Resumen

La presente memoria detalla el proceso de creación de un módulo de inteligencia artificial diseñado para detectar personas y animales en vídeo de vigilancia capturado con cámaras montadas en drones. En particular, el objetivo de este trabajo es detectar intrusos en streaming de vídeo de seguridad. El módulo se desarrolló en colaboración con Manta Beach, empresa que hace parte del programa de vinculación de la universidad y que propuso el proyecto como parte de su plan de investigación y desarrollo.

El desarrollo de este trabajo hace un uso extensivo de los conocimientos de la rama de visión por computadora a través de redes neuronales convolucionales. De igual manera, se ha requerido una base sólida en Python, teniendo en cuenta que se partió de un algoritmo establecido con anterioridad.

I

Resumen

La presente memoria detalla el proceso de creación de un módulo de inteligencia artificial, diseñado para detectar personas y animales en vídeo de vigilancia capturado con cámaras montadas en drones. En particular, lo que se busca con el desarrollo de este trabajo es detectar intrusos en streaming de vídeo como parte de un sistema más grande de vigilancia, que será implementado por la empresa Manta Beach. Esta empresa, que hace parte del programa de vinculación de la universidad, propuso el trabajo como parte de su plan de investigación y desarrollo.

Para la creación de este módulo,se hace un uso extensivo de los conocimientos de la rama de visión por computadora a través de redes neuronales convolucionales. De igual manera, se ha requerido una base sólida en Python, teniendo en cuenta que se partió de un algoritmo establecido con anterioridad.

Índice general

v

Índice general

Índice de figuras

VII

Índice de figuras

Índice de tablas

IX

Índice de tablas

Modelos YOLO - Lanzamiento	
Modelo	Fecha de Lanzamiento
YOLOv3	Abril de 2018
YOLOv3 Lite	Abril de 2020
YOLOv6	Junio de 2022

TABLA 1.1. Fecha de lanzamiento de distintos modelos YOLO.

1.3. Estado del arte

Existen varias técnicas y varios modelos que se pueden utilizar para implementar visión por computadora. Uno de los métodos más comunes a la fecha es el algoritmo YOLO, que se caracteriza por ser el primer modelo que no requiere hacer un doble análisis de la imagen [3](de aquí proviene su nombre: *You Only Look Once*), y en consecuencia, analiza las imágenes en un tiempo mucho menor.

A la fecha de redacción de este documento, las versiones más avanzadas de los modelos de la familia YOLO son las versiones v6 y v7, que ofrecen grandes mejoras en términos de precisión y rendimiento en comparación con sus antecesores. Sin embargo, como se puede ver en la tabla 1.1, estos modelos no estuvieron disponibles hasta una fecha posterior al planteamiento del trabajo.

Por otro lado, las versiones 4 y 5 fueron descartadas para la implementación de este trabajo, visto que son dos versiones que fueron desarrolladas por personas diferentes a los desarrolladores originales, quienes se retiraron del proyecto por conflictos éticos con las capacidades de la técnica y sus potenciales usos [4]. En consecuencia, es muy extendido el uso de la versión 3 del modelo a pesar de no ser el más actualizado.

Ahora bien, es importante tener en cuenta que existen diferentes versiones del modelo, cada una adaptada a algún caso de uso particular. Destacan dentro de estas versiones la arquitectura YOLOv3 y la arquitectura YOLOv3 Lite:

- YOLOv3: es la versión "base" de la arquitectura YOLO. Representa un avance incremental con respecto a las arquitecturas YOLOv1 y YOLOv2. Su principal ventaja por encima de los modelos anteriores (en particular comparado con RetinaNet-50 y RetinaNet-101) es una reducción significativa en los tiempos de detección. El *dataset* sobre el que fue entrenado es el *dataset* COCO.
- YOLOv3 Lite: es una versión reducida de la versión YOLOv3 diseñada específicamente para ser ejecutada en dispositivos con menor potencia, o que no cuentan con tarjeta gráfica [5]. Si bien es un modelo que es capaz de reducir sustancialmente la complejidad del equipo requerido, también cuenta con una menor precisión en el momento de hacer la detección correspondiente.

Por otro lado, además de la arquitectura, es de vital importancia tener en cuenta el *dataset* que fue utilizado para el entrenamiento del modelo. En este caso, se trata del dataset COCO, que cuenta con más de 200.000 imágenes etiquetadas (a la fecha de redacción de este documento), con más de 1.5 millones de objetos detectados en ellas. [6]

1.3. Estado del arte 3

TABLA 1.1. Fecha de lanzamiento de distintos modelos YOLO.

Modelos YOLO - Lanzamiento	
Modelo	Fecha de Lanzamiento
YOLOv3	Abril de 2018
YOLOv3 Lite	Abril de 2020
YOLOv6	Junio de 2022

1.3. Estado del arte

Existen varias técnicas y varios modelos que se pueden utilizar para implementar visión por computadora. Uno de los métodos más comunes a la fecha es el algoritmo YOLO, que se caracteriza por ser el primer modelo que no requiere hacer un doble análisis de la imagen [3](de aquí proviene su nombre: *You Only Look Once*), y en consecuencia, analiza las imágenes en un tiempo mucho menor.

