Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

Escuela de Ingeniería en Mecatrónica

Trabajo de Grado Previo a la Obtención del Título de Ingeniero en Mecatrónica

Tema:

SISTEMA DE SEGUIMIENTO DE OBJETOS EN TIEMPO REAL MEDIANTE MOVIMIENTOS DE CÁMARA PAN-TILT

Autor: Jonathan Paúl Brasales Pérez

Director: Carlos Xavier Rosero Chandi

Ibarra-Ecuador

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR	
CÉDULA DE IDENTIDAD	1003352513
APELLIDOS Y NOMBRES	BRASALES PÉREZ JONATHAN PAÚL
DIRECCIÓN	OTAVALO, CALLE LOS PUCHOS Y SIETE CUENTOS
EMAIL	jpbrasalesp@utn.edu.ec
TELÉFONO	0980061852
DATOS DE LA OBRA	
TÍTULO	“SISTEMA DE SEGUIMIENTO DE OBJETOS EN TIEMPO REAL MEDIANTE MOVIMIENTOS DE CÁMARA PAN-TILT"
AUTOR	BRASALES PÉREZ JONATHAN PAÚL
FECHA	12/08/2021
PROGRAMA	PREGRADO
TÍTULO POR EL QUE OPTA	INGENIERO EN MECATRÓNICA
DIRECTOR	CARLOS XAVIER ROSERO CHANDI

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra. 12 de agosto de 2021

Jonathan Paúl Brasales Pérez
C.I.: 1003352513

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "SISTEMA DE SEGUIMIENTO DE OBJETOS EN TIEMPO REAL MEDIANTE MOVIMIENTOS DE CÁMARA PAN-TILT", presentado por el egresado JONATHAN PAÚL BRASALES PÉREZ, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 12 de agosto de 2021

Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

Carlos Xavier Rosero
DIRECTOR DE TESIS

Dedicado a
mis padres que siempre me motivaron a cumplir mi metas y son un apoyo fundamental en mi formación.

Agradecimientos

Quiero extender mi mayor agradecimiento a quienes hicieron posible la culminación de esta etapa en mi vida con una mención especial para mis padres, mi hermano y mi tío. Gracias por apoyarme, ser fuente de inspiración y fortaleza.

Mi gratitud, también a Escuela de ingeniería, a mi tutor de tesis y a todo el cuerpo docente, quienes con su apoyo y enseñanzas me brindaron las bases para una vida profesional.

¡Muchas gracias a todos!

Resumen

El seguimiento continuo de objetos móviles en video es un campo de gran interés dentro del área de visión artificial y su comunidad, ya que las cámaras estáticas están limitadas por su rango de visión y por la posición en la que son instaladas, el seguimiento continuo haciendo uso de cámaras motorizadas es cada vez mas empleado ya que nos brinda una solución a las limitaciones de visibilidad. Este trabajo consiste en desarrollar un sistema prototipo de seguimiento de objetos basado en software y hardware que consiste en un dispositivo capaz de realizar el seguimiento continuo de un objeto por medio de movimientos Pan-Tilt, de modo que una vez detectado el objeto se activa el sistema de seguimiento, el cual mantiene dicho objeto centrado en el centro de la pantalla y puede ser monitoreado mediante un monitor o haciendo uso de un servidor VNC. En la segunda sección de documento, se exponen conceptos teóricos que contextualizan el estudio del aprendizaje profundo y seguimiento de objetos. La tercera sección se centra en la propuesta de la arquitectura y el desarrollo de la misma, donde encontramos el algoritmo utilizado para reentrenamiento de la red neuronal convolucional y el usado para el control del dispositivo con el modelo, en este caso se entrenaron 2 modelos haciendo uso de la plataforma de Google Colab. Por último en la cuarta sección encontramos la implementación del modelo quantificado y los algoritmos en una raspberry pi 4. Los resultados de las pruebas realizadas tanto a los modelos como al sistema general de seguimiento demuestran que tener una base de datos mas diversificada brinda mejores resultados en las detecciones, además se comprobó que el uso del acelerador TPU ofrece inferencias de aprendizaje automático de alta velocidad, llegando a mantener al objetivo centrado con un error medio de 34 ± 25 y 32 ± 20 píxeles en el eje horizontal y vertical respectivamente. El dispositivo desarrollado resulta una plataforma optima para el estudio de algoritmos de visión y control.

Abstract

Continuous tracking of moving objects in video is a field of great interest in the area of machine vision and its community, since static cameras are limited by their range of vision and the position in which they are installed, continuous tracking using motorized cameras is increasingly used as it provides a solution to the limitations of visibility. This work consists of developing a prototype object tracking system based on software and hardware that consists of a device capable of continuously tracking an object by means of Pan-Tilt movements, so that once the object is detected the tracking system is activated, which keeps the object centered in the center of the screen and can be monitored by means of a monitor or by using a VNC server. In the second section of the document, theoretical concepts that contextualize the study of deep learning and object tracking are presented. The third section focuses on the proposed architecture and its development, where we find the algorithm used for retraining the convolutional neural network and the one used to control the device with the model, in this case 2 models were trained using the Google Colab platform. Finally, in the fourth section we find the implementation of the quantified model and algorithms in a Raspberry Pi 4. The results of the tests performed to both the models and the general tracking system show that having a more diversified database provides better results in the detections, it was also found that the use of the TPU accelerator offers high-speed machine learning inferences, reaching to keep the target centered with an average error of 34 ± 25 and 32 ± 20 pixels in the horizontal and vertical axis respectively. The developed device is an optimal platform for the study of vision and control algorithms.

Índice general

Índice de figuras

Índice de tablas

Introducción

Problema

Hoy en día la visión artificial es empleada en muchos ámbitos de estudio, es una técnica de bajo costo, confiable y accesible, estando sus principales aplicaciones dentro de la seguridad, deportes, etc [1]. El seguimiento continuo de objetos móviles en video es un campo de gran interés dentro del área de visión artificial y su comunidad. Sin embargo, es uno de los campos que menos evolución ha tenido comparándolo con los demás.

Los problemas que se presentan cuando se detecta y sigue objetos con cámaras en movimiento son diversos. Normalmente, las técnicas usadas con cámaras estáticas no son las mismas que con cámaras móviles [2], ya que las primeras cuentan con un rango de visión limitado por la posición en la que se instalan, siendo este uno de los principales impedimentos al momento de realizar el seguimiento de objetos.

Por otro lado, la epidemia COVID-19 que se presenta en la actualidad es feroz, altamente contagiosa y causa un impacto fuerte en la salud de cualquier persona. Para una reactivación económica a través del retorno progresivo al trabajo, los empleadores deberán tomar nuevas medidas de seguridad y salud, acordes con los riesgos laborales propios de sus actividades [3, 4].

Con la implementación de un sistema de seguimiento mediante cámaras con movimiento Pan-Tilt se obtiene un rango dinámico de mayor visión, siendo Pan el movimiento panorámico y Tilt el cambio de altura de la cámara [5], este sistema es usado cada vez más para diversas necesidades, como son las conferencias de sala inteligente, sistemas de video vigilancia, seguimiento de objetos, control de aglomeraciones, aplicaciones militares, etc [6]. Al contar con este tipo de plataforma en conjunto con algoritmos de visión artificial se podrá brindar mayor dinamismo en procesos de seguimiento de objetos y se ayudará en el control de procesos donde las cámaras estáticas no son suficientes.

Objetivos

Objetivo general

Desarrollar un dispositivo para el reconocimiento y seguimiento de objetos en tiempo real en un escenario controlado.

Objetivos específicos

- Diseñar la arquitectura del sistema teniendo en cuenta la literatura existente sobre seguimiento de objetos y control de movimiento Pan-Tilt
- Desarrollar el sistema de la cámara considerando adquisición de video, control Pan-Tilt y seguimiento de objetos.
- Evaluar los sistemas de la cámara a través de pruebas de funcionamiento en escenarios controlados.

Justificación

La visión por computadora es una rama de inteligencia artificial que está en constante crecimiento, presente en muchos de los procesos actuales. Mediante este trabajo de grado se propone un sistema de seguimiento de objetos no estáticos mediante el uso de cámaras con un sistema de movimiento Pan-Tilt, la cual servirá para futuras investigaciones y proyectos que se basen en el área de objetos móviles, además supondrá un avance en recintos con alto tránsito, dando dinamismo a los procesos y ayudando al progreso de la tecnología.

La implementación de los sistemas de visión artificial en las diferentes áreas contribuye una mejora en la calidad de vida de las personas y en su entorno, tanto en su seguridad, su confort y otros muchos factores. Cuenta con un gran aporte en la industria como es en el ámbito de la robótica y la automatización industrial, mediante el presente trabajo de grado ofrecemos una solución factible al momento de aumentar la seguridad en el trabajo de las personas en recintos donde sea necesario mayor control.

El presente trabajo de grado aborda un tema nuevo y desconocido, motivando el aprendizaje de una nueva disciplina al trabajar en el tema de seguimiento de objetos cuando estos están en movimiento, haciendo uso de dispositivos de captura de video y librerías de procesamiento de imágenes. La investigación fue seleccionada como un tema de actualidad que brindará apoyo a la vida profesional.

Alcance

El presente trabajo de grado desarrollará un dispositivo capaz de dar seguimiento a un objeto en tiempo real mediante el uso de una cámara con movimientos tipo Pan-Tilt, haciendo uso de métodos y librerías ya existentes. Además, el desempeño del dispositivo se validará mediante pruebas en escenarios controlados.

Capítulo 1

Revisión literaria

1.1. Revisitando aprendizaje de máquina

1.1.1. Aprendizaje profundo

El aprendizaje profundo (Deep Learning) es un subcampo del aprendizaje automático (Machine learning) que enseña a las computadoras a abordar problemas intuitivos mediante la experiencia, su entrenamiento es mediante un amplio conjunto de datos etiquetados y unas arquitecturas de redes neuronales que contienen muchas capas, llegando a obtener una precisión que, en ocasiones, supera el rendimiento humano [7, 8].

Mientras que el aprendizaje automático trabaja con árboles de decisión, el aprendizaje profundo usa redes neuronales que funcionan de forma muy parecida a las conexiones neuronales biológicas de nuestro cerebro, estos algoritmos aprenden directamente de los datos etiquetados, sin una extracción manual de las características y van mejorando a medida que el tamaño de los datos etiquetados aumenta [7, 9].

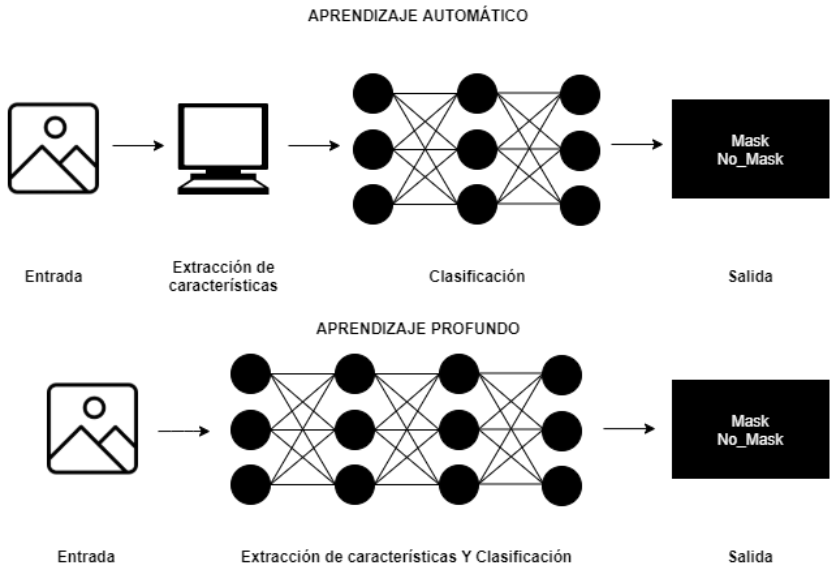


Figura 1: Comparación del aprendizaje profundo con el aprendizaje de la máquina

En el aprendizaje profundo se encuentran las redes neuronales artificiales (ANN), y como dos subconjuntos de ésta se tienen a las redes neuronales convolucionales (CNN)

y redes neuronales recurrentes (RNN). En este trabajo nos centraremos en como re-entrenar las redes neuronales convolucionales, donde actualmente existen tres técnicas de entrenamiento que emplean con éxito las CNN: la formación de la CNN desde cero, el uso de las características de la CNN pre-entrenada y la transferencia de aprendizaje [10].

1.1.1.1. Entrenamiento desde cero

El entrenamiento de una CNN desde cero requiere de muchos datos etiquetados para que tenga un buen rendimiento, su proceso de capacitación requiere de ajustes contantes debido a cuestiones de convergencia y sobreajuste (overfitting). Debido a la gran cantidad de datos y la velocidad de aprendizaje, es necesario una unidad de procesamiento gráfico (GPU) de alto rendimiento para obtener mejores resultados y reducir el tiempo de entrenamiento, el proceso suele tardar días o semanas en entrenar estas redes y resulta muy útil para las aplicaciones nuevas o las aplicaciones que tendrán un número muy elevado de categorías de salida [7, 11].

1.1.1.2. Extracción de características

Esta es la técnica menos habitual donde las características son extraídas de una red en cualquier punto de su entrenamiento ya que cada capa tiene su tarea asignada con sus características de aprendizaje definidas. Estas características extraídas pueden ser usadas en la entrada de datos de un nuevo modelo de aprendizaje automático [7].

1.1.1.3. Aprendizaje por transferencia

Es una de las técnicas mas utilizadas para aplicaciones de aprendizaje, un proceso que implica el ajuste detallado de los modelos CNN (supervisados) previamente entrenados para realizar una tarea nueva, donde se necesitan muchos menos datos y en consecuencia el tiempo de cálculo se reduce (se procesan miles de imágenes en lugar de millones). Esta técnica de aprendizaje por transferencia puede ser aplicado de dos maneras, como línea de base o como generador de características [7, 11], los modelos CNN pre-entrenados sirven como base para mejorar significativamente muchos problemas de detección de objetos y de segmentación de imágenes [10].

1.1.2. Redes neuronales convolucionales

Una red neuronal convolucional (CNN) es un tipo de red neuronal artificial que trata de imitar a las neuronas en la corteza visual de un cerebro, teniendo una efectividad mayor a otros modelos de aprendizaje automático [12]. Este tipo de redes poseen la operación de convolución que se realiza a partir de un filtro kernel que pasa sobre cada coordenada (i,j) de la imagen de entrada y se almacenan en las mismas coordenadas de la imagen de salida. La convolución de un kernel (K) con un tamaño definido (m x n) y colocado en el píxel (i, j) sobre una imagen I está descrita en la siguiente ecuación.

$$(I * K)(i, j) = \sum_{m} \sum_{n} (i + m, j + n) K(m, n)$$

Las CNNs están compuestas por diferentes capas de una o más dimensiones, donde las primeras extraen las características para a continuación localizar sus formas y proceder con la detección de las características de alto nivel. Las últimas capas son las encargadas de otorgar una predicción mediante las características extraídas.

