“Sistema de localización de Obstáculos en altura para personas con Discapacidad Visual”

‘Trabajo de Fin de Máster’

Máster en Internet de las Cosas

Jesús Campo López

Tutor:

Álvaro García López

RESUMEN

Este proyecto de fin de Máster tiene como finalidad el diseño y construcción de un sistema de detección de obstáculos en altura para personas con discapacidad visual, acompañado de un análisis sobre la actual tecnología aplicada sobre este sector.

Principalmente, el sistema se despliega sobre una Raspberry Pi que va a ser equipada con una cámara, un sensor de ultrasonidos y un pequeño altavoz, con el objetivo de detectar objetos como potenciales obstáculos, calcular distancias sobre dichos obstáculos y notificar vía voz al usuario.

El sistema presenta una arquitectura hardware con los diferentes elementos mencionados, y ejecuta un software que ha sido desarrollado para realizar todos los procesos necesarios para detectar obstáculos y notificar al usuario acerca de los tipos de obstáculos detectados y la distancia hasta los mismos.

Palabras Clave:

Obstáculo, Raspberry Pi, Python, Detección de Objetos, Sensor Ultrasonidos.

ABSTRACT

The main purpose of this master's final project is the design and construction of a system for detecting obstacles at height for people with visual disabilities, accompanied by an analysis of the current technology applied to this sector.

Mainly, this system is deployed on a Raspberry Pi that will be equipped with a camera, an ultrasonic sensor, and a small speaker, with the aim of detecting objects as obstacles, calculating distances over these obstacles and notifying the user through voice.

The system presents a hardware architecture with the different elements mentioned and runs software that has been developed to carry out all the necessary processes to detect obstacles and notify the user about the types of obstacles detected and the distance to them.

Keywords:

Obstacle, Raspberry Pi, Python, Object Detection, Ultrasonic sensor.

ÍNDICE

1. Introducción

1.1. Marco del Proyecto

Este Trabajo de Fin de Máster va a consistir en la elaboración del análisis, diseño y construcción de un sistema de detección de obstáculos en altura para personas con algún tipo de discapacidad visual.

Para lograrlo, este sistema va a estar construido sobre una Raspberry Pi 3B+ [1], junto a una cámara pi bajo la funcionalidad de detección de obstáculos, un sensor de ultrasonidos encargado de medir las distancias a los obstáculos y de un altavoz mediante el que se notificará al usuario al respecto.

1.2. Motivación y Objetivos

Desde hace varias décadas, la tecnología ha ido avanzando a pasos agigantados y de manera exponencial, lo cual ha ido cubriendo y solventando muchos de los problemas que tiempos atrás podían llegar a ser impensables de solucionar. A su vez, han ido apareciendo ayudas y nuevos instrumentos en las vidas de cada ser humano que nunca nos hubiésemos imaginado que necesitaríamos.

En esta ocasión, gracias a los avances en inteligencia artificial, en telecomunicaciones o en robótica, durante los últimos años han aparecido una gran cantidad de sistemas capaces de complementar a aquellas personas que sufren de discapacidades físicas. Como ejemplo se puede tomar la aparición de los brazos o piernas robóticas que sustituyen a las extremidades naturales, pequeños dispositivos electrónicos implantados para recuperar la audición en una persona, o incluso sistemas como el que usaba Stephen Hawking para poder expresarse mediante un simple movimiento de mejilla o gesto facial.

Es por esto por lo que se ha creído conveniente indagar en un problema muy grave como es la discapacidad visual que por desgracia sufren hoy muchas personas de todos los lugares del mundo, ya sea parcial o total.

Siempre se ha asumido que las personas ciegas de alguna manera pueden valerse por sí mismas mediante el uso de bastones o perros guía, pero ciertamente, si uno se para a pensarlo, con estas herramientas una persona ciega puede ser capaz de

detectar obstáculos que nacen del suelo, o incluso escalones, caídas o desniveles. El problema viene cuando aparecen objetos a una cierta altura, como la del pecho o la de los ojos, donde pese a que se lleve un bastón o un perro guía, va a ser inevitable de esquivar en muchas ocasiones, como puede ser una señal de tráfico que sobresale, un cartel de publicidad, una parada de metro con un techo más bajo o cualquier otro elemento que pueda aparecer.

Por tanto, tal y como se ha comentado, el principal objetivo de la elaboración de este sistema es el de solventar el problema que padecen las personas con discapacidad visual a la hora de encontrarse con obstáculos situados a cierta altura, ya que les pueden causar todo tipo de accidentes, golpes o lesiones.

Dentro del diseño e implementación del proyecto, uno de los objetivos que opaca una parte importante del mismo es el de aprovechar y optimizar modelos de detección de objetos ya creados, para adaptarlos y ser capaces de localizar potenciales obstáculos a través de una cámara. De esta manera, el usuario será capaz de tener una noción del tipo de obstáculo al que se enfrenta.

A su vez, se puede considerar como una meta principal el cálculo exacto y en tiempo real de la distancia más próxima al obstáculo más cercano, para lo que se hará uso del correspondiente sensor de ultrasonidos.

Finalmente, se busca que el sistema se pueda acoplar de una manera cómoda y llevadera, de forma que el usuario pueda utilizarlo con total normalidad e incorporarlo a su vida cotidiana como un complemento más de ayuda, donde se pueda instaurar en el interior de un gorro o unas gafas.

1.3. Descripción del documento

En el segundo capítulo de esta memoria se hace un estudio de los elementos y las tecnologías implementadas en el desarrollo del proyecto, como las placas de desarrollo, las tecnologías de detección de objetos o el Machine Learning, y se aporta un contexto actual sobre cómo funcionan, que aplicaciones tienen y qué son exactamente.

A continuación, el documento presenta un tercer capítulo en el que se explica todo el hardware que contiene el sistema, desglosado por cada elemento y detallando como

se comunican unos elementos con otros. A su vez se aporta un esquema general de cómo se conecta cada uno de los elementos, para poder observar la arquitectura general del sistema.

Con el hardware detallado, en el cuarto capítulo se pasa a explicar cómo se ha desarrollado el software del proyecto, incluyendo las funcionalidades que cubre el mismo, las librerías utilizadas, la preparación necesaria, las funciones implementadas y el despliegue del programa principal, donde además, se muestra un diagrama de flujo que muestra las comunicaciones existentes entre los diferentes módulos del software.

En el siguiente capítulo se hallan las diferentes pruebas y demostraciones del funcionamiento y rendimiento de las diferentes partes que componen el sistema, como el modelo de detección, el tiempo de procesado o el tiempo de reproducción del audio. A su vez, se explica como configurar y preparar la Raspberry Pi para que el sistema pueda desplegarse en exteriores.

Finalmente se presentan una serie de propuestas a futuro para mejorar e implementar nuevas funcionalidades al sistema, terminando con unas conclusiones sobre cómo ha sido desarrollar este proyecto, sobre los resultados obtenidos y los conocimientos adquiridos.

2. Estado del Arte

2.1. Tecnologías existentes

Existen diversas tecnologías [2] para la detección de todo tipo de objetos u obstáculos, las cuales desempeñan un papel muy importante en campos donde se busca que los sistemas sean capaces de entender el entorno que les rodea y poder tomar decisiones al respecto. Algún ejemplo puede ser el sistema que se plantea en este trabajo, en el que se busca identificar obstáculos para evitar colisiones, o podría ser un sonar cuyo objetivo es buscar basura marítima en el fondo del mar mediante la detección de objetos.

Para poder realizar este tipo de funcionalidades, durante las últimas décadas han ido apareciendo diferentes tecnologías que se han integrado en multitud de escenarios y que han ido evolucionando, donde se pueden destacar algunas como las siguientes:

- **Radar:** Se trata de un dispositivo que sirve para detectar y localizar objetos en su entorno mediante el uso de ondas electromagnéticas, normalmente ondas de radio o microondas. Comúnmente se conoce su funcionalidad de determinar la velocidad de los objetos, como en el caso de los coches en las carreteras, pero también se puede utilizar para determinar distancias o características físicas de los objetos, como su tamaño y forma, lo cual resulta idóneo para sistemas como el que se trata en este proyecto [3].

Figura 1. *Radar de detección de obstáculos para drones DJI M300 RTK.*

- **Visión Artificial:** Los sistemas de detección de obstáculos u objetos mediante visión artificial, se basan en incorporar consigo una o varias cámaras que de manera fluida van extrayendo fotogramas del entorno que les rodea, las cuales son procesadas por determinados algoritmos encargados de identificar características de este, como puede ser el tipo de objeto que se observa, su tamaño, la distancia entre un objeto y la cámara, etc.… Un ejemplo de uso puede ser la visión artificial de un coche autónomo, cuya funcionalidad es la de evitar colisiones con cualquier tipo de obstáculo, o la visión artificial de un propio dron, para detectar cualquier anomalía en el entorno [4].