A la fecha de redacción de este documento, las versiones más avanzadas de los modelos de la familia YOLO son las versiones v6 y v7, que ofrecen grandes mejoras en términos de precisión y rendimiento en comparación con sus antecesores. Sin embargo, como se puede ver en la tabla 1.1, estos modelos no estuvieron disponibles hasta una fecha posterior al planteamiento del trabajo.

Por otro lado, las versiones 4 y 5 fueron descartadas para la implementación de este trabajo, visto que son dos versiones que fueron desarrolladas por personas diferentes a los desarrolladores originales, quienes se retiraron del proyecto por conflictos éticos con las capacidades de la técnica y sus potenciales usos [4]. En consecuencia, es muy extendido el uso de la versión 3 del modelo a pesar de no ser el más actualizado.

Ahora bien, es importante tener en cuenta que existen diferentes versiones del modelo, cada una adaptada a algún caso de uso particular. Destacan dentro de estas versiones la arquitectura YOLOv3 y la arquitectura YOLOv3 Lite:

- YOLOv3: es la versión "base" de la arquitectura YOLO. Representa un avance incremental con respecto a las arquitecturas YOLOv1 y YOLOv2. Su principal ventaja por encima de los modelos anteriores (en particular comparado con RetinaNet-50 y RetinaNet-101) es una reducción significativa en los tiempos de detección. El *dataset* sobre el que fue entrenado es el *dataset* COCO.
- YOLOv3 Lite: es una versión reducida de la versión YOLOv3 diseñada específicamente para ser ejecutada en dispositivos con menor potencia, o que no cuentan con tarjeta gráfica [5]. Si bien es un modelo que es capaz de reducir sustancialmente la complejidad del equipo requerido, también cuenta con una menor precisión en el momento de hacer la detección correspondiente.

Por otro lado, además de la arquitectura, es de vital importancia tener en cuenta el *dataset* que fue utilizado para el entrenamiento del modelo. En este caso, se trata del dataset COCO, que cuenta con más de 200.000 imágenes etiquetadas (a la fecha de redacción de este documento), con más de 1.5 millones de objetos detectados en ellas. [6]

1.4. Objetivos y alcance

A continuación, se presentan los requisitos establecidos en la etapa de apertura del trabajo y su estado actual a la fecha. Es necesario tener en cuenta que, debido a cambios operativos, algunos de estos requisitos no pudieron ser cumplidos en su totalidad.

4 Capítulo 1. Introducción general

A continuación se puede ver una ilustración del funcionamiento de las redes neuronales convolucionales a través de las que funciona el modelo YOLOv3:

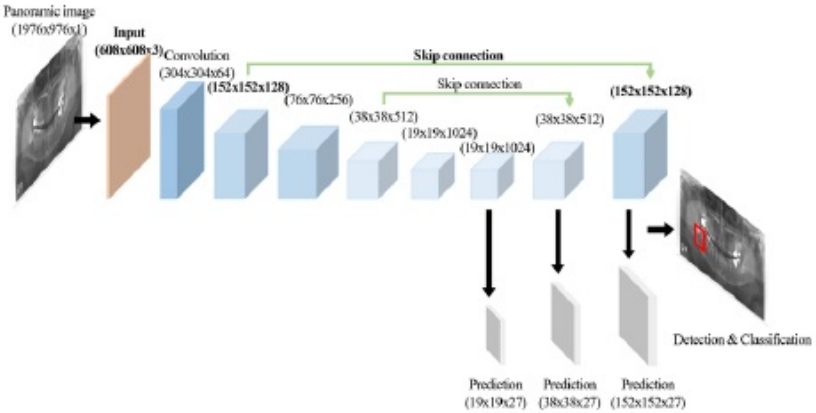

FIGURA 1.1. Esquema de redes neuronales de YOLOv3[7]

1.4. Objetivos y alcance

A continuación, se presentan los requisitos establecidos en la etapa de apertura del trabajo y su estado actual a la fecha. Es necesario tener en cuenta que, debido a cambios operativos, algunos de estos requisitos no pudieron ser cumplidos en su totalidad.