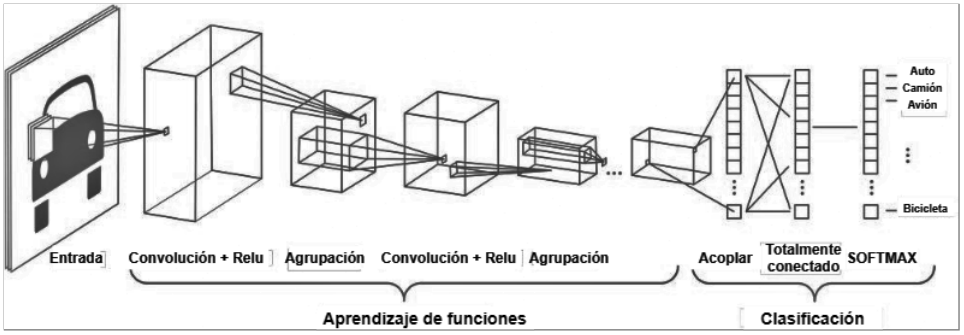


Figura 2: Ejemplo red neuronal convolucional de muchas capas [7]

Las Redes neuronales convolucionales aprenden a reconocer una diversidad de objetos y para hacerlo es necesario un entrenamiento previo con una cantidad importante de muestras por cada objeto, de donde las neuronas podrán captar las características y generalizarlas.

1.1.3. Marcos de implementación

En la actualidad existe una gran variedad de entornos para desarrolladores en el ámbito del aprendizaje profundo, entre las librerías mas conocidas se encuentran TensorFlow, Keras, Pytorch ó Caffe.

1.1.3.1. TensorFlow

En noviembre de 2015, Google publicó TensorFlow, una de las bibliotecas de software de código abierto más popular para el aprendizaje automático que permite a los desarrolladores experimentar con nuevas optimizaciones y entrenar algoritmos, está desarrollada para la computación numérica de alto rendimiento compatible con CPUs, GPUs y unidades de procesamiento tensoriales (TPUs) [13, 14].

TensorFlow soporta una variedad de aplicaciones, con un enfoque en la formación y la inferencia en redes neuronales profundas, es considerado un sistema de aprendizaje automático que funciona a gran escala y en entornos heterogéneos, además cuenta con una conversión directa con su plataforma más ligera, TensorFlow Lite, la cual reduce el número de pasos a realizar durante el desarrollo de modelos más ligeros [14].

1.1.3.2. TensorFlow Lite

Hoy en día con el uso en gran medida de dispositivos móviles e integrados es necesario que tener una plataforma como TensorFlow Lite que funciona en una gran variedad de dispositivos, desde microcontroladores pequeños hasta teléfonos celulares potentes.

TensorFlow Lite es un conjunto de herramientas creadas para ayudar a los desarrolladores a ejecutar modelos de TensorFlow en este tipo de dispositivos. Permite la

inferencia de aprendizaje automático en dispositivos con una latencia baja y un tamaño de objeto binario pequeño, consta de dos componentes principales [15]:

- El *conversor de TensorFlow Lite* convierte y optimiza los modelos de TensorFlow en un formato eficiente para que los use el intérprete.
- El *intérprete de TensorFlow Lite* ejecuta modelos optimizados en varios tipos de hardware diferentes.

1.2. Detección de objetos

En la detección de objetos sobre imágenes es muy común contar con herramientas basadas en arquitecturas de redes neuronales convolucionales profundas para detectar una cantidad específica de objetos [16]. Este tipo de detección no es tarea fácil, ya que en una imagen los objetos pueden estar en cualquier punto de la misma, tener diferente posición y tamaño, e incluso el objeto puede no aparecer entero.

1.2.1. Redes para la detección de objetos

Existen aplicaciones donde no es suficiente con reconocer y clasificar objetos en una imagen, donde es necesario localizar cada uno de ellos y obtener sus coordenadas para identificarlo a través de un delimitador trazado alrededor del objeto. Se han desarrollado varios algoritmos con características diferentes para la detección de objetos, como los algoritmos basados en R-CNN y otros algoritmos con estructuras distintas como son YOLO y SSD [17].

- **R-CNN: Búsqueda selectiva.** Se trata de una red CNN de dos fases donde primero se determina las regiones de interés dentro de la imagen para luego realizar clasificación de imágenes sobre esas áreas usando una red pre-entrenada. Este algoritmo utiliza la búsqueda selectiva para extraer 2000 regiones de diversos tamaños donde son pasadas por la CNN y se validan mediante un clasificador para posteriormente localizarlo mediante un regresor [16, 17].

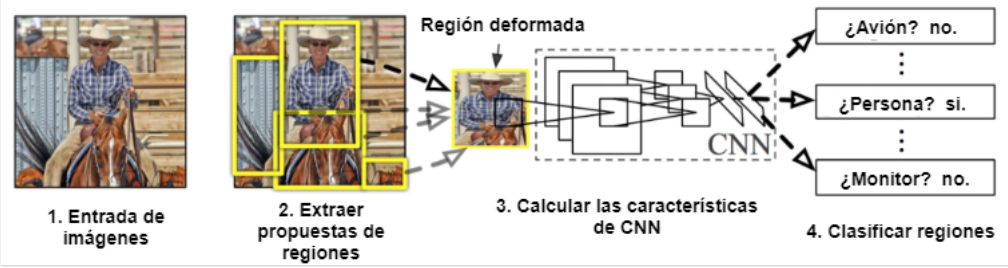


Figura 3: Resumen del sistema de trabajo de una R-CNN [13]

Existen diversos inconvenientes al usar este algoritmo, como el solapamiento del mismo objeto en diversas áreas, donde existen métodos como *IoU* (Intersection over Union) y *NMS* (Non Maximum Supression) que nos ayudan con este tipo de problemas, sin embargo, aún con el uso de todas estas mejoras en el algoritmo, la detección de objetos sobre una sola imagen toma un tiempo considerable de

segundos, siendo ineficiente para aplicaciones en tiempo real, donde el tiempo de detección toma un papel importante [16].

- **Detección Rápida: YOLO.** YOLO es una red CNN de una sola fase que significa *You Only Look Once*, esta red predice simultáneamente múltiples cuadros delimitadores y probabilidades de clase para esos cuadros utilizando un enfoque completamente diferente [18]. En el modelo original la arquitectura tiene 24 capas convolucionales para la extracción de características y dos capas adicionales completamente conectadas para calcular las predicciones y probabilidades de los cuadros delimitadores [19].

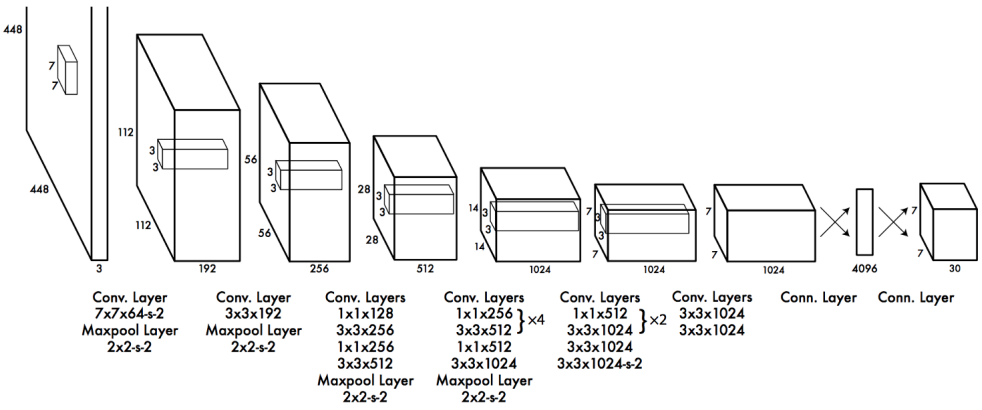


Figura 4: Conformación de la arquitectura YOLO [16]

El entorno original de YOLO es Darknet, sin embargo, es posible utilizarse con varias librerías como TensorFlow u OpenCV, la implementación con este último presenta varias ventajas como la integración inmediata, la calibración sin reentrenamiento y una mayor velocidad [20], ya que no es una marca registrada podemos encontrar un sin número de versiones distintas.

- **SSD: Single Shot Detection.** Las redes de una sola fase SSD tienen una estructura piramidal en la que las capas van disminuyendo gradualmente. Esto le permite al algoritmo poder detectar grandes y pequeños objetos, su enfoque se basa en una red convolucional de retroalimentación que produce recuadros delimitadores de tamaño fijo con puntajes para la presencia de objetos, seguido de un paso de supresión para producir las detecciones finales mostradas en pantalla [16, 21].

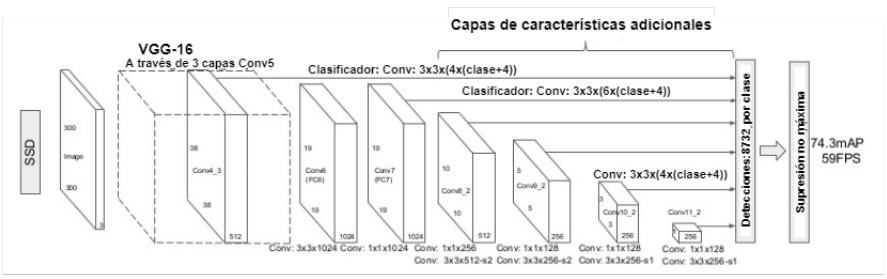


Figura 5: Ejemplo de una arquitectura SSD [13]

Es un detector de disparo único como YOLO que genera las regiones de interés y la clasificación de estas regiones en un solo paso y al mismo tiempo proporciona los cuadros de limitación en la imagen [22], toma solo un disparo para detectar múltiples objetos presentes, es considerado un algoritmo de detección de objetos de alta precisión y es el que se usará el el presente trabajo [23].

Existen muchos otro modelos para la detección de objetos, como el Google Spinet [24] de código abierto que rompe con la estructura piramidal y propone una arquitectura denominada modelo de escala permutada, lo podemos encontrar en el repositorio de GitHub de TPU de Tensorflow 1 y 2 [25]. Por su cuenta Facebook propone "End to End object detection with Transformers" una técnica de redes neuronales especializada en imágenes [26].

1.2.2. Modelos previamente entrenados

La mayoría de aplicaciones para la detección y clasificación de hoy en día comienza con un modelo previamente entrenado con el conjunto de datos de Imagenet, para luego este modelo ser ajustado a un nuevo conjunto de datos. A pesar de que el nuevo entrenamiento pueda tratarse de un conjunto de datos totalmente diferente, muchas de las características aprendidas de Imagenet también son útiles para el nuevo conjunto de datos, por lo que es posible ajustar el modelo sin necesidad de tener grandes cantidades de datos [16].

La integración de Mobilenet con el marco SSD es un tema de investigación muy atractivo, en gran parte debido a prácticas de ejecutar poderosas redes neuronales en dispositivos de gama baja como teléfonos móviles o portátiles y ser implementados en proyectos de tiempo real.

1.3. Seguimiento de objetos en tiempo real

El seguimiento de objetos en tiempo real es un campo de gran interés dentro del área de la visión artificial, sin embargo cuando un sistema de visión estática no es suficiente para realizar el seguimiento de un objeto ya sea por las limitaciones del rango de visibilidad que estas presentan, surgen técnicas y algoritmos en la literatura para dar una solución al seguimiento de objetos.

- *Sistemas de seguimiento con múltiples cámaras:* El seguimiento multicámara de objetos es considerada como un paso básico para el diseño de aplicaciones de vigilancia inteligentes, logrando un seguimiento mas preciso en condiciones de oclusiones [27]. Los métodos de seguimiento mediante el uso de múltiples cámaras se pueden clasificar de diferentes maneras según diferentes criterios como: los métodos utilizados para la detección del movimiento, la posición de las cámaras y la fusión de datos entre diferentes sensores. El objetivo de los sistemas de seguimiento multicámara es fusionar los datos para dar un seguimiento donde el plano de visión es aumentado [27, 28].

- *Sistemas de seguimiento con una única cámara motorizada:* Donde se propone una cámara motorizada en tres ejes que dan el seguimiento de un objeto, obteniendo un movimiento de paneo en el eje horizontal, un movimiento de inclinación

para el movimiento vertical y por último una ciclotorsión que es el giro sobre el eje óptico de la cámara [29]. Existen sistemas de seguimiento que omiten la ciclotorsión, dando lugar a los sistemas de movimiento Pan-Tilt [6]. El objetivo de este tipo de sistema es mantener aproximadamente centrado a un objeto dentro del marco de visión de la cámara.

- *Sistemas de seguimiento con un par estéreo motorizado:* Se trata de un sistema de seguimiento de dos cámaras motorizadas, cada una de ellas cuenta con los giros de paneo e inclinación y se usan controladores dinámicos para el control de cada uno de sus grados de libertad. Su objetivo es emular los sistemas visuales biológicos para dar seguimiento a un objeto donde se percibe una amplia porción de su entorno [30, 31].

- *Sistema de seguimiento con una cámara estática ubicada sobre un robot:* El sistema se seguimiento de trayectorias en el espacio que se realiza mediante movimientos de un robot, donde la cámara permanece fija en todo momento [32]. Existen varios métodos que abordan este problema como el sistema ojo en mano, donde la cámara se encentra sobre un brazo mecánico motorizado y esta interactúa con el entorno para dar los movimientos necesarios al sistema [33].

Capítulo 2

Desarrollo

Una vez descrito y estudiado a cerca del aprendizaje profundo y los diferentes métodos para la detección de objetos, se presentará la propuesta de la arquitectura que se usará para este proyecto comentando la estrategia y métodos a realizar para implementar el dispositivo.

2.1. Arquitectura

2.1.1. Requerimientos del sistema

El dispositivo hará uso de la detección de objetos de un modelo de aprendizaje profundo para dar seguimiento a un objeto mediante movimientos de cámara Pan-Tilt. A continuación, se enumeran las características generales con las que que debe contar el dispositivo para cumplir con los requerimientos del presente trabajo.

- Un modelo de detección de objetos capaz de ejecutarse sobre una raspberry y tenga una compatibilidad con las inferencias sobre una TPU.
- Sistema mecánico para la orientación de la cámara en dos ejes de libertad para mantener centrado a un objetivo.
- Alimentación externa exclusiva del sistema de orientación.

