UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Sistema empotrado de análisis flexible de imágenes mediante redes neuronales pre-entrenadas.

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Gonzalez Romero, Sebastián

Tutor/a: Simó Ten, José Enrique

CURSO ACADÉMICO: 2021/2022

SISTEMA EMPOTRADO DE ANÁLISIS FLEXIBLE DE IMÁGENES MEDIANTE REDES NEURONALES PRE-ENTRENADAS

Autor
SEBASTIÁN GONZÁLEZ ROMERO
(sgonrom@posgrado.upv.es)

Director: JOSÉ ENRIQUE SIMÓ TEN
(jsimo@disca.upv.es)

TRABAJO FIN DE MÁSTER
MÁSTER DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Valencia, septiembre 2022

Dedicado a todos los que han hecho posible la finalización de este máster.

AGRADECIMIENTOS

A José Enrique Simó Ten, por su inconmensurable ayuda a la hora de llevar a cabo este proyecto. También a mi familia, pareja y amigos, que me han servido de apoyo desde siempre.

TABLA DE CONTENIDO

LISTA DE FIGURAS

RESUMEN EXTENDIDO

El proyecto plantea el desarrollo sobre plataforma "Raspberry Pi" (aunque puede ser exportado a otras plataformas como las tarjetas Jetson de Nvidia) de un sistema empotrado de identificación y localización de objetos en una imagen.

El sistema funcionará ejecutando redes neuronales pre-entrenadas del tipo YOLO y desempeñará la función de nodo sensorial de un sistema distribuido en el que cada nodo comunique al "grupo" o al "integrador" el resultado de la detección para determinar el estado global del sistema.

El desarrollo del proyecto incluye la integración de la red YOLO, el desarrollo de un sistema de comunicaciones que permita tanto la comunicación de los resultados de detección como la configuración del sistema empotrado de detección, y una aplicación cliente sobre PC que permita el envío y recepción de mensajes desde múltiples nodos.

La configuración en tiempo de ejecución del sistema empotrado mediante mensajes permitirá la selección de modos de ejecución (selección de objeto, conteo de objeto, detección múltiple, localización de objeto, etc.) así como la selección de diferentes redes neuronales pre-entrenadas.

1. INTRODUCCION

El presente trabajo nace de la necesidad de combinar dos sistemas previamente existentes, los cuales supondrán la base de todo el proyecto y que procederemos a explicar brevemente.

El primero de los sistemas viene conformado por la red neuronal pre-entrenada para análisis de imágenes YOLOv5 (del acrónimo "*You Only Look Once*" en su versión 5, ver F16). Se trata de un algoritmo de detección de objetos que divide imágenes en un sistema de cuadrícula, dónde cada celda de la cuadrícula es responsable de detectar objetos dentro de sí misma. YOLOv5 destaca entre otras cosas por la efectividad y velocidad de sus algoritmos de predicción, por la facilidad y variedad de opciones de personalización y por ser *open source*.

El segundo de los sistemas es un monitor de comunicaciones que emplea sockets TCP/IP para la comunicación. Se trata de un sistema creado por el propio director de este TFM (José Enrique Simó Ten), el cual emplea un protocolo para la estructuración de los mensajes del cual se hablará más adelante (ver Anexo 4. Formato de los mensajes intercambiados entre el PC y el equipo remoto.).

Así, el trabajo nace bajo la premisa de combinar ambos sistemas, de tal manera que, haciendo uso del sistema de comunicaciones podamos establecer conexión de manera remota con la red neuronal YOLOv5 para poder hacer uso de ella y sus características.

2. JUSTIFICACION

El desarrollo de este nuevo sistema (que como ya hemos dicho previamente nace de la combinación de dos previamente existentes) busca satisfacer varios requisitos como lo son:

- La reutilización de código previamente existente.
- La consolidación de un sistema sobre el cual en un futuro poder desarrollar nuevas funcionalidades con las que sacar provecho a la red YOLOv5 y las cuales podrían ser beneficiosas para diferentes sectores, como podrían serlo, por ejemplo:
    - El control de aforos, para fines sanitarios o de seguridad.
    - Comprobación de posibles espacios vacíos para apagar las luces, con fines de ahorro energético.
    - Posibilidad de transformar las características de una imagen a sonido, con fines de ayuda a discapacitados visuales.
    - Detección de elementos en diversos entornos.
- Extraer las máximas capacidades posibles del desarrollo *open source* de YOLOv5.

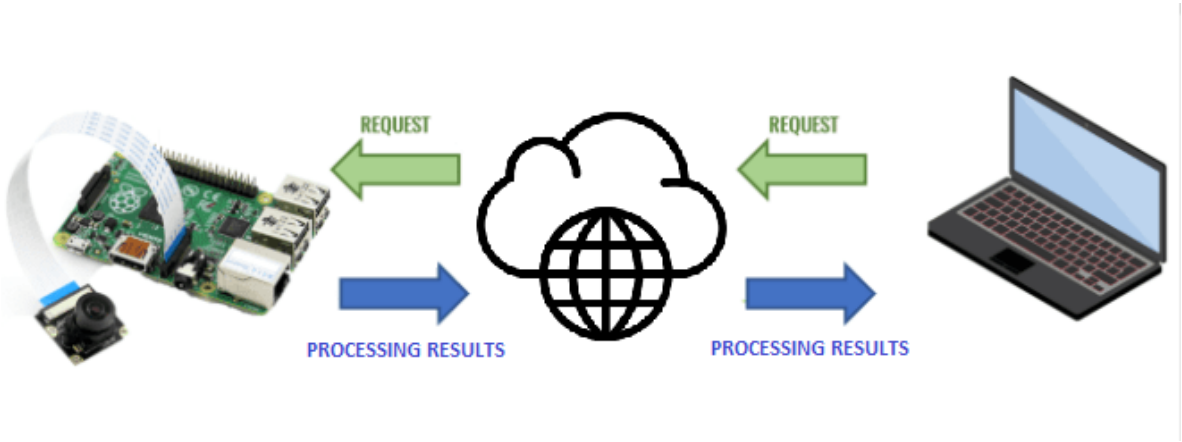
A todo lo anterior, además, podemos sumar posibles mejoras futuras que se quieran hacer sobre el proyecto, ampliando funcionalidades o incluso mejorando las características de las existentes. En este aspecto ahondaremos más adelante en el epígrafe “RECOMENDACIONES Y TRABAJOS FUTUROS”.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

El objetivo general de este proyecto será la de disponer de un sistema funcional el cual nos permita comunicarnos a través de una interfaz gráfica desde un dispositivo “A” con un dispositivo (o conjunto de dispositivos) “B” remoto el cual se encuentre corriendo la red YOLOv5 y dispone de una cámara.

Figura 1. Diagrama de la solución.



De esta manera, se espera que desde el dispositivo “A” podamos configurar y utilizar las distintas capacidades que ofrece YOLOv5 de manera remota, obteniendo una respuesta de ésta en forma de información útil como imágenes.

Además, el desarrollo de este proyecto nos podrá ayudar a asentar y ampliar conocimientos acerca del ámbito de las telecomunicaciones. Trabajando con elementos ya estudiados durante el presente máster, así como enfrentándonos a nuevos problemas que supondrán la asimilación de nuevos conocimientos.

3.2 OBJETIVOS ESPECÍFICOS

Entre los objetivos específicos que hemos planteado para el proyecto se encuentran los siguientes:

- Conseguir combinar ambos sistemas (el monitor para las comunicaciones y la YOLOv5).

- *Streaming* de vídeo desde el dispositivo “B” al dispositivo “A”.

- Posibilidad de obtención de imágenes en el instante en que se solicite.

- Obtención de información en texto sobre la situación actual del entorno en el que se encuentre el dispositivo “B”.

- Posibilidad de configurar el dispositivo remoto “B” desde el dispositivo “A”.

- Obtención de información a través de la locución (en forma de sonido por un altavoz) de las características extraídas sobre la situación actual del entorno en el que se encuentre el dispositivo “B”.

- Posibilidad de enviar imágenes desde el dispositivo “A” al “B” para que YOLOv5 analice la imagen.

- Posibilidad de aplicar técnicas de visión artificial para la obtención de características del espacio en el que se encuentra el dispositivo “B”.

4. ESTADO DEL ARTE

En este apartado procederemos a hacer un repaso sobre el estado del arte de los diferentes elementos que componen el proyecto.

4.1 SOFTWARE

Los distintos componentes software que componen la solución.

4.1.1 YOLOv5. Se trata de un detector de objetos a través de visión artificial, el cual se encuentra actualmente en su quinta versión y caracterizado por su rapidez y precisión a la hora de hacer predicciones.

Con un buen rendimiento a la hora de procesar imágenes, incluyendo vídeo en tiempo real, se trata de una red neuronal que está ganando enorme popularidad en el desarrollo de múltiples proyectos, sobre todo por su carácter *open-source*.

El algoritmo basado en aprendizaje profundo de YOLOv5 nos permite implementar en el presente proyecto toda la parte de tratamiento de imágenes y detección de objetos, así como la extracción de las características presentes en éstos.

4.1.1.1 Estructura de YOLOv5. La red YOLOv5 es un detector de objetos, que de forma similar a la mayoría de los detectores de objetos consta de tres componentes bien diferenciados. Estos son (ver Y20, S20 y K21):

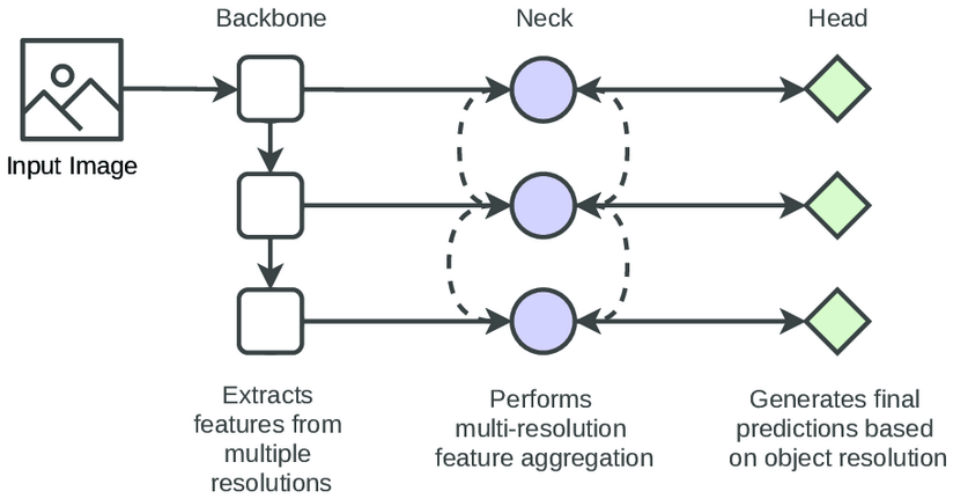
- *Backbone*: se trata de redes neuronales convolucionales que son empleadas principalmente para la extracción de las características más importantes de la imagen (o las imágenes) dada. Más concretamente, YOLOv5 emplea redes CSP (*Cross Stage Partial Networks*) como *backbone*. Esta parte de YOLOv5 se encarga de toda la fase de preentrenamiento de la red. Este componente ha sido mejorado en cuanto a rendimiento (tiempo de procesamiento) en las sucesivas versiones de YOLO.

- *Neck*: define la forma en que la información se propaga a través de la red neuronal. Este componente es utilizado para generar un “mapa (o pirámide) de características”. El modelo utilizado por YOLOv5 para la generación de pirámides de características es conocido como PANet (*Path Aggregation Network*). Las pirámides de características ayudan entre otras cosas al modelo a generalizar bien en el escalado de los objetos, es decir, ayudan a

identificar un mismo objeto con diferentes tamaños. Se encuentra en la parte intermedia de la red, entre el *backbone* y el *head*.

- *Head*: empleado en la fase final de detección de YOLOv5, para la detección de categorías (tipos de objeto) y generación de cuadros delimitadores sobre los objetos. En resumen, este componente es el que genera la salida final de la red (con las diferentes características de identificación que ofrece YOLOv5).

Figura 2. Diagrama sobre los diferentes componentes que componen YOLOv5 y sus interacciones. Imagen extraída de la página web researchgate.net.



Es el conjunto de todos estos elementos es el que compone el funcionamiento de la red YOLOv5 (ver Figura 2).

4.1.2 Monitor de comunicaciones. Creado por el director de éste TFM, José Simó Ten, se trata de un conjunto de ficheros que contienen código desarrollado tanto para lenguaje Java como para lenguaje C/C++.

Este software permite establecer comunicaciones a través de sockets TCP/IP, tolerando múltiples conexiones simultáneas y tanto el envío de mensajes individuales como de múltiples de manera sostenida (*streaming*).

En cuanto a sus mensajes poseen un protocolo que queda explicado en el Anexo 4. Formato de los mensajes intercambiados entre el PC y el equipo remoto.. Cabe destacar también la robustez del monitor y la tolerancia a fallos de este.

4.2 HARDWARE

Los elementos hardware que componen la solución. Hablaremos acerca de las tarjetas de desarrollo en general ya que la solución podría ser exportable a diversos modelos. Además, incidiremos en especial en la tarjeta de desarrollo Raspberry Pi 4 por ser la tarjeta en la que hemos desarrollado concretamente este proyecto.

4.2.1 Tarjetas de desarrollo. Las tarjetas de desarrollo son placas o circuitos, por lo general de tamaño reducido, dotadas de un microprocesador. Las tarjetas nos permiten ejecutar diversos programas suministrados, incluyendo sistemas operativos.

Figura 3. Un ejemplo de tarjeta de desarrollo es la Beagle Bone Black empleada en las prácticas del máster.

Actualmente las tarjetas de desarrollo han obtenido gran popularidad y en el mercado podemos encontrar multitud de opciones a la hora de adquirir una, con gran diversidad de capacidades, características y precios.

Estas tarjetas son empleadas en multitud de escenarios. Entre sus aplicaciones más destacadas podemos encontrar:

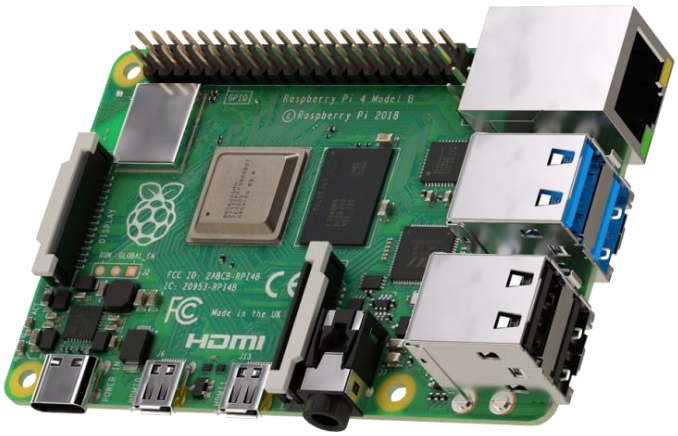
- Diseño electrónico.
- Desarrollo de dispositivos *IoT* y *wereables*.
- Elaboración de proyectos *DIY*.

- Sistemas relacionados con la *IA* y las redes neuronales.

4.2.2 Raspberry Pi 4. La tarjeta de desarrollo Raspberry Pi 4 es básicamente una mini-computadora. Se trata de una tarjeta de desarrollo capaz de correr distintos sistemas operativos, ampliamente utilizada para desarrollar sistemas de bajas capacidades.

En un inicio nacieron como tarjetas concebidas para fomentar el aprendizaje en conocimientos de programación, con su primer modelo siendo lanzado en febrero de 2012. El modelo más reciente de Raspberry Pi es el lanzado a partir de junio de 2019, el conocido como "Raspberry Pi 4 - Modelo B", del cual en los años posteriores han ido creando variantes como "Raspberry Pi 400".

Figura 4. Raspberry Pi 4 es la tarjeta escogida para implementar nuestra solución.



El modelo 4 ofrece una mejora considerable con respecto a los modelos anteriores, tanto en conexiones como en potencia o memoria. Más adelante hablaremos algo más en profundidad de las características y prestaciones que ofrece este modelo concreto.

Finalmente, es reseñable hablar sobre el incremento de precio que ha experimentado Raspberry Pi 4 desde su lanzamiento, siendo hoy en día más difícil y caro conseguir este modelo que cuando fue sacada al mercado, debido al desabastecimiento fruto de una crisis de relacionada a la fabricación de procesadores.

5. COMPONENTES Y HERRAMIENTAS EMPLEADAS EN EL DESARROLLO DEL PROYECTO

En este apartado hablaremos acerca de las diferentes herramientas y elementos que han intervenido en el desarrollo del TFM, tanto software como hardware, haciendo hincapié en sus especificaciones técnicas.

5.1 HARDWARE

5.1.1 Raspberry Pi 4 - Model B. Se trata de la tarjeta de desarrollo sobre la que hemos desplegado parte de la solución elaborada. Modelo similar al que podemos ver en Figura 4.