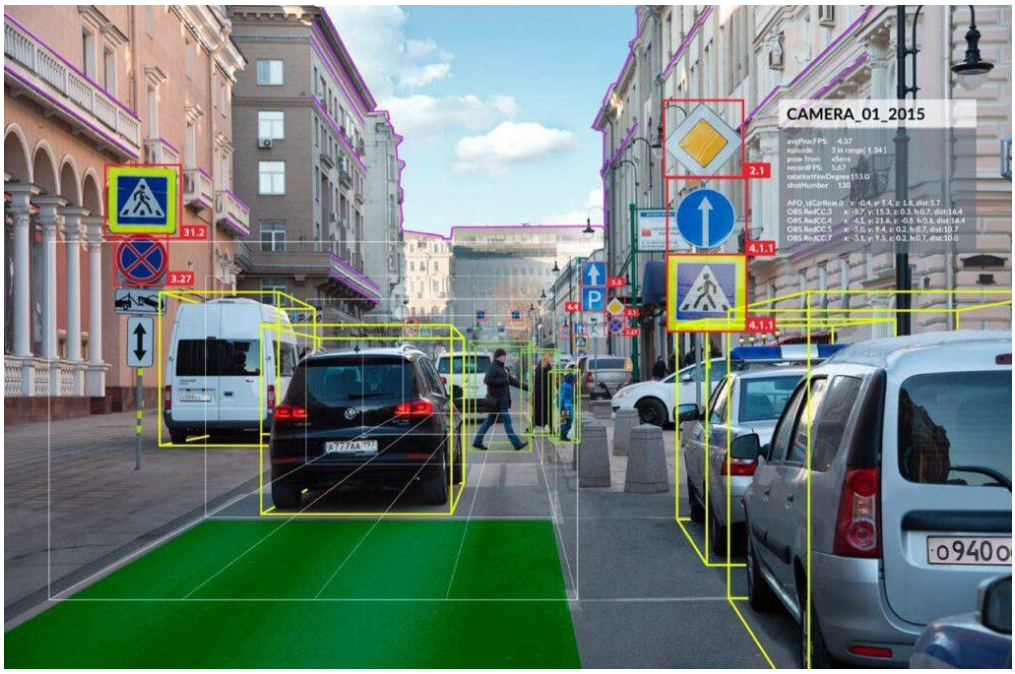

Figura 2. *Detección de obstáculos mediante visión artificial en un coche inteligente.*

- **LIDAR (Light Detection and Ranging):** Esta tecnología de detección remota hace uso de pulsos de luz láser para poder visualizar su entorno y generar mapas tridimensionales del mismo [5]. Son muy comunes en los vehículos autónomos con el objetivo de detectar todo tipo de anomalías u obstáculos en el entorno y poder reaccionar ante las mismas, así como en aviones o drones que desean escanear el entorno que sobrevuelan.

  Básicamente, su funcionamiento comienza mediante la generación de pulsos cortos de luz láser, la cual cuando impacta contra cualquier tipo de objeto, parte de la luz que rebota se refleja hacia el LIDAR. De esta forma, teniendo en

cuenta el tiempo que tarda la luz en viajar ida y vuelta, se puede determinar la distancia hacia dicho punto del objeto, lo cual si se combina con medidas desde diferentes ángulos y hacia diferentes puntos de un objeto, se pueden generar mapas tridimensionales detallados del entorno que abarca el propio LIDAR.

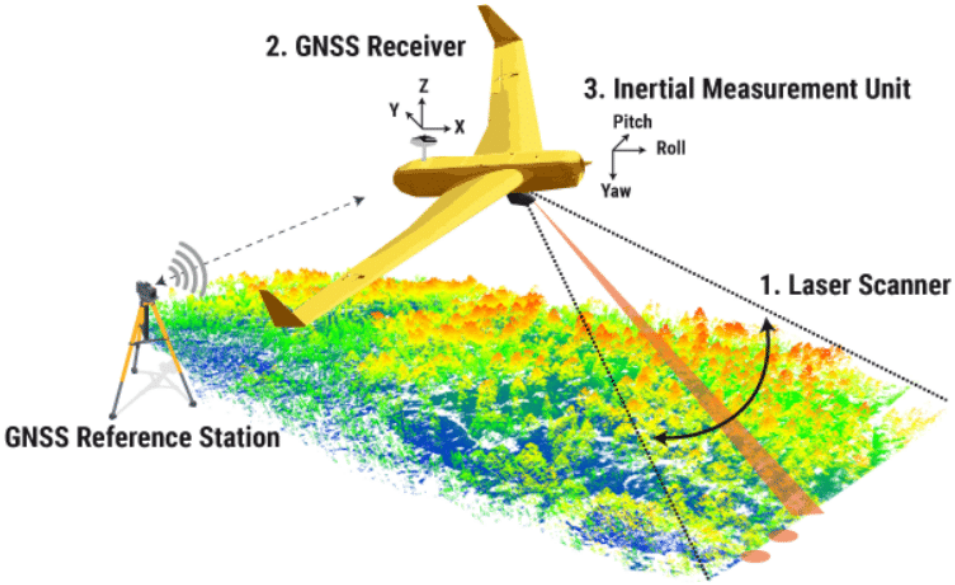

Figura 3. *Esquema de funcionamiento de un LIDAR en una avioneta.*

- **Sensores Infrarrojos:** Como su propio nombre indica, detectan la presencia de los objetos mediante la reflexión que se produce en la luz y gracias a su capacidad para detectar la radiación térmica emitida por los objetos la cual convierten en señales eléctricas.

  Su modo de funcionamiento se basa en la radiación infrarroja que emiten los objetos por naturaleza en forma de calor. Estos sensores están compuestos por un emisor que emite una pequeña cantidad de radiación infrarroja y un receptor que detecta la radiación infrarroja emitida por los objetos. De esta forma, si existe algún objeto dentro del rango infrarrojo del sensor, este reflejará la radiación emitida por el propio sensor, la cual llegará con menor intensidad al receptor. Esta radiación se transforma en una señal eléctrica, la cual en función de su intensidad permitirá estimar la distancia hasta dicho objeto y con varias iteraciones se podría crear un esquema de la forma y distancia hasta el mismo.

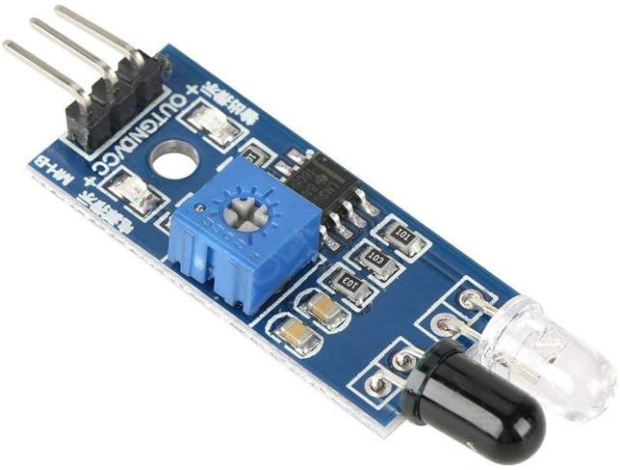

Figura 4. *Módulo de Sensor Infrarrojo [6].*

2.2. Machine Learning

Este término hace referencia a una de las ramas más interesantes y de mayor relevancia de la inteligencia artificial y de la informática, la cual se basa en el uso de datos y algoritmos que los procesan con la intención de asimilar la forma en la que aprende el cerebro humano. En lugar de seguir instrucciones estáticas, los algoritmos de Machine Learning van evolucionando su rendimiento y la calidad de sus resultados a medida que se les proporcionan más datos y que los procesan más veces. Esta capacidad ha llevado a numerosos avances importantes en una gran variedad de campos como la informática, las telecomunicaciones, la medicina, la enseñanza…

Por lo tanto, se puede resumir en que el Machine Learning [7] se focaliza en desarrollar algoritmos y modelos que en base a los datos que reciben, son capaces de aprender a proporcionar respuestas 'inteligentes' sobre tareas específicas. Generalmente, estos modelos o algoritmos se basan en estas etapas:

- **Extracción de datos:** Se recopila el conjunto de datos que va a ser utilizado el cual contiene ejemplos asociados a respuestas, lo cual va a permitir al algoritmo saber qué respuesta se asocia a cada ejemplo. Es decir, si el conjunto de datos son imágenes de rostros y se quiere crear un modelo capaz de distinguir entre rostros tristes o contentos, a cada imagen del conjunto de datos se le tiene que asociar si el rostro esta triste o contento, para que el algoritmo pueda diferenciarlos.

- **Preprocesamiento:** A veces se requiere esta fase para transformar el conjunto de datos iniciales para que todos tengan el mismo formato, para eliminar datos irrelevantes, etc.…
- **Entrenamiento:** Aquí en primer lugar se divide el conjunto de datos total en un conjunto de datos de entrenamiento y otro de validación. Después, se selecciona un algoritmo adecuado para resolver el problema que se enfrenta, y se entrena pasando a dicho algoritmo cada uno de los datos de entrenamiento, atendiendo a diversos parámetros como puede ser la tasa de aprendizaje, el número de épocas (iteraciones realizadas sobre el conjunto de datos), el tamaño del lote, etc.… De esta forma, el algoritmo ira asociando las características comunes que tienen los datos asociados a la misma respuesta. Tomando el ejemplo anterior, el algoritmo aprendería los patrones que siguen los rostros contentos y por otro lado los rostros tristes.
- **Validación:** Se prueba que el algoritmo proporciona respuestas coherentes primero pasándole como datos de entrada los datos de entrenamiento y luego los de validación, es decir, datos que no han sido utilizados para el entrenamiento. De esta forma se estudia la tasa de acierto y de error sobre ambos conjuntos de datos, y se puede ver si ha existido sobre-entrenamiento, si el entrenamiento esta ajustado, si faltan iteraciones, etcétera.