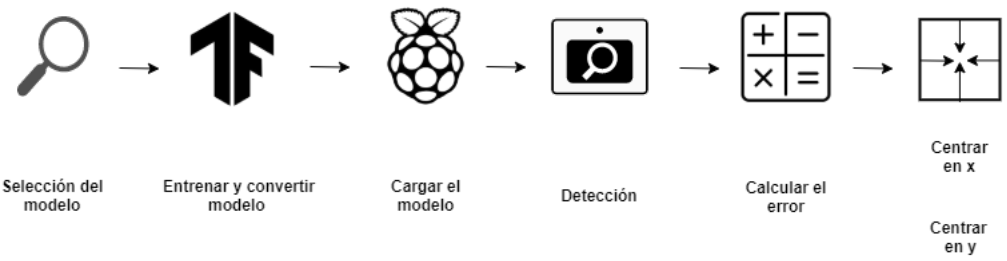


Figura 6: Propuesta a seguir para la implementación del dispositivo de seguimiento de objetos

2.1.2. Arquitectura propuesta

Definidos los requerimientos necesarios, se procede a definir la propuesta a implementar en el presente trabajo. La idea general del sistema se basa en reentrenar un modelo de detección objetos para que trabaje en tiempo real con un dispositivo de movimientos Pan-Tilt y dar el seguimiento a un objeto, todo esto implementado sobre una raspberry.

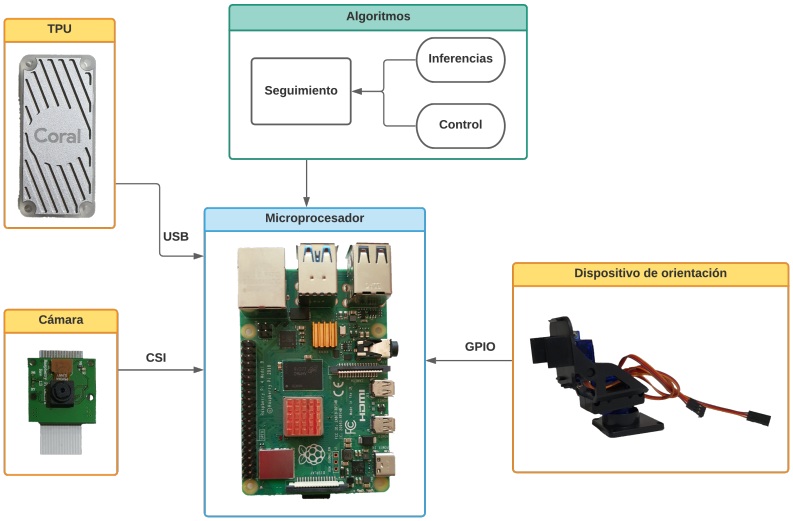


Figura 7: Esquema general del dispositivo a implementar

La figura 7 muestra un esquema general del dispositivo y el tipo de conexión que tendrán los elementos que conforman el dispositivo, tomando en cuenta que el movimiento del dispositivo de orientación de lo realiza mediante servomotores. A continuación, se expondrá el flujo de trabajo que se seguirá en esta propuesta, tanto para el software como la implementación del hardware.

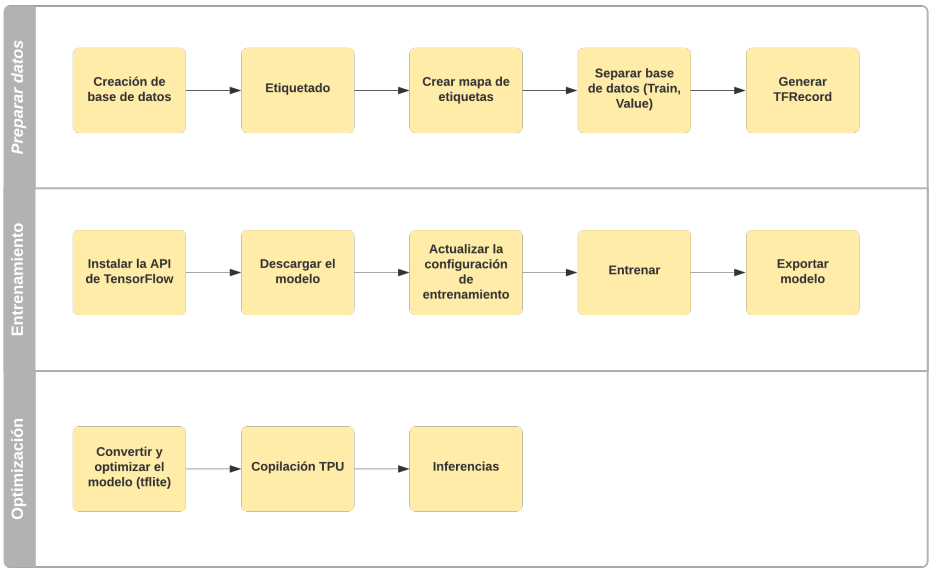


Figura 8: Flujo de trabajo para el entrenamiento y exportación del modelo

La figura 8 muestra el flujo de trabajo se que empleará desde la preparación del conjunto de datos hasta la optimización del modelo en el entorno de Google Colab, los códigos necesarios para la interacción del modelo con el sistema de seguimiento se dividirán en *control*, *inferencias* y *seguimiento*. Por otro lado la figura 9 presenta el flujo de trabajo empleado en la configuración del sistema y ensamblado del dispositivo.

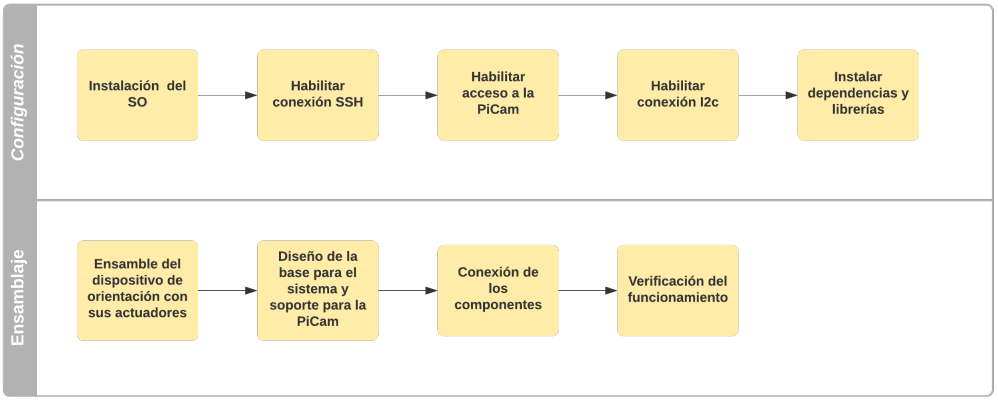


Figura 9: Flujo de trabajo para la configuración de la raspberry y ensamble del sistema de seguimiento

2.2. Caracterización

2.2.1. Hardware

El sistema de seguimiento de objetos mostrado en la figura 7 consta de una serie de componentes necesarios para su implementación, a continuación serán enumerados brindando una breve descripción del uso que tendrán dentro del trabajo.

- *Dispositivo de orientación (Pan-Tilt):* Se trata de una base para el acoplamiento de los actuadores y la sujeción de la cámara. Se optó por un sistema pre-armado para asegurar la precisión en los movimientos y la confiabilidad del sistema.
- *Actuadores:* Se dispondrá de 2 servomotores SG90 de 2.5 kg-cm de torque, serán usados para la manipulación de sus dos grados de libertad.
- *Cámara:* La picam v2 de una resolución de 2592×1944 pixeles se empleará para la adquisición de video, estará conectada a la raspberry y se ubicará sobre el dispositivo de orientación.
- *Microprocesador:* La raspberry pi 4 será empleada para realizar las conexiones de los diferentes componentes y donde se implementará el modelo y los algoritmos de seguimiento y control.
- *Coral edge TPU:* El Acelerador USB especializado para operaciones de Tensor-Flow Lite acelerará la velocidad de inferencia (predicción) en el microprocesador.
- *Alimentación Raspberry:* La raspberry Pi 4 será alimentada con los valores que el fabricante nos solicita para que funcione correctamente, siendo usado un cargador de 5v a 3A.

- *Alimentación externa:* Esta alimentación será exclusiva para los servomotores y evitará futuros problemas, se hará uso de un cargador de 5v a 2A.

Como se acaba de mencionar, los servomotores tendrán una alimentación completamente independiente, el esquema de conexiones de la figura 10 muestra los pines de la raspberry que son usados para el control del movimiento de los servomotores y la alimentación de estos mismos. Los servomotores se encuentran conectados en serie con una resistencia de 1K Ohm que sirve como protección para la raspberry.

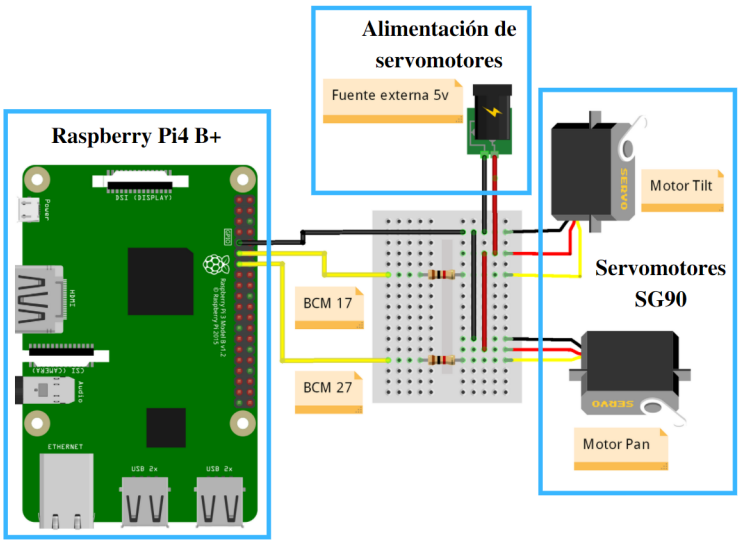


Figura 10: Esquema de conexiones para los servomotores

El esquema eléctrico de la figura 11 se implementó sobre una pequeña placa, donde se colocaron todas las conexiones hacia la raspberry y la alimentación de los servomotores.

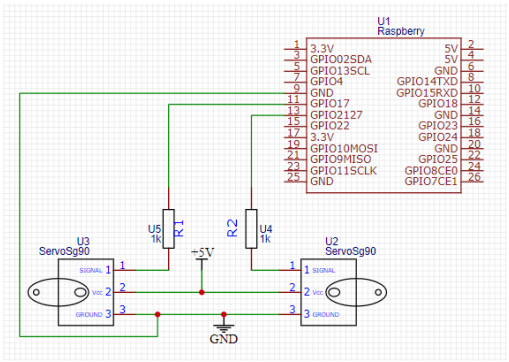


Figura 11: Esquema eléctrico de las conexiones de los servomotores con la raspberry

Para que el sistema de seguimiento fuese lo menos modular posible se realizó una carcasa que funciona como chasis para la raspberry y la placa de conexiones. Ademas sirve como base para el dispositivo de orientación, su diseño fue realizado en SolidWorks y posteriormente impreso en PLA.

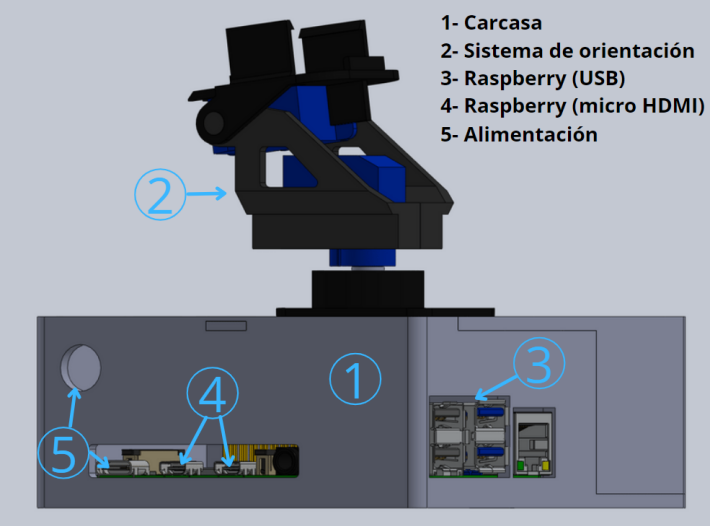


Figura 12: Diseño final de la carcasa, vista en 3D del modulo

2.2.2. Software

Como se mencionó anteriormente se hará uso de la plataforma de Google Colab para todo el proceso de re-entrenamiento del modelo, de aquí en adelante solo se lo mencionará como entrenamiento. Para empezar con el entrenamiento se vio necesario realizar una serie de instrucciones que contengan los directorios que se emplean durante la preparación de los datos, el entrenamiento y la exportación del mismo como se observa en la figura 13. Todo esto se realiza para tener un código más limpio, organizado y fácil de modificar para el entrenamiento del segundo modelo.

```
#CUSTOM_MODEL_NAME = 'ssdlite_mobiledet_edgetpu_320x320_coco_2020_05_19'                    a
#CUSTOM_MODEL_NAME = 'ssd_mobilenet_v1_quantized_300x300_coco14_sync_2018_07_18'

APIMODEL_PATH = 'models'                                                                   b
WORKSPACE_PATH = 'workspace/MASK'
MODEL_PATH = WORKSPACE_PATH + '/models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH + '/pre-trained-models'
CHECKPOINT_PATH = MODEL_PATH + '/' + CUSTOM_MODEL_NAME

IMAGE_PATH = '/content/carga/MyDrive/workspace/images'                                     c
ANNOTATION_PATH = '/content/carga/MyDrive/workspace/annotations/tf1'
OUTPUT_PATH = '/content/carga/MyDrive/workspace/tf1/'+ CUSTOM_MODEL_NAME + '/output'
EXPORT_PATH = '/content/carga/MyDrive/workspace/tf1/'+ CUSTOM_MODEL_NAME + '/export'
```

Figura 13: Configuración de las rutas necesarias para el entrenamiento

En la figura 13.a se inicializan los nombres de los modelos que usaremos en los entrenamientos, en 13.b se declara las rutas que tendrán los archivos del modelo descargado, como sus puntos de control y los archivos de configuración y por último en 13.c se especifican las rutas de la base de datos (IMAGE_PATH), los archivos tfrecord, labelmap (ANNOTATION_PATH), los directorios donde se guardarán los resultados del entrenamiento (OUTPUT_PATH) y la exportación del modelo (EXPORT_PATH).

2.2.2.1. Entrenamiento del modelo para la detección de objetos

Para lograr la detección de una serie de objetos que no se encuentran originalmente en el modelo, se hizo uso de la API de TensorFlow para realizar un entrenamiento mediante el uso de la técnica de aprendizaje por transferencia. Siguiendo el flujo de trabajo establecido en el capítulo 3 para el entrenamiento del modelo comenzamos con la preparación de los datos, seguido del entrenamiento y terminando con las optimizaciones necesarias para su compatibilidad con el acelerador.

Preparación de los datos

Para realizar un entrenamiento se debe tratar el conjunto de datos que se usará. Esta sección detalla el procedimiento desde la creación de la base de datos hasta obtención de los TFRecord de entrenamiento y evaluación.