Capítulo 2

Introducción específica

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

2.1. Definición de Intruso

Dado que el trabajo tiene un objetivo específico en mente, la definición de intruso dada por el cliente incluye únicamente personas o animales de gran porte. El *dataset* de COCO permite el reconocimiento de una serie de animales de gran porte que no hacen parte de la fauna argentina, por lo que se decidió en conjunto con el cliente que algunas etiquetas se filtrarían a fin de evitar que el modelo trate de reconocer estas clases. Algunos ejemplos de etiquetas filtradas incluyen:

- Elefantes
- Jirafas
- Cebras

Por otro lado, y a fin de mejorar el reconocimiento de intrusos, se decidió incorporar algunas clases que corresponden a objetos que suelen cargar las personas y vehículos. De esta manera se hace posible una suerte de reconocimiento indirecto, dándole al modelo una "segunda oportunidad" en casos en los que por algún motivo generó un falso negativo en el reconocimiento de una persona. Es importante tener en cuenta que estos objetos propuestos ya hacían parte del *dataset* COCO, por lo que no se requirió reentrenar el modelo para lograr su detección. Dentro de estos objetos relacionados encontramos:

- Vehículos
- Objetos de porte personal, como maletines o sombrillas

Finalmente, es importante tener en cuenta que cualquier detección de las clases indicadas dentro del *streaming* de vídeo será marcada como intruso. Es decir, no será necesario considerar que la presencia de cualquiera de estas clases en un lote vecino no deberá ser marcada como intruso. Dicho esto, las clases que generarán un reporte positivo para intruso al ser detectadas son:

9

Capítulo 2

Introducción específica

En este capítulo, se presentan los conceptos necesarios para poner en funcionamiento el proyecto de detección. Se inicia con la descripción de los objetos a detectar, seguido de los modelos de visión por computadora disponibles. De igual manera, se aborda los protocolos de transmisión y recepción del vídeo y se finaliza con los requerimientos computacionales necesarios para llevar a cabo la detección.

2.1. Definición de Intruso

Para cumplir con el objetivo específico del trabajo, es importante tener en cuenta la definición de intruso dada por el cliente, que incluye la detección de personas y animales de gran porte, lo que implica que la respuesta del módulo deberá activarse con cualquier animal más grande que un perro o un gato. Es importante tener en mente que el *dataset* COCO permite el reconocimiento de una serie de animales de gran porte que no hacen parte de la fauna argentina, por lo que se decidió en conjunto con el cliente que algunas etiquetas se filtrarían a fin de evitar que el modelo intente reconocer estas clases. Algunos ejemplos de etiquetas filtradas incluyen:

- Elefantes
- Jirafas
- Cebras

Por otro lado, y a fin de mejorar el reconocimiento de intrusos, se decidió incorporar algunas clases que corresponden a objetos que suelen cargar las personas y vehículos. De esta manera se hace posible una suerte de reconocimiento indirecto, dándole al modelo una "segunda oportunidad" en casos en los que por algún motivo generó un falso negativo en el reconocimiento de una persona. Es importante tener en cuenta que estos objetos propuestos ya hacían parte del *dataset* COCO, por lo que no se requirió reentrenar el modelo para lograr su detección. Dentro de estos objetos relacionados encontramos:

- Vehículos
- Objetos de porte personal, como maletines o sombrillas

Es importante tener en cuenta que el módulo marcará toda detección como una intrusión. Esto quiere decir que las detecciones en lotes vecinos no se filtrarán dado que se considera que la totalidad del *frame* es área vigilada. Dicho esto, las clases que generarán un reporte positivo para intruso al ser detectadas son:

TABLA 2.1. Clases detectadas como intruso por el módulo

Clases detectadas como intruso	
Persona	Vaca
Camión	Bus
Automóvil	Sombrilla
Motocicleta	Mochila
Bicicleta	Maletín
Perro	Maleta
Caballo	Gato

Hay que recordar que la sola detección de una de estas clases generará un reporte positivo de intruso. El modelo no está en capacidad de identificar personas específicas en el vídeo, por lo que le resultará imposible diferenciar entre personal autorizado en un área (el mismo personal de seguridad, o la persona al mando del dron), y a intrusos en la zona.

2.2. Modelo utilizado

El modelo utilizado para el desarrollo de la totalidad del trabajo es YoloV3. La principal ventaja que tiene este modelo es que es capaz de ejecutarse con una velocidad considerablemente mayor a la de modelos anteriores como RetinaNet-50 y RetinaNet-101. Este modelo funciona a través del uso de redes neuronales convolucionales y calcula tres parámetros:

- Intersection over Union (IoU), que puede ser interpretado como la precisión del modelo al rodear un objeto con una caja. Es decir, qué tan acertada es la posición de la caja propuesta con el modelo, en relación con la posición predicha del objeto anteriormente.
- Offset de la caja detectada en comparación con la caja ancla.
- Tensor con la probabilidad de ocurrencia de las 80 clases que forman parte del dataset COCO.

Es importante tener en cuenta que estos no son los parámetros finales arrojados por el modelo, sino que más adelante, la red calcula las coordenadas de la caja con la que se rodeará al objeto. Es decir, para cada objeto detectado, el modelo va a devolver (en ese orden):

- pc: probabilidad de que haya un objeto.
- bx: coordenada x del centro de la caja propuesta.
- by: coordenada y del centro de la caja propuesta.
- bh: altura de la caja propuesta.
- bw: ancho de la caja propuesta.
- c: tensor con la probabilidad de ocurrencia de las 80 clases que forman parte del dataset COCO.