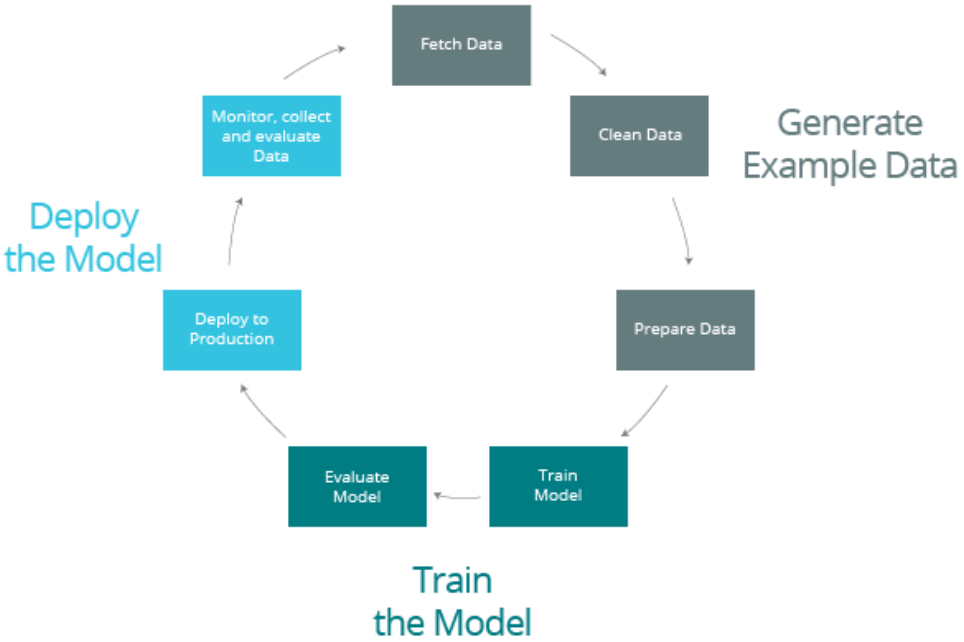

Figura 5. *Ciclo de vida de un modelo de Machine Learning.*

2.3. Redes Neuronales y Convolucionales

Una red neuronal consiste en un tipo de proceso de Machine Learning, conocido como Deep Learning o Aprendizaje Profundo, el cual intenta de alguna forma emular el comportamiento de las redes neuronales [8] del cerebro humano, caracterizado por el aprendizaje a través de la experiencia y el conocimiento a partir de conjuntos de datos. Para ello, está compuesta de unidades de procesamiento interconectadas entre sí, las cuales se denominan neuronas o nodos y que se organizan en capas. En el anterior apartado se comentó acerca de los diversos pasos que se dan a la hora de crear un modelo de Machine Learning. En esta ocasión, una red neuronal se situaría en la etapa de entrenamiento, recibiendo el conjunto de datos inicial y proporcionando una salida en función del problema planteado.

Existen tres tipos de capas dentro de una red neuronal. La **capa de entrada** está formada por uno o varias neuronas que reciben los datos iniciales los cuales trasladan a la capa siguiente, y donde cada una de las neuronas de esta capa está asociada a una dimensión o característica de los datos iniciales. Las **capas ocultas** de una red neuronal son las intermedias entre la capa de entrada y la de salida, cuyo número pueden ser las que el creador decida y que se encargan de procesar los datos recibidos de los datos de entrada aplicando transformaciones no lineales en función de pesos y funciones de activación. Finalmente, la **capa de salida** genera la salida final del modelo, la cual en función del problema planteado, puede contener una o varias neuronas.

Cada neurona tiene dos funcionalidades principales. La primera es que cada entrada que recibe, la multiplica por el peso asociado a dicha conexión y posteriormente suma todas las multiplicaciones realizadas para obtener una suma ponderada. En segundo lugar está la función de activación, la cual toma la suma ponderada obtenida y determina si la neurona será activada o no y cuál será su salida en tal caso.

Por tanto, para construir una red neuronal se debe determinar el número de capas ocultas que existirán, las funciones de activación de cada una de ellas, el número de neuronas de cada capa, etc.…

En concreto, la mayor parte de los modelos de detección de objetos se basan **en redes neuronales convolucionales** [9], como la que va a ser utilizada para este proyecto. Este tipo de redes neuronales se distinguen por su rendimiento superior en cuanto a conjuntos de datos de imagen, voz o señales de audio. Están compuestas de capas **convolucionales**, que pueden ir seguidas de otras capas convolucionales o de

capas de **agrupación**, pero donde la última capa siempre es una capa **totalmente conectada**. En las capas convolucionales se realizan la mayoría de los cálculos y procesamientos de los datos, en las capas de agrupación se reduce la dimensión del conjunto de datos mediante la reducción del número de parámetros de la entrada, mientras que la capa totalmente conectada es la encargada de realizar la clasificación basándose en las características extraídas de las capas anteriores. Sus campos de aplicación abarcan la clasificación de imágenes, procesamiento del lenguaje natural, segmentación semántica o la detección de objetos.

2.4. Modelos de Detección de Objetos

Es la técnica [10] cuyo fin es detectar y diferenciar patrones específicos de una clase de objetos dentro un conjunto finito de imágenes, como puede ser un árbol, una persona, un perro o un coche. Haciendo uso de las redes neuronales convolucionales (CNN) explicadas previamente, se pueden crear este tipo de modelos, donde a diferencia de los modelos de clasificación, en esta ocasión se pueden detectar y clasificar más de un objeto en cada imagen. Traducido a la aplicación de este proyecto, se pueden detectar más de un obstáculo en el camino del viandante, identificando sus coordenadas, el tipo de obstáculo y la tasa de confianza entre 0 y 1 de que el tipo de obstáculo asociado se corresponda al verdadero tipo al que corresponde.

Existen varias técnicas para afrontar el problema de la detección de objetos, destacando la **ventana deslizante,** que consiste en deslizar un modelo clasificador a través de la imagen, el método de **dos etapas**, donde la primera etapa delimita zonas donde pudieran ubicarse objetos y la segunda etapa clasifica los objetos según su clase en cada una de las zonas delimitadas, y finalmente, el método de **una etapa**, en donde tan sólo existe una red CNN encargada de realizar las dos funciones del método anterior, donde se obtiene una mayor rapidez pero una menor precisión.

En los últimos 10 años, han ido apareciendo y evolucionando varios modelos de detección de objetos, donde cabe destacar el más conocido y utilizado por todos cuyo nombre es **YOLO** [11]. Entre las grandes ventajas que presenta frente al resto de los modelos de detección, se puede observar una eficiencia en tiempo real superior, es capaz de detectar múltiples objetos de múltiples clases en una imagen sin tener que procesarla varias veces, es muy robusta en términos de detección de objetos en diferentes escalas y posiciones y a su vez presenta una fácil implementación.

Como alternativas, se puede encontrar Single Shot Detector (SSD), RetinaNet, Spinet (Google) o DETR (Facebook) [12].

Pese a que el modelo YOLO es el más conocido y utilizado, sus archivos de configuración y pesos necesarios para su implementación resultan demasiado complejos y pesados como para ser ejecutados sobre la Raspberry Pi, lo cual produce bastantes fallos y retardos en cuanto al procesamiento.

Teniendo en cuenta las limitaciones que supone utilizar una Raspberry Pi, hay que utilizar un modelo de detección de objetos que sea lo bastante ligero como para poder ser ejecutado sobre la propia placa de desarrollo y que a su vez sea lo suficientemente funcional como para poder detectar una amplia gama de objetos.

Para ello, se ha optado en este proyecto por hacer uso del modelo 'MobileNet-SSD' [13], el cual ha sido diseñado para ser rápido y eficiente en términos computacionales, siendo mucho más ligero en comparación con modelos más grandes y complejos como YOLO. Esto hace que este modelo sea ideal para ser utilizado sobre dispositivos como la Raspberry Pi los cuales tienen recursos limitados.

Este modelo puede ser entrenado por un conjunto de datos personalizados para poder adaptarse a la detección de objetos específicos o en situaciones muy concretas. Al mismo tiempo también existen modelos 'MobileNet-SSD' ya previamente entrenados con diferentes conjuntos de datos los cuales atienden a labores específicas y que pueden ser utilizados por cualquier persona, tal y como va a ser el caso de este proyecto, donde se va a utilizar un modelo' MobileNet-SSD' pre-entrenado.

2.5. Threading

En programación informática, la técnica del '***threading'*** [14], también conocida como 'paralelización', permite que una aplicación sea capaz de ejecutar simultáneamente varias operaciones o tareas en un mismo proceso, es decir, que un programa sea capaz de dividirse en varias tareas y que estas tareas sean ejecutadas paralelamente. Para ello, debe existir un flujo de ejecución para cada una de las tareas que se ejecutan en paralelo, los cuales se van a denominar hilos o '**threads'**.

Esta técnica se utiliza en programas y aplicaciones de todo tipo, como la computación científica o en inteligencia artificial, ya que ofrece posibilidades como realizar varias descargas en paralelo, realizar cálculos complejos rápidamente, lanzar diversas operaciones de búsqueda al mismo tiempo, etc.…

Así mismo, esta técnica presenta una gran variedad de ventajas, donde se pueden destacar las siguientes:

- **Mejora de Rendimiento.**
- **Comunicación entre módulos.**
- **Escalabilidad.**
- **Reducción de Tiempos de Espera.**
- **Eficiencia de Recursos.**
- **Tiempo Real.**

Dentro de la paralelización o '*threading'*, existen diversos elementos que se utilizan para establecer una buena sincronización entre los hilos y para compartir información entre ellos. Entre estos elementos se van a destacar los siguientes:

- **Mutex**: Es un mecanismo de sincronización el cual sirve para evitar que varios hilos accedan de manera simultánea a una región crítica del código, por lo que de alguna manera hacen que solo un hilo sea capaz de tener acceso de manera simultánea sobre una región específica del código.