Se encarga de ejecutar el monitor de comunicaciones y la red YOLOv5. Entre sus especificaciones encontramos las siguientes:

- Procesador: Broadcom BCM2711 SoC de 64 bits A72 (ARM v8).
- Memoria (según el modelo concreto): SDRAM 2GB LPDDR4, SDRAM 4GB LPDDR4 o SDRAM 8GB LPDDR4.
- Bluetooth: Bluetooth 5.0.
- WiFi: IEEE 802.11ac de 2,4 GHz/5,0 GHz.
- Ethernet: Gigabit Ethernet.
- USB: 2 puertos USB 2.0, 2 puertos USB 3.0.
- Conexión: Conector GPIO de 40 contactos.
- HDMI: 2 puertos micro HDMI (admite hasta 4Kp60).
- Vídeo: Puerto de visualización MIPI DSI de 2 carriles, puerto de cámara MIPI CSI de 2 carriles.
- Audio: Salida estéreo de 4 polos y puerto de vídeo compuesto.

- Multimedia: H.265 (decodificación 4Kp60), H.264 (decodificación 1080p60, codificación 1080p30). Gráficos OpenGL ES 3.0.

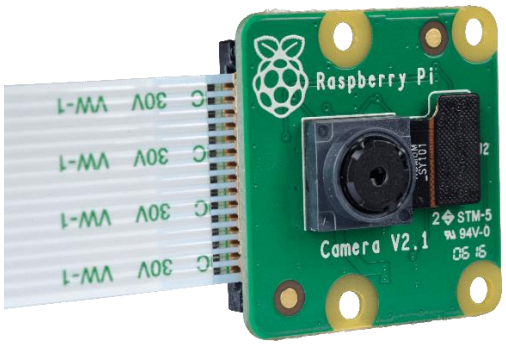
- Almacenamiento: Ranura de tarjeta microSD para cargar sistema operativo y almacenamiento de datos.

- Potencia de entrada: 5v DC a través de conector USB-C (mín. 3A), 5v DC a través de conector GPIO. Compatible con POE (requiere un HAT para PoE).

- Temperatura de funcionamiento: De 0 a 50 °C.

5.1.2 Cámara - RPI 8MP Camera Board. Se trata de la cámara que conecta con la Raspberry Pi a través del puerto de cámara MIPI CSI de la misma. Utilizada para la captura de imágenes que serán tratadas por la YOLOv5.

Figura 5. Cámara empleada como parte de la solución del sistema remoto.



Entre sus especificaciones encontramos:

- Chip: Sony IMX219.

- Resolución de imagen: 3280x2464 (8MP).

- Resolución de vídeo: 1080p30, 720p60, 640x480p90.

- Tamaño: 25mm x 23mm x 9mm.

- Temperatura Min/Max de Funcionamiento: -20°/+60° C.

- Filtro UV.

5.1.3 Ordenador Portátil - Lenovo G50-80. Es mi ordenador personal. En él he desarrollado íntegramente el proyecto del TFM.

Este es el equipo que deberemos tener como referencia a la hora de establecer las características mínimas necesarias para que funcione correctamente el proyecto (pues es en este equipo en el que hemos comprobado que el proyecto funciona correctamente y de manera fluida). En cualquier equipo con características inferiores no se garantiza un funcionamiento correcto o fluido (no testado) del proyecto.

Entre sus especificaciones técnicas podemos encontrar:

- Sistema Operativo: Windows 10 Home (versión 21H1).
- RAM: 8GB (2 módulos de 4GB) DDR3L 1600 MHz Single Channel.
- Procesador: Intel Core i7-5500U x64 2,4GHz.
- Disco: Kingston SA400S37240G SSD 240GB SATA3.
- Resolución: 1366 × 768 píxeles.
- Webcam: Sí.

5.2 SOFTWARE

5.2.1 Eclipse. Es una herramienta de desarrollo utilizada para programación informática. Ofrece un entorno de trabajo ampliable gracias a los diferentes *plugins* disponibles en este, lo que dota a Eclipse de una enorme versatilidad a la hora de facilitar la labor de la programación.

Figura 6. Logo de la herramienta de desarrollo Eclipse.

Además, Eclipse se consagra como uno de los entornos para el desarrollo en lenguaje Java más utilizados. No obstante, también es ampliamente utilizado para programación en lenguaje C/C++.

En este proyecto hemos utilizado la herramienta para programar tanto en C/C++ como en Java toda la parte del proyecto relacionada con las comunicaciones.

6. METODOLOGIA EXPERIMENTAL

La metodología empleada para la realización de este proyecto es la conocida como método de desarrollo iterativo e incremental, en el cual comenzamos diseñando, elaborando y testeando una versión más simple y pequeña de lo que se tiene como objetivo final, para posteriormente y de manera progresiva ir incrementando esta versión más simple a otras con mayor complejidad para finalmente llegar a la versión final que se tenía en mente. Esto nos permite tener siempre una versión funcional que iremos mejorando en características y capacidades.

De esta manera, hemos partido de un código base funcional, para poco a poco ir introduciendo nuevas características al proyecto (incrementando el código existente), una a una, las cuales hemos ido poniendo a prueba antes de pasar a la inclusión de la siguiente.

Con el objetivo de explicar de la forma más clara y estructurada posible la elaboración del presente proyecto, dividiremos todo el proceso de desarrollo en varias subfases, tratando de seguir una especie de cronología paralela a la creación del TFM, en la cual, según avancemos de fase, se corresponderá con que serán expuestas a continuación.

6.1 PUESTA EN MARCHA

En esta primera subfase hablaremos acerca de los inicios de este trabajo, explicando los diferentes aspectos que en su día influyeron a la hora de iniciar este proyecto.

6.1.1 Instalación de los diferentes programas. Los programas que tuvimos que instalar en el PC para comenzar con el desarrollo fueron los siguientes:

- Eclipse: que emplearíamos para crear código para el monitor de comunicaciones en lenguaje Java (concretamente una versión compatible con el desarrollo en Java). A continuación, debimos instalar el plugin de WindowBuilder (tutorial aquí), necesario para crear la interfaz gráfica que será mostrada en la aplicación. Posteriormente fue necesario incluir dos librerías externas que no estaban incluidas con la instalación base que nos ofrece Eclipse, la librería Gson (para la lectura y tratamiento de ficheros *json*) y Javax.servlet (necesario para poner el funcionamiento el código proporcionado, está relacionada con el establecimiento de comunicaciones).

- Java JDK: necesario para poder ejecutar la aplicación, podemos obtener java desde su página oficial.

- Python 3: necesario para poder ejecutar un script capaz de transformar texto a una elocución sonora. Nuevamente podemos descargar su instalador a través de su página oficial.

- Oracle MV VirtualBox: que emplearíamos para instalar una máquina virtual en la que desarrollar el código que correría en la máquina remota (en nuestro caso Raspberry Pi 4), es decir, el código del monitor de comunicaciones de la parte remota (en este caso en lenguaje c/c++) y el código de las funcionalidades que nos ofrecerá la red YOLOv5. El sistema operativo escogido para correr en VirtualBox fue Kubuntu, en su versión 20.04 de 64 bits. Entre las diversas configuraciones que llevamos a cabo para tener listo el entorno de desarrollo en VirtualBox destacan las siguientes:

    - Actualización de los paquetes del sistema operativo a través del comando “apt-get upgrade”.

    - Instalación de los paquetes necesarios para programar en C/C++ haciendo uso del comando “apt install build-essential”.

    - Instalación de Eclipse para sistema operativo Linux, en su versión para desarrollo en C/C++.

    - Instalación de Python en su versión 3.9, Pip3 y múltiples dependencias necesarias para el funcionamiento de YOLOv5 (las dependencias pueden ser consultadas en el GitHub oficial de YOLOv5).

    - Descarga del código de la red YOLOv5 (también disponible en el GitHub oficial de YOLOv5).

    - Configuración de la máquina virtual, para poder hacer uso de la webcam integrada del PC para llevar a cabo las diferentes pruebas y para poder conectar a través de la red con los programas que correrían en ella.

6.1.2 Preparación de la Raspberry Pi 4. De cara a en el futuro poder exportar el código desarrollado en la máquina virtual Kubuntu a la RPi para su correcto despliegue, deberemos instalar una serie de librerías y herramientas. Serán las siguientes:

- Preparación inicial: una vez instalado el sistema operativo que correrá en nuestra RPi, deberemos llevar a cabo una serie de configuraciones, entre ellas:

    - Actualizar los paquetes del SO, a través de los comandos “sudo apt-get update” y “sudo apt-get upgrade”, tras lo cual deberemos reiniciar el SO ayudándonos con el comando “sudo reboot now”.

    - Establecer en la configuración de Raspberry Pi (accesible mediante el comando “sudo raspi-config”) la cámara en “Legacy Mode” (en el apartado de interfaces) y desactivar el modo de reposo de la Raspberry Pi (para que cuando esté corriendo no se bloquee cuando no detecte uso) accediendo en el menú de configuración a “Display Options” y desactivando el “Screen Blanking”.

- Instalación de las librerías necesarias para C/C++:

    - Con el comando sudo “apt-get install build-essential” podremos instalar el compilador de C/C++ así como una serie de librerías de desarrollo.

- Instalación de OpenCV4 para Raspberry Pi:

    - Con este objetivo hemos seguido este tutorial.

- Instalación de YOLOv5 y todas sus dependencias:

    - Nuevamente, hemos recurrido a un tutorial ya existente, concretamente a la página oficial de YOLOv5 para llevar a cabo la correcta instalación. La guía paso a paso se encuentra en este enlace (concretamente en la parte de la documentación).

- Instalación de Python 3 y Pip 3:

    - Mediante el comando “sudo apt-get install python3” podremos instalar Python3.

    - Mediante el comando “sudo apt-get install python3-pip” podremos instalar Pip3.

6.2 DESARROLLO DEL CÓDIGO

Esta segunda subfase engrosa toda la parte relativa a la escritura del código necesario para la consecución de los diversos objetivos planteados previamente.

Así comenzaremos hablando sobre la solución software creada desde un punto de vista más general para posteriormente hablar acerca del papel que cumple cada uno de los componentes software que la conforman. Finalmente, en este subapartado veremos el enfoque iterativo e incremental llevado a cabo durante el desarrollo.

La solución software generada durante este proyecto pretende lograr la conexión entre un equipo personal y un equipo remoto con el objetivo de sacar el máximo partido posible a la red neuronal YOLOv5. Esto es logrado a través de la utilización de monitores de comunicación, que a su vez conectan con otros elementos como YOLOv5 o una interfaz gráfica.

Como ya hemos mencionado, y desde el punto de vista del desarrollo de software, la solución consta de dos partes bien diferenciadas, el software creado para nuestra computadora personal y el software desarrollado para el equipo remoto (en este proyecto una Raspberry Pi 4).

6.2.1 Desarrollo software desde el punto de vista de nuestro computador personal. También podemos considerarlo como la parte que cumple el rol de “cliente” en la solución.

Aquí podemos diferenciar dos componentes software, el desarrollo llevado a cabo en Java (monitor) y el desarrollo llevado a cabo en Python (script de locución).

- Desarrollo Java. Se trata de un proyecto desarrollado en lenguaje Java. Este proyecto abarca toda la parte del monitor de comunicaciones de la parte del “cliente”. Además, cuenta con una interfaz gráfica que nos facilita la utilización del monitor y que nos muestra los resultados obtenidos del equipo remoto de forma gráfica.

- Desarrollo Python. Consta de un solo fichero. Este fichero contiene un pequeño programa capaz de transformar texto en lenguaje natural a una locución sonora del mismo (ver P22). Es lanzado por la aplicación Java en el manejador que implementa el comando “*GET_VOICE*” (del que hablaremos más adelante).

6.2.2 Desarrollo software desde el punto de vista del equipo remoto (RPi). También podemos considerarlo como la parte que cumple el rol de “servidor” en la solución.

Aquí podemos diferenciar dos componentes software, el desarrollo llevado a cabo en C/C++ (monitor, ver E20, B12 y K22) y el llevado a cabo en Python (YOLOv5).

- Desarrollo C/C++. Se trata de proyectos creados en lenguaje C/C++. Estos proyectos conforman toda la parte relacionada con el monitor de comunicaciones desde el punto de vista del "servidor", es decir, el monitor de comunicaciones del equipo remoto. Este monitor, además, se mantiene en comunicación con el proyecto de YOLOv5 a través de unas tuberías *UNIX* a través de las cuales intercambian mensajes de texto.

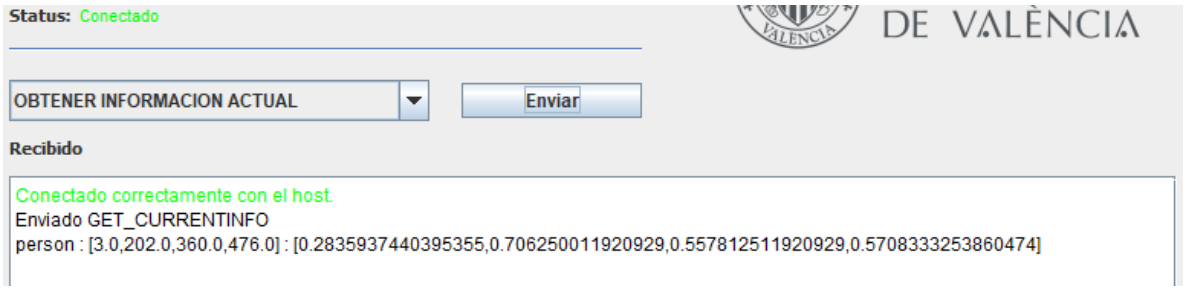
- Desarrollo Python. Se trata del proyecto de carácter *open-source* de la red neuronal YOLOv5 para identificación de imágenes (ver F16). El proyecto en su mayoría se ha mantenido inmutable con respecto al original, pero más adelante será tratado en mayor profundidad. Este *software* es el encargado de la extracción de las características de una imagen a través de la aplicación de técnicas de detección.

6.2.3 Desarrollo del proyecto desde el punto de vista del desarrollo iterativo e incremental. Plantearemos la explicación de este subapartado desde el enfoque de la metodología que hemos empleado para su elaboración, la metodología de desarrollo iterativa e incremental. Hablaremos así de como poco a poco hemos ido incluyendo nuevas funcionalidades, probándolas y finalmente pasando al desarrollo de la siguiente.

Para mejorar la comprensión de este punto se recomienda apoyarse en las figuras mostradas en el Anexo 5. Diagramas de flujo de los comandos creados.

6.2.3.1 Obtener información actual. Implementado a través del comando "COMMAND_GET_CURRENTINFO" (nombre que recibe a nivel de código fuente, ver Figura 42).

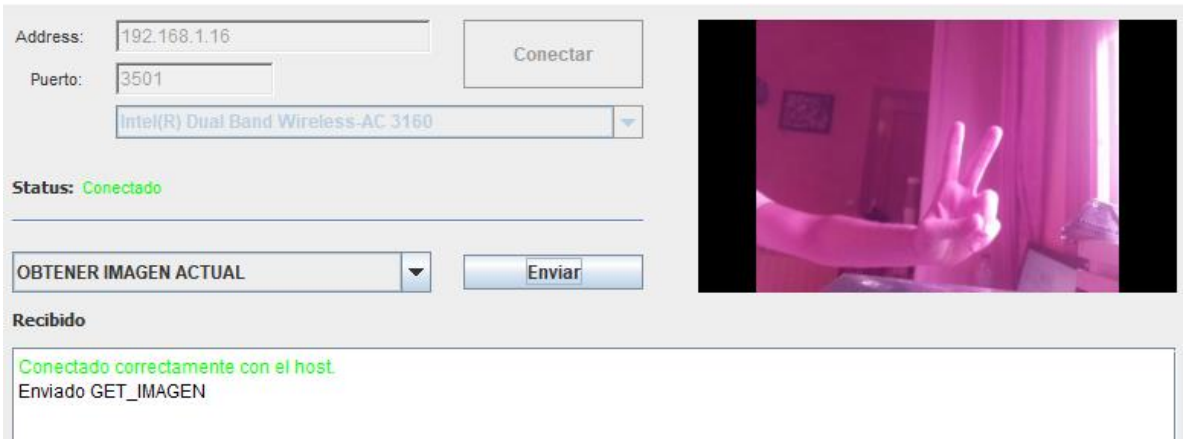
Figura 7.Ejemplo de funcionamiento de la característica "Obtener Información Actual".



Este comando es enviado desde nuestro PC al monitor de comunicaciones del equipo remoto, a continuación, el monitor le comunica el comando recibido a la red YOLOv5, tomando una imagen del instante actual (concretamente en el archivo *detect.py*), imagen la cual, a continuación, es procesada, obteniendo varias características como lo son: los tipos de objetos identificados en la imagen (móviles, personas, pantallas, …), las coordenadas relativas "x" e "y" con relación a los píxeles de la imagen, en las que se encuentra enmarcado dicho objeto y de forma similar también las coordenadas absolutas (con respecto al valor 1). Todas estas características son devueltas por el monitor de comunicaciones en formato "*json*" a nuestro PC, el cual nos las muestra finalmente por pantalla.

6.2.3.2 Obtener imagen actual. Implementado a través del comando "COMMAND_GET_IMAGEN" (ver Figura 43). Este comando nos permite la obtención de una imagen en el instante actual.

Figura 8. Ejemplo de funcionamiento de la característica "Obtener Imagen Actual".