Esto se logra a través del uso de Anchor Boxes: cajas de predicción predefinidas que el modelo va refinando a medida que analiza la imagen y calcula las métricas

10 Capítulo 2. Introducción específica

TABLA 2.1. Clases detectadas como intruso por el módulo

Clases detectadas como intruso	
Persona	Vaca
Camión	Bus
Automóvil	Sombrilla
Motocicleta	Mochila
Bicicleta	Maletín
Perro	Maleta
Caballo	Gato

Hay que recordar que la sola detección de una de estas clases generará un reporte positivo de intruso. El modelo no está en capacidad de identificar personas específicas en el vídeo, por lo que le resultará imposible diferenciar entre personal autorizado en un área (el mismo personal de seguridad, o la persona al mando del dron), y a intrusos en la zona.

2.2. Modelo utilizado

El modelo utilizado para el desarrollo de la totalidad del trabajo es YOLOv3. La principal ventaja que tiene este modelo es que es capaz de ejecutarse con una velocidad considerablemente mayor a la de modelos anteriores como RetinaNet-50 y RetinaNet-101.

Este modelo se basa en el uso de redes neuronales convolucionales (CNN) que, al dividir la imagen en una cuadrícula, permiten calcular la probabilidad de ocurrencia de objetos en cada celda. Con esta información, el modelo determina tres parámetros clave para su funcionamiento:

- *Intersection over Union* (IoU)[8], se trata del criterio utilizado para eliminar cajas redundantes, y asegurar que solo se mantenga una por cada objeto detectado, incluso si hay varios objetos cercanos en la imagen. De esta manera, se evita la detección de objetos duplicados y se obtiene una detección más precisa y clara de los objetos presentes en la imagen.

  Se calcula a través de la siguiente fórmula:

  $IoU = \frac{A_s}{A_u}$

  Donde A_s representa el área de superposición entre ambas cajas y A_u el área de unión.

  Para lograr esto, cada celda es responsable de calcular la probabilidad de que haya un objeto dentro de sus fronteras. se genera una serie de cajas propuestas alrededor del objeto, y a través del uso de redes neuronales convolucionales se calcula la probabilidad de ocurrencia.

  Finalmente, el modelo elimina las cajas que no cumplen con dos criterios importantes: En primer lugar, se descartan aquellas cajas que tienen una probabilidad de ocurrencia menor que la máxima del objeto de la zona. Este algoritmo es conocido como *Non-Maximum Suppression* [9] y utiliza la probabilidad de ocurrencia, también conocida como *objectness score* [10], para seleccionar solo las cajas más relevantes. En segundo lugar, se eliminan las

de cada uno de los objetos encontrados. A través de este método es posible eliminar la necesidad de una "ventana deslizante", lo que a su vez permite analizar la totalidad de la imagen en un tiempo mucho menor que los métodos basados en aquel método, y, en consecuencia, se hace posible llevar a cabo un análisis en tiempo real.

Dadas las bondades explicadas anteriormente, en conjunto con su facilidad de implementación en Python, se escogió partir del modelo YoloV3 y modificar su código fuente para lograr generar y extraer la información pertinente para completar el propósito del trabajo. Cabe aclarar que, la decisión de utilizar este modelo se vio reforzada una vez se ejecutó la primera versión en tanto que el modelo está optimizado para funcionar con la tarjeta gráfica (NVIDIA Titan X), el código se pudo ejecutar satisfactoriamente, sin dejar a un lado los requisitos del trabajo en CPU (AMD Ryzen 5).

2.3. Protocolos de vídeo y transmisión

El módulo del trabajo encargado de extraer y permitir el cálculo sobre cada uno de los frames es OpenCV, por lo que los protocolos de vídeo aceptados corresponderán a los aceptados por OpenCV. En este sentido, el código está en capacidad de detectar intrusos en:

- Vídeo guardado localmente
- Secuencia de imágenes
- URL de streaming de vídeo
- Pipeline Gstreamer
- Datos obtenidos desde Webcam

Sin embargo, es importante mencionar que, durante el desarrollo, se hizo pruebas únicamente con los métodos de vídeo guardado localmente y URL de streaming de vídeo. Los otros métodos no han sido probados, y, por lo tanto, no se podrá garantizar su correcto funcionamiento al solicitar al módulo ejecutarse utilizándolos como origen de datos.

Por otro lado, y recordando que se trata de la biblioteca que hace posible extraer la información del vídeo para hacer posible su análisis, es de vital importancia tener en mente cuáles son los formatos de vídeo admitidos por OpenCV. Dentro de ellos se encuentran:

- H264
- MPEG4
- AV1

Ahora bien, para acceder a ellos, bastará con pasar el link apropiado al método encargado. Dentro de los protocolos probados durante el desarrollo del trabajo se encuentran:

- MQTT: es un protocolo que permite la comunicación entre dispositivos desarrollado por IBM con gran énfasis en IoT. Es un protocolo muy seguro teniendo en cuenta con un componente que filtra las comunicaciones entre el servidor y el cliente, por lo que ninguno de los componentes conocerá los

cajas que tienen un *Intersection Over Union (IoU)* mayor a un valor determinado, que suele ser 0.45, aunque también se puede ajustar como parámetro en el modelo.