- **Semáforo**: Son mecanismos de sincronización que se utilizan para controlar el acceso de los hilos o procesos sobre los recursos compartidos, de manera que hacen que se evite que más de un hilo acceda al mismo recurso compartido de manera simultánea, lo cual produciría errores en la aplicación.

- **Cola**: Almacenan una lista de datos u objetos que se van añadiendo y quitando, y sirven para comunicarse entre hilos, de forma que desde un hilo se puedan añadir elementos sobre una cola y desde otro hilo se vayan leyendo dichos elementos.

- **Evento**: Sirven para notificarse entre hilos de algún suceso en específico. Por ejemplo, puede que un hilo esté esperando a que otro hilo realice una tarea en específico para continuar con su función. Entonces, para no estar continuamente comprobando si dicho hilo ha finalizado o no su tarea, en el momento en el que este último finaliza su tarea, notifica al primer hilo mediante un evento para que pueda continuar con su función.

En este proyecto se va a utilizar esta técnica para realizar de manera paralela varias tareas distintas, las cuales se comunicarán mediante el uso de colas y de eventos, tal y como se desarrollará en el apartado de *'Descripción del Software'*.

2.6. Placas de Desarrollo

Son dispositivos que incorporan un microcontrolador reprogramable que ejecuta instrucciones con un fin específico. Estas placas disponen a su vez de entradas y salidas análogas y digitales que permiten la comunicación con sensores externos.

Dentro del mercado existen diferentes opciones en cuanto a placas de desarrollo, y en este apartado se va a justificar la elección en este caso de la Raspberry Pi para este proyecto frente a otras alternativas como puede ser Arduino u Onion. En un principio para este proyecto, teniendo en cuenta el presupuesto y los objetivos, se han barajado las opciones de uso de Arduino y de Raspberry Pi [15].

La Raspberry Pi se trata de una placa de desarrollo de un tamaño reducido pero que a la vez es lo suficientemente potente como para albergar proyectos de este tipo y que además ha ganado popularidad durante los últimos años entre toda la comunidad de la informática y la tecnología. Respecto a las ventajas del uso de esta placa de desarrollo se pueden destacar las siguientes:

- **Potencia de cómputo:** En comparación con soluciones como Arduino u Onion, Raspberry proporciona un procesador más potente y una memoria RAM superior, lo cual hace que se puedan ejecutar programas y aplicaciones más complejas. Arduino presenta una potencia de cómputo y una memoria algo más limitada en comparación con Raspberry, lo cual disminuye las posibilidades de desarrollo en cuanto a complejidad.

- **Conectividad:** La Raspberry cuenta con puertos de todo tipo, Ethernet, USB, Wifi, salida de audio, HDMI, etc.… Esto hace que se faciliten la conexión tanto a la red como a distintos periféricos. Por otro lado, sí es cierto que existen últimos modelos de placas de Arduino que proporcionan conectividad avanzada al igual que lo mencionado en la Raspberry, pero la conectividad nativa de Arduino es más limitada en comparación con Raspberry Pi.

A su vez, también presenta desventajas para tener en cuenta, como las siguientes:

- **Consumo de Energía:** La Raspberry Pi tiende a consumir más energía en comparación con placas de desarrollo Arduino debido sobre todo a su mayor potencia de procesamiento, por lo que si hubiera restricciones de consumo sería un problema de cara al uso de baterías.

- **Coste Relativo:** Generalmente, las placas de la saga Raspberry Pi son más costosas que Arduino, por lo que es importante tener en cuenta respecto al presupuesto.

Teniendo en cuenta tanto ventajas como desventajas y sobre todo el objetivo y la funcionalidad principales de este proyecto, como ya se ha comentado, la elección de la placa de desarrollo se ha decantado por la Raspberry Pi.

Principalmente, la elección de la placa de desarrollo ha estado condicionado por el presupuesto, donde las opciones se barajaban entre un Arduino o una Raspberry Pi, ya que placas como 'Coral' [16], fabricada por Google, tienen precios superiores al presupuesto de este proyecto.

En este caso, se va a establecer una comparación entre el modelo de Raspberry Pi que ha sido utilizado para este proyecto (3B model +) y uno de los modelos de Arduino más completos y utilizados, Arduino Mega 2560 [17].

En la **figura 6** se puede observar una tabla comparativa entre ambas placas atendiendo a características fundamentales de las mismas.

	Raspberry Pi 3B +	**Arduino Mega 2560**
Microprocesador	✔	✘
Conectividad	✔	✘
Tamaño	✘	✔
Memoria	✔	✘
Interfaz	✔	✘
Inputs/Outputs	✘	✔

Figura 6. *Tabla comparativa entre la Raspberry Pi 3B + y la Arduino Mega 2560.*

Como se aprecia en las especificaciones de ambas placas [1] [17], la Raspberry presenta una mayor potencia en cuanto a capacidad de procesamiento gracias a su procesador de cuatro núcleos (quad-core), mientras que la placa Arduino tiene un microcontrolador de 8 bits que limita su capacidad de cálculo. En cuanto a conectividad, la Raspberry puede establecer conexiones Wifi y Bluetooth, mientras que

Arduino no puede, aunque en términos de conexiones físicas de entrada y salida Arduino presenta una mayor cantidad de pines GPIO, lo cual es interesante para proyectos que requieren muchos periféricos pero que en este caso no es de gran interés ya que solo se necesitan unos pocos pines GPIO. Por otro lado, Raspberry presenta una mayor memoria RAM.

Teniendo en cuenta el objetivo del proyecto y las especificaciones de las placas Raspberry y Arduino, como ya se ha comentado, se ha optado por utilizar Raspberry.

3. Descripción del Hardware

En esta sección se van a detallar los componentes fundamentales de este proyecto y con los cuales se constituye la funcionalidad principal del mismo. En primera instancia, en la **figura 7** se muestra un esquema de la arquitectura que constituye el sistema con cada uno de sus componentes y cómo están conectados entre sí.

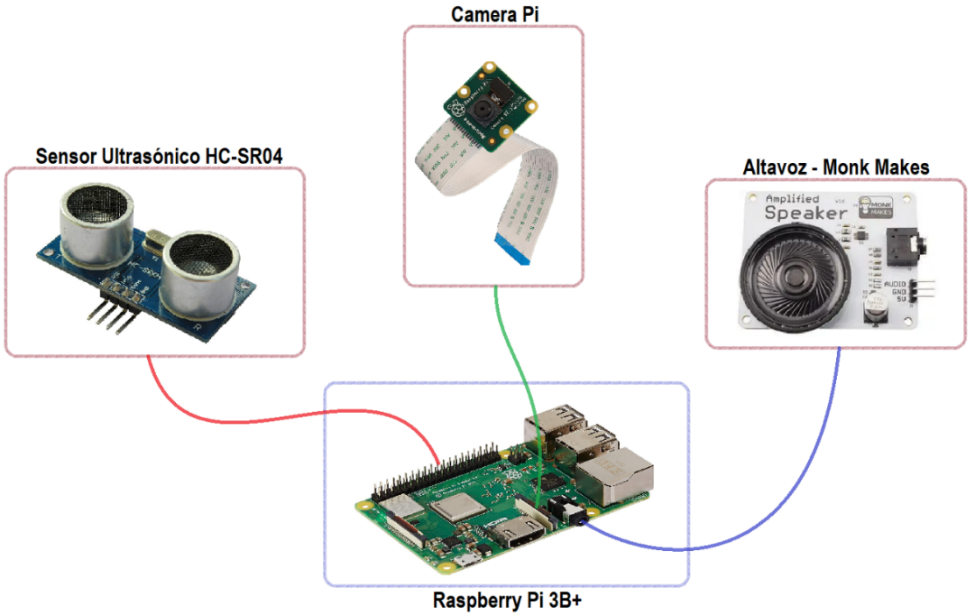

Figura 7. *Arquitectura Hardware del Sistema.*

3.1. Raspberry Pi

La Raspberry Pi, en concreto el modelo 3B+, conforma la plataforma central en la arquitectura del sistema de detección de obstáculos que se va a implementar. Se trata de una computadora que comprende una placa única de bajo costo y tamaño que ofrece una amplia gama de posibilidades en cuanto a conectividad y procesamiento [18]. Gracias a estas características, este dispositivo va a funcionar como cerebro del sistema, coordinando las diferentes funcionalidades que se plantean y se quieren cubrir, y proporcionando una interfaz que permita implementar las mismas.

Sobre esta placa de desarrollo, se van a conectar el resto de los componentes que aparecen en la **figura 7** a través de los diferentes puertos y pines que posee en su cubierta. Las funciones de estos componentes son variadas, sirviendo para recopilar información en forma de imágenes o señales del exterior (sensores) o para informar al

usuario (actuadores). En concreto, se va a contar con dos componentes que actuarán como sensores, que son el módulo de cámara y el sensor de ultrasonidos, mientras que se contará con un solo actuador, en este caso el altavoz, el cual funcionará como el único notificador sobre el usuario y a través del cual el propio usuario obtendrá toda la información necesaria sobre los obstáculos procesada por la propia Raspberry.

Comenzando por la cámara, la cual proporciona un flujo de video en tiempo real, la Raspberry aplica un modelo pre-entrenado de detección de objetos sobre cada uno de los fotogramas del flujo de video recibidos, identificando obstáculos potenciales en la trayectoria del usuario como se explicará más adelante.