- *Creación de la base de datos*

  La base de datos final se elaboró con un total de 2500 fotografías de acceso libre obtenidas de diferentes repositorios de Kaggel [34, 35], las cuales se encuentran divididas en imágenes con personas usando mascarilla y personas sin mascarilla.

Figura 14: Conjunto de imágenes usado para el entrenamiento del modelo

  Para tener una mejor organización y control sobre el conjunto de datos se procedió a cambiar el nombre todas las imágenes mediante la herramienta de renombrado predeterminada de Linux-Ubuntu como podemos observar en la figura 15.

- *Etiquetado*

  El etiquetado de las imágenes fue usado para proporcionar la información de lo que queremos detectar en el modelo, se realizaron pruebas en varios entornos, sin embargo se descartaron ya que fue necesario una suscripción para la exportación de los datos o los tiempos de carga eran altos.

  La figura 15 muestra el entorno usado para el etiquetado, donde 15.a es la lista de herramientas para gestionar el directorio de imágenes, 15.b gestiona las etiquetas y 15.c tiene una lista de todas las imágenes encontradas en el directorio. Dicha figura corresponde al entorno de LabelImg que se adaptó a las necesidades del trabajo, esta herramienta de anotación permitió etiquetar las imágenes en personas usando mascarilla (Mask) y personas sin mascarilla (No_Mask), estas anotaciones se guardaron como archivos *XML* en formato *pascal voc*.

Figura 15: Ejemplo del etiquetado de imágenes con LabelImg: (a) Herramientas, (b) Gestión de etiquetas y (c) lista de directorios

- *Creación de mapa de etiquetas*

  La creación del mapa de etiquetas (labelmap) para un conjunto de datos personalizado se la puede realizar de manera manual o mediante la implementación de un algoritmo, en este trabajo se lo realizó mediante un algoritmo con python, donde fue necesario especificar en nombre de la etiqueta (name) y su identificador (id). El mapa de etiquetas es usado como una fuente de registro independiente para las anotaciones de cada clase.

- *Separar la base de datos*

  Se trata de una operación común en todos los modelos de aprendizaje supervisado, donde dividimos la base de datos en al menos dos partes: una parte de entrenamiento, que corresponde a la mayor parte y que fue usada para entrenar el modelo y una parte de evaluación, de menor tamaño que fue usada para la evaluación del modelo. Del mismo modo se empleó un algoritmo capaz de realizar una separación de forma aleatoria con una relación 80 a 20 de las imágenes con sus respectivos XMLs para asegurarnos la máxima variabilidad de los datos.

- *Generar los TFRecord*

  Para poder entrenar el modelo con TensorFlow se generaron los archivos tfrecord que almacenan las secuencias de registros binarios, es decir, primero convertimos los datos de XML a un solo archivo de tipo CSV y después a tfrecord, que es el formato de imagen optimizado. EL algoritmo usado para esta operación se encuentra basado en la adaptación de varios repositorios de GitHub y así poder realizar ambas operaciones en una sola ejecución.

Figura 16: Archivos TFRedord compatibles con el entrenamiento con TensorFlow

Una vez finalice la ejecución de la celda en Google Colab obtenemos *train.record* y *test.record* de la figura 16 que son dos archivos que ya contienen la información

de todas las imágenes y de las coordenadas de los objetos que se marcaron con LabelImg.

Entrenamiento

Tras haber configurado correctamente las rutas y preparado los datos, se hizo uso de la técnica de transferencia de aprendizaje comúnmente usada para reducir los tiempos de entrenamiento para bases de datos pequeños, reduciendo el coste computacional del proceso ya que los modelos han sido previamente entrenados con enormes conjuntos de imágenes generalizadas para este propósito.

- *Descargar e instalar la API*

  Para empezar hacer uso de las diferentes librerías y algoritmos que nos proporciona TensorFlow es necesario clonar e instalar la API para la detección de objetos, la cual es compatible con TensorFlow 1 y 2, sin embargo se optó por realizar un entrenamiento con la primera versión ya que los modelos proporcionados y recomendados por el fabricante del acelerador TPU fueron entrenados con esta versión.

```
# Si no existe, clonar el repositorio de modelos tensorflow
if "models" in pathlib.Path.cwd().parts:
  while "models" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path('models').exists():
  !apt-get install protobuf-compiler
  !git clone https://github.com/tensorflow/models.git
```

Figura 17: Clonado e instalación de la API en la ruta APIMODEL_PATH

- *Descargar el modelo*

  Como se mencionó anteriormente, fueron seleccionados dos modelos previamente entrenados con la API de TensorFlow 1. Estos modelos fueron entrenados bajo los mismo parámetros de datos para posteriormente comparar los resultados que brinden en una serie de imágenes de prueba y seleccionar el mas adecuado para este trabajo. Los modelos concretos que fueron usados son los siguientes:

  1. ssd_mobilenet_v1_quantized_300x300_coco14_sync_2018_07_18
  2. ssdlite_mobilenet_edgetpu_320x320_coco_2020_05_19

- *Actualizar la configuración de entrenamiento*

  Todo entrenamiento por transferencia realizado mediante la API de TensorFlow hace uso de un archivo de configuración *pipeline.config* que contiene las rutas de los archivos necesarios (modelo, TFRecord, label_map), de igual manera en este archivo se definen los parámetros básicos necesarios para el entrenamiento como el número de clases a detectar y el tamaño de lote (batch_size) que depende de la capacidad de la memoria ram de la GPU, además en esta sección se especificó el número de pasos en los que empezará a realizar el entrenamiento consciente de cuatificación.

- *Entrenar*

  Con el archivo de configuración listo, ya es posible comenzar con el entrenamiento del modelo. El algoritmo (model_main.py) de la API de TensorFlow facilitó el proceso de entrenamiento, ya que proporcionando una serie de especificaciones se logra comenzar con el entrenamiento.

  - *pipeline_config_path:* Este corresponde a la dirección donde se encuentra almacenado el archivo de configuración, con las modificaciones previamente realizadas.
  - *model_dir:* Corresponde a la dirección donde se almacenarán los datos del entrenamiento.
  - *num_train_steps:* Corresponde al número de iteraciones de entrenamiento, que en este caso fueron 25000 y posteriormente se incrementaron.
  - *num_eval_steps:* Corresponde al número de iteraciones de evaluación, que para este caso fue de 500.
  - *logtostderr:* Envía los registros al archivo estándar STDERR es decir, envía los datos de la salida al terminal y observar los datos del entrenamiento.

  El entrenamiento de la red neuronal se llevó a cabo durante un tiempo aproximado de 6 horas. El proceso de entrenamiento puede ser retomado con un nuevo grupo de imágenes ya que TensorFlow permite almacenar puntos de control del entrenamiento cada cierto tiempo.

```
!python $APIMODEL_PATH/research/object_detection/model_main.py \
    --pipeline_config_path=$CHECKPOINT_PATH/pipeline.config \
    --model_dir=$OUTPUT_PATH \
    --num_train_steps=25000 \
    --num_eval_steps=500 \
    --logtostderr=true
```

Figura 18: Comando para el entrenamiento del modelo usando la API de TensorFlow

- *Exportar modelo*

  Del mismo modo para la exportación del modelo desde el último punto de control, se hizo uso del código proporcionado por la librería de Tensorflow (export_tflite_ssd_graph.py) con la que se crea un gráfico congelado con las operaciones compatibles con Tensorflow Lite, el modelo de la figura 19 ya se puede usar para realizar inferencias, pero aun no es compatible con el acelerador TPU.

Figura 19: Archivos generado al exportar el modelo

Optimización

- *Convertir modelo*

  Para poder usar un modelo de TensorFlow sobre un dispositivo móvil o un microprocesador se procede a convertir el gráfico congelado a un formato de búfer plano de TensorFlow Lite con la ayuda del comando tflite_converter. El modelo obtenido del entrenamiento ya puede ser usado directamente para realizar detecciones en una raspberry, sin embargo es necesario un paso extra para poder ser usado sobre el acelerador.

- *Copilar en Coral TPU*

  Para poder realizar las inferencias sobre el acelerador TPU es necesario que los modelos de TensorFlow Lite sean convertidos a un modelo compatible, esto se logró haciendo uso del copilador de Edge TPU que toma el modelo y nos devuelve uno compatible, teniendo en cuenta que no todas las operaciones de TensorFlow Lite son compatibles con Edge TPU, si en el modelo existen operaciones no admitidas como lo fue en este caso, el copilador nos dará la información de las operaciones que se ejecutarán en el acelerador y las que se ejecutaran en la CPU de la raspberry, por lo que la velocidad de inferencias no llegará a ser la máxima soportada por el acelerador.

```
Edge TPU Compiler version 15.0.340273435

Model compiled successfully in 944 ms.

Input model: model.tflite
Input size: 5.34MiB
Output model: model_edgetpu.tflite
Output size: 5.71MiB
On-chip memory used for caching model parameters: 5.62MiB
On-chip memory remaining for caching model parameters: 2.04MiB
Off-chip memory used for streaming uncached model parameters: 0.00B
Number of Edge TPU subgraphs: 1
Total number of operations: 64
Operation log: model_edgetpu.log

Model successfully compiled but not all operations are supported by the Edge TPU.
Number of operations that will run on Edge TPU: 63
Number of operations that will run on CPU: 1
See the operation log file for individual operation details.
```

Figura 20: Salida del modelo Edge TPU

2.2.2.2. Seguimiento de objetos

El seguimiento de objetos se lo realizó haciendo uso de una raspberry donde se implementaron los diferentes códigos y el sistema de orientación Pan-Tilt. El seguimiento consta de 3 códigos en el lenguaje de programación python creados usando el editor de código fuente Visual Code.

Inferencias

El objetivo de este código es detectar el objeto en sí y calcular las coordenadas centrales *(ObjX,ObjY)* de dicho objeto. Se encuentra dividido en las clases VideoStream y ObjDetector, donde el primero se encarga de manejar la transmisión de video en un hilo de procesamiento separado [36] y mejorar el rendimiento, mientras que el segundo se encarga de trabajar en las detecciones junto al modelo y las etiquetas.

- Class VideoStream: Esta clase controla la trasmisión de video de la Picam, donde se hace uso de los subprocesos para mejorar la tasa de procesamiento de FPS y reducir la latencia de E/S en la raspberry, teniendo en cuenta que cuanto mas procesamiento se tenga en el código, más tiempo tomará cada bucle, lo que disminuirá la tasa general de procesamiento FPS.

- Class ObjDetector: Se inicia especificando las rutas, cargando el mapa de etiquetas y el modelo sobre el interprete. Dentro de esta clase contamos con la método *start* donde comenzamos obteniendo los detalles necesarios del modelo e iniciando el video, posteriormente inicia el bucle donde tratamos la imagen de salida configurando su gama de colores y tamaño, al realizar la detección se recuperan datos necesarios como:

  1. boxes : Aquí se obtienen las coordenadas de la caja delimitadora de los objetos detectados.
  2. classes: Obtiene el índice de las clases de los objetos para compararlos con los entrenados.
  3. scores: Almacena los datos de confianza de los objetos detectados.

  Con estos datos se recorre todas las detecciones y procede a dibujar el cuadro de detección solo si la confianza se encuentra sobre un umbral preestablecido, en esta sección del código se calcularon los centros de los objetos detectados para al final mostrar en pantalla todas las etiquetas y resultados de las detecciones.

Control

Este script implementa la fórmula de PID con solo la importación de la librería *time*, consta de tres métodos:

- *__init__:* Es el constructor, donde definimos tres valores de ganancias como constantes.

- *initialize:* Inicializa los valores PID y establece la marca de tiempo actual y tiempo anterior.

- *update:* Se realizan los cálculos para obtener delta time, delta error y los términos de control:

  1. *cP:* Este término es igual al error obtenido de la resta del centro.
  2. *cI:* Se obtiene al multiplicar el error por delta time.
  3. *cD:* Es obtenido de la división de delta error sobre delta time.

  Por último se guardan los valores de tiempo y error, para ser usados en la siguiente actualización y se retorna la sumatoria de los términos PID por las constantes.

Las actualizaciones se realizan en un ciclo rápido debido a la velocidad de inferencia, por lo que se estableció un retardo de manera experimental para ajustarse con las limitaciones mecánicas y los protocolos de comunicación.

Seguimiento

Se trata del código principal que controla el seguimiento de objetos con giro e inclinación durante las detecciones, realiza cuatro procesos independientes mediante el uso de la librería *multiprocessing* [37]:

- prossObjCenter: Hace uso de la función Obj_center, el cual tiene como objetivo comenzar con las detecciones haciendo uso del script *Inferencias*.
- prossPan: Proceso que hace uso de la función pid_pross para inicializar el script *Control* y calcular el error en el eje horizontal (x) para ser devuelto.
- prossTilt: Proceso que hace uso de la función pid_pross para inicializar el script *Control* y calcular el error en el eje vertical (y) para ser devuelto.
- prossSetServos: El último proceso hace uso de la función set_servos para mover los servomotores, contiene un bucle que primero adapta los valores de entrada y los suma a la ultima posición de los servomotores para posicionarlos si este movimiento se encuentra dentro del rango seguro de los servomotores.

En la función pid_pross inicializa el control PID de ambos movimientos mediante los valores de Kp, Ki, Kd, que fueron obtenidos mediante un ajuste manual descrito a continuación:

- Se colocaron los valores Ki y Kd a su mínimo.
- Se aumentó Kp hasta observar una oscilación mantenida en la salida, para así establecer el valor a la mitad aproximadamente.
- Se aumentó Ki hasta que los desplazamientos se corrijan rápidamente, sabiendo que un valor demasiado alto causaría inestabilidad.
- Por último se aumentó Kd hasta que la salida se asiente en la referencia deseada rápidamente después de una perturbación de carga. Se tomó en cuenta que demasiado Kd causará una respuesta excesiva y hará que la salida se exceda.

Para finalizar los procesos se hace uno de un manejador de señal ejecutado en segundo plano que detiene los procesos al presionar CTRL+C.

Capítulo 3

Implementación y pruebas

3.1. Implementación

Para poder ejecutar los códigos realizados en la sección anterior es necesario la instalación de librerías como también la configuración del entorno de trabajo.

3.1.1. Configuraciones iniciales

La configuración del entorno de trabajo que utilizaremos en el seguimiento de objetos en tiempo real se la realizó siguiendo los pasos a continuación descritos.

- *Habilitar la cámara Pi:* Una vez conectada la PiCam V2 dentro de la configuración de interfaz de la raspberry se activó el uso de cámara.

- *Crea un entorno virtual:* Es un paso fundamental ya que nos aseguramos de no tener conflictos de versiones con librerías que ya tengamos instaladas en la raspberry, en este caso recibe el nombre de *tesis-env*.

- *Instalar dependencias del sistema:* En este paso se instaló todas las librerías necesarias tanto para realizar las inferencias como el control de los servomotores, donde solo se usaron librerías y bibliotecas libres.