- *Offset*: el *offset* se refiere a la distancia medida desde la esquina superior izquierda de la celda con mayor probabilidad de contener un objeto [11]. Esta medida es una forma eficiente de representar la posición de las cajas y reduce significativamente la cantidad de cálculos necesarios, lo que aumenta la eficiencia computacional del modelo [12]. Una vez obtenido el offset, se utilizan estos valores para calcular las coordenadas de las cajas que no fueron descartadas.
- Tensor: hace referencia a un vector utilizado para representar la probabilidad [13] de que cada uno de los objetos incluidos en el *dataset* COCO [6] se encuentre en la caja . Esta representación no solo permite determinar la posición de diferentes objetos en la imagen, sino también discriminarlos por clases. La clase con mayor probabilidad es entonces reportada por el modelo.

Es importante tener en cuenta que estos no son los parámetros finales arrojados por el modelo, sino que más adelante, la red calcula las coordenadas de la caja con la que se rodeará al objeto. Es decir, para cada objeto detectado, el modelo va a devolver (en ese orden):

- pc: *objectness score*, la probabilidad de que haya un objeto.
- bx: coordenada x del centro de la caja propuesta.
- by: coordenada y del centro de la caja propuesta.
- bh: altura de la caja propuesta.
- bw: ancho de la caja propuesta.
- c: tensor con la probabilidad de ocurrencia de las 80 clases que forman parte del dataset COCO.

Esto se logra a través del uso de Anchor Boxes: cajas de predicción predefinidas que el modelo va refinando a medida que analiza la imagen y calcula las métricas de cada uno de los objetos encontrados. A través de este método es posible eliminar la necesidad de una "ventana deslizante", lo que a su vez permite analizar la totalidad de la imagen en un tiempo mucho menor que los métodos basados en aquel método, y, en consecuencia, se hace posible llevar a cabo un análisis en tiempo real.

Dadas las bondades explicadas anteriormente, en conjunto con su facilidad de implementación en Python, se escogió partir del modelo YoloV3 y modificar su código fuente para lograr generar y extraer la información pertinente para completar el propósito del trabajo. Cabe aclarar que, la decisión de utilizar este modelo se vio reforzada una vez se ejecutó la primera versión en tanto que el modelo está optimizado para funcionar con la tarjeta gráfica (NVIDIA Titan X), el código se pudo ejecutar satisfactoriamente, sin dejar a un lado los requisitos del trabajo en CPU (AMD Ryzen 5).

datos del otro componente. En cambio, el cliente solicita una suscripción a un tema que le interesa, y el agente se encarga de distribuir los mensajes enviados por el servidor. Cabe mencionar que MQTT está pensado especialmente para dispositivos IoT con recursos limitados y poco ancho de banda.

- RTMP: es un protocolo diseñado específicamente para la transmisión de audio y vídeo con gran rendimiento. Esto lo logra dividiendo la información en pequeños fragmentos, cuyo tamaño puede ser negociado de manera dinámica entre el cliente y el servidor de forma que la transmisión de vídeo se adapta mejor a las condiciones de la red de la que depende la transmisión. Sin embargo, dado que se trata de un protocolo diseñado para la transmisión de vídeo y audio con el menor delay posible, este protocolo puede no ser el más adecuado cuando las condiciones de la red son limitadas.

2.4. Software y hardware utilizados

Para su funcionamiento, este trabajo requiere la presencia de algunos paquetes de software instalados en la máquina, los cuales son llamados directamente por el usuario cuando requiera la funcionalidad (por ejemplo, OBS Studio), o bien, que ya vienen incorporados en el código y por lo tanto, son un requisito indispensable para el proyecto y deberán encontrarse instalados. Dicho esto, es posible clasificar el software utilizado en tres categorías:

1. Paquetes de Python: son paquetes importados en el código a través del comando *import*. Cabe mencionar que todos los paquetes son de código abierto. Ejemplos de estos paquetes son:

- Numpy
- Matplotlib
- OpenCV

2. Código en Python: en especial para la implementación de la metodología YOLOV3. En internet está disponible el código fuente en formato Python (py).

3. Software completo: en particular para la retransmisión del vídeo procesado, se utilizó OBS estudio. Esta decisión responde a que, dentro del alcance del trabajo, no se definió ninguna actividad que no se encuentre directamente relacionada con la implementación de la detección de intrusos. Sin embargo, dado que se trata de un trabajo en el que se debe devolver la imagen procesada, es importante asegurarse que la retransmisión de los datos funcione de la manera correcta.