Por otra parte, el sensor de ultrasonidos proporciona información a través de los pines de la Raspberry sobre las señales ultrasónicas que emite y que rebotan contra el obstáculo más cercano. De esta manera, la Raspberry se encarga de procesar toda esa información para determinar la distancia más cercana a un obstáculo.

Por último, la Raspberry procesa toda la información recibida por el conjunto de sensores, cámara y sensor de ultrasonidos, tal y como ha sido comentado previamente, y emite periódicamente por el altavoz vía audio la información que el usuario debe de conocer acerca de los obstáculos presentes en el entorno.

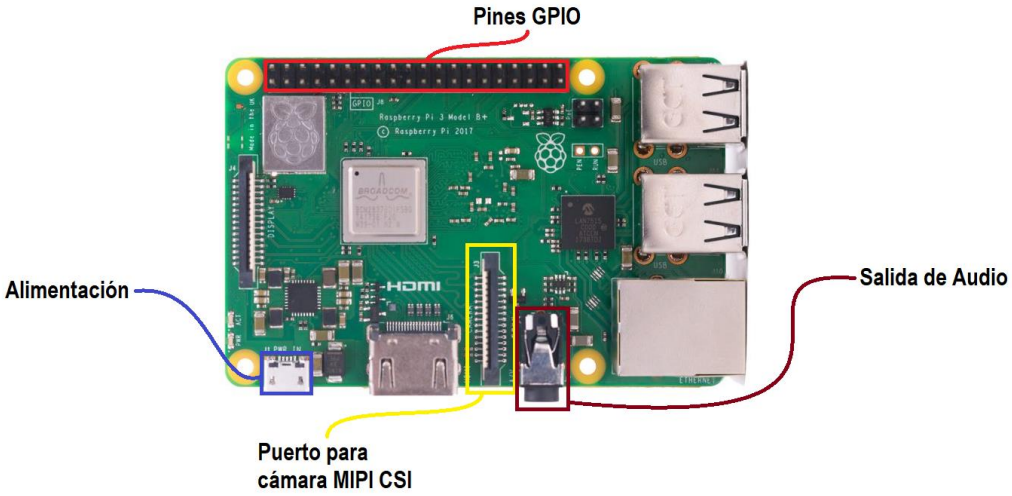

Figura 8. *Componentes HW relevantes de la Raspberry Pi 3B+.*

Como ya ha sido mencionado, la Raspberry Pi 3B+ cuenta con una amplia variedad de pines y puertos que dan lugar a diversas posibles conexiones con diferentes módulos. En concreto, en la **figura 8** aparecen destacados aquellos componentes de la

Raspberry relevantes para este proyecto y los cuales van a ser fundamentales para establecer la conexión con el resto de los componentes HW del sistema.

En color rojo se ubica la sección de pines GPIO (General Purpose Input/Output), los cuales se utilizan para la entrada y salida de señales digitales y analógicas. Entre estas entradas y salidas, se pueden destacar los pines de alimentación, a los cuales se tienen que conectar la mayoría de los módulos para poder funcionar, donde existen pines de 3.3 voltios, de 5 voltios y de tierra (0 voltios). También existen los pines UART para establecer comunicación serial, los pines SPI que permiten la comunicación en serie entre dispositivos, los cuales se utilizan en aplicaciones que requieren una alta velocidad de comunicación o los pines I2C que permiten conectar varios dispositivos en un bus común. A parte de los ya mencionados, el resto de los pines los cuales conforman la mayoría de ellos se tratan de pines GPIO. Estos son los pines más versátiles y utilizados, pudiendo ser configurados como entradas o salidas digitales para interactuar con cualquier tipo de componente externo.

En la parte inferior de la **figura 8** se observa en color azul el puerto de alimentación de la Raspberry y en color marrón la salida de audio por donde se va a establecer la conexión con el altavoz para que el sistema pueda emitir la salida correspondiente. En color amarillo, se puede apreciar el puerto de cámara MIPI CSI, el cual se trata de un conector físico de la propia Raspberry el cual atiende al estándar “Mobile Industry Processor Interface – Camera Serial Interface”. Este protocolo de interfaz está diseñado para transmitir datos de imágenes desde una cámara a un procesador como el que posee la Raspberry. De esta manera, este puerto va a ser utilizado para conectar el módulo de Camera Pi para la captura de imágenes y videos.

Finalmente, también se han utilizado los puertos USB y HDMI correspondientes para la conexión de los periféricos pertinentes que han servido de ayuda para desarrollar el software, tales como el ratón, el teclado y el monitor.

3.2. Camera Pi

Camera Pi o Raspberry Pi Camera Module [19], se trata de una cámara específicamente diseñada para usarse sobre una placa Raspberry. Este dispositivo presenta entre sus características principales, su funcionalidad como sensor de imagen, que a pesar de su pequeño tamaño proporciona una alta calidad. A parte de imágenes fijas, también funciona como capturador de video en diferentes resoluciones y tasas de cuadros.

Como ya se comentó en el apartado de la Raspberry Pi, esta cámara se conecta mediante el puerto MIPI CSI, como se muestra en la **figura 8,** haciendo uso de dicho estándar, lo cual le permite una conexión directa y de alta velocidad con el procesador de imágenes de la Raspberry.

Figura 9. *Conexión de la Camera Pi a la Raspberry Pi 3B+.*

En la **figura 9** se aprecia como se conecta la cámara a través de dicho puerto de la Raspberry Pi.

En cuanto a las características técnicas del modelo de Camera Pi utilizado en este proyecto:

- Sensor de imagen Sony IMX219 de 8 Megapíxeles
- Resolución de imagen fija de 3280 x 2464
- Captura de vídeo a 1080p – 30fps y 720p – 60fps
- Dimensiones: 23.86 x 25 x 9mm
- Cable plano de 15 contactos

3.3. Sensor Ultrasónico HC-SR04

Para el cálculo de distancias se ha utilizado el sensor HC-SR04 [20]. Se trata de un dispositivo que utiliza señales ultrasónicas para medir distancias a objetos cercanos mediante el principio de tiempo de vuelo. Este se obtiene mediante el tiempo que tarda la señal ultrasónica en ir, rebotar y volver desde el sensor al objeto.

Este sensor ultrasónico cuenta fundamentalmente con dos componentes: un transductor ultrasónico emisor y un transductor ultrasónico receptor. A través del emisor se emiten una serie de pulsos ultrasónicos generalmente de alrededor de 40KHz, lo cual conforma ondas de sonido de alta frecuencia a las que los humanos no tienen capacidad de escuchar. Dichas ondas, tras ser emitidas llega un punto en el que se encuentran con un objeto en su trayectoria y rebotan siguiendo la misma trayectoria de regreso, por lo que la onda vuelve hasta el sensor y se recoge a través del receptor, tal y como se puede apreciar en la **figura 10**. Teniendo en cuenta la velocidad del sonido en el aire, la cual es constante siempre y con un valor de 343 m/s, y conociendo el tiempo que transcurre desde que se emite la onda hasta que regresa, se puede calcular la distancia que hay entre el sensor y el objeto contra la que ha rebotado la onda.

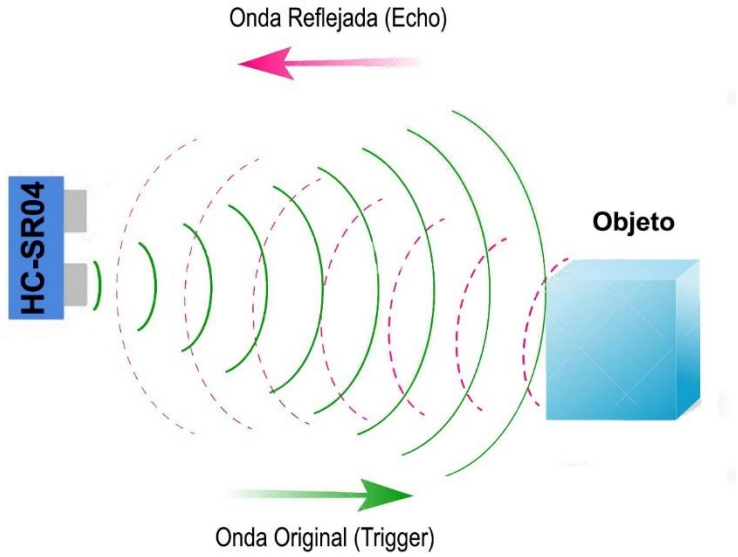

Figura 10. *Esquema del funcionamiento del sensor ultrasónico HC-SR04 [21].*

En cuanto a la conectividad con la placa Raspberry, este sensor dispone de 4 pines que permiten su conexión.

Uno de estos pines corresponde con el de alimentación, denominado VCC, el cual debe de ir conectado a uno de los pines de alimentación de la Raspberry Pi, de 5V. Otro de los pines corresponde con el que se conecta a tierra, GND, el cual debe unirse a uno de los pines destinados a tierra de la Raspberry, estableciendo de esta manera un potencial de referencia común. Finalmente, los otros dos pines corresponden con Trigger (Trig) y Echo, los cuales pueden ir conectados a cualquiera de los pines GPIO de la Raspberry. El primero de ellos se utiliza como pin de activación para emitir un pulso ultrasónico, es decir, cuando este pin recibe un valor 1, el sensor emite un pulso ultrasónico. Por otro lado, el pin Echo es donde se detecta el regreso de la onda ultrasónica, de manera que la duración del pulso sobre este pin, la cual se mide en microsegundos, es proporcional al tiempo que la onda ultrasónica tardó desde su emisión hasta que regresó tras rebotar en un objeto.