- *Instalar el tiempo de ejecución de Edge TPU:* Una vez dentro del entorno virtual con las dependencias instaladas se procedió a descargar e instalar el tiempo de ejecución de Edge TPU, que es la librería necesaria para la interacción con el acelerador.

3.1.2. Carga de archivos

Una vez terminada la configuración de nuestro entorno de trabajo con la ayuda de Filezilla [38] procedemos al intercambio de archivos con la raspberry obteniendo una organización de los directorios de la figura 21.

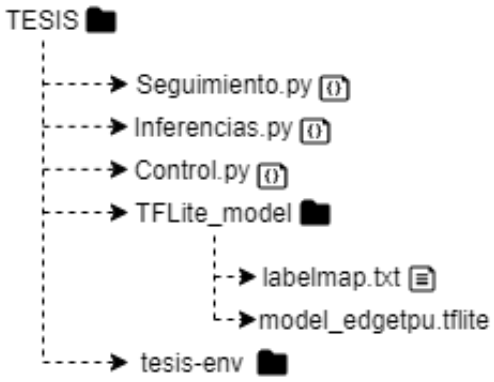


Figura 21: Árbol de directorios usados para la organización del entorno de trabajo en la raspberry

El entorno de trabajo está conformado por los 3 códigos usados para las inferencias y control de los motores, seguidos por el directorio donde guardamos el modelo y el mapa de etiquetas, también encontramos el directorio del entorno virtual con todas las librerías y bibliotecas instaladas. Con todo listo, haciendo uso de un servidor VNC para la manipulación de la raspberry comenzamos con las serie de comandos dentro del terminal para dar inicio al seguidor de objetos.

1. Es necesario la ubicación dentro del directorio *Tesis*.

2. Activar el entorno virtual para poder hacer uso de las librerías.

3. Activar el dominio *pigpio* usado para el control de las entradas y salidas de GPIO.

4. Por último se procede a ejecutar el script principal de seguimiento.

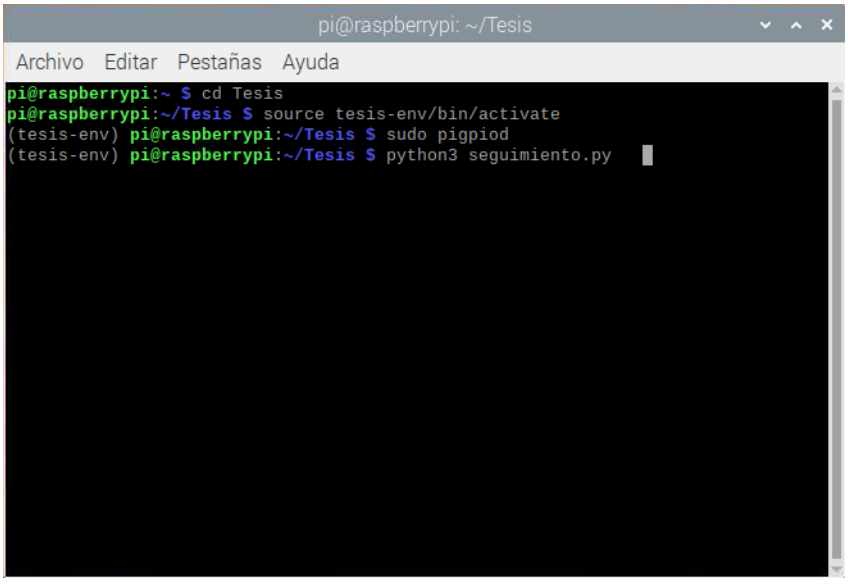


Figura 22: Lanzador del seguidor de objetos haciendo uso del terminar de la raspberry

3.2. Pruebas

Se realizarán 3 pruebas para comprobar el funcionamiento del sistema de seguimiento, primeramente comparando los resultados obtenidos desde TensorBoard de los dos modelos, siguiendo por una pequeña evaluación tanto en imágenes como en video, y por último el trabajo en conjunto con el dispositivo de seguimiento.

Tabla 1: Parámetros de los modelos en diferentes entrenamientos

Parámetros	**Entrenamiento I**		**Entrenamiento II**	
	Modelo 1	**Modelo 2**	**Modelo 1**	**Modelo 2**
Imágenes entrenamiento	1500		2560	
Imágenes evaluación	375		640	
Iteraciones	2500		50000	
Tamaño de imagen	300x300	320x320	300x300	320x320
Taza de aprendizaje (Base)	0.2	0.8	0.2	0.8

La tabla 1 brinda información de los parámetros usados en los entrenamientos, además de estos datos se realizó un nuevo entrenamiento desde el punto de control del entrenamiento I, donde se comprobó que al aumentar la base de datos y el número de iteraciones el modelo mejoraba los resultados sin la necesidad de realizar un entrenamiento desde 0, los resultados podemos observarlos en la sección de anexos.

3.2.1. Resultados obtenidos del entrenamiento

Existen una gran variedad de métricas que se pueden considerar dentro de la API de TensorFlow, para este caso se usaron las métricas de detección de COCO [39], que anteriormente se especificó en el archivo de configuración *pipeline*, consta de 12 métricas para caracterizar el rendimiento del detector. En esta sección se presentan los resultados obtenidos del entrenamiento del modelo por lo que no se entrará en detalle.

Tabla 2: Resumen de los datos obtenidos con las métricas de COCO

Valores						
Modelo	Entrenamiento I					
	mAP	mAP@.50IoU	mAP@.75IoU	AR@1	AR@10	Loss
1	0.3265	0.6466	0.2937	0.2711	0.4052	0.4599
2	0.1974	0.4670	0.1087	0.1919	0.3225	0.7492
Modelo	Entrenamiento II					
	mAP	mAP@.50IoU	mAP@.75IoU	AR@1	AR@10	Loss
1	0.4335	0.7875	0.4210	0.3471	0.5032	0.2055
2	0.2960	0.6303	0.2539	0.2817	0.4174	0.5701

La tabla 2 muestra un resumen de 5 métricas obtenidas al finalizar los entrenamientos, agregando la perdida final con los diferentes modelos. En el resumen podemos comprobar que el modelo 1 cuenta con mejores resultados en la precisión y recuperación promedio además que su perdida final se acerca mas al valor mínimo deseado. Del mismo modo se aprecia mejores resultados al realizar un nuevo entrenamiento desde 0 hasta las 50000 iteraciones con una mayor base de datos, por lo que el entrenamiento II será usado para las siguientes evaluaciones.

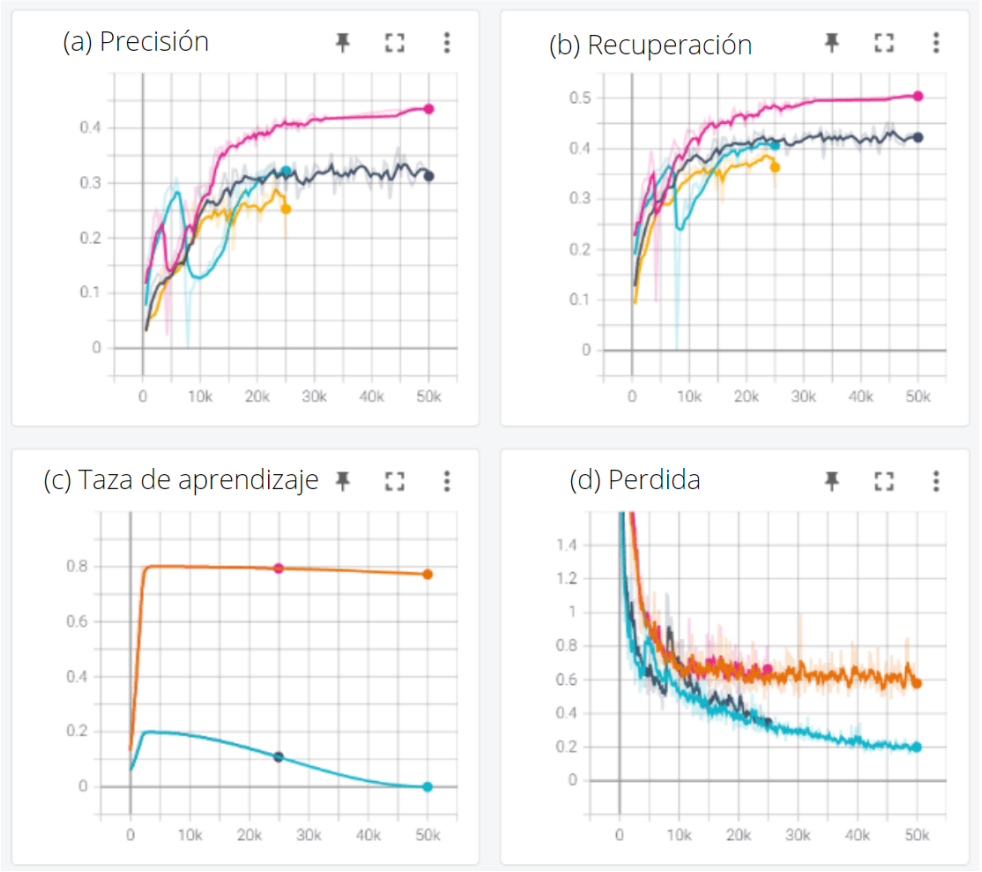


Figura 23: Resultados del entrenamiento obtenidos por TensorBoard: (a) Precisión promedio, (b) Recuperación promedio, (c) Taza de aprendizaje y (d) Perdida final

La figura 23 muestra muestra gráficamente el proceso de entrenamiento de los modelos obtenidos por TensorBoard donde se hace uso de las métricas de COCO [39], los valores finales se comentaron en la tabla 2.

Para comparar el rendimiento de los modelos se hizo uso de matrices de confusión, para obtener los valores de precisión, sensibilidad y valor de referencia. Las formulas utilizadas y el significado de las siglas se muestran a continuación.

Precisión:

$$Presición = \frac{TP}{TP + FP} \tag{3.1}$$

Sensibilidad:

$$Sensibilidad = \frac{TP}{TP + FN} \tag{3.2}$$

Valor de referencia:

$$Valor\,de\,referencia = 2 * \frac{Precisión * Sensibilidad}{Precisión + Sensibilidad} \tag{3.3}$$

Donde:

- Verdaderos positivos (TP): Se refiere a la cantidad de detecciones realizadas de forma correcta a la clase *Mask*.

- Falsos Positivos (FP): Son el total de objetos detectados que no pertenecen al objeto de interés y el modelo lo toma como positivo.
- Falsos Negativos (FN): Son el total de objetos que el modelo no ha detectado.
- Verdaderos Negativos (TN): Se refiere a la cantidad de detecciones realizadas de forma correcta de las clases que no son de interés.

Aunque el modelo fue entrenado para detectar dos clases diferentes, solo se realizará la evaluación de las personas usando mascarilla ya que es al que se le realizará el seguimiento.

3.2.2. Rendimiento del modelo sobre imágenes

Las evaluaciones se las realizó en el entrono de Google Colab, la prueba consiste en pasar el modelo sobre un conjunto de imágenes para obtener los datos que se usarán para el cálculo de precisión, sensibilidad y valor de referencia, con el objetivo de comprobar que tan bien funcionan los modelos en diferentes entornos y así seleccionar el que se ajuste a las necesidades del proyecto. Se realizó pruebas sobre un conjunto de 50 imágenes que tiene las siguientes características:

Tabla 3: Datos del conjunto de imágenes usado en la evaluación

Datos para la evaluación	
Imágenes para prueba	50
Total de detecciones	173
Personas con mascarilla	104
Personas sin mascarilla	75

Al realizar las inferencias sobre el conjunto de imágenes se dibuja un rectángulo para indicar la ubicación de la detección, además nos brinda un valor de confianza de los objetos detectados.

Figura 24: Resultado de las detecciones realizadas, obteniendo los cuadros delimitadores con su respectiva confianza

Del mismo modo para comprobar el rendimiento del modelo en su forma mas optimizada se realizaron las mismas pruebas, como se observa en la figura 25.

Figura 25: Ejemplo de las detecciones realizadas con los modelos optimizados a su versión Lite

La tabla 4 muestra los resultados obtenidos en la prueba con los modelos congelados, mientras que la tabla 5 muestra los resultados de las pruebas con los modelos en su forma optimizada.

Tabla 4: Resultados obtenidos en las pruebas sobre un conjunto de 50 imágenes

Modelo 1 - 300x300			
Matriz de confusión		Predicción	
		Mask	No Mask
Real	Mask	93	11
	No Mask	2	72
Resultados			
Precisión			0,9789
Sensibilidad			0,8942
Valor de referencia			0,9347

Modelo 2 - 320x320			
Matriz de confusión		Predicción	
		Mask	No Mask
Real	Mask	82	21
	No Mask	4	57
Resultados			
Precisión			0,9535
Sensibilidad			0,7961
Valor de referencia			0,8677

Tabla 5: Resultados obtenidos con los modelos de TensorFlow Lite en las pruebas sobre un conjunto de 50 imágenes

Modelo 1 - Lite 300x300			
Matriz de confusión		Predicción	
		Mask	No Mask
Real	Mask	90	14
	No Mask	1	71
Resultados			
Precisión			0,9890
Sensibilidad			0,8654
Valor de referencia			0,9231

Modelo 2 - Lite 320x320			
Matriz de confusión		Predicción	
		Mask	No Mask
Real	Mask	78	24
	No Mask	3	60
Resultados			
Precisión			0,9630
Sensibilidad			0,7647
Valor de referencia			0,8525

En la tabla 4 se observa que el rendimiento en cuanto a la detección de personas usando mascarilla es bastante bueno, el modelo 1 logró detectar correctamente 93 casos de los 104 existentes, mientras que el modelo 2 detectó 82.

Mientras que en la tabla 5 se aprecia que el modelo de TensorFlow Lite obtiene un menor número de detecciones correctas en ambos modelos, sin embargo el valor de referencia comprueba que la perdida que existe es del 1.25 % y el modelo puede seguir siendo usado para realizar inferencias. Los modelos optimizados a su versión lite mejoran su latencia ya que no existe ida y vuelta de información con un servidor, por lo que no se requiere ningún tipo de conexión a internet por ende existe mayor privacidad [40].

Con los resultados obtenidos podemos verificar el modelo 1 tiene un mejor funcionamiento como los valores de TensoBoard nos había predicho anteriormente. Debemos agregar que ambos modelos contenían un post procesamiento de supresión no máxima dentro de su configuración de entrenamiento, sin embargo durante esta prueba en el modelo 2 se encontraron una mayor cantidad de casos de doble detección en ambas clases del entrenamiento como se puede observar en la siguiente figura.

Figura 26: Ejemplo de las dobles detecciones del modelo 2

3.2.3. Pruebas del modelo en video

En esta sección se realizarán las pruebas del modelo 1 ya que fue quien ofreció mejores resultados en el entrenamiento y las diferentes evaluaciones mencionadas anteriormente. Haciendo uso del algoritmo de inferencias se realizaron las pruebas en tiempo real para comprobar la velocidad con la que el modelo realiza las inferencias con y sin el uso del acelerador TPU.