Más específicamente, las diferentes librerías y paquetes de software utilizados para el desarrollo de este trabajo, y sus versiones correspondientes son:

12 Capítulo 2. Introducción específica

2.3. Protocolos de vídeo y transmisión

El módulo del trabajo encargado de extraer y permitir el cálculo sobre cada uno de los frames es OpenCV, por lo que los protocolos de vídeo aceptados corresponderán a los aceptados por OpenCV. En este sentido, el código está en capacidad de detectar intrusos en:

- Vídeo guardado localmente
- Secuencia de imágenes
- URL de streaming de vídeo
- Pipeline Gstreamer
- Datos obtenidos desde *webcam*

Sin embargo, es importante mencionar que, durante el desarrollo, se hizo pruebas únicamente con los métodos de vídeo guardado localmente y URL de streaming de vídeo. Los otros métodos no han sido probados, y, por lo tanto, no se podrá garantizar su correcto funcionamiento al solicitar al módulo ejecutarse utilizándolos como origen de datos.

Por otro lado, y recordando que se trata de la biblioteca que hace posible extraer la información del vídeo para hacer posible su análisis, es de vital importancia tener en mente cuáles son los formatos de vídeo admitidos por OpenCV. Dentro de ellos se encuentran:

- H264
- MPEG4
- AV1

Ahora bien, para acceder a ellos, bastará con pasar el link apropiado al método encargado. Dentro de los protocolos probados durante el desarrollo del trabajo se encuentran:

- MQTT: es un protocolo que permite la comunicación entre dispositivos desarrollado por IBM. Este protocolo es considerado muy seguro debido a que utiliza un intermediario llamado MQTT Broker para el intercambio de información entre el servidor y el cliente. De esta forma, ninguna de las partes tiene acceso directo a los datos del otro, visto que solo cuentan con la información para acceder al MQTT Broker. En cambio, el cliente solicita una suscripción a un tema que le interesa (en este caso la transmisión de vídeo), y el agente se encarga de distribuir los mensajes enviados por el servidor. Está pensado especialmente para dispositivos IoT con recursos limitados y conexiones con poco ancho de banda.

- RTMP: es un protocolo diseñado específicamente para la transmisión de audio y vídeo con gran rendimiento. Esto lo logra dividiendo la información en pequeños fragmentos, cuyo tamaño puede ser negociado de manera dinámica entre el cliente y el servidor de forma que la transmisión de vídeo se adapta mejor a las condiciones de la red de la que depende la transmisión. Sin embargo, dado que se trata de un protocolo diseñado para la transmisión de vídeo y audio con el menor delay posible, este protocolo puede no ser el más adecuado cuando las condiciones de la red son limitadas.

TABLA 2.2: Software requerido para el desarrollo y la ejecución del módulo

Software requerido				
Paquete	Tipo	Licencia	Función	Versión
PIP	Gestor de bibliotecas	MIT	Gestor de bibliotecas para Python	22.0.2
Numpy	Biblioteca de Python	BSD	Cálculos matriciales	1.21.5
OpenCV	Biblioteca de Python	Apache 2	Manipulación de imágenes y visión por computadora	4.5.4
Matplotlib	Biblioteca de Python	BSD	Creación de visualizaciones con Python	3.6.2
Tensorflow	Biblioteca de Python	Apache 2	Framework de machine learning	2.10.1
Pycharm	Software completo	Apache 2	Ambiente de desarrollo	2022.3.3 Community edition
OBS Studio	Software completo	GPLv2	Streaming de vídeo	28.1.2 (64 Bit)
Python	Lenguaje de programación	Python Software Foundation License Version 2	Lenguaje en el que fue programado el módulo	3.10.6
Ubuntu	Sistema operativo	Creative Commons CC-BY-SA version 3.0 UK	Sistema operativo	22.04.1 LTS (64 Bit)

2.4. Software y hardware utilizados 13

2.4. Software y hardware utilizados

Para su funcionamiento, este trabajo requiere la presencia de algunos paquetes de software instalados en la máquina, los cuales son llamados directamente por el usuario cuando requiera la funcionalidad (por ejemplo, OBS Studio), o bien, que ya vienen incorporados en el código y por lo tanto, son un requisito indispensable para el trabajo y deberán encontrarse instalados. Dicho esto, es posible clasificar el software utilizado en tres categorías:

1. Paquetes de Python: son paquetes importados en el código a través del comando *import*. Cabe mencionar que todos los paquetes son de código abierto. Ejemplos de estos paquetes son:

   - Numpy
   - Matplotlib
   - OpenCV

2. Código en Python: en especial para la implementación de la metodología YOLOV3. En internet está disponible el código fuente en formato Python (py).

3. Software completo: en particular para la retransmisión del vídeo procesado, se utilizó OBS estudio. Esta decisión responde a que, dentro del alcance del trabajo, no se definió ninguna actividad que no se encuentre directamente relacionada con la implementación de la detección de intrusos. Sin embargo, dado que se trata de un trabajo en el que se debe devolver la imagen procesada, es importante asegurarse que la retransmisión de los datos funcione de la manera correcta.

Más específicamente, las diferentes librerías y paquetes de software utilizados para el desarrollo de este trabajo, y sus versiones correspondientes son:

Es importante tener en cuenta que la totalidad de programas y librerías utilizadas, tanto para el desarrollo, como para la ejecución del trabajo son software libre, por lo que no se requerirá adquirir ningún tipo de licencia para ponerlo en funcionamiento.