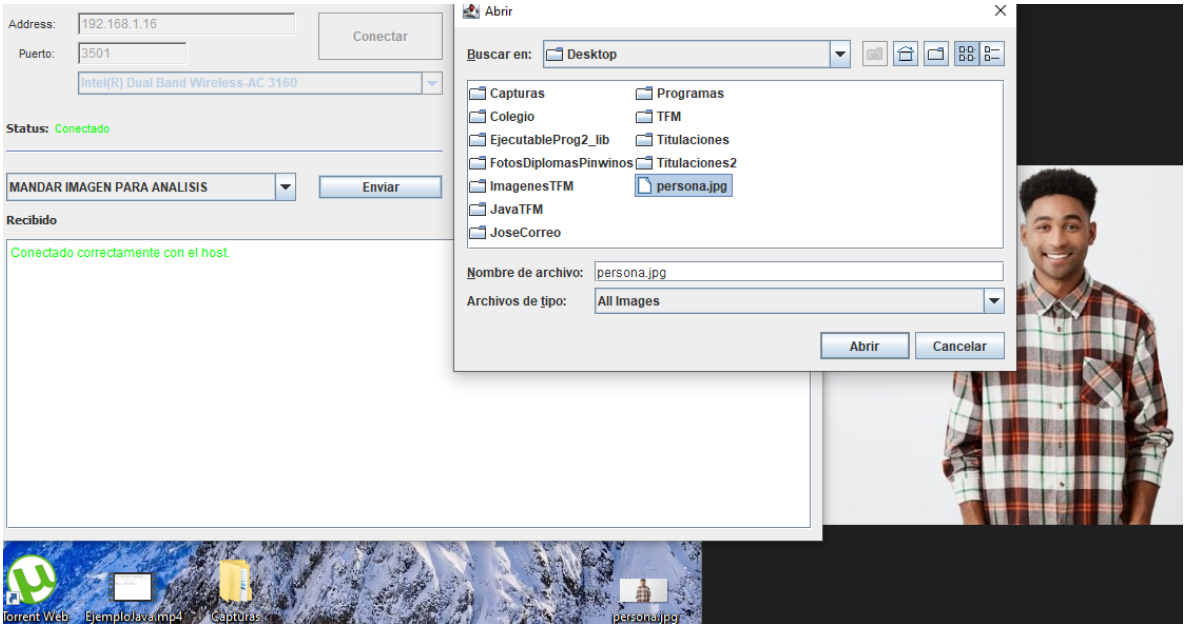
El comando es enviado desde nuestro PC al monitor del equipo remoto, el cual nuevamente se comunica con el archivo "*detect.py*" para que tome una imagen del instante actual, no obstante, a diferencia del comando anterior, la imagen no es procesada. Posteriormente la imagen es almacenada de manera temporal, comunicando la parte de YOLOv5 al monitor la ubicación de ésta, que es serializada (ver C17) y enviada a nuestro PC que finalmente la muestra por pantalla.

Algo recalcable de mencionar también acerca de la Figura 8 es el tinte rosa que presenta la imagen. Este tinte es debido a dos factores, que la imagen

fue tomada en una situación de intensa luz solar y la presencia de un filtro ultra-violeta (UV) en la propia cámara.

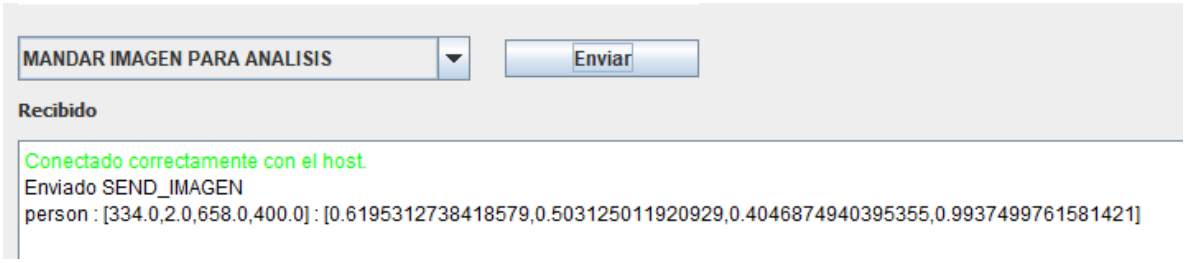
6.2.3.3 Mandar imagen para análisis. Implementado a través del comando “COMMAND_SEND_IMAGEN” (ver Figura 44). Este comando nos permitirá seleccionar una imagen almacenada en nuestro PC para que el equipo remoto (la red YOLOv5) la analice.

Figura 9. Ejemplo nº1 de funcionamiento de la característica "Mandar Imagen Para Análisis".



Desde nuestro PC, a través de un selector de ficheros (que acepta formatos de imagen jpg, jpeg, tif, tiff, bmp y png, ver K14), podremos escoger la imagen que queremos enviar para ser analizada (Figura 9).

Figura 10. Ejemplo nº2 de funcionamiento de la característica "Mandar Imagen Para Análisis".



Tras ser seleccionada, la imagen es serializada en nuestro PC y enviada al monitor del equipo remoto, tras esto, el monitor almacenará la imagen de manera temporal, indicando a continuación a YOLOv5 (detect.py) el comando recibido y la ubicación de la imagen. Posteriormente, la imagen es analizada por YOLOv5, extrayendo sus características y devolviéndolas a nuestro PC de forma idéntica a como ocurría la funcionalidad “Obtener información actual.”.

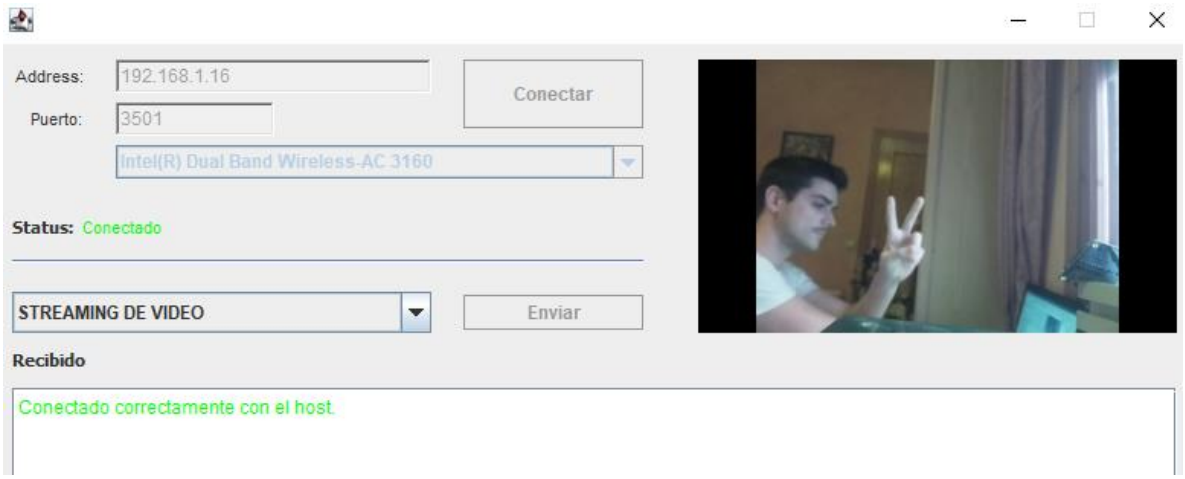
6.2.3.4 Mandar imagen obtener cuadro. Implementado a través del comando “COMMAND_SEND_IMAGEN_GETBOX” (ver Figura 45). Este comando nos permite enviar una imagen almacenada en nuestro PC para ser analizada y devuelta con sus características y un recuadro que enmarca el elemento identificado junto con su nivel de certeza en la identificación.

Figura 11. Ejemplo de funcionamiento de la característica “Mandar Imagen Obtener Cuadro”.

Este comando implementa el funcionamiento del anterior, añadiendo que, en el momento en el que YOLOv5 ha analizado la imagen, generando además en ella un marco rojo e indicando además el tipo de objeto identificado y la estimación de la fiabilidad de predicción de dicho objeto (con valor sobre 1), ésta es almacenada de forma temporal, devolviendo “*detect.py*” al monitor las características extraídas en formato “*json*” y el “*path*” donde se encuentra almacenada la imagen analizada (ya con el marco). Finalmente, la imagen es serializada y enviada junto con las características obtenidas a nuestro PC, el cual se encargará de mostrar todo por pantalla.

6.2.3.5 Streaming de vídeo. Implementado a través del comando "COMMAND_GET_STREAMING" (ver Figura 46). Este comando nos permite conectar con la cámara del equipo remoto, la cual enviará a nuestro PC imágenes en el instante actual, que serán mostradas como vídeo en la interfaz gráfica de la aplicación que corre en nuestro PC.

Figura 12. Ejemplo de funcionamiento de la característica "Streaming De Vídeo".



El funcionamiento de este comando es algo diferente al de los anteriores. Para esta característica, el monitor de comunicaciones de nuestro PC crea otro socket para mantener la comunicación en "*streaming*". El socket creado emplea la misma IP que la establecida a la hora de crear la conexión inicial con el equipo remoto, sin embargo, le suma una unidad al puerto previamente establecido (es decir, emplea para este ejemplo el puerto 3502), ya que el equipo implementa el servicio de "*streaming*" en el puerto inmediatamente superior al de la conexión inicial.

Una vez han creado la conexión ambos equipos, comenzará un bucle de peticiones de mensaje solicitando una imagen al equipo remoto, que éste contestará continuamente con mensajes que contengan una imagen del instante actual, que según vayan llegando serán mostradas en la interfaz gráfica del PC.

Algo destacable de este proceso es que es llevado enteramente a cabo por ambos monitores de comunicación (el del PC (TciTVClient.java) y el de la RPi(CommProviderService.cpp)), sin que en ningún momento tenga que intervenir YOLOv5.

6.2.3.6 Reproducir voz. Implementado a través del comando "COMMAND_GET_VOICE" (ver Figura 47). Este comando nos permite la obtención de información en forma de texto locutado (forma sonora) de lo que está captando en ese instante la cámara del equipo remoto.

Figura 13. Ejemplo de funcionamiento de la característica "Reproducir Voz".

Para esta característica el proceso seguido es el mismo que el seguido para la implementación de la característica "Obtener información actual.". Una vez nos son devueltas las características extraídas a nuestro PC, la rutina de tratamiento para este comando se encara de transformar los datos (recordemos que llegan con estructura de *json*) a una cadena de texto con sentido, es decir, a lenguaje natural. Finalmente, esta cadena es pasada como argumento a un pequeño programa Python (que a su vez es lanzado en este momento) llamado *Text2Speech.py*", capaz de transformar el texto que le es pasado como argumento a discurso sonoro, que es reproducido por nuestro PC. Finalmente, el monitor de comunicaciones espera a la finalización del programa *Python* para continuar su ejecución.

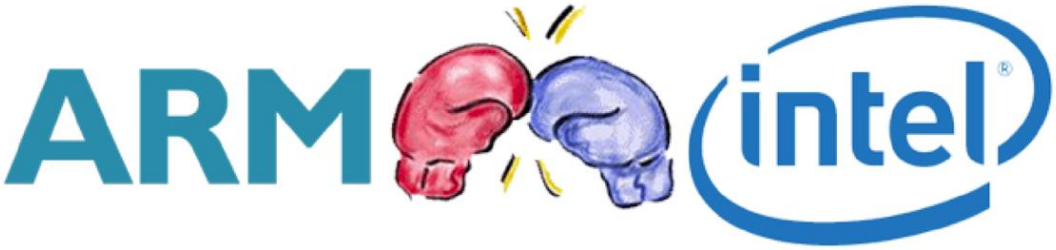
Es evidente que la existencia de una interfaz gráfica para poder utilizar este comando hace que pierda parte de su utilidad para la ayuda a la invidencia, no obstante, la funcionalidad está desarrollada y puede ser empleada en proyectos futuros que sí sean de utilidad real para invidentes.

Para finalizar, es destacable que las imágenes de ejemplo de las Figura 12 y Figura 13, por obvias razones no podemos apreciar enteramente el funcionamiento de este comando y el anterior, sin embargo, podemos ver un ejemplo del funcionamiento en forma de vídeo en este link.

6.3 EXPORTACIÓN DEL PROYECTO A RASPBERRY PI 4 Y PUESTA EN FUNCIONAMIENTO EN ARRANQUE

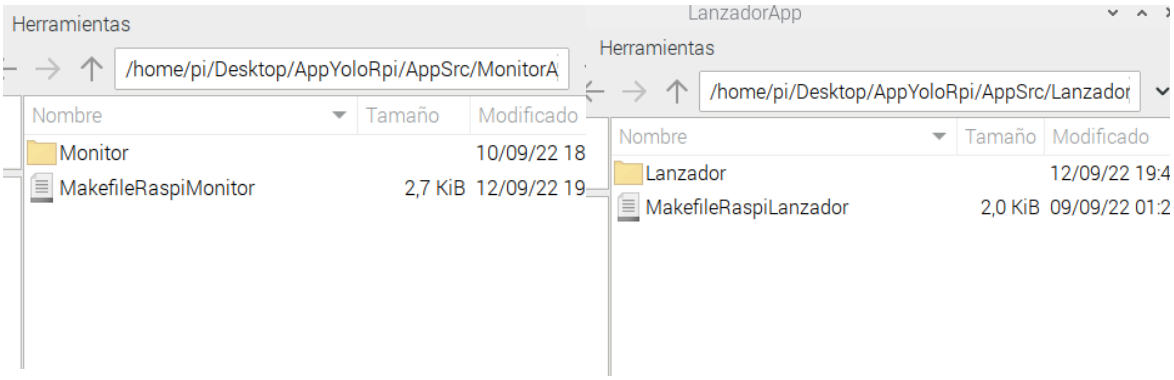
Con el objetivo de facilitar el desarrollo del software del equipo remoto (el que correrá la red YOLOv5 y el monitor de comunicaciones C/C++), decidimos que el mejor entorno de desarrollo sería una máquina virtual Ubuntu, que entre otras muchas cosas nos ofrecía un buen rendimiento (al correr en PC y no directamente en Raspberry Pi) además de otras ventajas como la facilidad de instalación de ciertas librerías o herramientas para codificar como Eclipse.

Figura 14. Ilustración satírica sobre las incompatibilidades de los procesadores ARM e Intel.



El hecho de desarrollar el código en una plataforma diferente a la que sería la que finalmente correría el código provocaba ciertas incompatibilidades, por ejemplo, que los ejecutables generados estuvieran creados para ser ejecutados en un procesador *Intel*, en lugar de un procesador *ARM* (incompatibles debido a que emplean juegos de instrucciones completamente diferentes). Otra de las incompatibilidades más destacables podría ser el hecho de que las rutas de instalación de ciertas librerías son diferentes en una máquina con respecto a la otra.

Figura 15. Imagen extraída de la carpeta "AppSrc" que contiene el código funte junto con los Makefile.



La solución que hemos decidido darle a este problema de incompatibilidades pasa por conservar el código fuente desarrollado en la máquina virtual y generar un fichero *makefile* (en realidad 2). Un archivo *makefile* contiene un conjunto de órdenes que deben ser ejecutadas por la utilidad "*make*" de *GNU*, así como las dependencias entre los distintos módulos del proyecto. En sí mismo, un archivo *makefile* no deja de ser un fichero de texto que nos permite compilar un código agregando a la compilación diversos *flags* como librerías y otras dependencias necesarias para generar archivos ejecutables. Los dos *makefile* generados para este propósito se corresponden con los dos proyectos C/C++ creados para la Raspberry Pi 4, el primero para compilar el lanzador y el segundo para compilar el monitor de comunicaciones. Cabe mencionar que el código de YOLOv5 no necesita de *makefile* ya que está escrito en *Python* (que es un lenguaje interpretado en lugar de compilado).

Finalmente, una vez resueltos los problemas derivados de exportar el proyecto de una máquina virtual Ubuntu a Raspberry Pi, también decidimos que los ejecutables generados por la compilación del código deberían iniciarse cuando la Raspberry Pi entrase en funcionamiento. Con este motivo generamos *un Shell script* (.sh) de nombre *my_startup_service.sh*, el cual se encargase de comprobar en primer lugar que existan los ejecutables (para en caso de que no existan generarlos empleando los *makefile*) para posteriormente de lanzarlos. Para lograr que en el inicio de la *RPi* se ejecute el archivo ".sh", hemos debido crear un servicio que se encargue de ello, tal y como aprendimos en la asignatura "Interfaces físicos y sistemas empotrados" del presente máster.

7. ESTRUCTURA DE LA SOLUCIÓN SOFTWARE

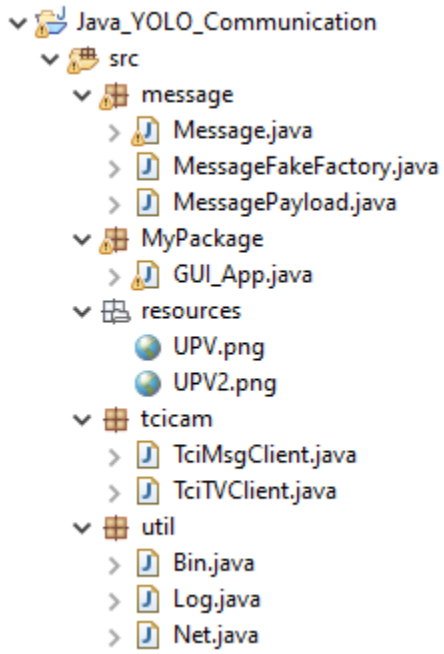
En este apartado veremos cómo se han estructurado los diferentes desarrollos software que componen en su conjunto la solución. Para ello, dividiremos el código en dos grupos, el software creado para que corra en nuestro ordenador personal y el desarrollado para que funcione en el equipo en remoto.