En la **figura 11** se puede observar cómo ha sido establecida la conexión entre el sensor y la Raspberry Pi durante el desarrollo de este proyecto. Cabe decir que las conexiones entre pines se realizan con conectores hembra-hembra.

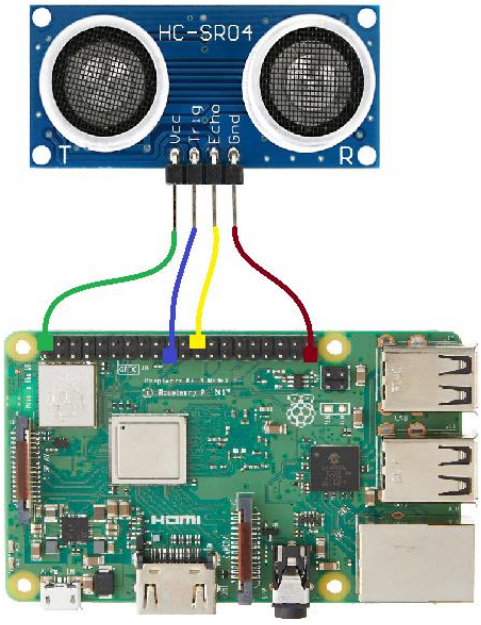

Figura 11. *Conectividad entre el sensor HC-SR04 y la Raspberry Pi 3B+. En verde, los pines de alimentación (VCC). En azul y amarillo, los pines Trig y Echo, respectivamente. En marrón, los pines de tierra (GND).*

3.4. Amplified Speaker Monk Makes

Para la salida de audio, se va a hacer uso de este dispositivo [22], el cual se trata de un pequeño altavoz amplificado que está diseñado específicamente para correr sobre placas de desarrollo como Raspberry o Arduino. Sobre este, se van a emitir los audios correspondientes acerca de la distancia a la que se encuentra el objeto más cercano al sistema la cual se obtiene mediante el sensor de ultrasonidos, y también se emitirá audio sobre el tipo de obstáculos que se pueden encontrar en el camino los cuales se obtienen gracias a aplicar el modelo de detección sobre el video capturado por la Camera Pi.

Respecto a su conexión con la propia placa de Raspberry, este altavoz cuenta con tres pines y con un conector Jack de audio de 3.5mm. De los tres pines, sólo van a ser necesarios el uso de dos de ellos, en concreto el destinado a conectarse a alimentación y el destinado a conectarse a tierra, de manera que deben de conectarse con los pines de la Raspberry dedicados a ello. Por otro lado, como ya se ha comentado, contiene una entrada de audio de 3.5 mm, sobre la cual se va a conectar un conector Jack entre dicha entrada y la misma entrada de audio que contiene la Raspberry Pi al borde de la placa, tal y como se puede ver en la **figura 12.** Con todo conectado, el altavoz estará preparado para poder emitir cualquier salida de audio procesada por la propia Raspberry Pi.

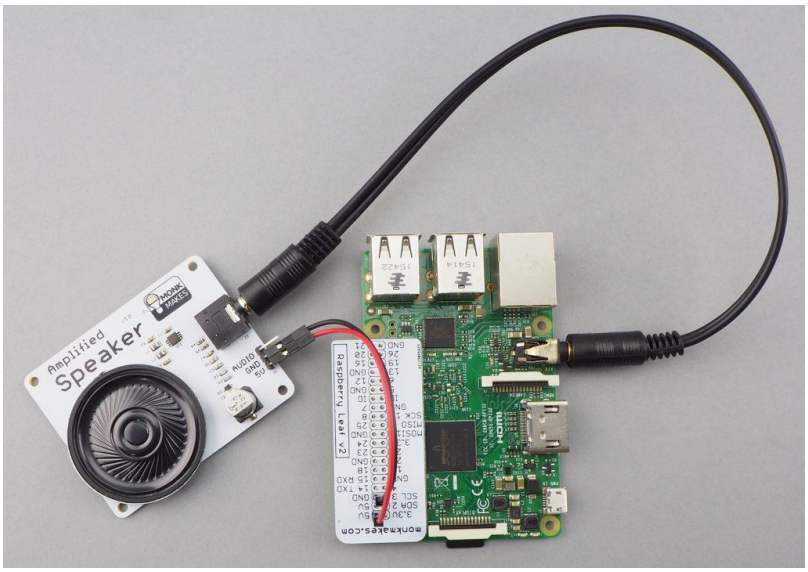

Figura 12. *Conectividad entre el altavoz y la Raspberry Pi 3B+ [22].*

4. Descripción del Software

Tras realizar el diseño de la arquitectura Hardware del sistema, el siguiente paso es el diseño y construcción del Software que va a implementar todas las funcionalidades mencionadas en los objetivos. Para ello será necesario la captura de video a través de la cámara, la detección de obstáculos, la medición de distancias con el sensor HC-SR04 y la salida de audio para informar al usuario. El Software en cuestión ha sido realizado con el lenguaje de programación 'Python'.

En primera instancia se van a presentar de manera resumida y justificada las funcionalidades más generales del Software. Posteriormente, se va a realizar una descripción del propio código, comenzando por las librerías utilizadas, las funciones definidas y el programa principal que recoge todas las funcionalidades y ejecuta el proceso completo.

4.1. Funcionalidades

Este programa debe de abarcar una serie de funcionalidades para cumplir con el objetivo principal del proyecto, el cual es avisar al usuario en caso de que se aproxime a un obstáculo acerca de la distancia hacia el mismo y el tipo de obstáculo que es. Entre estas, se van a destacar las siguientes, de las cuales alguna ya ha sido mencionada:

- **Captura de vídeo en tiempo real**: Mediante la 'Camera Pi', se va a realizar una captura de vídeo constante, de forma que se van obteniendo todos los fotogramas de dicho vídeo en tiempo real.

- **Procesado de fotogramas:** La captura de vídeo va generando fotogramas constantemente, pero en este caso, solo van a ser procesados los fotogramas obtenidos cada 6-7 segundos, ya que no es necesario procesar todos los fotogramas de vídeo obtenidos porque el propio procesado de cada fotograma se demora un tiempo entre 1-2 segundos para detectar los objetos y alrededor de 3-4 segundos en notificar al usuario por voz. A su vez, si se procesaran todos los fotogramas, existiría un gran retraso o 'delay' en el sistema.

- **Uso del modelo de detección de obstáculos ‘MobileNet-SSD’:** Se va a hacer uso de este modelo de detección de obstáculos debido a su ligereza y su marco de clases de objetos capaz de detectar, como ya fue explicado en el apartado de ‘*Modelos de Detección de Objetos’* en el *‘Estado del arte’*.

- **Obtención del área de ubicación en la cámara de cada objeto detectado:** En cada fotograma que se procesa, para cada objeto detectado se va a necesitar obtener el marco delimitador que determina al área que ocupa dicho objeto en la cámara, para posteriormente poder determinar si el objeto es o no un obstáculo.

- **Establecimiento de un marco que determina el área de proyección del sensor de ultrasonidos:** Se va a establecer un marco imaginario en el centro de la proyección de la cámara, el cual determina el área que abarca o hacia donde apunta el sensor de ultrasonidos, para poder examinar a posteriori los potenciales obstáculos.

- **Comprobar con cada objeto si se encuentra dentro del marco de proyección del sensor:** Conociendo el área que abarca el objeto en cada fotograma y el marco imaginario de proyección del sensor, se va a calcular el centro de cada objeto y se determinará si está dentro del marco o no.

- **Determinar el objeto ubicado dentro del marco que ocupe más área:** Si existe más de un objeto dentro del marco de proyección del sensor, se escogerá el que ocupe más área como el obstáculo más próximo al sensor.

- **Medición de distancias en tiempo real con el sensor HC-SR04:** Se harán los cálculos precisos utilizando los datos obtenidos a través del sensor de ultrasonidos HC-SR04 para determinar la distancia hacia el obstáculo más próximo.

- **Notificación de obstáculo:** En caso de que se encuentre un obstáculo a menos de 5 metros, se emitirá por el altavoz un mensaje informando sobre la distancia hacia el mismo y el tipo de obstáculo que es (el cuál se determina como ha sido recién explicado en los puntos anteriores). En caso de que existiera un obstáculo que fuera un tipo de objeto que el modelo no es capaz

de detectar, el mensaje emitido informará sobre la distancia y sobre que se aproxima a un obstáculo, sin determinar el tipo.

4.2. Librerías

A continuación, se detallan las librerías que han sido utilizadas por el programa realizado, explicando de que tratan y para qué han sido empleadas.

En primera instancia y de más importancia, se encuentra la librería '***OpenCV'*** [23]. Se trata de una biblioteca de código abierto en Python enfocada en el procesamiento de imágenes y de vídeo. Esta librería se emplea para una amplia gama de tareas relacionadas con la visión por computadora, como el reconocimiento de patrones, análisis de imágenes médicas, realidad aumentada o, como se va a implementar en este proyecto, la detección de objetos. En este trabajo, esta librería se va a utilizar principalmente para capturar el vídeo a través de la Camera Pi, para crear el modelo de detección de objetos basado en una red neuronal convolucional, aplicarlo sobre las imágenes y para mostrar por pantalla la detección de los objetos con sus clases y su marco delimitador (Esto último se ha utilizado durante el desarrollo, ya que en el sistema final no hace falta mostrar nada por pantalla).