De igual manera, cabe resaltar que OBS Studio es el programa de retransmisión elegido teniendo en cuenta que esta es una característica que se encuentra fuera del alcance del trabajo y por lo tanto, cuya implementación es necesaria a través de este método mientras que hacerlo a través de software integrado en el código fuente (como FFMpeg) será parte del alcance de trabajos posteriores.

El paquete de Python ya viene instalado por defecto en la instalación de Ubuntu, por lo que no se requerirá su instalación. El resto de los paquetes, sin embargo, deberán ser instalados de acuerdo con el manual de instalación que se entrega al cliente, en el que se detalla de manera minuciosa cuál es el procedimiento que se debe seguir a fin de instalar correctamente todos los paquetes.

De igual manera, se debe tener en cuenta que dadas las restricciones de la máquina en la que se desarrolló el trabajo, la totalidad del desarrollo fue llevado a cabo en máquinas virtuales a través del uso de programas de virtualización como VMWare Workstation. Dentro de los requerimientos mínimos de hardware del trabajo, se tienen los siguientes:

- Procesador: AMD Ryzen 5
- Tarjeta Gráfica: no requerida
- Memoria RAM: 6 GB.

Es importante tener en cuenta que los listados anteriormente son los requisitos mínimos del sistema, que van a garantizar que el proyecto se ejecute cumpliendo con los requerimientos del cliente. Sin embargo, se recomiendan las siguientes características del sistema para un rendimiento óptimo a la hora de ejecutar el módulo.

- Procesador: Intel Core i7
- Tarjeta Gráfica: NVidia Tesla T4 o NVidia Titan X.
- Memoria RAM: 8 GB.

El modelo YoloV3, sobre el que está basado el desarrollo del trabajo utiliza de manera extensiva la API Cuda de NVIDIA. Por este motivo, tratar de ejecutar el proyecto en tarjetas gráficas de AMD resultará en el modelo siendo llevado al procesador por Tensorflow. En consecuencia, se afirmará que el trabajo resulta incompatible con estas tarjetas gráficas. Esto fue comprobado con el uso de una tarjeta AMD Radeon Vega 8.

TABLA 2.2. Software requerido para el desarrollo y la ejecución del módulo

Software requerido				
Paquete	Tipo	Licencia	Función	Versión
PIP	Gestor de bibliotecas	MIT	Gestor de bibliotecas para Python	22.0.2
Numpy	Biblioteca de Python	BSD	Cálculos matriciales	1.21.5
OpenCV	Biblioteca de Python	Apache 2	Manipulación de imágenes y visión por computadora	4.5.4
Matplotlib	Biblioteca de Python	BSD	Creación de visualizaciones con Python	3.6.2
Tensorflow	Biblioteca de Python	Apache 2	Framework de machine learning	2.10.1
Pycharm	Software completo	Apache 2	Ambiente de desarrollo	2022.3.3 Community edition
OBS Studio	Software completo	GPLv2	Streaming de video	28.1.2 (64 Bit)
Python	Lenguaje de programación	Python Software Foundation License Version 2	Lenguaje en el que fue programado el módulo	3.10.6
Ubuntu	Sistema operativo	Creative Commons CC-BY-SA version 3.0 UK	Sistema operativo	22.04.1 LTS (64 Bit)

Capítulo 3

Diseño e implementación

3.1. Consideraciones generales

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno lstlisting con el siguiente código:

```
\begin{lstlisting}[caption= "un epígrafe descriptivo"]
las líneas de código irían aquí...
\end{lstlisting}
```

A modo de ejemplo:

```
#define MAX_SENSOR_NUMBER 3
#define MAX_ALARM_NUMBER  6
#define MAX_ACTUATOR_NUMBER 6

uint32_t sensorValue[MAX_SENSOR_NUMBER];
FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
state_t alarmState[MAX_ALARM_NUMBER];           //ON or OFF
state_t actuatorState[MAX_ACTUATOR_NUMBER];     //ON or OFF

void vControl() {

  initGlobalVariables();

  period = 500 ms;

  while(1) {

    ticks = xTaskGetTickCount();

    updateSensors();

    updateAlarms();

    controlActuators();

    vTaskDelayUntil(&ticks, period);
  }
}
```

CÓDIGO 3.1. Pseudocódigo del lazo principal de control.

Es importante tener en cuenta que la totalidad de programas y librerías utilizadas, tanto para el desarrollo, como para la ejecución del trabajo son software libre, por lo que no se requerirá adquirir ningún tipo de licencia para ponerlo en funcionamiento.

De igual manera, cabe resaltar que OBS Studio es el programa de retransmisión elegido teniendo en cuenta que esta es una característica que se encuentra fuera del alcance del trabajo y por lo tanto, cuya implementación es necesaria a través de este método mientras que hacerlo a través de software integrado en el código fuente (como FFMpeg) será parte del alcance de trabajos posteriores.