Trataremos de explicar los diversos ficheros que componen los dos grupos, explicando el papel que desempeñan dentro de éstos. Además, veremos cómo quedan estructurados los ficheros dentro de la propia solución.

7.1 SOFTWARE DEL PC

Comenzaremos hablando acerca del código implementado en lenguaje Java. Podemos apreciar el árbol de directorios de esta solución en la Figura 16.

Figura 16. Árbol de directorios del proyecto Java durante su desarrollo.



Podemos apreciar que la carpeta "*src*" contiene un total de 5 paquetes, cada uno de los cuales contiene a su vez varios recursos del proyecto. Siguiendo el orden en sentido descendente:

- Paquete “message”: contiene los ficheros java que se encargan de la creación de los distintos mensajes que serán intercambiados entre el PC y el remoto.

    - Message.java: se trata de uno de los ficheros más importantes de esta solución. Contiene la clase “Message”, conformada por la enumeración de los distintos comandos existentes para su uso, así como la definición de los diversos campos que conforman la estructura de los mensajes (ver Anexo 4. Formato de los mensajes intercambiados entre el PC y el equipo remoto.) y finalmente los diversos métodos que definen el comportamiento de los distintos comandos que podemos emplear en nuestra aplicación, rellenando los diversos campos que conforman un mensaje con información diferente en base al comando seleccionado y esperando diferentes respuestas también en consecuencia.

    - MessageFakeFactory.java: es un fichero encargado de generar mensajes que contengan campos con información aleatoria. Su utilidad radica a la hora de hacer pruebas de conexión entre el PC y el remoto.

    - MessagePayload.java: se trata del fichero que contiene la clase “Payload”. El payload es parte esencial de la estructura de nuestros mensajes, éste se encarga de almacenar la información que queremos enviar (imágenes, texto, números). Esta clase es utilizada por la clase “Message” como uno de los campos que la conforman. La clase “Payload” además contiene métodos de gran utilidad a la hora de interpretar y transformar la información que queremos recibir o enviar, por ejemplo, contiene un método capaz de transformar un byteArray (forma en la que enviamos las imágenes entre los dispositivos) a imagen.

- Paquete “MyPackage”: contiene tan solo un único fichero java, este fichero es el que contiene el código de la interfaz gráfica de la aplicación.

    - GUI_APP.java: como ya hemos comentado, este fichero contiene todo el código de la interfaz gráfica de la aplicación. Hace uso de las librerías de “Window Builder”.

- Paquete “resources”: contiene imágenes que son mostradas en la aplicación, concretamente se trata de dos logos de la UPV.

- Paquete “tcicam”: contiene el código relacionado con el establecimiento y mantenimiento de las comunicaciones.

- TciMsgClient.java: contiene la definición de la clase con el mismo nombre. Se encarga de establecer conexión desde nuestro PC con el equipo remoto a través de un socket TCP-IP. Además, este fichero contiene los métodos encargados de recibir y enviar el *stream* de datos relativo al intercambio de mensajes (getOutputStream y getInputStream).

- TciTVClient.java: contiene la definición de la clase con el mismo nombre. De manera similar al anterior fichero, se encarga de establecer conexión desde nuestro PC con el equipo remoto a través de un socket TCP-IP, no obstante, esta conexión no será utilizada para el envío de mensajes, si no para soportar la característica de streaming de vídeo de la aplicación. De esta forma, esta clase se encarga de recibir las imágenes del equipo remoto.

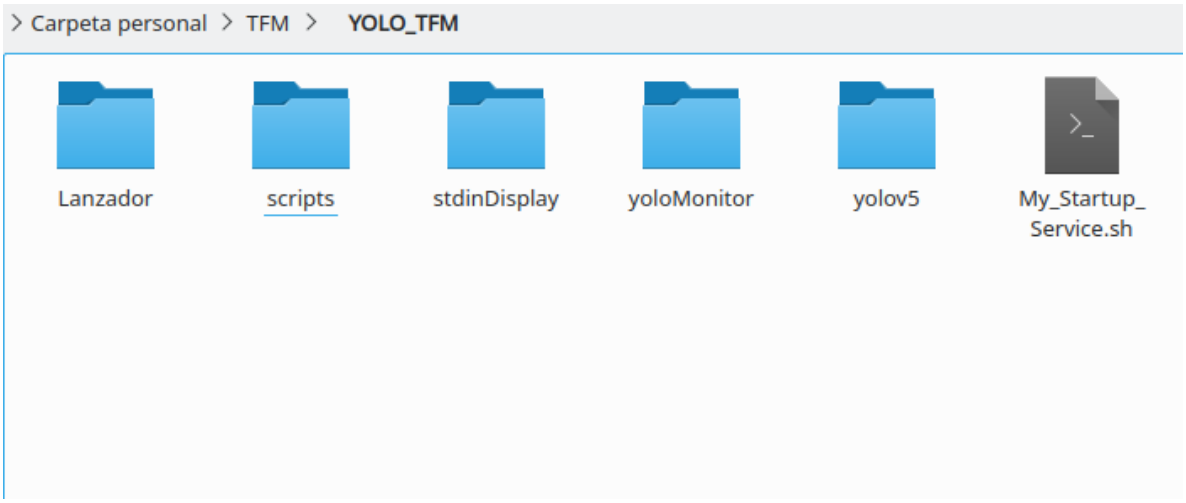
- Paquete “util”: se trata de un paquete que contiene 3 ficheros con código que nos puede ser de utilidad a la hora de ampliar la aplicación (como por ejemplo para comprobar si una IP es válida). No entraremos en demasiados detalles pues no son de gran relevancia para el funcionamiento del proyecto.

Cabe destacar la existencia de un pequeño programa en Python que lanzamos desde el método que define el comportamiento del comando “COMMAND_GET_VOICE” (en el fichero Message.java). Este programa Python es el encargado de transformar texto a sonido para posteriormente ser reproducido.

7.2 SOFTWARE DEL EQUIPO REMOTO (RASPBERRY PI 4)

Finalizaremos este apartado hablando del software que corre en el equipo remoto, en nuestro caso en la Raspberry Pi 4. Este software se divide en dos bloques, el primero, el cual es el encargado de gestionar las comunicaciones con nuestro PC y el segundo, el conformado por la red YOLOv5.

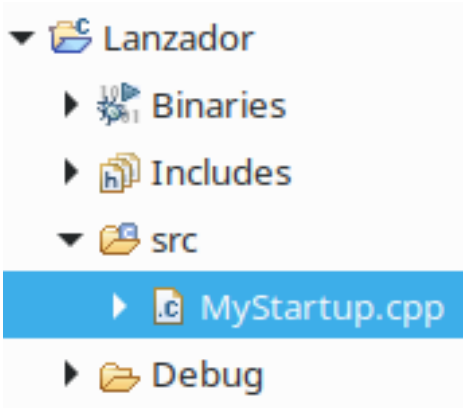
Figura 17. Directorio de desarrollo de la aplicación del equipo remoto en Kubuntu.



El primero de los bloques viene conformado por dos proyectos en c/c++, “Lanzador” y “yoloMonitor”.

- Lanzador: se trata de un proyecto que contiene una única clase, “MyStartup.cpp”.

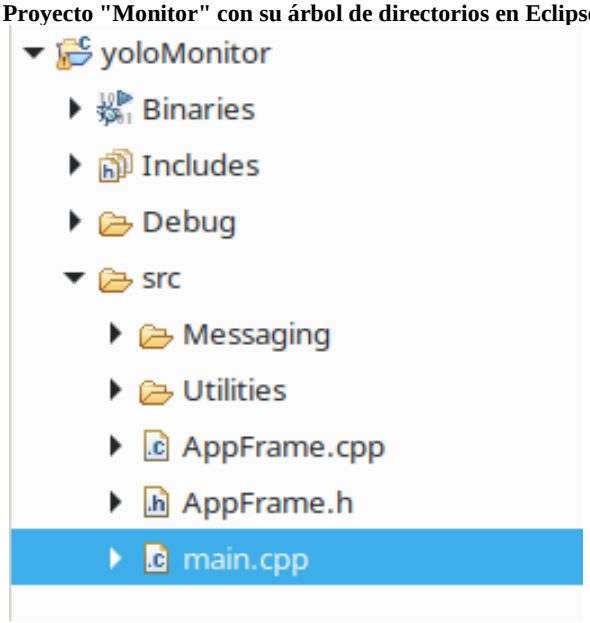
Figura 18. Proyecto "Lanzador" con su árbol de directorios en Eclipse.



Esta clase será la encargada de poner en funcionamiento tanto la red YOLOv5 como el monitor de comunicaciones cuando arranque la Raspberry. Además, cumple un papel muy importante a la hora de establecer comunicación entre el monitor de comunicaciones y la red YOLOv5, ya que se encarga de crear dos “*pipe*” para que ambos programas puedan traspasar información entre ellos.

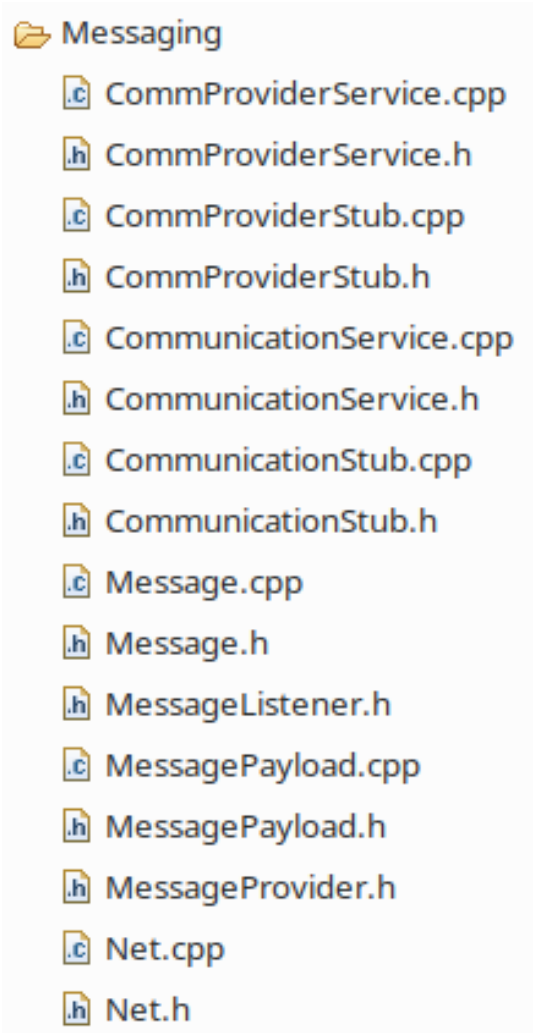
- YoloMonitor: se trata del proyecto que contiene el código del monitor de comunicaciones, que emplearemos para mantener conexión con nuestro PC.

Figura 19. Proyecto "Monitor" con su árbol de directorios en Eclipse.



Siguiendo un orden descendente nos encontramos primeramente con la carpeta "*Messaging*", encargada de contener la mayoría de los archivos que conforman el comportamiento del monitor, entre estos archivos podemos encontrar los mostrados en la Figura 20.

Figura 20. Directorio "*Messaging*" del proyecto "*yoloMonitor*".



Pasaremos a continuación a hablar del rol que cumplen algunos de los ficheros más relevantes para el funcionamiento del programa.

- CommProviderService: se trata del fichero que contiene el código para el establecimiento de una conexión de mensajes en *streaming*. En él se establece una conexión a través de un socket TCP/IP para posteriormente proceder al envío de imágenes empleando mensajes que son enviados a través de éste. Cabe destacar la posibilidad de establecer múltiples conexiones simultáneas, así como la existencia de tolerancia a posibles fallos en la comunicación.

- CommunicationService: se trata del fichero que contiene el código para el establecimiento de una conexión para el intercambio de

mensajes de manera puntual. Posee un comportamiento muy similar al fichero anterior.

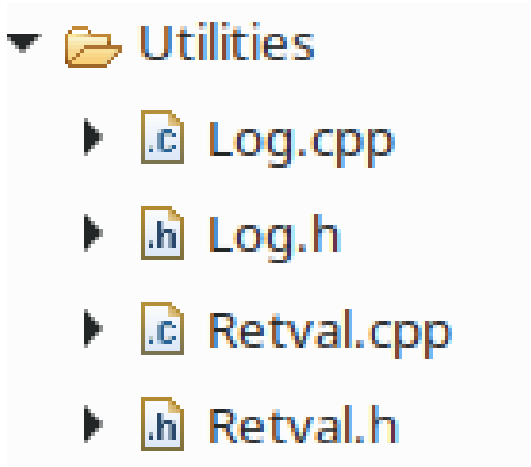
- Message: se trata del fichero que contiene el código para la definición de la estructura de los mensajes (los campos que conforman un mensaje), así como la definición de los diferentes comandos que podrán ser enviados/recibidos durante la comunicación (y que conforman uno de los campos de los mensajes). Finalmente, contiene los métodos para leer y escribir mensajes a través del socket de la comunicación.

- MessagePayload: se trata del fichero que contiene el código que define el campo *payload* del mensaje. Este campo es el encargado de albergar la información que es enviada/recibida (imágenes, cadenas de texto o números). Además, contiene métodos destinados al tratamiento de la información, como por ejemplo un método para transformar un byte array a una imagen.

- Net: se trata del fichero que contiene el código de un menú para la configuración de las opciones de red (como por ejemplo la selección de la interfaz de red para el establecimiento de las comunicaciones).

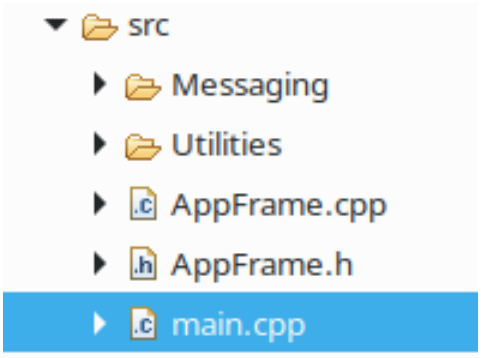
Por otra parte, la carpeta "*Utilities*", al igual que en el anteriormente explicado monitor de comunicaciones escrito en lenguaje java, contiene una serie de ficheros para añadir funcionalidades extra al monitor, como por ejemplo un "*Logger*" para guardar un histórico de lo que ocurre en nuestro monitor. No obstante, no es necesario que profundicemos más en esta carpeta.

Figura 21. Directorio "*Utilities*" del proyecto "*yoloMonitor*".



Finalmente, existen tres archivos (dos en realidad si tomamos como uno único al "*.cpp*" y al .h que poseen el mismo nombre) que se encuentran dentro de la propia carpeta "*src*".

Figura 22. Árbol de directorios del código fuente del proyecto "*yoloMonitor*" visto desde Eclipse.



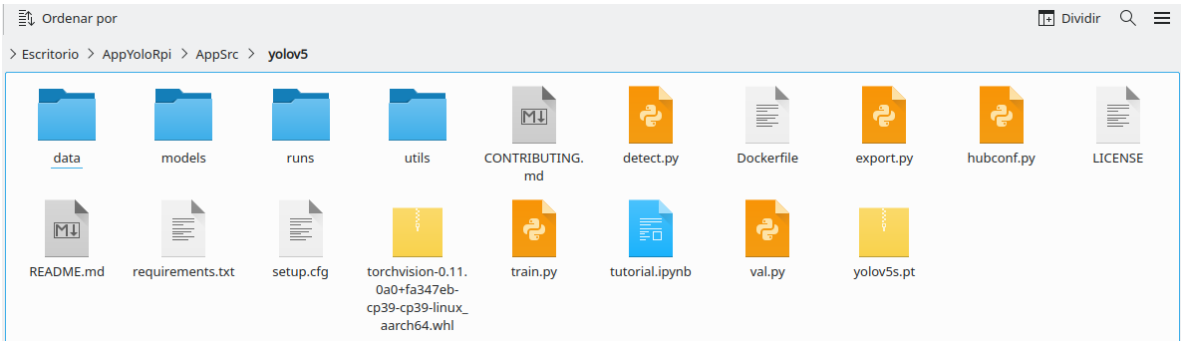
Nuevamente, en sentido descendiente, los archivos son:

- AppFrame: se trata del archivo que contiene el código para el procesamiento de los mensajes. Contiene diferentes métodos que albergan la lógica que deberá llevarse a cabo en función del comando que venga adjunto al mensaje recibido. Es en este fichero donde realmente se comunican la red YOLOv5 y nuestro monitor de comunicaciones. Es uno de los archivos más importantes del monitor.

- main: se trata del archivo que contiene el código de arranque de este proyecto (recordemos que, a su vez, éste es lanzado por el proyecto "Lanzador"). Este fichero se encarga, entre otras cosas, de recibir los descriptores de archivo a través de los cuales nuestro monitor de comunicaciones establecerá diálogo con la red YOLOv5, también se encargará de arrancar los servicios de comunicación, tanto el de comunicación por *streaming* como el de comunicación "normal" de un único mensaje. Finalmente, cabe destacar que este fichero también se encarga de obtener una instancia de "AppFrame" (que actúa como un *singleton*) para posteriormente ser utilizada por los servicios de comunicación.