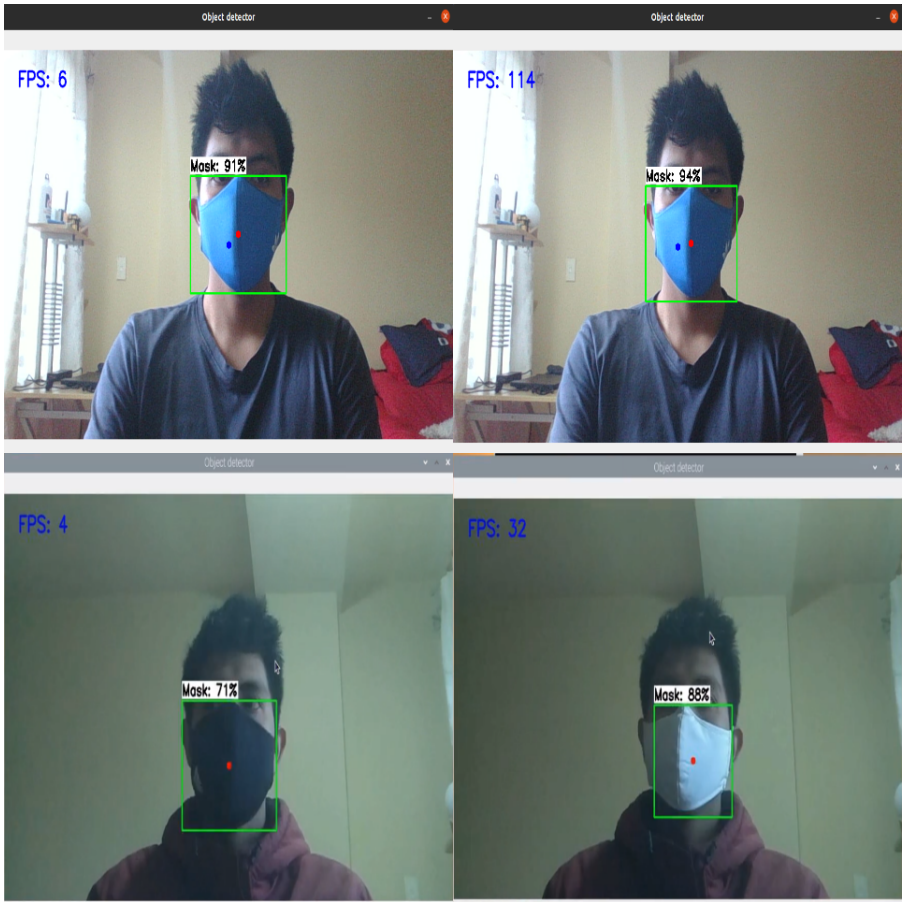


Figura 27: Velocidades máximas alcanzadas en las inferencias sobre diferentes dispositivos

La tabla 6 muestra tanto las características del dispositivo donde se realizó la prueba como el resultado obtenido durante una prueba de video de un minuto.

Tabla 6: Especificaciones y FPS máximos alcanzados con y sin el uso del acelerador TPU

Métrica	Laptop		Raspberry	
	Sin TPU	Con TPU	Sin TPU	Con TPU
Sistema operativo	Ubuntu		Raspbian	
Conexión USB	-	3.0	-	3.0
Tamaño del modelo (MB)	5.34	5.71	5.34	5.71
FPS alcanzados	6	114	4	32

Tanto en la laptop como en la Raspberry se hizo uso del conector USB 3.0 ya que sus velocidades de inferencia pueden diferir según el sistema y el tipo de conexión que se use. En la tabla 6 se puede observar una mejora considerable cuando corremos el algoritmo haciendo uso del acelerador edge TPU, comprobando que el dispositivo ofrece inferencia de aprendizaje automático de alta velocidad.

3.2.4. Pruebas del sistema de seguimiento

En esta sección se muestran los resultados de 3 experimentos del modelo trabajando con el sistema de seguimiento. Estos experimentos consisten en el seguimiento de un objeto durante tres trayectorias: horizontal (pan), vertical (tilt) y combinada (pan-tilt) obtener sus coordenadas para posteriormente realizar un mapeo de su trayectoria. En la figura 28 se presenta la secuencia de la trayectoria que se refiere al desplazamiento en el eje horizontal (pan), el movimiento vertical no es tomado en cuenta ya que para este experimento se desactivó el proceso que controla el cambio de posición de este eje.

Figura 28: Experimento 1, la secuencia de frames superior representa la trayectoria original y la secuencia inferior representa las imágenes captadas por el seguimiento

En la figura 29 se presenta la secuencia de la trayectoria del experimento 2, que se refiere al desplazamiento en el eje vertical (tilt), del mismo modo el movimiento horizontal no es tomado en cuenta ya que se desactivó el proceso que controla su cambio de posición.

Figura 29: Experimento 2, la secuencia de frames superior representa la trayectoria original y la secuencia inferior representa las imágenes captadas por el seguimiento

Por último la figura 30 muestra la secuencia de la trayectoria del experimento 3, que hace referencia al desplazamiento en ambos ejes, es decir se produce un seguimiento pan-tilt.

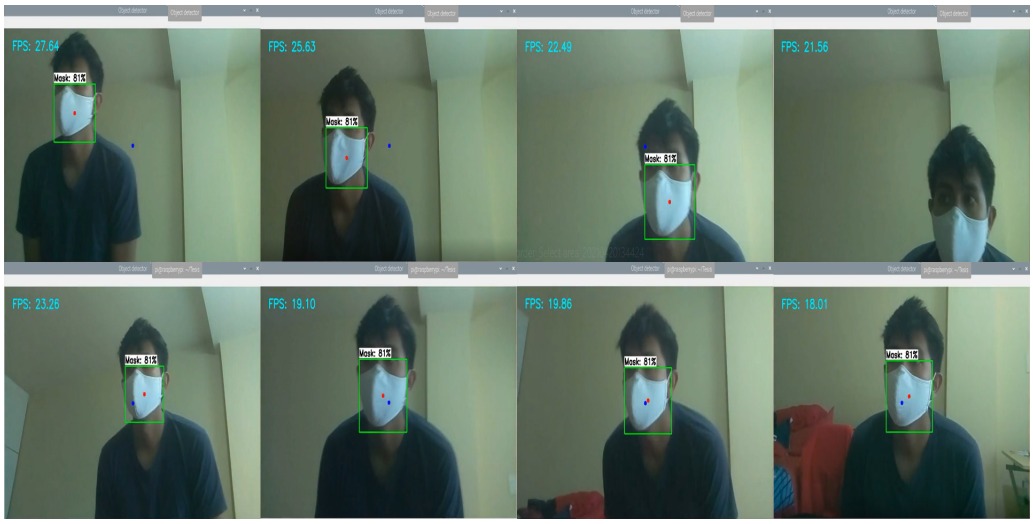


Figura 30: Experimento 3, la secuencia de frames superior representa la trayectoria original y la secuencia inferior representa las imágenes captadas por el seguimiento

Las figuras 31, 32, 33 muestran la trayectoria que sigue el centroide del objeto cuando se realiza o no el seguimiento.

En la figura 31.a se muestra la trayectoria que sigue el centroide del objeto si no se realiza seguimiento, mientras que en la figura 31.b se muestra un mapeo de la posición del centroide cuando se realiza el seguimiento del objeto.

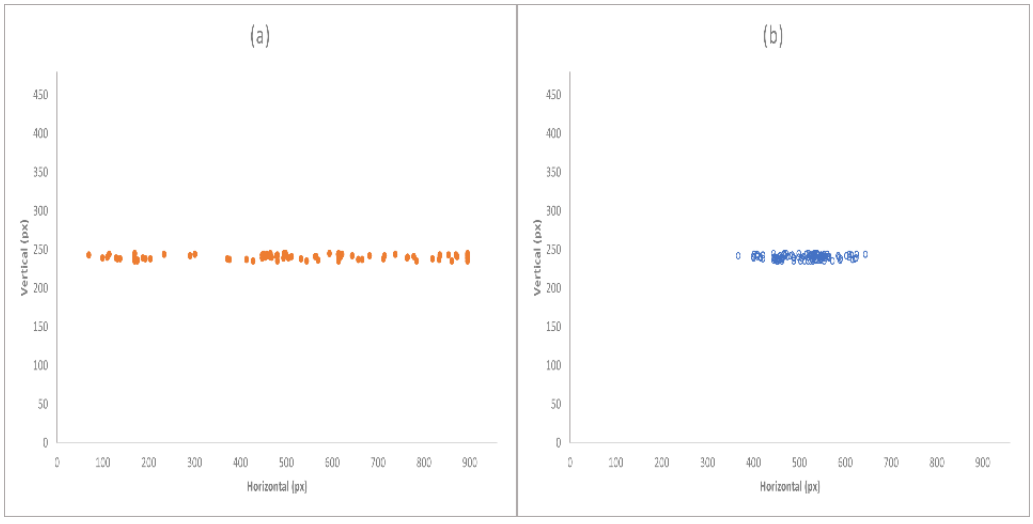


Figura 31: Posición del centroide del objeto en el experimento 1: (a) sin seguimiento y (b) con seguimiento

De modo similar para el segundo experimento observamos la trayectoria original del centroide en la figura 32.a y el mapeo de la trayectoria seguida del objeto en la figura 32.b.

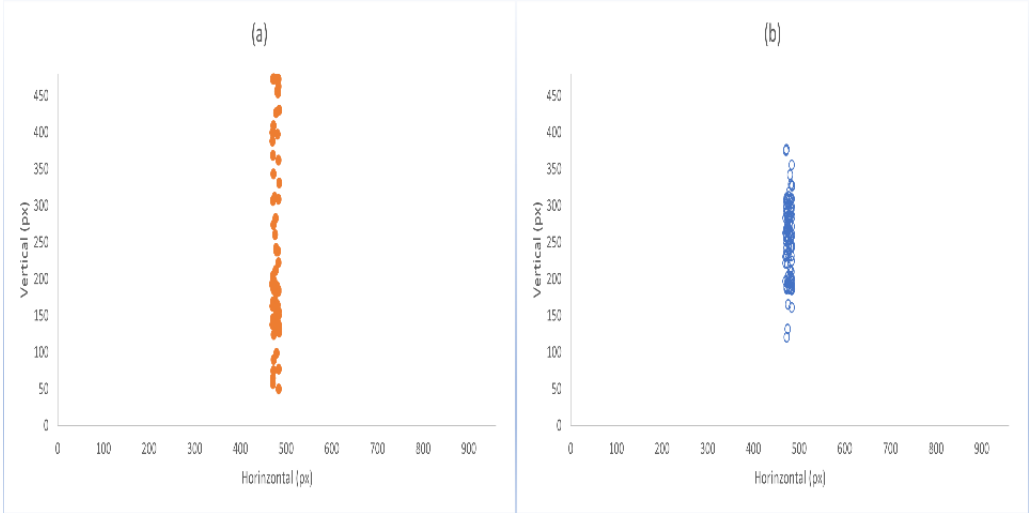


Figura 32: Posición del centroide del objeto en el experimento 2: (a) sin seguimiento y (b) con seguimiento.

Finalmente, en la figura 33 se visualiza el mapeo del centroide del último experimento cuando se realiza o no el seguimiento.

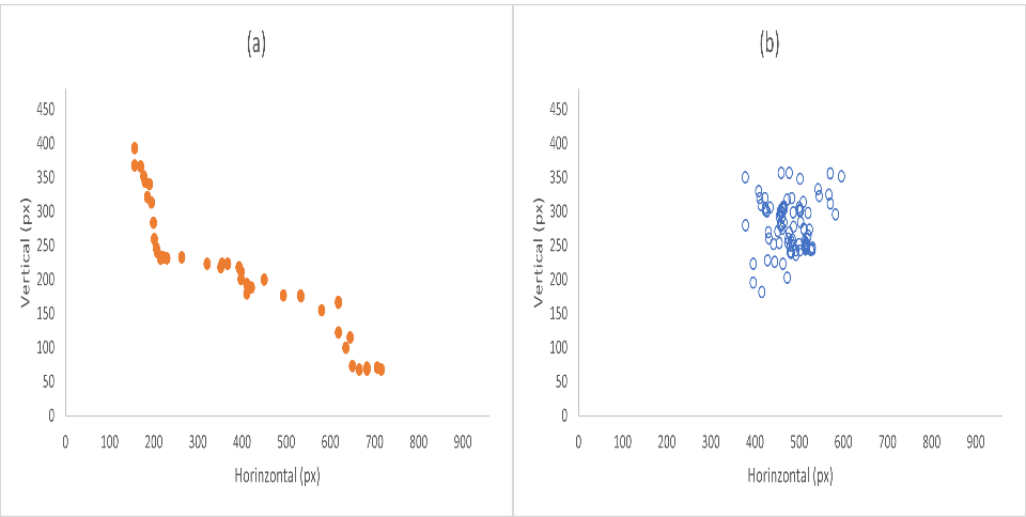


Figura 33: Posición del centroide del objeto en el experimento 3: (a) sin seguimiento y (b) con seguimiento.

Como podemos observar el las figuras anteriores, de manera independiente al dar seguimiento horizontal a un objeto el algoritmo consigue mantener aproximadamente el objetivo en el centro de la imagen con un error medio de 48±26 píxeles en el eje. Por otro lado, el seguimiento el el eje vertical cuenta con un error medio de 40±26 píxeles en el eje, tomando en cuenta algunas detecciones dispersas, ya que al momento de iniciar el algoritmo primero debe posicionarse sobre el objeto al cual se dará seguimiento.

Por último en la figura 33 representa el seguimiento combinado, activando ambos procesos de seguimiento se logró mantener aproximadamente centrado al objeto, donde el sistema es capaz de seguir el objeto con un error medio de 34±25 píxeles en el eje horizontal y de 32±20 píxeles en el eje vertical.

Capítulo 4

Conclusiones y trabajos a futuro

Se cumplió con el objetivo de realizar un sistema de seguimiento de objetos en tiempo real, a continuación se describirán las conclusiones del presente trabajo:

- Se diseñó una arquitectura que permite obtener una solución al seguimiento de objetos, que consiste en el re-entrenamiento de un modelo de redes neuronales convolucionales, para posteriormente ser optimizado y usado en el sistema de seguimiento que involucra movimientos de tipo Pan-Tilt.
- Al ejecutar las validaciones en dos modelos, se logró verificar que los datos proporcionados por TensorBoard y los datos de las pruebas de rendimiento sobre un conjunto de imágenes dan mejores resultados en el modelo 1 del entrenamiento II, que cuenta con una base de datos mas equilibrada con un número de iteraciones superior, obteniendo un 89.4 % de aciertos en las detecciones de la clase Mask.
- En las pruebas de velocidad realizadas para comprobar el máximo de FPS que podía alcanzar el modelo al usar el acelerador Coral edge TPU, se logra comprobar que el uso de este tipo de dispositivos brinda una máxima velocidad de inferencias en modelos de aprendizaje automático que cuenten con una arquitectura SSD, los únicos en la actualidad optimizados para ser usado con este dispositivo.
- Se logró el seguimiento de un objeto mediante el uso de un control PID donde se estableció los valores mediante un ajuste manual, en las pruebas realizadas para determinar la funcionalidad de sistema se obtuvo un error medio de 34 ± 25 pixeles para el movimiento horizontal y 32 ± 20 pixeles en el eje vertical, tomando en cuenta los valores de su posicionamiento inicial sobre el objeto.
- Se vio necesario la implementación de tiempos de delay en el código para la toma de cada coordenada, debido a que se debía disminuir la cantidad de datos que le llegaban al controlador PID y poder realizar el envió de la posición a los motores.

