples objetos, tomando ventaja de algoritmos tradicionales que vienen por defecto en la librería de Open CV como el KCF o TLD.

- La programación elegida para la implementación de los algoritmos es Python debido a que cuenta con librerías especiales para visión artificial en continuo desarrollo y además es el lenguaje de programación manejado por varias SBC como Raspberry Pi o Jetson, utilizadas ampliamente en aplicaciones de visión computacional.

- Debido a que no se cuenta con una SBC las pruebas realizadas fueron ejecutadas en una PC, considerando algoritmos utilizados en diversas investigaciones, donde se desarrollaron aplicaciones varias tomando en cuanto los algoritmos utilizados en este proyecto, en donde se demuestra que el rendimiento de estos disminuye insignificantemente en la implementación física.

## 4.2 Recomendaciones

- La etapa de entrenamiento de cualquier algoritmo de detección es importante, en el trabajo realizado se ha desarrollado aplicaciones con redes pre – entradas con objetos que el algoritmo si es capaz de detectar “carros, personas”, sin embargo, si se requiere detectar algún objeto particular, se recomienda utilizar “Transfer Learning”, es decir aumentar la data de entrenamiento ya existente, con una

cantidad bastante grande de imágenes de dicho objeto, tomadas desde distintos ángulos y alturas construyendo un “Dataset propio” con el fin de obtener nuevos pesos característicos de la red para su posterior ejecución.

- Los sistemas implementados fueron ejecutados en una PC, sin embargo, estos tendrán que ser evaluados físicamente en una SBC montada en el mini – dron, por lo que se recomienda contar con una tarjeta que tenga prestaciones considerables en cuanto a procesamiento y sobre todo que cuente con una GPU “unidad de tarjeta gráfica”, para que realice el procesamiento de imágenes con mayor rapidez y eficiencia, como el módulo Jetson de la marca NVIDIA.

- La mayor capacidad computacional se necesita en la etapa de detección de objetos, por lo tanto, al ejecutar los algoritmos de detección en una SBC y observar un comportamiento no adecuado del mismo, se recomienda probar versiones más antiguas o utilizar otro algoritmo llamado SSD, debido a que una SBC no tiene las misas prestaciones que una PC.

# 5 REFERENCIAS BIBLIOGRÁFICAS


[1] H. Shakhatreh *et al.*, “Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges,” *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 48572–48634, 2019. doi: 10.1109/ACCESS.2019.2909530.

[2] B. Mohammed, “Integración de Visión Artificial en drones - multi rotor para gestión de misiones autónomas,” Universidad Carlos 3 de Madrid. Accessed: Aug. 16, 2022. [Online]. Available: https://e-archivo.uc3m.es/bitstream/handle/10016/29423/TFG_Mohammed_Bouayad_Boughroum.pdf?sequence=1&isAllowed=y

[3] R. Ormaechea, “Detección Y Seguimiento De Objetos Móviles En Secuencias De Video,” no. August, 2015, doi: 10.13140/RG.2.2.19162.90567.

[4] F. G. Becker *et al.*, *Técnicas y Algoritmos Básicos de Visión Artificial*, vol. 7, no. 1. 2015. [Online]. Available: https://www.researchgate.net/publication/269107473_What_is_governance/link/548173090cf22525dcb61443/download%0Ahttp://www.econ.upf.edu/~reynal/Civil wars_12December2010.pdf%0Ahttps://think-asia.org/handle/11540/8282%0Ahttps://www.jstor.org/stable/41857625

[5] J. A. Cancelas, R. C. González, I. Álvarez, and J. M. Enguita, *Conceptos y Métodos*

*en Visión por Computador*, vol. 1. 2016. [Online]. Available: https://intranet.ceautomatica.es/sites/default/files/upload/8/files/ConceptosyMetodosenVxC.pdf

[6] E. Albino and L. López, “Visión Computacional para la traducción en tiempo real del lenguaje de señas a texto en idioma español,” pp. 0–3, 2016.

[7] G. Mariela *et al.*, “Redes neuronales paralelas aplicadas a la visión computacional,” vol. 273, pp. 725–728, 2021, [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/120407

[8] X. Bosogain, “Redes Neuronales Artificiales y sus Aplicaciones,” 2020, [Online]. Available: http://cvb.ehu.es/open_course_ware/castellano/tecnicas/redes_neuro/contenidos/pdf/libro-del-curso.pdf

[9] HeartexLab, “labelImg,” 2022. https://github.com/heartexlabs/labelImg (accessed Aug. 17, 2022).

[10] S. Mallat, “Understanding deep convolutional networks,” *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 374, no. 2065, Apr. 2016, doi: 10.1098/RSTA.2015.0203.

[11] J. Sanchez, “Evaluación de algoritmos de detección de objetos basados en deep learning para detección de incidencias en carreteras,” p. 102, 2020, [Online]. Available: https://uvadoc.uva.es/bitstream/handle/10324/43277/TFG-G4450.pdf?sequence=1&isAllowed=y

[12] J. Bagnato, “Modelos de Detección de Objetos | Aprende Machine Learning,” 2020. https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/ (accessed Aug. 17, 2022).

[13] P. Viola and M. Jones, “Robust Real-Time Face Detection.,” 2004. https://sci-hub.se/https://doi.org/10.1023/B:VISI.0000013087.49260.fb (accessed Aug. 17, 2022).

[14] S. Raneros and M. Jiménez, “Estudio de la arquitectura YOLO para la detección de objetos mediante deep learning,” Universidad de Valladolid, 2021. Accessed: Aug. 17, 2022. [Online]. Available: https://uvadoc.uva.es/bitstream/handle/10324/45359/TFM-G1316.pdf?sequence=1

[15] A. Rosebrock, “Simple object tracking with OpenCV,” 2018. https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/ (accessed Aug. 17, 2022).