- YOLOv5: contiene el código fuente de la red neuronal YOLOv5. Explicar todos los archivos y directorios existentes en esta parte del proyecto sería muy extenso, además, no es el objetivo de este TFM explicar cómo funciona la red neuronal. En caso de querer algo de información, podemos encontrar documentación y tutoriales acerca de los ficheros y la estructura que componen a YOLOv5 en su página web.

No obstante, sí que hablaremos acerca de las modificaciones que hemos llevado a cabo sobre los archivos de YOLOv5.



En la imagen se muestran los archivos y carpetas que obtenemos al descargar YOLOv5. De esta imagen nos interesa analizar un archivo, "*detect.py*" y una carpeta, "*data*". Ambos son los elementos que hemos tenido que modificar para llevar a cabo nuestro proyecto.

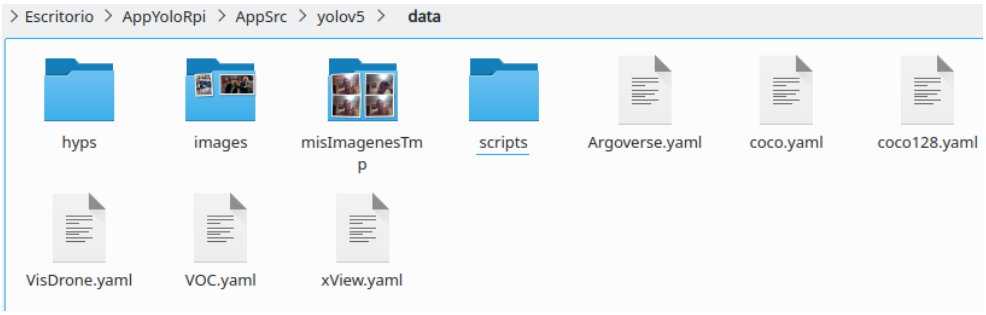
- Detect.py: se trata del archivo "principal" de YOLOv5. Es el fichero en el que se lleva a cabo el procesamiento y posterior identificación de los elementos en las imágenes. Actúa como una suerte de archivo "*main*", siendo en el que se inicia la aplicación y desde donde se hace uso de otros archivos.

  Este fichero fue modificado para poder establecer comunicación con el monitor (para establecer conexión con nuestro PC) que corre en paralelo con la red YOLOv5. De esta manera, los comandos solicitados llegan al monitor, el cual, y en caso de ser necesario, se los comunica a este archivo, que a su vez posee diversas rutinas de tratamiento para los distintos comandos.

  Así, por ejemplo, si llega a este archivo el comando "COMMAND_GET_CURRENTINFO", la rutina de tratamiento tomará una foto del instante actual (haciendo uso de la cámara conectada a la RPi) y a continuación, procesará la imagen para extraer sus características, las cuales serán comunicadas al monitor para que las transfiera a nuestro PC.

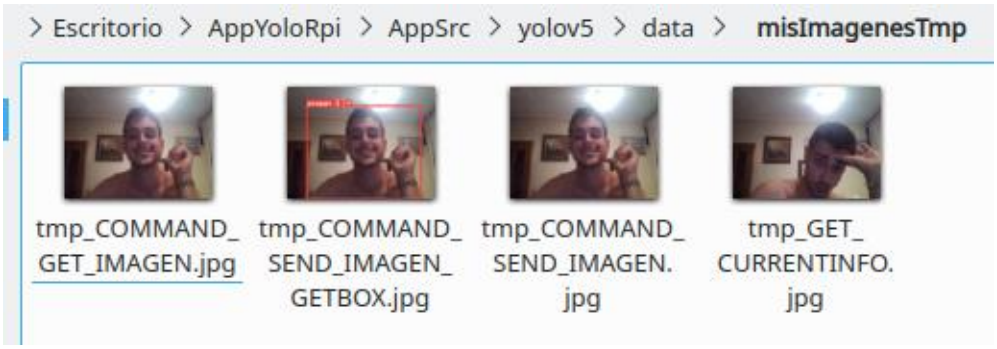
- Carpeta data: este directorio fue el escogido para almacenar información temporal relacionada con las imágenes generadas por YOLOv5 (concretamente *detect.py*).

Figura 23. Contenido de la carpeta "*data*" del árbol de directorios de YOLOv5.



Hablamos acerca de la subcarpeta “misImagenesTmp”, creada por nosotros mismos para albergar de manera temporal imágenes ligadas a ciertos comandos, las cuales se van sobreescribiendo sobre las anteriormente existentes.

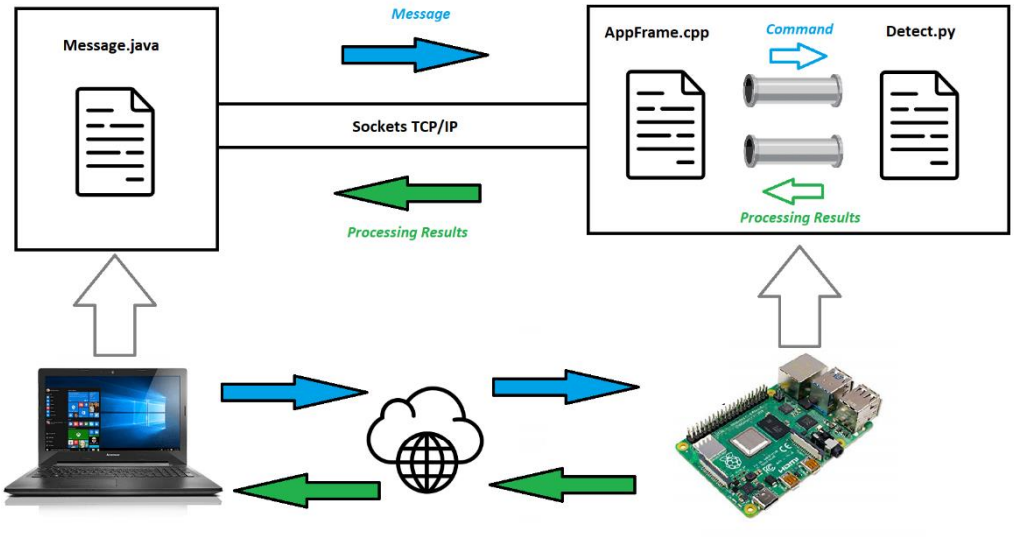
Figura 24. Contenido de la carpeta "misImagenesTmp" derivado de la utilización del proyecto.



Así, por ejemplo, el comando “COMMAND_GET_IMAGEN” generará un archivo “*.jpg*” de nombre “tmp_ COMMAND_GET_IMAGEN”.

Ahora que ya conocemos cómo se estructura nuestra solución software y que sabemos qué rol cumple cada archivo dentro del propio proyecto, podemos pasar a hablar acerca del funcionamiento del proyecto desde un punto de vista más general. Para tal objetivo nos ayudaremos con el diagrama que muestra la Figura 25.

Figura 25. Diagrama ilustrando el funcionamiento de la solución creada.



En el momento en el que disponemos de ambos monitores en funcionamiento (el de la parte del PC y el de la parte de la Raspberry), a través de la interfaz gráfica de la solución del PC podremos conectar con el equipo remoto. La conexión se establecerá a través del uso de sockets TCP/IP.

Una vez ya conectados, podremos hacer uso de los diferentes comandos a nuestra disposición. Al seleccionar y enviar uno, será enviado un mensaje que contendrá, entre otras cosas, con un campo con el nombre del comando escogido, además podrá ser enviada o no dentro del propio mensaje información adicional, como por ejemplo imágenes. Una vez enviado el mensaje, nuestro PC queda a la espera de una respuesta.

El mensaje será recibido por el monitor de la Raspberry Pi, llegando hasta el archivo "*AppFrame.cpp*", que cuenta con manejadores que implementan distintos comportamientos en función del comando recibido. A continuación, por lo general, el archivo "*AppFrame.cpp*" enviará al archivo "*Detect.py*" (que forma parte de YOLOv5) en forma de cadena de texto y a través de unas tuberías *UNIX* creadas para tal propósito, el valor del comando recibido y en algunas situaciones información adicional (como por ejemplo en *path* en el que se encuentra una imagen a procesar).

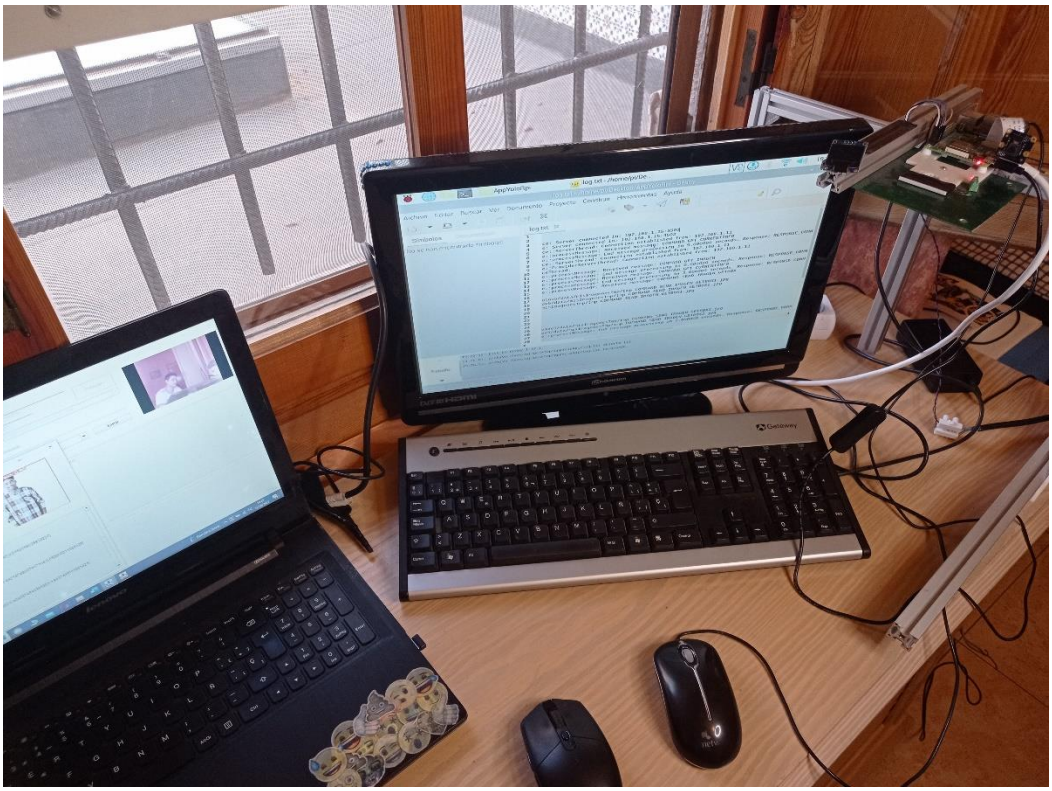
Tras el procesamiento de imágenes, el archivo "*Detect.py*" retorna al archivo "*AppFrame.cpp*" la información extraída a través de las *pipes* creadas en forma de texto (por ejemplo las características extraídas de una imagen son enviadas en formato *json*, que son cadenas de texto). A continuación, la información pertinente es enviada de vuelta a nuestro PC en forma de mensaje (ver Anexo 4. Formato de los mensajes intercambiados entre el PC y el equipo remoto.). El mensaje llegará a

través de la conexión TCP/IP hasta el archivo “*Message.java*”, donde la información será extraída del mensaje y mostrada a través de la interfaz gráfica.

8. RESULTADOS

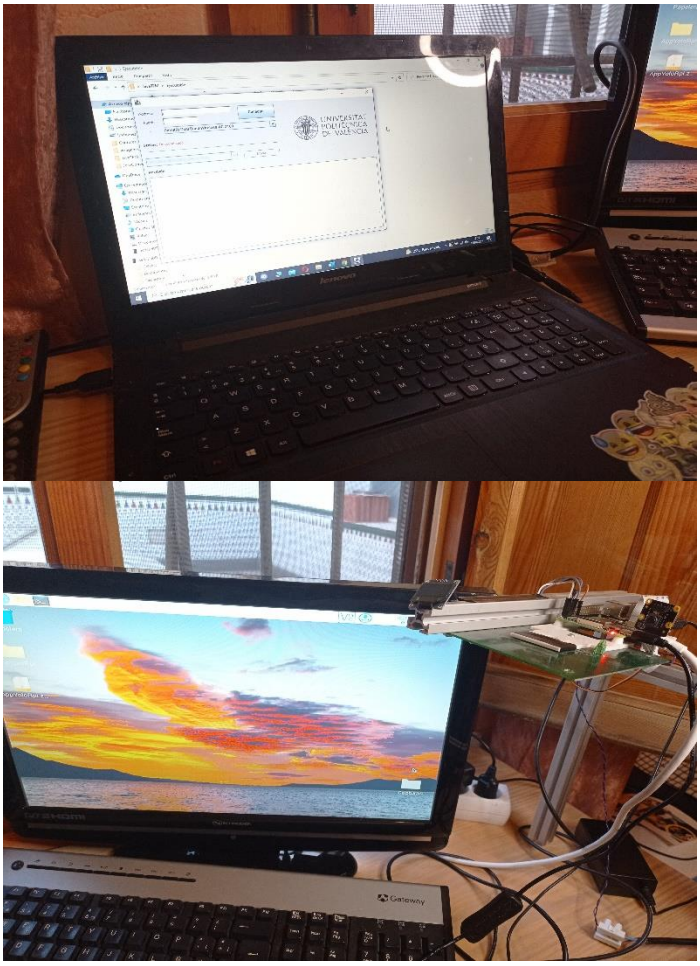
En este apartado nos centraremos en mostrar los resultados obtenidos frutos del desarrollo del trabajo. Mostraremos de esta forma el sistema funcionando en su conjunto.

Figura 26. Resultados obtenidos en funcionamiento.

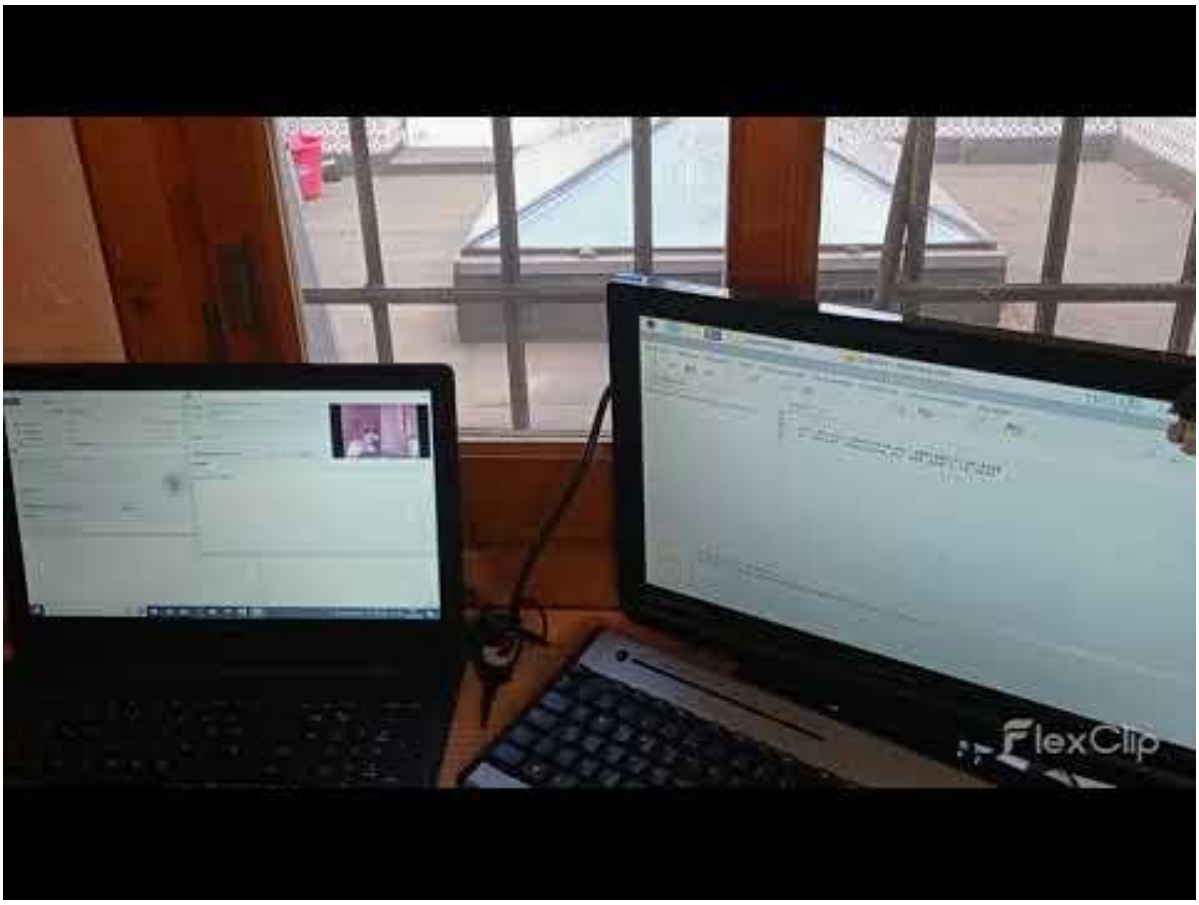


Primeramente, para este apartado concreto mostraremos los dos componentes que conforman la solución, el PC y el equipo remoto (Figura 27).