Como continuación de este trabajo de tesis, existen diversas líneas de investigación que quedan abiertas y en las que es posible continuar trabajando, algunas han ido surgiendo durante la realización del trabajo. A continuación se presentan algunos trabajos futuros que pueden desarrollarse como resultado de esta investigación:

- Como posibles mejoras del sistema mecánico de cara al futuro se plantea el cambio de cámara para permitir una mejor resolución y captación del objeto a seguir,

además de incluir una pantalla donde se muestren las imágenes del seguimiento sin la necesidad de optar por servidores VNC y una computadora. Como también el cambio de actuadores para proporcionar una mejor dinámica mejorando los tiempos de control con respuestas mas rápidas.

- En la parte de entrenamiento del modelo como una posible mejora, se aconseja la creación de un modelo desde cero con el uso de operaciones totalmente compatibles con las inferencias sobre el acelerador TPU, lo que brindará una mayor velocidad de inferencias al trabajar completamente con el dispositivo. Además es aconsejable diversificar la base de datos para mejorar los resultados sobre entornos con poca luz.

- De igual manera es aconsejable para futuros entrenamientos contar con una maquina personal que tenga una GPU integrada, donde se reducirían los tiempos de entrenamiento, ya que el servicio de Google Colab cuenta con limitaciones en cuanto al uso de una máquina virtual con GPU, además de contar con una sesión activa de 12 horas en donde se puede trabajar antes de actualizar el entorno.

Bibliografía

[1] C. Luo, X. Cai, and J. Zhang, "Robust object tracking using the particle filtering and level set methods: A comparative experiment," oct 2008.

[2] Y.-K. Jung, K.-W. Lee, and Y.-S. Ho, "Feature-based object tracking with an active camera," *Advances in Multimedia Information Processing*, pp. 1137–1144, 2002.

[3] Universo, "Coronavirus en ecuador: Normas para volver al trabajo en el sector público." Accedido en 10-05-2020 a https://www.eluniverso.com/noticias/2020/05/05/nota/7833156/coronavirus-ecuador-normas-volver-trabajo-sector-publico, 2020.

[4] M. Otazua, "Cámaras de detección y monitoreo térmico, tecnología de prevención frente al contagio y expansión del covid-19." Accedido en 03-05-2020 a https://www.tecnoseguro.com/analisis/cctv/camaras-deteccion-monitoreo-termico-prevencion-covid-19, 2020.

[5] L. S. Armajach, "Movimientos de la cámara - adictos al trabajo." Accedido en 03-05-2020 a https://www.adictosaltrabajo.com/2015/03/20/movimientos-de-camara/, 2020.

[6] S. R. Yosafat, C. Machbub, and E. M. I. Hidayat, "Design and implementation of pan-tilt control for face tracking," pp. 217–222, 2017.

[7] MathWorks, "Deep learning: Tres cosas que es necesario saber." Accedido en 03-12-2020 a https://la.mathworks.com/discovery/deep-learning.html, 2019.

[8] J. Marquez Diaz, "Inteligencia artificial y big data como soluciones frente a la covid-19," *Revista de Bioetica y Derecho*, no. 50, pp. 315–331, 2020.

[9] M. Gorini, "¿cuál es la diferencia entre el machine learning y el deep learning?." Accedido en 12-12-2020 a https://blog.bismart.com/es/diferencia-machine-learning-deep-learning, 2020.

[10] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[11] R. Mehra *et al.*, "Breast cancer histology images classification: Training from scratch or transfer learning?," *ICT Express*, vol. 4, no. 4, pp. 247–254, 2018.

[12] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[13] P. Goldsborough, "A tour of tensorflow," *arXiv preprint arXiv:1610.01178*, 2016.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[15] TensorFlow, "Deploy machine learning models on mobile and iot devices." Accedido en 10-12-2020 a https://www.tensorflow.org/lite/guide, 2019.

[16] J. Kopepasah, "Modelos de detección de objetos." Accedido en 15-09-2020 a https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/, Agosto. 21, 2017.

[17] P. Guerra Toni *et al.*, "Detección de objetos en imágenes urbanas de google street view," B.S. thesis, 2019.

[18] M. Burić, M. Pobar, and M. Ivašić-Kos, "Adapting yolo network for ball and player detection," in *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2019)*, pp. 845–851, 2019.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[20] J. Pérez Nasser, "Análisis de tráfico vehicular mediante visión artificial," Master's thesis, Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas ..., 2019.

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[22] J. Xu, "Uso del aprendizaje profundo para el reconocimiento de objetos." Accedido en 16-09-2020 a https://www.deeplearningitalia.com/uso-del-aprendizaje-profundo-para-el-reconocimiento-de-objetos/, 2018.

[23] R. Khandelwal, "Ssd : Single shot detector for object detection using multibox." Accedido en 16-09-2020 a https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca, 2019.

[24] D. Xianzhi and K. Jaeyoun, "Spinenet: A novel architecture for object detection discovered with neural architecture search." Accedido en 18-09-2020 a https://ai.googleblog.com/2020/06/spinenet-novel-architecture-for-object.html, 2020.

[25] X. Du, T.-Y. Lin, P. Jin, G. Ghiasi, M. Tan, Y. Cui, Q. V. Le, and X. Song, "Spinenet: Learning scale-permuted backbone for recognition and localization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11592–11601, 2020.

[26] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.

[27] R. Iguernaissi, D. Merad, K. Aziz, and P. Drap, "People tracking in multi-camera systems: a review," *Multimedia Tools and Applications*, vol. 78, no. 8, pp. 10773–10793, 2019.

[28] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer, "Multi-camera multi-person tracking for easyliving," in *Proceedings Third IEEE International Workshop on Visual Surveillance*, pp. 3–10, IEEE, 2000.

[29] W. Mayol, B. Tordoff, and D. W. Murray, "Wearable visual robots," *Personal and Ubiquitous Computing*, vol. 6, no. 1, pp. 37–48, 2002.

[30] A. Bernardino and J. Santos-Victor, "Binocular tracking: integrating perception and control," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 6, pp. 1080–1094, 1999.

[31] A. F. Roos, H. V. Neto, *et al.*, "Towards saliency-based gaze control in a binocular robot head," in *Proceedings of the 6th UNICAMP Symposium on Signal Processing, Campinas, Brazil*, 2015.

[32] G. J. Garcia, J. Pomares, and F. Torres, "A new time-independent image path tracker to guide robots using visual servoing," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pp. 957–964, IEEE, 2007.

[33] H. Wang, D. Guo, H. Xu, W. Chen, T. Liu, and K. K. Leang, "Eye-in-hand tracking control of a free-floating space manipulator," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 4, pp. 1855–1865, 2017.

[34] W. Intelligence, "Face mask detection dataset." Accedido en 19-09-2020 a https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset, Jun 2020.

[35] Sumansid, "Facemask dataset." Accedido en 19-09-2020 a https://www.kaggle.com/sumansid/facemask-dataset, Jun 2020.

[36] A. Talele, A. Patil, and B. Barse, "Detection of real time objects using tensorflow and opencv," *Asian Journal For Convergence In Technology (AJCT)*, 2019.

[37] T. Tataru, "Pixel, a virtual assistant with face recognition," 2021.

[38] FileZilla, "Documentation - filezilla wiki," 2020.

[39] COCO, "Common objects in context." Accedido en 10-12-2020 a https://cocodataset.org/detection-eval.

[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner,

I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

Apéndice A

En este apartado de apéndices muestra los diferentes códigos que se han realizado para llevar a cabo el desarrollo del sistema de seguimiento de un objeto en tiempo real.

Código principal: Seguimiento

```
#importar librerias necesarias
from multiprocessing import Manager
from multiprocessing import Process
from Inferencias import *
from Control import PID
import signal
import time
import sys
import cv2
import pigpio
import pandas as pd

# definir el rango seguro de los motores
servoRangePan = (500,2490)
servoRangeTlt = (1670, 2490)

# funcion para interrunpir la ejecucion con CTRL+C
def signal_handler(sig, frame):
    sys.exit()

def obj_center(objX,objY,centerX,centerY):
    signal.signal(signal.SIG_IGN,signal_handler)
    ObjDetector().start(objX,objY,centerX,centerY)


def pid_pross(output, p, i, d, objCoord, centerCoord):
    signal.signal(signal.SIGINT, signal_handler)
    # crear el PID e inicializarlo
    p = PID(p.value, i.value, d.value)
    p.initialize()
    contador = 0

    # loop indefinido
    while True:
        # calcular el error
        error = centerCoord.value - objCoord.value

        #Guardar los datos en un registro
        contador = contador + 1
```

```
40          df = open('panCE2.csv','a')
41          df.write(str(objCoord.value))
42          df.write('\n')
43
44          # actualizamos el valor de salida
45          if objCoord.value == 0  or error ==480 or error ==270:
46              output.value = 0
47          else:
48              output.value = p.update(error)
49          #print(error,output)
50
51  def in_range(val, start, end):
52      return val >= start and val <= end
53
54  def set_servos(pan_delta, tilt_delta):
55      #Declaramos pines y posicion inicial
56      signal.signal(signal.SIGINT, signal_handler)
57      pan = 27
58      tlt = 17
59      pi = pigpio.pi()
60      pi.set_mode(pan, pigpio.OUTPUT)
61      pi.set_mode(tlt, pigpio.OUTPUT)
62      pi.set_servo_pulsewidth(pan, 800)
63      time.sleep(0.3)
64      pi.set_servo_pulsewidth(tlt, 1790)
65
66      while True:
67          #Calcular movimientos
68          pan_change = pan_delta.value * -1
69          pan_pulse_width =  pi.get_servo_pulsewidth(pan) + pan_change
70          tlt_change = tilt_delta.value * -1
71          tlt_pulse_width =  pi.get_servo_pulsewidth(tlt) + tlt_change
72
73          #comprobar dentro del rango y movimiento
74          if in_range(pan_pulse_width, servoRangePan[0], servoRangePan[1]):
75              pi.set_servo_pulsewidth(pan, pan_pulse_width)
76              #print(pan_pulse_width)
77          if in_range(tlt_pulse_width, servoRangeTlt[0], servoRangeTlt[1]):
78              pi.set_servo_pulsewidth(tlt, tlt_pulse_width)
79              #print(tlt_pulse_width)
80
81          time.sleep(0.05)
82
83  if __name__ == "__main__":
84
85      # iniciar el gestor para las variables e iniciar los procesos
86      with Manager() as manager:
87
88          # valores enteros para las coordenadas del centro del frame (x, y
        )
89          centerX = manager.Value("i", 0)
90          centerY = manager.Value("i", 0)
91
92          # valores enteros para las coordenadas (x, y) del objeto
93          objX = manager.Value("i", 0)
94          objY = manager.Value("i", 0)
95
```

```
96          # valores de giro e inclinación serán gestionados por PIDs
    independientes
97          pan_delta = manager.Value("i", 0)
98          tlt_delta = manager.Value("i", 0)
99
100         # establecer los valores PID para el desplazamiento panoramico
101         panP = manager.Value("f", 0.062)
102         panI = manager.Value("f", 0.0055)
103         panD = manager.Value("f", 0.0022)
104
105         # establecer valores PID para la inclinación
106         tiltP = manager.Value("f", 0.07)
107         tiltI = manager.Value("f", 0.0045)
108         tiltD = manager.Value("f", 0.0022)
109
110         #4 procesos independientes
111         prossObjCenter = Process(target=obj_center, args=(objX,objY,
    centerX,centerY)) #localiza los objetivos
112         prossPan = Process(target=pid_pross, args=(pan_delta, panP, panI,
     panD, objX, centerX)) #determina el valor de giro
113         prossTilt = Process(target=pid_pross, args=(tlt_delta, tiltP,
    tiltI, tiltD, objY, centerY)) #determina el valor de inclinación
114         prossSetServos = Process(target=set_servos, args=(pan_delta,
    tlt_delta)) #acción de motores
115
116         # iniciamos los 4 procesos
117         prossObjCenter.start()
118         prossPan.start()
119         prossTilt.start()
120         prossSetServos.start()
121
122         # Unimos los 4 procesos
123         prossObjCenter.join()
124         prossPan.join()
125         prossTilt.join()
126         prossSetServos.join()
```

Código 1: Código del seguimiento

Código de control

```
1
2 # importar librerias necesarias
3 import time
4
5 class PID:
6   def __init__(self, kP=1, kI=0, kD=0): # Constructor
7     # inicializar las ganancias
8     self.kP = kP
9     self.kI = kI
10    self.kD = kD
11
12  def initialize(self):
13    # inicializar el tiempo actual y el anterior
14    self.currTime = time.time()
```

```
15      self.prevTime = self.currTime
16
17      # inicializar el error anterior
18      self.prevError = 0
19
20      # inicializar las variables de resultado de los términos
21      self.cP = 0
22      self.cI = 0
23      self.cD = 0
24
25    # Metodo donde se realizan los calculos
26    def update(self, error, sleep=0.2): #Valor de sleep debe tomar en
        cuenta limitaciones mecanicas y computacionales
27      # pausa por un momento
28      time.sleep(sleep)
29
30      # toma la hora actual y calcula el tiempo delta
31      self.currTime = time.time()
32      deltaTime = self.currTime - self.prevTime
33
34      # error delta
35      deltaError = error - self.prevError
36
37      # termino proporcional
38      self.cP = error
39
40      # termino proporcional
41      self.cI += error * deltaTime
42
43      # termino derivado y evitar la division por cero
44      self.cD = (deltaError / deltaTime) if deltaTime > 0 else 0
45
46      # guardar el tiempo y el error anteriores para la siguiente
        actualizacion
47      self.prevTime = self.currTime
48      self.prevError = error
49
50      # suma las condiciones y devuelve
51      return sum([
52        self.kP * self.cP,
53        self.kI * self.cI,
54        self.kD * self.cD])
```