Para el cálculo de las distancias con el sensor de ultrasonidos HC-SR04, se ha necesitado de las librerías ***'GPIO'*** y ***'time'***. La primera de estas se utiliza para configurar los pines de la Raspberry sobre los cuales está conectado el sensor y para obtener a través de ellos los datos referentes a los pulsos ultrasónicos. La librería 'time', se complementa con esta última para realizar el cálculo de la distancia, como se explicará más adelante.

Por otro lado, para generar y emitir el mensaje por el altavoz, se ha hecho uso de las librerías '***gTTS'*** [24] (*Google Text-to-Speech*) y '***playsound'.*** La primera de estas librerías se utiliza para convertir el mensaje que se quiere emitir en voz, es decir, para la síntesis de voz, mientras que la última de estas se utiliza para la reproducción del propio mensaje por el altavoz.

Finalmente, en este programa se hace uso de paralelización, que es una técnica que permite dividir un programa (una única tarea) en varias tareas, y que estas tareas se ejecuten simultáneamente. Para poder implementar la paralelización, se ha hecho uso de las librerías '***threading'*** y '***queue'***. En los siguientes apartados se entrará más en profundidad con esta técnica utilizada.

4.3. Preparación

Previo a implementar cada una de las funciones y el programa principal del software de este proyecto, se deben seguir una serie de pasos con respecto al modelo de detección de objetos.

En el '*Estado del Arte'*, se indicó el funcionamiento básico de las redes neuronales convolucionales y los modelos de detección de objetos. En esta ocasión, no va a ser posible utilizar un modelo propio de detección principalmente por falta de recursos y de tiempo para su preparación y entrenamiento. De esta manera, se va a utilizar un modelo existente y entrenado previamente, el cual es capaz de detectar 91 tipos de objetos diferentes, de entre los cuales muchos de ellos son potenciales obstáculos que uno se puede encontrar, como personas, señales de tráfico, coches o semáforos. Este modelo recibe el nombre de ***'MobileNet-SSD'***, en concreto la versión 'v3'.

Para implementar este modelo y poder aplicarlo sobre las imágenes que van a ser capturadas, se necesita descargar una serie de ficheros fundamentales, los cuales se pueden encontrar en la web oficial de TensorFlow (*https://tfhub.dev*), la cual contiene una gran cantidad de modelos pre-entrenados. El primero de estos ficheros es el de configuración, que en este caso recibe el nombre de ***'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'***, el cual contiene información sobre la arquitectura de la red neuronal que utiliza el modelo y sobre los hiperparámetros que este ha utilizado. El segundo fichero corresponde con el archivo de pesos del modelo, ***'frozen_inference_graph.pb'***, que contiene los pesos entrenados de la red neuronal para realizar la detección de objetos. Finalmente, se necesita también de un fichero que contenga una lista ordenada con los nombres de las clases de objetos que el modelo puede detectar, siguiendo el mismo orden de clases que sigue el fichero de pesos y de configuración. Los nombres de cada clase de objetos han sido traducidos al español y el fichero se llama ***'spanish.names'*.**

4.4. Variables Globales

En esta región del código, se declaran las variables globales que van a ser utilizadas por las distintas funciones del programa, así como objetos accesibles desde cualquier parte de este. Concretamente, en este software se van a declarar las variables y los objetos necesarios para la creación del modelo de detección y para la configuración del sensor de ultrasonidos.

Comenzando por la detección de objetos, se declara una variable global, '*classNames'*, que contendrá una lista con los nombres de todas las clases de objetos que el modelo es capaz de detectar, es decir, con los nombres recogidos en el fichero '*spanish.names'*. A su vez, se crea una instancia del modelo de detección, haciendo uso de la librería OpenCV y de la función *'dnn_DetectionModel()'*, la cual necesita como argumentos de entrada los ficheros de configuración y de pesos mencionados en el apartado anterior para poder crear la instancia del modelo. El modelo se configura de la siguiente manera:

```
net.setInputSize(320,240)      # Dimensiones de la imagen de entrada en
                               la red neuronal
net.setInputScale(1.0/ 127.5)  # Escala de entrada en el modelo
net.setInputMean((127.5, 127.5, 127.5))   # Media de entrada (RGB)
net.setInputSwapRB(True)       # Se intercambian los canales de color de
                               entrada al modelo rojo y azul
```

Cuando se detecta un objeto, el modelo le asocia un parámetro llamado '*tasa de confianza'*, el cual contiene un valor entre 0 y 1 que corresponde con la confianza que tiene el modelo de que el tipo de objeto detectado sea realmente el tipo de objeto que es en la realidad, es decir, cuanto mayor es este valor, mayor es la confianza que tiene el modelo en que ha acertado en su detección. Por lo tanto, es necesario crear una variable global que contenga un umbral de tasa de confianza, de manera que el modelo solo detecte objetos para los cuales tenga una tasa de confianza superior al umbral declarado. En esta ocasión, se ha fijado dicho valor en 0.55, aunque se puede modificar.

Por otro lado, como se comentó en el apartado de las funcionalidades, se establece un marco que delimita el área de proyección del sensor de ultrasonidos. Dicho límite se declara variable global.

```
limit = [180, 135, 280, 210]
```

En otra instancia, se debe de realizar la configuración y la declaración de variables pertinentes para poder obtener información a través de los pines del sensor de ultrasonidos. Se comienza estableciendo el modo de numeración de pines GPIO, en esta ocasión será el modo BCM (Broadcom SOC Chanel), lo cual hace que los números de los pines se refieran a los números de GPIO específicos de la Raspberry Pi, por lo que por ejemplo, el número 8, corresponderá con el pin GPIO8.

```
GPIO.setmode(GPIO.BCM)
```

Posteriormente, se declaran los números de los pines ‘Trigger’ y ‘Echo’. En este caso, como están conectados a los pines GPIO8 y GPIO10 respectivamente, se les asocia dicho número.

```
GPIO_TRIGGER = 8
GPIO_ECHO = 10
```

Finalmente, como fue explicado en el ‘*Estado del Arte’*, el pin ‘Trigger’ es el que determina cuando se lanza un pulso ultrasónico en función de si está activado o no, por lo tanto, es un pin de salida. Por otro lado, el pin ‘Echo’, detecta cuando se recibe un pulso ultrasónico, por lo tanto es un pin de entrada. Hay que declararlos como tal.

```
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
```

4.5. Funciones implementadas

Importadas las librerías, descargados los ficheros para la preparación y declaradas las variables globales correspondientes, en esta sección se van a explicar de manera detallada cada función implementada en el Software desarrollado, describiendo las partes de código más complejas y el funcionamiento de cada una de ellas.

4.5.1. getObjects()

Esta función va a ser la encargada de aplicar el modelo de detección de objetos sobre cada fotograma que se vaya a procesar.

Como entrada recibe los siguientes **argumentos**:

- **img**: Imagen sobre la que se va a aplicar el modelo de detección
- **thres**: Indica el valor mínimo de confianza de un objeto para que sea detectado
- **nms**: Es el valor que indica cuánta supresión de no máximos se aplicará a las detecciones.
- **objects**: Lista de nombres de clases de objetos que se quieren detectar por el modelo.

Como **retornos** de la función se presentan los siguientes:

- **img:** Imagen ya procesada con cada objeto detectado junto a sus respectivas cajas delimitadoras y sus nombres dibujados

- **objectName:** Lista con los nombres de los objetos detectados.
- **inside:** Lista con valores 1 y 0, que indican si el objeto está dentro (1) o no (0) del marco de proyección del sensor delimitado por la variable '*limit'.* El orden de la lista corresponde con el orden de la lista '*objectName'*, es decir, el primer elemento de esta lista corresponde con el primero de su lista y así sucesivamente.
- **size:** Lista que almacena las áreas que ocupan en la imagen cada objeto, siguiendo el mismo orden que las anteriores dos listas.

Lo primero que se realiza es una llamada al método '***detect****()'* del modelo, la cual se encarga de procesar la imagen de entrada '*img'*, teniendo en cuenta el valor mínimo de confianza *'thres'* y el valor '*nms'.*

```
classIds, confs, bbox = net.detect(img, confThreshold=thres, nms=nms)
```

Este método va a devolver en la variable ***'classIds'*** una lista con los ids de las clases de objetos que han sido detectados en la imagen. Cabe recordar que existe un id entre el 1 y el 91, asignados por orden a cada una de las 91 clases de objetos que el modelo es capaz de detectar. A su vez, en la variable ***'confs'*** se recogen las tasas de confianza respectivas a cada objeto detectado, mientras que en la variable ***'bbox'*** se almacenan las cajas delimitadoras de cada objeto, es decir, las coordenadas de la caja que delimita el área que ocupa el objeto detectado en la imagen. Por ejemplo, se va a suponer que el método *'detect()'* devuelve lo siguiente:

```
classIds=[4,17]; confs=[0.36, 0.71]; bbox=[[40,120,5,5], [50,80,10,6]]
```

Esto significa que se han detectado los objetos con id igual a 4 y 17, los cuales tienen una tasa de confianza de 0.36 y de 0.71 respectivamente, y que a su vez su área en la imagen está delimitado por una caja con las coordenadas recogidas en *'bbox'*, cuyos valores corresponden, respectivamente, con la coordenada X de la esquina superior izquierda de la caja, la coordenada Y de la esquina superior izquierda de la caja, la anchura de la caja y la altura de la caja.