[16] W. Jiang and H. Di. Schotten, “A Comparison of Wireless Channel Predictors: Artificial Intelligence Versus Kalman Filter,” *IEEE Int. Conf. Commun.*, vol. 2019-May, May 2019, doi: 10.1109/ICC.2019.8761308.

[17] Y. Zhang, Z. Chen, and B. Wei, “A Sport Athlete Object Tracking Based on Deep Sort and Yolo V4 in Case of Camera Movement,” *2020 IEEE 6th Int. Conf. Comput. Commun. ICCC 2020*, pp. 1312–1316, Dec. 2020, doi: 10.1109/ICCC51575.2020.9345010.

[18] S. Sanchez, “Nvidia Jetson Nano - Hardware Especializado en Inteligencia Artificial - ¿Que puedo desarrollar?” https://www.youtube.com/watch?v=Hcfiurw8uM4 (accessed Aug. 17, 2022).

[19] C. M. Samsudin, “Diseño de un sistema de Visión Artificial par la detección y control de presencia de zopilote negro en aeródromos,” vol. 68, no. 1, pp. 1–12, 2020, [Online]. Available: http://dx.doi.org/10.1016/j.ndteint.2014.07.001%0Ahttps://doi.org/10.1016/j.ndteint.2017.12.003%0Ahttp://dx.doi.org/10.1016/j.matdes.2017.02.024

[20] NVIDIA, “High Performance AI at the Edge - NVIDIA Jetson TX2,” 2022. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/ (accessed Aug. 17, 2022).

[21] Python, “Download Python | Python.org,” 2021. https://www.python.org/downloads/ (accessed Aug. 17, 2022).

[22] P. Python, “PyPI · The Python Package Index,” 2022. https://pypi.org/ (accessed Aug. 17, 2022).

[23] J. Howse, *OpenCV Computer Vision with Python*. 2013. [Online]. Available: www.it-ebooks.info

[24] O. Python, “Home - OpenCV,” 2021. https://opencv.org/ (accessed Aug. 17, 2022).

[25] K. Cameron and M. S. Islam, “Multiple Objects Detection using HSV,” *Proc. IEEE Natl. Aerosp. Electron. Conf. NAECON*, vol. 2018-July, pp. 270–273, Dec. 2018, doi: 10.1109/NAECON.2018.8556711.

[26] G. Solano, “DETECCIÓN DE COLORES Y Tracking en OpenCV,” 2019. https://omes-va.com/deteccion-de-colores2/ (accessed Aug. 17, 2022).

[27] C. H. Setjo, B. Achmad, and Faridah, “Thermal image human detection using Haar-cascade classifier,” *Proc. - 2017 7th Int. Annu. Eng. Semin. Ina. 2017*, Oct. 2017, doi: 10.1109/INAES.2017.8068554.

[28] O. Python, “Haar Cascade Master,” 2020. https://github.com/opencv/opencv/tree/master/data/haarcascades (accessed Aug. 17, 2022).

[29] R. M. Pérez, J. S. Arias, and A. M. Porras, “Introducción al Aprendizaje Automático con YOLO,” *Rev. la Fac. Ing. y Tecnol. la Inf. y Comun.*, vol. 2, pp. 52–58, 2019, [Online]. Available: https://developer.nvidia.com/cuda-

[30] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, Accessed: Aug. 18, 2022. [Online]. Available: https://github.com/WongKinYiu/PyTorch_YOLOv4

[31] D. Coco, “COCO - Common Objects in Context,” *2020*. https://cocodataset.org/#home (accessed Aug. 17, 2022).

[32] R. Padilla, S. L. Netto, and E. A. B. Da Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” *Int. Conf. Syst. Signals, Image Process.*, vol. 2020-July, pp. 237–242, 2020, doi: 10.1109/IWSSIP48289.2020.9145130.

[33] Z. Liu, Z. Chen, Z. Li, and W. Hu, “An Efficient Pedestrian Detection Method Based on YOLOv2,” *Math. Probl. Eng.*, vol. 2018, 2018, doi: 10.1155/2018/3518959.

[34] B. Castelo and B. López, “Diseño e implementación de un sistema pra la detección y seguimiento de una persona a través de un cuadricóptero de tamaño reducido empleando visión artificial.” p. 129, 2022. [Online]. Available: http://bibdigital.epn.edu.ec/handle/15000/22126

[35] R. Alvarez-Cedrón García-Zarandieta, “Implementación de un modelo de detección y seguimiento de jugadores de waterpolo para el análisis de modelos de juego,” 2020, Accessed: Aug. 17, 2022. [Online]. Available: https://oa.upm.es/62753/

# 6 ANEXOS

## 6.1 Archivos Varios

En la carpeta compartida contiene los siguientes archivos:

- Carpeta Algoritmos Yolo – Deep Sort:
    - ✓ Código Fuente Aplicación de Seguimiento Particular
    - ✓ Código Fuente Aplicación de Monitoreo de Tráfico
    - ✓ Archivos importantes para el funcionamiento del algoritmo de detección.

- Carpeta Algoritmos Yolo – Centroide Tracking:
    - ✓ Código Fuente Aplicación de Seguimiento Particular
    - ✓ Código Fuente Aplicación de Monitoreo de Tráfico
    - ✓ Archivos importantes para el funcionamiento del algoritmo de detección.

- Carpetas imágenes y video para evaluar las dos aplicaciones implementadas
- Código de Filtrado de Color en tiempo Real
- Videos con distintas pruebas realizadas comprobando el funcionamiento de los algoritmos planteados.

**Link:**

`https://drive.google.com/drive/folders/1VMiklha6owSoVnzcu956rXA6iihfGxUm?usp=sharing`