Código 2: Código de los cálculos PID

Código de las inferencias

```
1
2 # Importando librerias necesarias
3 import os
4 import cv2
5 import numpy as np
6 import sys
7 import time
8 from threading import Thread
```

```
9  import importlib.util
10 import pyautogui
11
12
13
14 '''Definir la clase VideoStream para manejar la
15    transmisión de video en un hilo de procesamiento separado por Adrian
      Rosebrock'''
16
17 class VideoStream:
18     def __init__(self,resolution,framerate):
19         # Inicializar la PiCamera y el flujo de imágenes de la cámara
20         self.stream = cv2.VideoCapture(0)
21         ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc
       (*'MJPG'))
22         ret = self.stream.set(3,resolution[0])
23         ret = self.stream.set(4,resolution[1])
24
25         # Leer el primer fotograma del flujo
26         (self.grabbed, self.frame) = self.stream.read()
27
28       # Variable para controlar cuándo se detiene la cámara
29         self.stopped = False
30
31     def start(self):
32       # Iniciar el hilo que lee los fotogramas del flujo de vídeo
33         Thread(target=self.update,args=()).start()
34         return self
35
36     def update(self):
37         # Mantener el bucle indefinidamente hasta que el hilo se detenga
38         while True:
39             # Si la cámara está detenida, detenga el hilo
40             if self.stopped:
41                 # Cerrar los recursos de la cámara
42                 self.stream.release()
43                 return
44
45             # En caso contrario, coge el siguiente fotograma del flujo
46             (self.grabbed, self.frame) = self.stream.read()
47
48     def read(self):
49       # Devuelve el fotograma más reciente
50         return self.frame
51
52     def stop(self):
53       # Indicar que la cámara y el hilo deben detenerse
54         self.stopped = True
55
56
57 class ObjDetector:
58     def __init__(self):
59
60         #Definimos variables y modelos
61         MODEL_NAME = 'TFLite_model'
62         GRAPH_NAME = 'model_edgetpu.tflite'
63         LABELMAP_NAME = 'labelmap.txt'
```

```
64          self.min_umbral = 0.7
65          self.imW, self.imH = 960,540
66          use_TPU = True
67          pkg = importlib.util.find_spec('tflite_runtime')
68
69          #Verificamos la instalación de la librería para uso de TPU
70          if pkg:
71              from tflite_runtime.interpreter import Interpreter
72              if use_TPU:
73                  from tflite_runtime.interpreter import load_delegate
74                  print('Usando_TPU')
75          else:
76              from tensorflow.lite.python.interpreter import Interpreter
77              if use_TPU:
78                  from tensorflow.lite.python.interpreter import
    load_delegate
79
80          # Ruta del directorio de trabajo actual
81          CWD_PATH = os.getcwd()
82
83          # Ruta al archivo .tflite, que contiene el modelo
84          PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
85
86          # Ruta de acceso al archivo de mapa de etiquetas
87          PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
88
89          # Cargar el mapa de etiquetas
90          with open(PATH_TO_LABELS, 'r') as f:
91              self.labels = [line.strip() for line in f.readlines()]
92
93          if use_TPU:
94              self.interpreter = Interpreter(model_path=PATH_TO_CKPT,
    experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
95              print(PATH_TO_CKPT)
96          else:
97              self.interpreter = Interpreter(model_path=PATH_TO_CKPT)
98
99          self.interpreter.allocate_tensors()
100
101     def start(self,objX, objY,centerX,centerY):
102         # Obtener detalles del modelo a ejecutar
103         input_details = self.interpreter.get_input_details()
104         output_details = self.interpreter.get_output_details()
105         height = input_details[0]['shape'][1]
106         width = input_details[0]['shape'][2]
107         floating_model = (input_details[0]['dtype'] == np.float32)
108
109         #Valores para normalizar si fuese necesario
110         input_mean = 127.5
111         input_std = 127.5
112
113         # Inicializar el cálculo de la velocidad de fotogramas
114         frame_rate_calc = 1
115         freq = cv2.getTickFrequency()
116
117         # Inicializar videoStream
```

```
118         videostream = VideoStream(resolution=(self.imW, self.imH),
    framerate=30).start()
119         time.sleep(1)
120 
121         while True:
122 
123             # Temporizador de inicio (para calcular la velocidad de
    fotogramas)
124             t1 = cv2.getTickCount()
125 
126             # Tomar un fotograma del video stream
127             frame1 = videostream.read()
128 
129             # Adquirir el marco y cambiar el tamaño
130             frame = frame1.copy()
131             frame = cv2.flip(frame,0)
132             frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
133             frame_resized = cv2.resize(frame_rgb, (width, height))
134             input_data = np.expand_dims(frame_resized, axis=0)
135 
136             # Normalizar los valores de los píxeles si se utiliza un
    modelo flotante (es decir, si el modelo no está cuantificado)
137             if floating_model:
138                 input_data = (np.float32(input_data) - input_mean) /
    input_std
139 
140             # Realizar la detección real ejecutando el modelo con la
    imagen como entrada
141             self.interpreter.set_tensor(input_details[0]['index'],
    input_data)
142             self.interpreter.invoke()
143 
144             # Recuperar los resultados de la detección
145             boxes = self.interpreter.get_tensor(output_details[0]['index'
    ])[0] # Coordenadas de la caja delimitadora de los objetos detectados
146             classes = self.interpreter.get_tensor(output_details[1]['
    index'])[0] # Índice de clase de los objetos detectados
147             scores = self.interpreter.get_tensor(output_details[2]['index
    '])[0] # Confianza de los objetos detectados
148 
149             # Recorrer todas las detecciones y dibujar el cuadro de
    detección si la confianza está por encima del umbral mínimo
150             for i in range(len(scores)):
151                 if ((scores[i] > self.min_umbral) and (scores[i] <= 1.0))
    :
152 
153                     # Obtener las coordenadas de la caja delimitadora y
    dibujarla dentro del marco
154 
155                     ymin = int(max(1,(boxes[i][0] * self.imH)))
156                     xmin = int(max(1,(boxes[i][1] * self.imW)))
157                     ymax = int(min(self.imH,(boxes[i][2] * self.imH)))
158                     xmax = int(min(self.imW,(boxes[i][3] * self.imW)))
159 
160                     objX.value = 0
161                     objY.value = 0
162 
```

```
163                     #Verificamos el objeto detectado
164                     if self.labels[int(classes[i])] =="Mask":
165 
166                         objX.value = (xmax + xmin)/2
167                         objY.value = (ymax + ymin)/2
168                         centerX.value=self.imW/2
169                         centerY.value=self.imH/2
170 
171                         cv2.circle(frame, (int(objX.value),int(objY.value)), 5, (0, 0, 255), -1) #centro del objeto
172                         cv2.circle(frame, (int(centerX.value),int(centerY.value)), 5, (255, 0, 0), -1) #Centro del marco
173 
174                     cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
175 
176                     # Dibujar etiquetas
177                     object_name = self.labels[int(classes[i])] # Busca el nombre del objeto en la matriz
178                     label = '%s: %d%%' %(object_name, int(scores[i]*100))
179                     labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Obtener el tamaño de la fuente
180                     label_ymin = max(ymin, labelSize[1] + 10) # se asegura de no salirse del marco
181                     cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)
182                     cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Dibujar el texto de la etiqueta
183                     time.sleep(0.001)
184 
185             #  Calcular la velocidad de fotogramas
186             t2 = cv2.getTickCount()
187             time1 = (t2-t1)/freq
188             frame_rate_calc= 1/time1
189 
190             # mostrar la velocidad de fotogramas
191             cv2.putText(frame,'FPS: {0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)
192             cv2.imshow('Object detector', frame)
193 
194             # Salir
195             if cv2.waitKey(1) == ord('q'):
196                 print('Cerrando')
197                 break
198 
199         # Limpiar
200         cv2.destroyAllWindows()
201         #pyautogui.hotkey('ctrl', 'c')
202         videostream.stop()
```

Código 3: Código de las inferencias sobre el acelerador TPU

Códigos usados en Colab

```

#CONFIGURACIÓN DE RUTAS
#CUSTOM_MODEL_NAME = 'ssdlite_mobiledet_edgetpu_320x320_coco_2020_05_19'
CUSTOM_MODEL_NAME = '
    ssd_mobilenet_v1_quantized_300x300_coco14_sync_2018_07_18'

APIMODEL_PATH = 'models'
WORKSPACE_PATH = 'workspace/MASK'
MODEL_PATH = WORKSPACE_PATH + '/models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH + '/pre-trained-models'
CHECKPOINT_PATH = MODEL_PATH + '/' + CUSTOM_MODEL_NAME

IMAGE_PATH = '/content/carga/MyDrive/workspace/images-todo'
ANNOTATION_PATH = '/content/carga/MyDrive/workspace/annotations'
OUTPUT_PATH = '/content/carga/MyDrive/workspace/tf1/'+ CUSTOM_MODEL_NAME
    + '/output'
EXPORT_PATH = '/content/carga/MyDrive/workspace/tf1/'+ CUSTOM_MODEL_NAME
    + '/export'
```

Código 4: Código usado para definir rutas

```

import os
import re
from shutil import copyfile
import math
import random


source = IMAGE_PATH
dest = source
ratio = 0.2
copy_xml = True

source = source.replace('\\', '/')
dest = dest.replace('\\', '/')
train_dir = os.path.join(dest, 'train')
test_dir = os.path.join(dest, 'test')

if not os.path.exists(train_dir):
    os.makedirs(train_dir)
if not os.path.exists(test_dir):
    os.makedirs(test_dir)

images = [f for f in os.listdir(source) if re.search(r'([a-zA-Z0-9\s_
    \\.\-\(\):])+(.jpg|.jpeg|.png)$', f)]

num_images = len(images)
num_test_images = math.ceil(ratio*num_images)

for i in range(num_test_images):
    idx = random.randint(0, len(images)-1)
    filename = images[idx]
    copyfile(os.path.join(source, filename), os.path.join(test_dir,
    filename))
```

```
    if copy_xml == True:
        xml_filename = os.path.splitext(filename)[0]+'.xml'
        copyfile(os.path.join(source, xml_filename), os.path.join(
    test_dir,xml_filename))
        images.remove(images[idx])

for filename in images:
    copyfile(os.path.join(source, filename), os.path.join(train_dir,
    filename))
    if copy_xml == True:
        xml_filename = os.path.splitext(filename)[0]+'.xml'
        copyfile(os.path.join(source, xml_filename), os.path.join(
    train_dir, xml_filename))
```

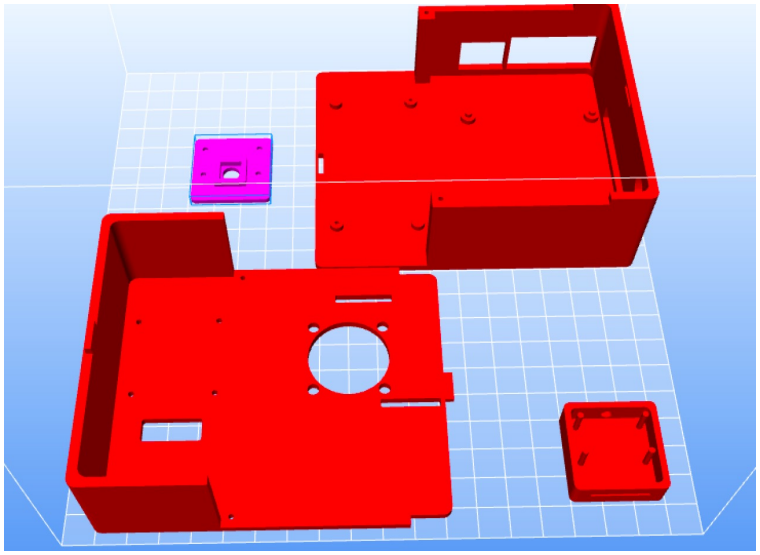
Código 5: Código usado en la base de datos

```

import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format

CONFIG_PATH = CHECKPOINT_PATH + '/pipeline.config'
config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)

pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)

pipeline_config.model.ssd.num_classes = 2
pipeline_config.train_config.batch_size = 30
pipeline_config.train_config.fine_tune_checkpoint = PRETRAINED_MODEL_PATH
    +'/'+ CUSTOM_MODEL_NAME + '/model.ckpt'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH + '/
    label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
    [ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH + '
    /label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:]
    = [ANNOTATION_PATH + '/test.record']

# Entrenamiento conciente de quantización
pipeline_config.graph_rewriter.quantization.delay = 15000
pipeline_config.graph_rewriter.quantization.weight_bits = 8
pipeline_config.graph_rewriter.quantization.activation_bits = 8

config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
    f.write(config_text)
```

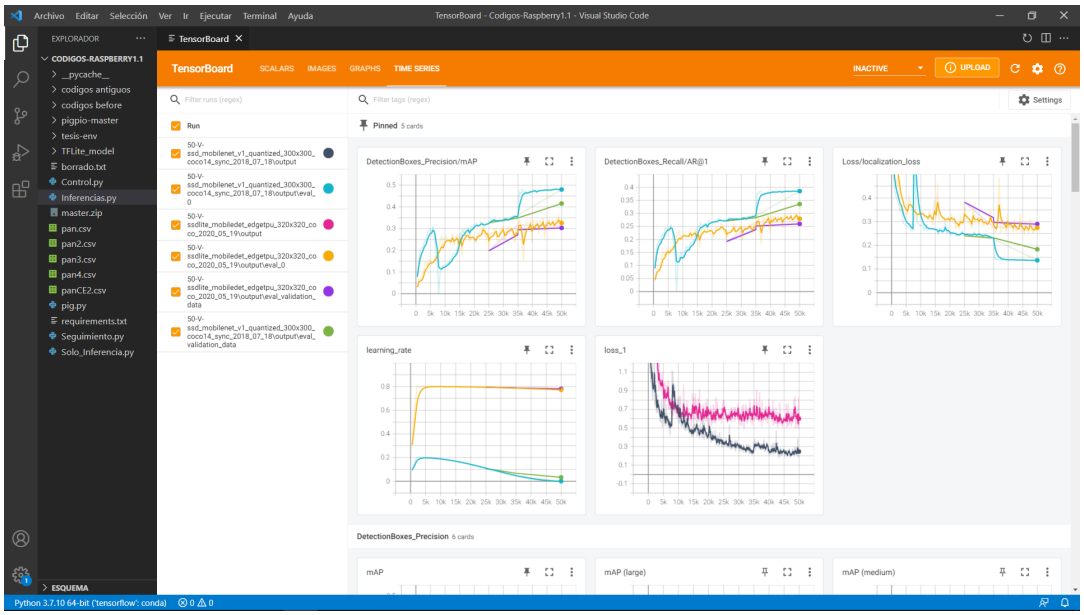
Código 6: Código usado para la modificación del archivo de configuración

El resto de códigos usados se encuentran en el repositorio de Github.

Apéndice B



Anexo 1: Archivos para impresión 3D de carcasa y soporte de cámara



Anexo 2: Resultados de la recuperación obtenidos de Tensorboard para el entrenamiento desde un punto de control