El paquete de Python ya viene instalado por defecto en la instalación de Ubuntu, por lo que no se requerirá su instalación. El resto de los paquetes, sin embargo, deberán ser instalados de acuerdo con el manual de instalación que se entrega al cliente, en el que se detalla de manera minuciosa cuál es el procedimiento que se debe seguir a fin de instalar correctamente todos los paquetes.

De igual manera, se debe tener en cuenta que dadas las restricciones de la máquina en la que se desarrolló el trabajo, la totalidad del desarrollo fue llevado a cabo en máquinas virtuales a través del uso de programas de virtualización como VMware Workstation.[14] Dentro de los requerimientos mínimos de hardware del trabajo, se tienen los siguientes:

- Procesador: AMD Ryzen 5
- Tarjeta Gráfica: no requerida
- Memoria RAM: 6 GB.

Es importante tener en cuenta que los listados anteriormente son los requisitos mínimos del sistema, que van a garantizar que el trabajo se ejecute cumpliendo con los requerimientos del cliente. Sin embargo, se recomiendan las siguientes características del sistema para un rendimiento óptimo a la hora de ejecutar el módulo.

- Procesador: Intel Core i7
- Tarjeta Gráfica: NVidia Tesla T4 o NVidia Titan X.
- Memoria RAM: 8 GB.

El modelo YoloV3, sobre el que está basado el desarrollo del trabajo utiliza de manera extensiva la API Cuda de NVIDIA. Por este motivo, tratar de ejecutar el módulo en tarjetas gráficas de AMD resultará en el modelo siendo llevado al procesador por Tensorflow. En consecuencia, se afirmará que el trabajo resulta incompatible con estas tarjetas gráficas. Esto fue comprobado con el uso de una tarjeta AMD Radeon Vega 8.

3.2. Arquitectura del proyecto

3.3. Esquema del módulo

3.4. Esquema de modificaciones

3.5. Ajustes para cumplir con el rendimiento

3.6. Entrega de resultados al resto del software

Capítulo 4

Ensayos y resultados

4.1. Descripción del proceso de pruebas

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

4.2. Pruebas en Raspberry Pi

4.3. Caso de Uso

Capítulo 3

Diseño e implementación

3.1. Consideraciones generales

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno lstlisting con el siguiente código:

```
\begin{lstlisting}[caption= "un epigrafe descriptivo"]
las lineas de código irían aquí...
\end{lstlisting}
```

A modo de ejemplo:

```
#define MAX_SENSOR_NUMBER 3
#define MAX_ALARM_NUMBER  6
#define MAX_ACTUATOR_NUMBER 6

uint32_t sensorValue[MAX_SENSOR_NUMBER];
FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
state_t alarmState[MAX_ALARM_NUMBER];           //ON or OFF
state_t actuatorState[MAX_ACTUATOR_NUMBER];     //ON or OFF

void vControl() {

  initGlobalVariables();

  period = 500 ms;

  while(1) {

    ticks = xTaskGetTickCount();

    updateSensors();

    updateAlarms();

    controlActuators();

    vTaskDelayUntil(&ticks, period);
  }
}
```

CÓDIGO 3.1. Pseudocódigo del lazo principal de control.

3.2. Arquitectura del proyecto

3.3. Esquema del módulo

3.4. Esquema de modificaciones

3.5. Ajustes para cumplir con el rendimiento

3.6. Entrega de resultados al resto del software

Capítulo 5

Conclusiones

5.1. Resultados Obtenidos

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Tiempos de ejecución

Acá se indica cómo se podría continuar el trabajo más adelante.

5.3. Monitoreo de resultados

Capítulo 4

Ensayos y resultados

4.1. Descripción del proceso de pruebas

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

4.2. Pruebas en Raspberry Pi

4.3. Caso de Uso

Bibliografía

[1] Drone AI Services. *Drones de seguridad privada ¿Cómo funcionan?* Último acceso el 28 de marzo de 2023. 2020. URL: https://www.droneservices.com.ar/industria-4-0/drones-de-seguridad-privada/.

[2] Jordan Henrio y Tomoharu Nakashima. «Anomaly Detection in Videos Recorded by Drones in a Surveillance Context». En: oct. de 2018, págs. 2503-2508. DOI: 10.1109/SMC.2018.00429.

[3] Joseph Redmon y col. «You Only Look Once: Unified, Real-Time Object Detection». En: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[4] Joseph Redmon y Ali Farhadi. «YOLOv3: An Incremental Improvement». En: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[5] Heng Zhao y col. «Mixed YOLOv3-LITE: A Lightweight Real-Time Object Detection Method». En: *Sensors (Basel)* 20.7 (2020), pág. 1861. ISSN: 1424-8220. DOI: 10.3390/s20071861. URL: https://doi.org/10.3390/s20071861.

[6] Common Objects in Context. *COCO*. Último acceso el 28 de marzo de 2023. URL: https://cocodataset.org/#home.

Capítulo 5

Conclusiones

5.1. Resultados Obtenidos

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Tiempos de ejecución

Acá se indica cómo se podría continuar el trabajo más adelante.

5.3. Monitoreo de resultados