Figura 27. Portátil corriendo el monitor de comunicaciones (Arriba, PC) y Raspberry Pi 4 corriendo el sistema operativo insttalado (Abajo, equipo remoto).



Finalmente, podemos considerar que la mejor forma de mostrar los resultados es a través de un vídeo (aunque la calidad de imagen no es demasiado buena).


FlexClip

9. ANALISIS Y DISCUSION DE RESULTADOS

En este nuevo apartado discutiremos acerca de los resultados obtenidos tras la elaboración completa del proyecto que plantea este TFM. Hablaremos para ello acerca de los resultados obtenidos respecto al marco teórico, la metodología, equipos usados, etc.

Desde un punto de vista general, los resultados obtenidos (expuestos en el apartado anterior, RESULTADOS) son los esperados. Podemos comprobar cómo los objetivos generales de este TFM, conseguir un sistema conformado por un equipo remoto y un equipo personal capaces de establecer comunicación y sacando partido a la red neuronal YOLOv5, han sido cumplidos.

El marco teórico planteaba la utilización de un monitor de comunicaciones para establecer las comunicaciones, al cual poder tener acceso de forma gráfica. Este objetivo es logrado a través de la utilización del plugin de Eclipse "*Window Builder*", con el que hemos podido crear en Java una interfaz capaz de sacar partido al monitor de comunicaciones que nos fue facilitado por el director de este TFM.

También se planteaba el hecho de poder sacar partido a la red neuronal de carácter "*open-source*" Yolov5. Si bien el jugo que se le puede extraer a ésta es muchísimo mayor, nuestro desarrollo implementa las características básicas que esta ofrece, refiriéndonos por ejemplo a la extracción de las propiedades de una imagen (desde el punto de vista del análisis de imagen que ofrece YOLOv5). Siendo estas características además extendidas con la inclusión por ejemplo de un mecanismo de locución de texto relacionado con las propiedades extraídas de la imagen.

En general, durante la etapa de desarrollo del software de la solución, no ha habido grandes problemas a la hora de adaptarnos al marco planteado inicialmente para la consecución de las características que queríamos implementar. Sin embargo, sí que ha habido características más complejas de desarrollar que otras, como podría ser el traspaso de imágenes a través de los monitores de comunicación, así como otras características que o bien no ha dado tiempo de implementar o no hemos logrado su desarrollo, como podría serlo la utilización de forma simultánea por parte de dos conexiones de la cámara de la Raspberry Pi.

En lo relacionado al hardware empleado en este proyecto, desde un principio fue planteado el finalmente utilizado. La Raspberry Pi fue escogida por su disposición inmediata ya que la facultad de informática disponía de varias. Lo mismo sucede con la cámara que ésta emplea. En cuanto al PC utilizado fue escogido por ser el ordenador personal del que el creador de este documento disponía, con el aliciente de poseer gran parte del software finalmente utilizado ya instalado, como por ejemplo Python 3, Eclipse o Virtualbox. Por último, las herramientas software utilizadas para desarrollar la aplicación fueron escogidas en base a la familiaridad (es *software* que el desarrollador del proyecto ya ha utilizado en otras ocasiones, de hecho, en este máster) y libre disposición (todo el *software* empleado se encuentra disponible de forma gratuita) de los programas.

10. CONCLUSIONES

En este nuevo apartado, queremos plasmar las conclusiones que fueron obtenidas del desarrollo práctico del proyecto.

En cuanto a la consecución de los objetivos generales, desde el punto de vista didáctico, el desarrollo del proyecto en su conjunto nos ha servido para afianzar y ampliar los conocimientos acerca del ámbito de las telecomunicaciones, estudiado en diversas asignaturas del presente máster.

Durante el desarrollo de la solución nos hemos enfrentado a problemas relacionados con conocimientos sobre los que ya teníamos una base como lo son las telecomunicaciones empleando sockets TCP/IP que implementa Java. No obstante, en su mayoría los problemas a resolver estaban relacionados con conceptos y tecnologías las cuales eran desconocidas hasta este momento, algunos ejemplos de esta afirmación podrían ser:

- Las estructuras de datos “Json”. Empleadas a la hora de transmitir las características obtenidas de una imagen desde el equipo remoto hasta nuestro PC.
- Serialización de imágenes. Empleada para enviar imágenes entre los equipos (ver C17).
- Creación de archivos *makefile* y en general exportación de proyectos a otra plataforma.
- …

Finalmente, el objetivo general planteado en un inicio podemos darlo por cumplido.

Este objetivo consistía en conseguir establecer comunicación entre nuestro PC y un equipo remoto a través de un monitor que sirva a ese propósito y con el objetivo de poder disponer de manera remota de las características que nos ofrece YOLOv5, siendo además capaz de configurarla de algún modo.

Aunque la parte de la configuración de la red YOLOv5 es algo más limitada en este desarrollo, en general el resto de los aspectos planteados en el objetivo general se ven satisfechos.

De manera general, también podemos destacar que en su mayoría los objetivos específicos que fueron planteados inicialmente se han visto implementados en la solución final. Así:

- Hemos conseguido combinar la YOLOv5 con el monitor de comunicaciones. Implementado a través de "*pipes*" propias de los sistemas operativos *UNIX*.

- También ha sido desarrollado de forma satisfactoria la capacidad de ofrecer *Streaming* de vídeo desde el dispositivo remoto. Aún así, es reseñable una carencia de la implementación y es que a pesar de ser posible la conexión de varios equipos (PC's) con el terminal remoto, éste no ofrece la posibilidad de mostrar en dos conexiones el *streaming* de forma simultánea. Esto es debido a que una de las conexiones toma el control de la cámara impidiendo a la otra hacer uso de ella.

- Se ha logrado la característica de obtención de imágenes en el instante en que se solicite. Implementada por el comando "COMMAND_GET_IMAGEN".

- La obtención de información en texto sobre la situación actual del entorno en el que se encuentre el dispositivo "B" ha sido también implementada gracias al comando "COMMAND_GET_CURRENTINFO".

- La característica para la locución de texto en forma sonora también fue lograda gracias a la creación de un pequeño programa *Python*.

- Por último, la posibilidad de enviar imágenes desde el dispositivo "A" al "B" para que YOLOv5 analice la imagen fue incluida en la aplicación desarrollada en Java, pudiendo seleccionar la imagen a través de un desplegable gráfico que también permite la búsqueda por tipo de imagen (formato de la imagen).

Sin embargo, no todos los objetivos específicos han sido finalmente logrados, existen algunos de los planteados originalmente que no se encuentran en la solución final generada:

- El objetivo planteado para la configuración de la red YOLOv5 no ha sido logrado enteramente. Si bien algunos comandos configuran de una u otra manera las características de la red, no hemos implementado un medio a través del cual ser capaces de configurarla de la forma deseada en el momento deseado.

- La característica planteada inicialmente acerca de la aplicación de técnicas de visión por computador a las imágenes obtenidas finalmente no ha sido incluida en ningún aspecto del desarrollo final.

De forma resumida y ya a modo de reflexión final acerca del proyecto. A pesar de las dificultades atravesadas durante el desarrollo, la cantidad de conceptos, conocimientos, tecnologías, … sobre las que hemos aprendido ha sido notoria. Además, el proyecto si bien no ha logrado todos los objetivos que planteaba en un inicio, ha acabado siendo un sistema bastante complejo y funcional que de manera general cumple con lo que pretendía ser.

11. RECOMENDACIONES Y TRABAJOS FUTUROS

En este apartado trataremos acerca de los posibles trabajos futuros que puedan ser implementados sobre la solución final (más bien soluciones), relatando además posibles consejos que puedan ser de utilidad a la hora llevar a cabo esta labor.

Dividiremos de esta forma el apartado en dos subapartados, que siguiendo el propio título del epígrafe serán: recomendaciones y trabajos futuros.

11.1 TRABAJOS FUTUROS

Sin mayores rodeos pasaremos a plantear una lista con una serie de ideas acerca de posibles ampliaciones sobre el proyecto.

11.1.1 Ayuda real a los discapacitados visuales. Como ya fue comentado previamente en este documento, la característica para la locución de texto de forma sonora posee un limitante muy importante a la hora de servir como una ayuda real.

Nos referimos a la existencia de una interfaz gráfica a la hora de utilizar la característica mencionada, lo cual la hace incompatible con discapacitados visuales de alto grado.

A modo de solución, podría ser implementado un dispositivo físico (por ejemplo, un botón) que pudiera conectar con el equipo remoto y que al ser pulsado mandara la orden para la ejecución de la característica.

11.1.2 Posibilitar la utilización simultánea de la cámara a través de dos conexiones distintas. Como hemos podido ver en los ejemplos mostrados en vídeo de los resultados en funcionamiento, es posible la coexistencia de múltiples conexiones al mismo tiempo con el equipo remoto.

Sin embargo, a pesar de ello, aún sigue sin ser posible que las conexiones utilicen la funcionalidad de vídeo en *streaming* simultánea. Esto es debido a que mientras una conexión está empleando la cámara, el resto no puede hacer uso de ella debido a que ya se encuentra ocupada.

En cuanto a esta futura mejora no se nos ha ocurrido ninguna posible solución al respecto.

11.1.3 Mayor tolerancia a fallos. A pesar de la enorme robustez que han demostrado tener los monitores de comunicaciones, aún existen situaciones en las que las aplicaciones pueden fallar (por ejemplo, la mencionada en el punto anterior sobre la utilización simultánea de la cámara por parte de dos conexiones, lo cual hace que el sistema falle).

El carácter remoto de una parte de la solución hace que sea preciso una buena tolerancia a los fallos (para ahorrar viajes al equipo remoto).

11.1.4 Implementación de nuevas características. Existen muchas otras características que pueden ser implementadas a través de la creación de nuevos comandos y manejadores.

Para ello se puede seguir la estructuración ya establecida en el código, pudiendo tomarse de ejemplo comandos ya creados que implementan ciertas características.

Para este apartado entra en juego la imaginación e ingenio del futuro desarrollador que retome el proyecto.

11.2 RECOMENDACIONES

Sirva este punto como una serie de consejos y observaciones que puedan ser de ayuda para futuros desarrolladores de la solución.

11.2.1 Lectura y apoyo en la documentación existente. Empezando por este propio documento y pasando por los documentos anexos y bibliografía.

Esto nos puede facilitar enormemente el entender la solución creada.

11.2.2 Utilización de la herramienta Eclipse. Los proyectos han sido creados originalmente en el programa de desarrollo Eclipse. Desde la opinión más personal, Eclipse ha demostrado ser una herramienta muy potente y versátil, que ha permitido facilitar enormemente la labor de desarrollo.

Otro motivo para el uso de Eclipse es el de evitar posibles incompatibilidades con el código generado, ya que éste ha sido creado en Eclipsees seguro que podemos abrir directamente los proyectos en el programa y ejecutarlos sin que surja ningún posible inconveniente.

11.2.3 Análisis del código y comprensión de las estructuras creadas. Para comprender mejor a qué nos referimos en este apartado pondremos un ejemplo.

A la hora de desarrollar un nuevo comando que implemente alguna nueva característica, el futuro desarrollador podría tomar como ejemplo las estructuras ya creadas para las características ya existentes, lo cual puede facilitar su futura labor.

12. REFERENCIAS BIBLIOGRÁFICAS

[K21]
Kateb,F.A.;Monowar M.M.; Hamid, M.A.; Ohi, A.Q.; Mridha M.F. FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards. *Agronomy.* Noviembre, 2021. Recuperado de: https://www.researchgate.net/figure/A-detection-model-contains-a-backbone-neck-head-module-The-backbone-module-exploits_fig1_356638292#:~:text=The%20backbone%20module%20exploits%20the,of%20objects%20in%20different%20resolutions.

[Y20]
YOLO V5 — Explained and Demystified (2020, julio 1). ''YOLO V5 — Explained and Demystified; YOLO V5 — Model Architecture and Technical Details Explanation''. Recuperado de: https://towardsai.net/p/computer-vision/yolo-v5%E2%80%8A-%E2%80%8Aexplained-and-demystified

[S20]
SOLAWETZ, J. (2020, junio 5): ''YOLOv5 New Version - Improvements and Evaluation''. Recuperado de: https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[F16]
FARHADI, A; REDMON, J. (2016 diciembre- 2020 mayo): ''YOLOv5 Documentation''. Recuperado de: https://docs.ultralytics.com

[E20]
ECKEL, B.: ''Thinking in C++, Volume 1, 2nd Edition''. *Planet PDF*. Enero, 2020. Recuperado de: https://www.micc.unifi.it/bertini/download/programmazione/TICPP-2nd-ed-Vol-one-printed.pdf

[P22]
PHYTON: ''Python 3.10.7 documentation''. Recuperado de: https://docs.python.org/

[K14]
KAR, C. (2014, enero 13): Add JFileChooser to Eclipse Window Builder. *Stack overflow.* Recuperado de: https://stackoverflow.com/questions/24202474/add-jfilechooser-to-eclipse-window-builder

[K22]
KLOOS, T. (2022, JULIO 8): ''Socket Programming in C/C++''. *Geeksforgeeks*. Recuperado de: https://www.geeksforgeeks.org/socket-programming-cc/

[B12]
BANAHAN, M; BRADY, D; DORAN, M.: The C Book. *Addison Wesley*. Septiembre, 2012. Recuperado de: https://openlibra.com/es/book/the-c-book

[C17]
CPLUSPLUS (2017, JUNIO 8): ‘’How to pass image file as pointer and reference as std::vector<char> ?’’. Recuperado de: https://cplusplus.com/forum/general/217487/

13. ANEXOS

Este apartado cumplirá la función de complementar o completar la información previa que hemos podido leer a lo largo de este documento.

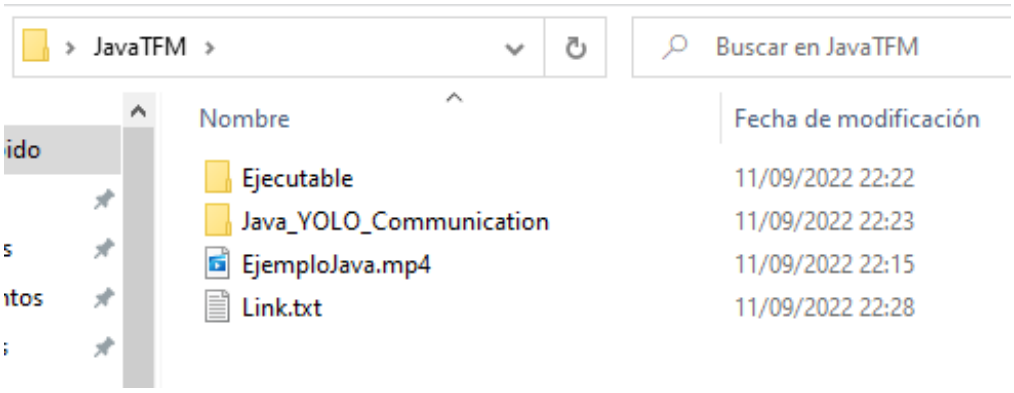
13.1 ANEXO 1. TUTORIAL PARA LA DESCARGA Y EJECUCIÓN DEL SOFTWARE DESARROLLADO PARA NUESTRAS COMPUTADORAS PERSONALES.

Sirva este anexo como guía para poder utilizar la solución Java creada en otro PC. Para ello haremos un repaso punto por punto de cómo podemos hacer funcionar la aplicación en nuestro equipo.

1. Descargaremos el archivo zip disponible para su descarga aquí.

2. Descomprimimos el archivo en la ruta que deseemos, obteniendo el siguiente resultado:

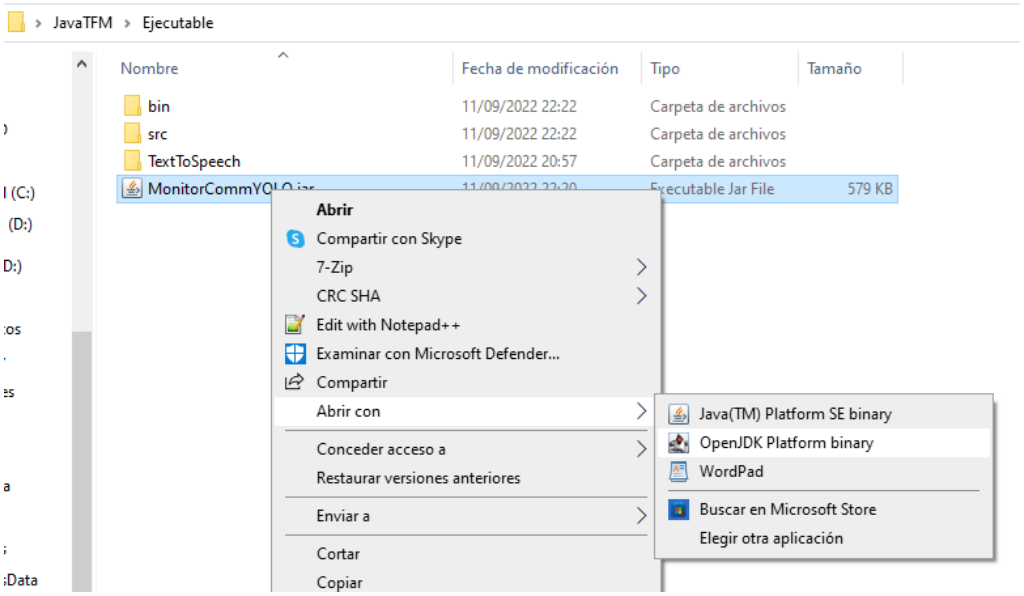
Figura 28. Contenido de la carpeta final fruto del desarrollo de la parte del PC.



Aquí podremos observar la carpeta “Ejecutable”, en la que se encontrará un archivo “.jar” que podremos lanzar para correr la aplicación. También aparece el proyecto java completo, de nombre “Java_YOLO_Communication”, donde se encuentra todo el código fuente de la aplicación (disponible en caso de querer abrir el proyecto en algún entorno de desarrollo como Eclipse). El resto lo conforman un vídeo mostrando el funcionamiento de la aplicación y un link del mismo vídeo subido a YouTube.

3. Accediendo a la carpeta “Ejecutable” y cumpliendo con los requisitos previos necesarios para la puesta en marcha, correremos la aplicación llamada “*MonitorCommYOLO.jar*” tal y como se muestra en la Figura 29.

Figura 29. Tutorial sobre cómo correr el archivo ejecutable "*MonitorCommYolo.jar*".



Tras esto, se abrirá la aplicación que nos permitirá conectar con la RPi para mandarle distintos comandos.

Figura 30. Interfaz gráfica de la aplicación desarrollada en java para la parte del PC.

También, mencionaremos que el contenido de la carpeta “Ejecutable”, que contiene las carpetas “*bin*” y “*src*”, ambas carpetas contienen los binarios y el código fuente de la aplicación respectivamente, mientras que la carpeta “*TextToSpeech*” contiene un breve fichero de *Python* de nombre “*Text2Speech.py*”, capaz de transformar texto a discurso sonoro.

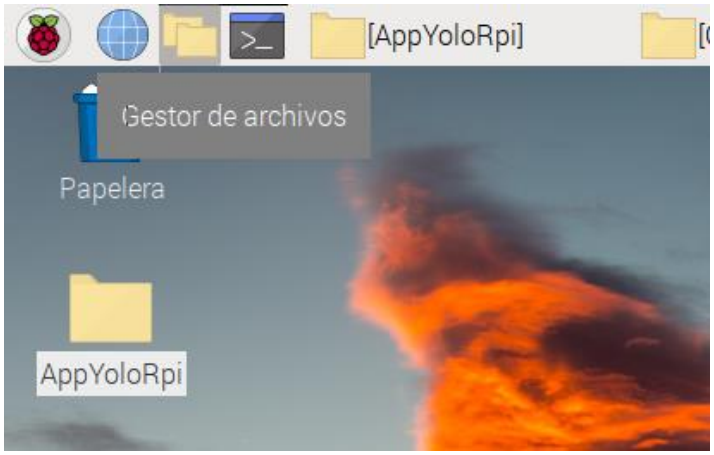
13.2 ANEXO 2. TUTORIAL PARA LA DESCARGA Y EJECUCIÓN DEL SOFTWARE DESARROLLADO PARA EL EQUIPO REMOTO (RASPBERRY PI).

A continuación, y una vez hemos visto los diversos archivos que conforman esta parte de la solución, pasaremos a explicar punto por punto cómo poner en marcha este software en nuestra RPi (teniendo en cuenta configuraciones previas de las que ya hemos hablado):

1. Descargamos el archivo zip que contiene la solución en el escritorio de la RPi donde queremos desplegar el programa. Disponible online en este enlace.

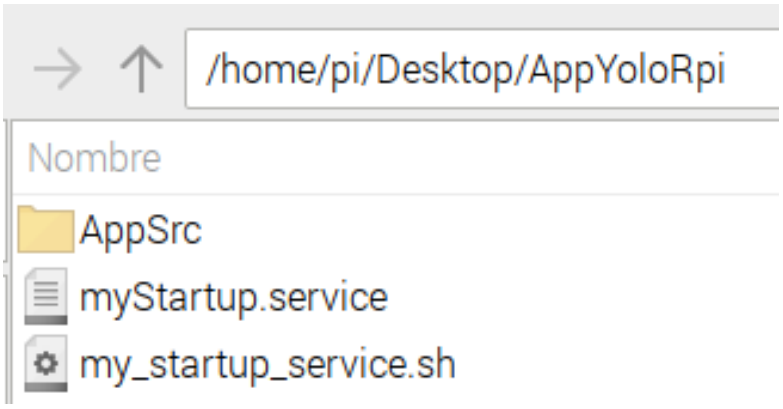
2. Movemos el archivo descargado al escritorio (/home/pi/Desktop) y lo extraemos.

Figura 31. Tutorial para la utilización del código desarrollado para RPi. Descarga del archivo.



Dentro de esta carpeta podremos encontrar el contenido mostrado en la Figura 32.

Figura 32. Tutorial para la utilización del código desarrollado para RPi. Contenido de "AppYoloRpi".



- AppSrc: que contiene el código fuente de la solución, así como sus respectivos *makefile* y diversos subdirectorios para el funcionamiento de la aplicación.

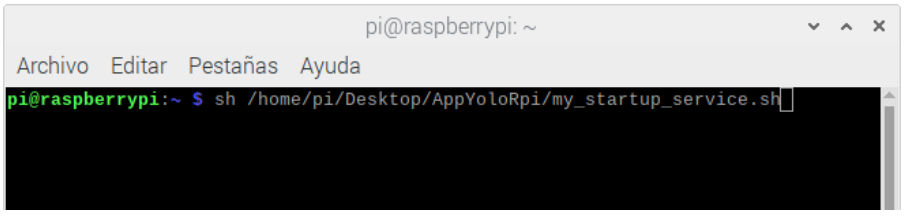
- myStartup.service: se trata de un archivo de texto que contiene el código para lanzar la aplicación como un servicio en el arranque de la RPi.

- My_startup_service: es un *Shell script* que se encarga de poner en funcionamiento los ejecutables y en caso de que estos no hayan sido generados aún, se encarga de compilar el código fuente para crearlos.

3. A continuación, disponemos de dos alternativas a la hora de lanzar la aplicación:

   - La primera y más simple, es la que consiste en lanzar la aplicación desde un terminal de la RPi, de forma similar a la mostrada en la Figura 33.

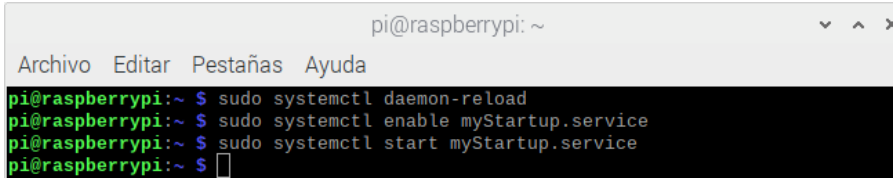
Figura 33. Tutorial para la utilización del código desarrollado para RPi. Lanzamiento del proyecto a través de la línea de comandos.



Esto provocará que se generen los ejecutables y se lance la aplicación (al correrlo por primera vez).

- La segunda alternativa es algo más compleja y pasa por establecer la aplicación como servicio en el arranque de la RPi. Para ello, deberemos copiar el archivo "*myStartup.service*" en la ruta "*/lib/systemd/system*", tras lo cual deberemos ejecutar una serie de comando en el terminal para que corra el servicio cuando inicie el dispositivo, tal y como se muestra en la Figura 34.

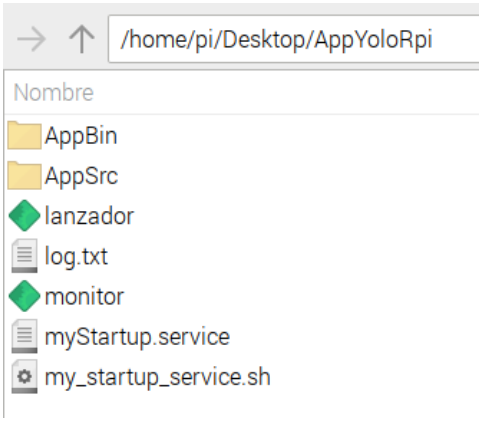
Figura 34. Tutorial para la utilización del código desarrollado para RPi. Establecimiento del lanzamiento de la aplicación como un servicio en arranque.



Finalmente, al reiniciar la RPi podremos ver cómo la aplicación se pone en marcha automáticamente.

4. Por último, sea cual sea la alternativa escogida, al correr por primera vez la aplicación se generarán una serie de carpetas y archivos en la carpeta de la aplicación.

Figura 35. Tutorial para la utilización del código desarrollado para RPi. Archivos y carpetas generados en el primer lanzamiento de la aplicación.



- AppBin: se trata de un directorio creado para generar de forma correcta los ejecutables, originalmente es una copia de AppSrc que es

sometida a una serie de modificaciones (para poder mantener AppSrc intacto en caso de que queramos volver a generar los ejecutables).

- Lanzador/Monitor: son los ejecutables generados por la compilación de los proyectos C/C++.

- Log.txt: solo es generado si escogemos la opción de lanzar la aplicación como un servicio en el arranque de RPi. Su función es la de mostrar *feedback* de la aplicación (ya que es ejecutada sin consola).

13.3 ANEXO 3. TUTORIAL PARA EL USO DE LA INTERFAZ GRÁFICA DE LA APLICACIÓN DESARROLLADA EN JAVA (PC).

En este apartado, mostraremos de forma resumida (ya que posee una interfaz muy sencilla) cómo utilizar la aplicación desarrollada en Java una vez ejecutada.

1. Dispondremos en un primer momento de varios parámetros a configurar para establecer conexión con el dispositivo remoto (RPi).

Figura 36. Tutorial sobre cómo conectar con el equipo remoto desde el PC.

- Address: se trata del campo en el que especificaremos la IP vinculada al equipo remoto.

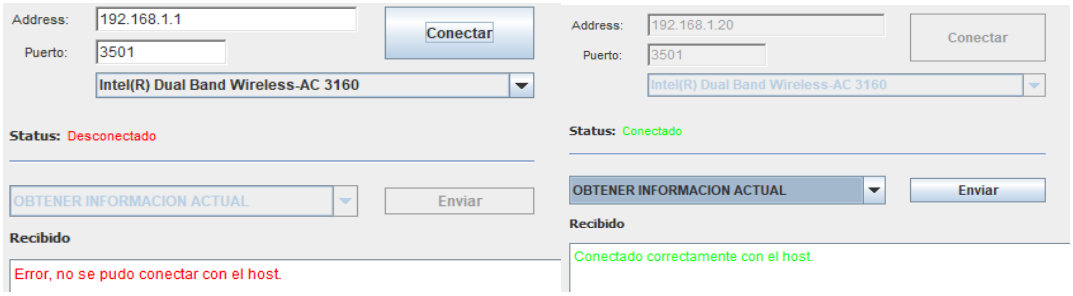
- Puerto: se trata del campo referente al puerto en el que se encuentra escuchando el monitor de comunicaciones del equipo remoto (en principio siempre debería ser el puerto 3501).

- Dropdown: el desplegable (*dropdown*) que se muestra bajo los campos anteriores nos sirve para escoger del PC a través de la

cual queremos establecer la comunicación. Por ejemplo, en el caso de tener el PC conectado a la red vía *WiFi* y *Ethernet* al mismo tiempo, podríamos escoger a través de cuál de los dos queremos establecer la comunicación.

2. Una vez establecidos los parámetros de la conexión, haremos *click* el botón “Conectar”.

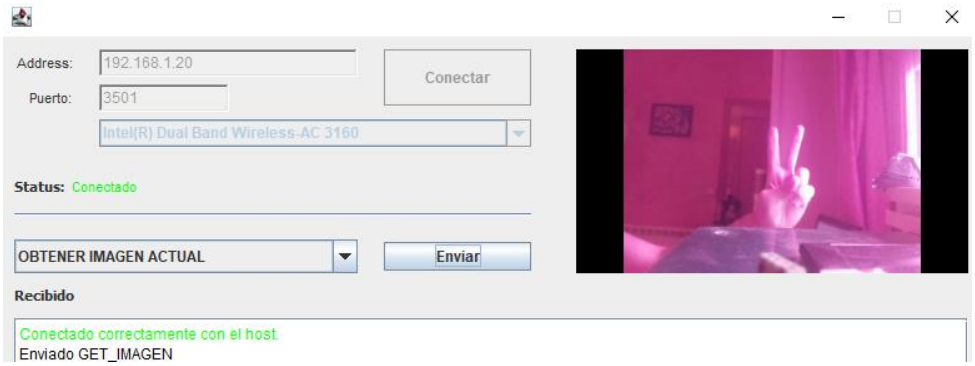
Figura 37. Tutorial, ejemplos sobre cómo responde la interfaz ante las conexiones.



En caso de que no se pueda establecer comunicación, la aplicación arrojará mensajes de error. En caso contrario, la aplicación indicará que la conexión ha sido exitosa.

3. Una vez conectados de forma exitosa podremos hacer uso del desplegable que antes aparecía bloqueado, el cual nos da acceso a los diferentes comandos programados. Tras seleccionar un comando le deberemos clicar al botón “Enviar”, tras lo cual recibiremos una respuesta del equipo remoto.

Figura 38. Tutorial, muestra de un comando en funcionamiento.

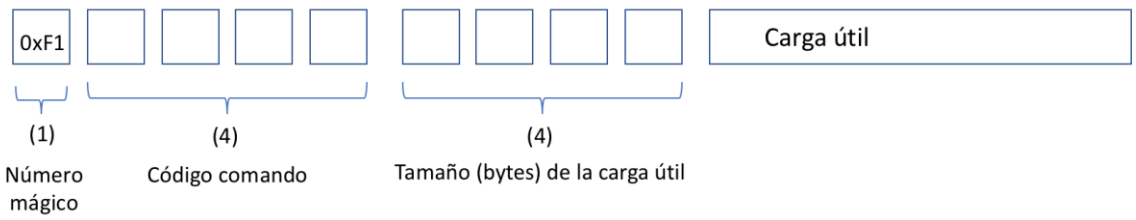


13.4 ANEXO 4. FORMATO DE LOS MENSAJES INTERCAMBIADOS ENTRE EL PC Y EL EQUIPO REMOTO.

En primer lugar, me gustaría destacar que el contenido de este apartado no es de autoría propia, si no que el autor es José Enrique Simó Ten, director de este trabajo.

Los mensajes que usa TCILOCATOR para comunicarse con las aplicaciones cliente siguen un formato uniforme que consiste en (ver Figura 39):
Número mágico: Un byte, siempre 0xF1, se usa para detectar el principio del mensaje en el caso en se pierda sincronismo en la comunicación
Código comando: Cuatro bytes. Es un número entero que identifica la operación a realizar. La lista de comandos soportados es limitada y bien conocida.
Tamaño de la carga útil: Cuatro bytes. Dependiendo del comando que se envíe, el mensaje debe ir acompañado de información adicional (carga útil). Este campo es un número entero que indica el tamaño de la carga útil.

Figura 39. Diagrama que muestra la composición de los mensajes (concretamente centrándose en la cabecera) empleados en la comunicación remota del proyecto.



El campo del mensaje que corresponde a la carga útil también sigue un formato bien definido. La carga útil puede contener alguno o todos de los siguientes campos:
Bloque de bytes en binario: normalmente corresponde a una imagen codificada en formato jpg.
Cadena de caracteres: es un mensaje de tamaño arbitrario.
Vector de números: es una lista de longitud arbitraria de valores numéricos en formato de punto flotante. Cada número en este vector está representado en punto flotante de precisión simple de tamaño cuatro bytes (float).
El tamaño de estos campos (que puede ser cero en cualquiera de ellos) se codifica en la cabecera de la carga útil tal y como se representa en la Figura 40.

Figura 40. Diagrama que muestra la composición de los mensajes (concretamente el payload) empleados en la comunicación remota del proyecto.

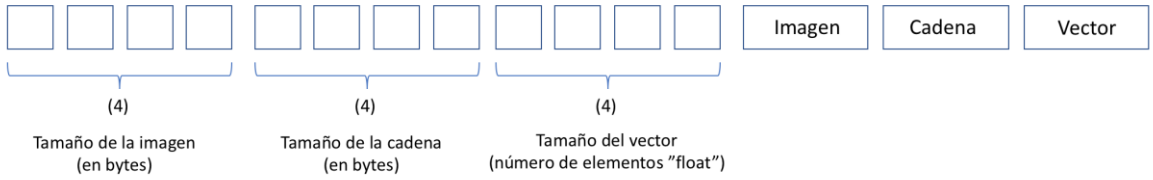
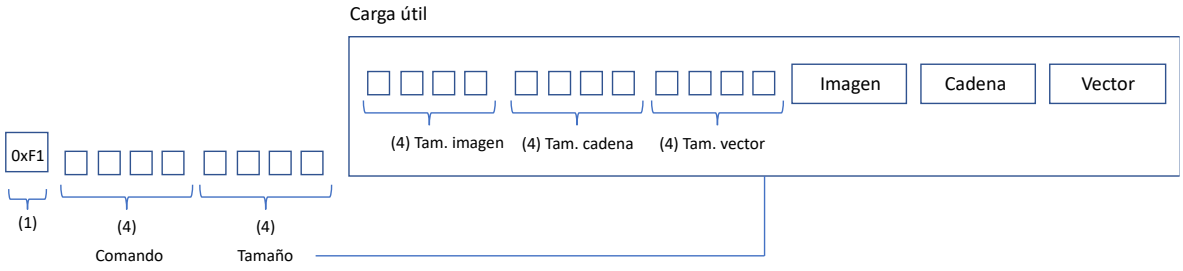


Figura 41. Diagrama que muestra la composición de los mensajes (cabecera y payload a detalle) empleados en la comunicación remota del proyecto.



Nota: El manejo de los mensajes: creación, lectura y envío se implementa en la clase "Message" del módulo "Messaging". Este módulo también contiene la clase "MessagePayload" que implementa las utilidades de manejo del campo de carga útil (composición y extracción de los campos).

13.5 ANEXO 5. DIAGRAMAS DE FLUJO DE LOS COMANDOS CREADOS

Este apartado pretende ser una fuente de información con el objetivo de ampliar los conocimientos acerca del funcionamiento del proyecto.

Figura 42. Diagrama de flujo del comando "*GET_CURRENTINFO*".

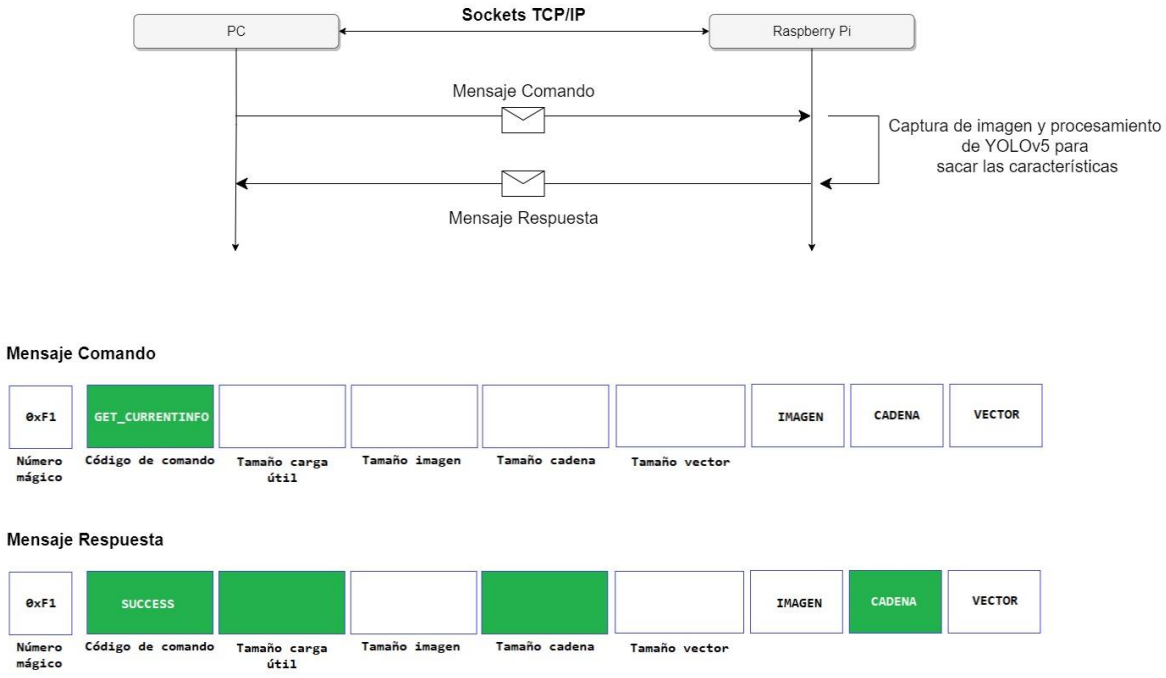


Figura 43. Diagrama de flujo del comando "*GET_IMAGEN*".

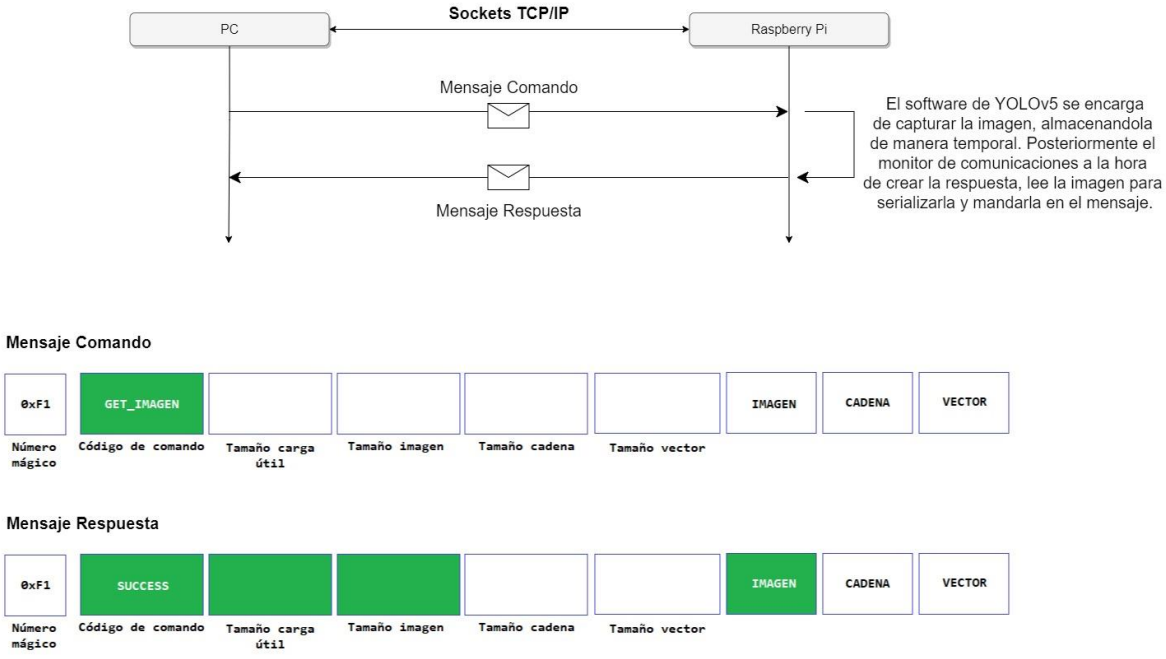


Figura 44. Diagrama de flujo del comando "*SEND_IMAGEN* ".

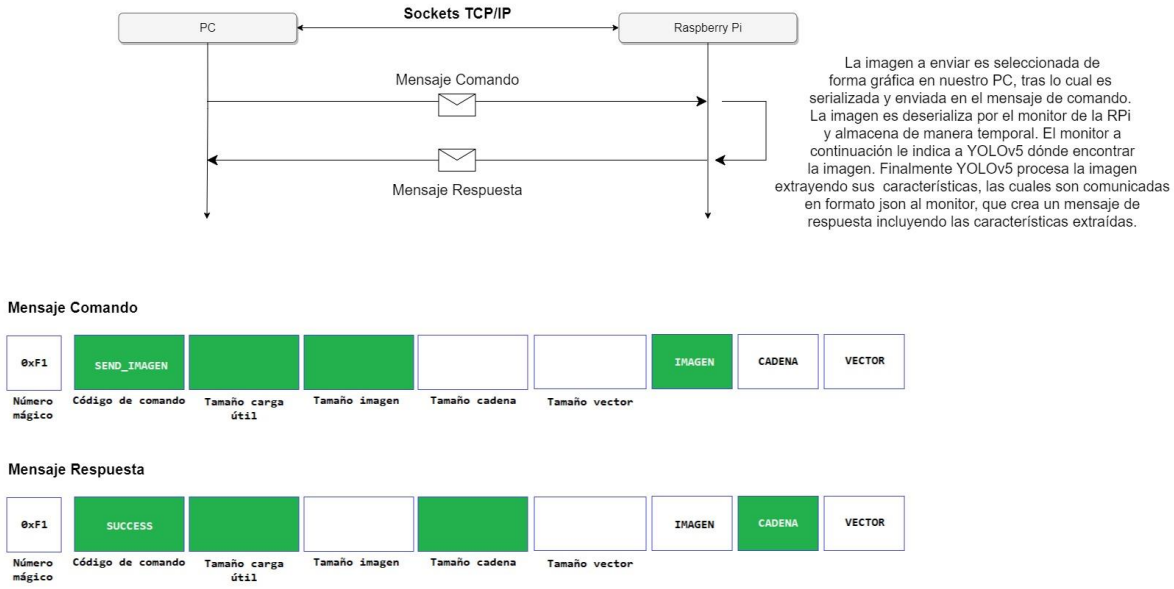


Figura 45. Diagrama de flujo del comando "*SEND_IMAGEN_GETBOX*".

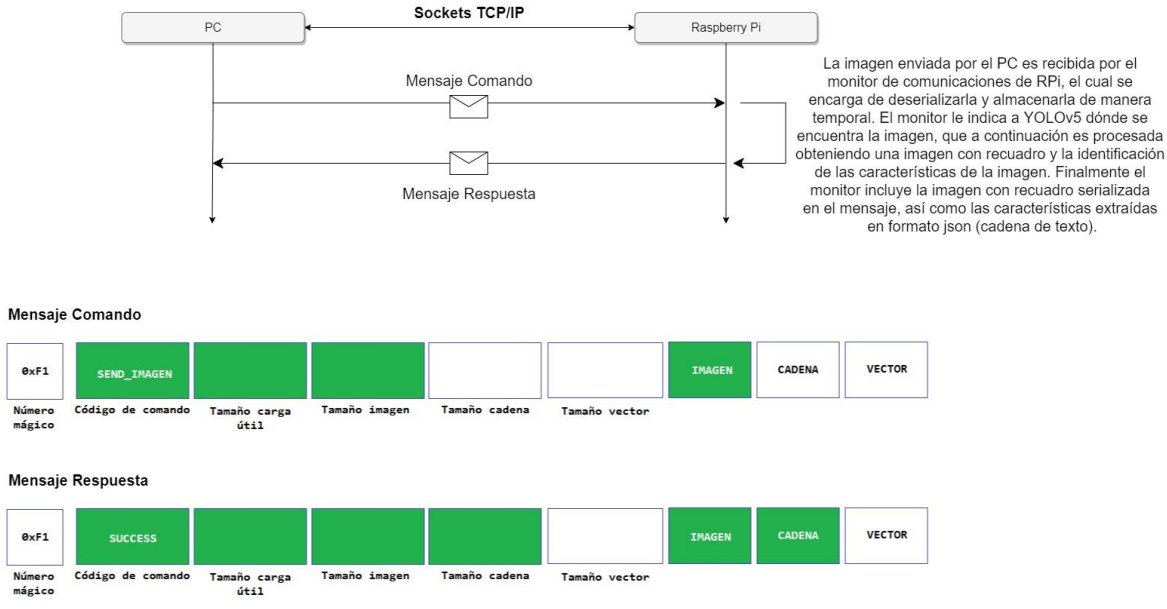


Figura 46. Diagrama de flujo del comando "*GET_STREAMING*".

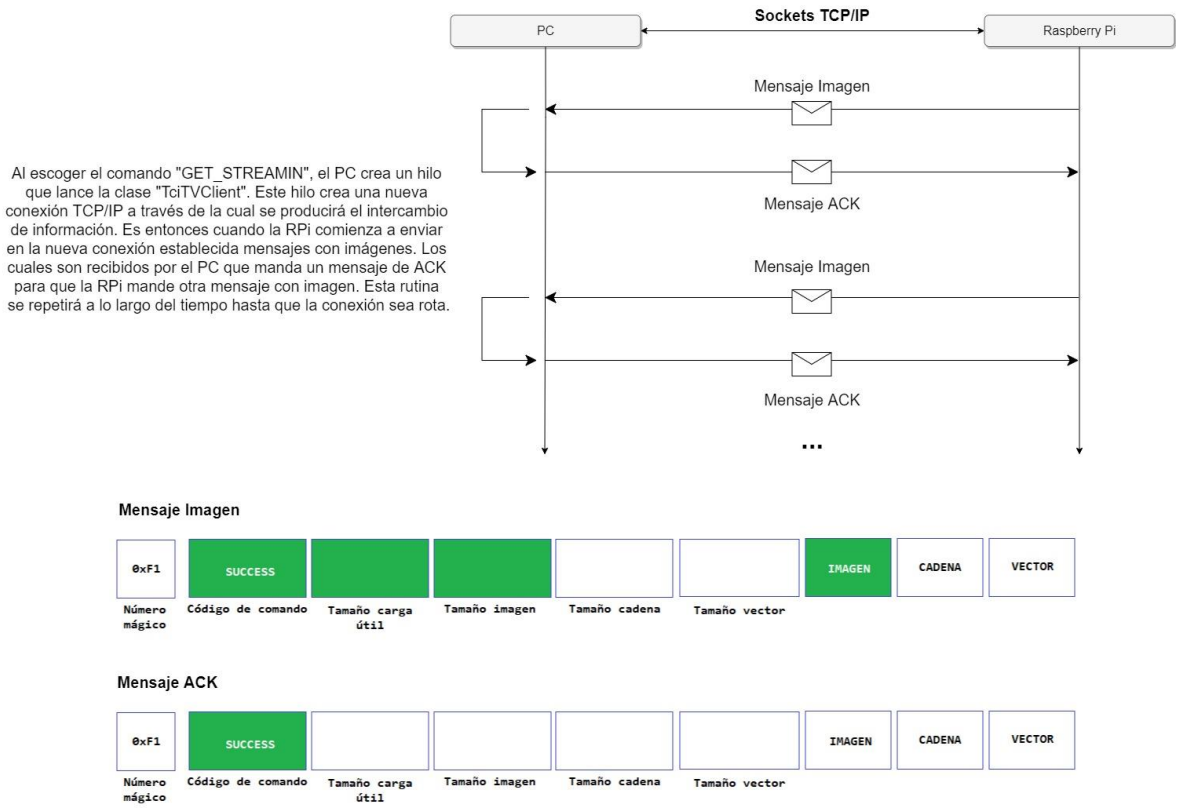
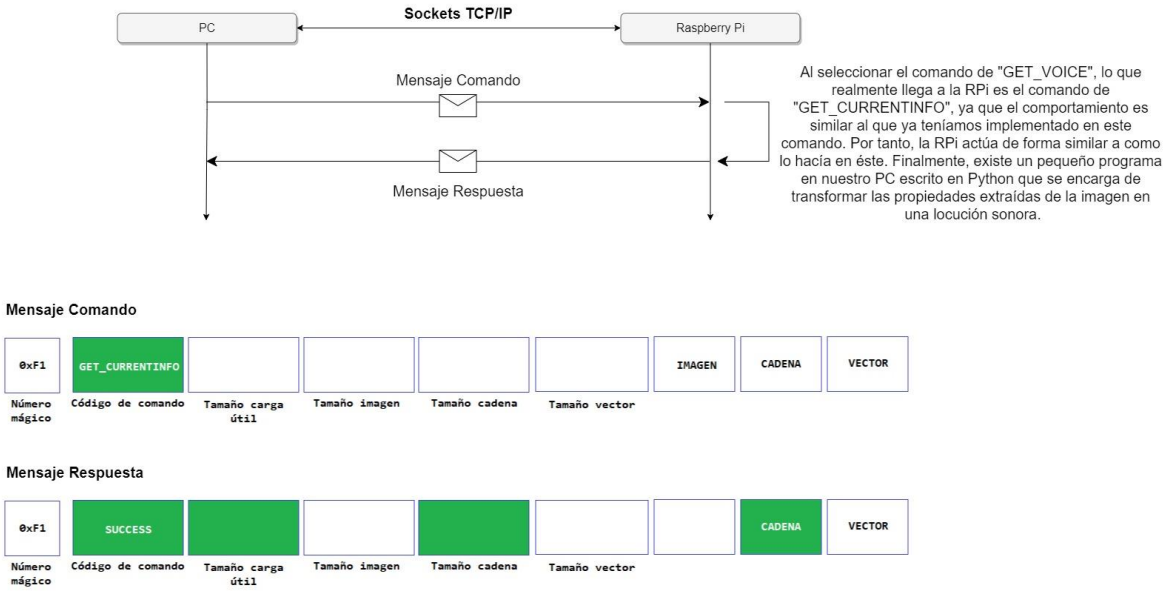


Figura 47. Diagrama de flujo del comando "*GET_VOICE*".



13.6 ANEXO 6. PROYECTOS DESCARGABLES.

Este anexo se limitará a adjuntar los enlaces de descarga de los distintos códigos fuente generados. Estos enlaces ya han sido mencionados en otras secciones del trabajo, no obstante, consideramos que localizarlos en este anexo puede facilitar la tarea de búsqueda de estos.

- Código fuente (incluye ejecutables) de la aplicación del PC: https://drive.google.com/file/d/1Tj713DWh9MYzZ5ZSX0F-gumPRg6YYtYd/view?usp=sharing

- Código fuente de la aplicación de RPi: https://drive.google.com/file/d/1SwHI3uv3g69EyJ0iUJpc2ovJkLlukmv1/view?usp=sharing

- Finalmente, adjuntamos el enlace a una carpeta archivo zip que contiene todo el código fuente del proyecto, organizado por componentes: https://drive.google.com/file/d/1iKPNhj6lduxgvWBbHkVm5AZUoXDlVHvi/view?usp=sharing