Una vez detectados todos los objetos presentes en la imagen junto a sus propiedades, el siguiente paso es comprobar si el argumento de entrada '*objects'* está vacío o no. En caso de estar vacío, significa que no se han especificado ninguna lista de objetos específica para detectar, por lo cual se procede a utilizar la lista completa con todas

las clases de objetos posibles a detectar, recogida en la variable '*classNames'* ya mencionada y declarada como global.

```
if len(objects) == 0: objects = classNames
```

Acto seguido, se comprueba si el modelo ha detectado algún objeto, es decir, comprueba si la variable '*classIds'* no está vacía. En caso de no estar vacía, se procede a procesar cada uno de los objetos detectados, para lo cual se abre un bucle '*for'* de la siguiente manera:

```
for classId,conf,box in zip(classIds.flatten(),confs.flatten(),bbox)
```

Este bucle va a recorrer las tres listas (*classIds*, *confs* y *bbox*) proporcionadas por el método '*detect()'*, de forma que se pueda ir analizando cada objeto uno a uno.

Para cada objeto, del cual se conoce su 'id', se extrae su nombre de la lista de nombres '*classNames'* y se comprueba que dicho nombre figura en la lista '*objects'*, la cual recoge los objetos que quieren ser detectados. En caso afirmativo, se añade el nombre de dicho objeto a la lista '*objectName'* (la cual se retorna al finalizar la función) y se procede a dibujar sobre la imagen la caja delimitadora del objeto, el centro de dicha caja, el nombre del objeto y la tasa de confianza sobre el mismo. La caja delimitadora del objeto se obtiene del método '*detect()'* y se recoge en la variable '*bbox'*, por lo que para calcular el centro de dicha caja bastará con realizar los siguientes cálculos:

```
x = (2*box[0] + box[2]) / 2
y = (2*box[1] + box[3]) / 2
```

Tras dibujar la caja delimitadora y los datos del objeto sobre la imagen, finalmente se va a comprobar si dicho objeto se encuentra dentro del marco de proyección del sensor, para lo cual se verifica si el centro del objeto, es decir, el centro de la caja delimitadora se encuentra dentro del marco de proyección marcado por la variable global '*limit'.* En caso afirmativo se añadirá un '1' a la lista '*inside'* y se calculará el área (anchura x altura) que ocupa el objeto y se añadirá a la lista '*size'*. En cambio, en caso negativo se añadirá un '0' a la lista '*inside'* y otro '0' a la lista '*size'*, ya que si el centro del objeto no se encuentra dentro del marco de proyección, no va a ser posteriormente procesado como posible obstáculo.

Finalmente, tras realizar todas las iteraciones necesarias en función del número de objetos detectados por el modelo, se van a retornar las listas '*objectName'*, '*inside'* y '*size'* ya mencionadas, junto a la imagen modificada '*img'*, que contendrá la imagen

original acompañada de las cajas delimitadoras de cada objeto detectado junto con sus nombres y sus tasas de confianza.

4.5.2. measurement()

Esta función es la encargada de calcular la distancia detectada a través del sensor de ultrasonidos HC-SR04 con el uso de los pines GPIO de la Raspberry Pi.

La función no recibe ningún argumento de entrada, y tan sólo retorna una variable, '***distance'***, la cual contiene el valor de la distancia medida por el sensor.

Como ya ha sido explicado, el sensor funciona mediante el envío de pulsos ultrasónicos, los cuales permiten que calculando el tiempo de ida y vuelta de uno de estos pulsos y conociendo la velocidad del sonido, se pueda calcular la distancia hacia el objeto contra el que haya rebotado dicho pulso ultrasónico.

Los pines tanto '**Echo'** como '**Trigger'**, pueden situarse en dos estados diferentes: 'High' o 'Low'. El pin 'Trigger' es el encargado de lanzar el pulso ultrasónico mientras que el pin 'Echo' es el encargado de recibir dicho pulso de regreso. Por lo tanto, para lanzar el pulso, en primera instancia se debe de establecer el pin de activación 'Trigger' en estado 'High', para que tras una pausa de 10 microsegundos necesaria para garantizar que el sensor esté listo para enviar y recibir señales, se establezca el estado 'Low' sobre el pin 'Trigger' lo cual indica al sensor que se debe enviar un pulso ultrasónico.

```
GPIO.output(GPIO_TRIGGER, True)
time.sleep(0.00001)
GPIO.output(GPIO_TRIGGER, False)
```

Acto seguido, se procede a inicializar las variables '*StartTime'* y '*StopTime'*, las cuales van a recoger el tiempo exacto en el que se lanzó el pulso y en el que se recibió de regreso, respectivamente.

El pin 'Echo' va a permanecer en estado 'Low' cuando no haya ningún pulso ultrasónico lanzado y va a permanecer en estado 'High' siempre que haya un pulso ultrasónico enviado y aún no haya regresado. Por lo tanto, para decretar el tiempo del 'StartTime', se entra en un bucle 'while' hasta que el pin 'Echo' cambie a estado 'High' y justo en ese instante se guarda el tiempo de comienzo. Luego, para decretar el tiempo del 'StopTime', se entra en un segundo bucle 'while' hasta que el pin 'Echo'

cambie a estado ‘Low’ de nuevo, lo cual indica que el pulso ha regresado y por tanto en ese mismo instante se guarda el tiempo de regreso.

```
while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()
```

El tiempo total que tarda el pulso ultrasónico en salir y regresar del sensor, va a ser el resultado de restar el tiempo de regreso ‘StopTime’ menos el tiempo de comienzo ‘StartTime’.

Conociendo ya el tiempo total que ha tardado el pulso en ir y volver y sabiendo la velocidad del sonido, se realiza el siguiente cálculo, el cual aparece esquematizado en la **figura 13**.

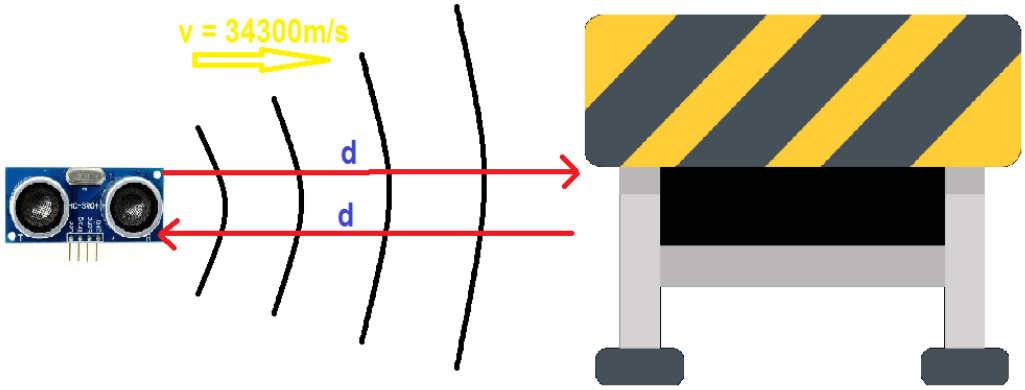

Figura 13. *Esquema para el cálculo de la distancia ‘d’ entre el sensor y un objeto.*

Para calcular la distancia ‘d’ que aparece en la figura, basta con calcular el tiempo que tarda el pulso ultrasónico en recorrer esa distancia. Como ya se ha calculado el tiempo total que tarda en ir y regresar, es decir, dos veces la distancia ‘d’, el tiempo que ha tarda en recorrer la distancia ‘d’ será ese tiempo total dividido entre dos. Con ese tiempo calculado y sabiendo que la velocidad del sonido es 34300 m/s, se multiplican ambos parámetros (d = v*t) y se obtiene la distancia entre el sensor y el objeto.

Finalmente, la función retorna el valor de la distancia en la variable de retorno ‘*distance’.*

4.5.3. capturador()

En el apartado '*Threading'* del '*Estado del Arte'*, se explicó con detalle cómo funciona la técnica de paralelización. En este caso, el programa desarrollado va a estar compuesto por tres tareas distintas las cuales van a ser ejecutadas paralelamente. Esta función contiene una de estas tres tareas, la cual va a ser ejecutada por uno de los tres hilos que va a contener el programa principal. A este hilo se le va a denominar **'Hilo Capturador'** y su funcionalidad principal va a ser la de capturar vídeo a través de la Camera Pi, e ir añadiendo a una cola todos los fotogramas que captura.

Lo primero que hace es comenzar la captura de vídeo, estableciendo las dimensiones de este, que en este caso van a ser 640x480 píxeles.

```
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
```

Tras comenzar con la captura de vídeo, la función entra en un bucle 'while' infinito (hasta que se finalice manualmente el programa) donde se van a estar añadiendo a una cola (denominada '*frames'*) todos y cada uno de los fotogramas capturados a través de la Camera Pi. Esta cola es de público acceso para los tres hilos del programa, por lo que se accederá a la misma desde el hilo encargado de procesar los fotogramas ('**Hilo Procesador'**) para obtener dichos fotogramas. Para notificar al 'Hilo Procesador', se va a utilizar un evento (ev1). Este evento va a ser activado por el 'Hilo Capturador' cada vez que añada un fotograma a la cola '*frames'*, mientras que el 'Hilo Procesador' se mantendrá esperando a que este evento se active para leer sobre la cola y realizar su función.

4.5.4. procesador()

Esta función compone la segunda de las tres tareas en las que se dividide el programa, la cual va a ser la encargada de procesar los fotogramas capturados por vídeo. Al hilo encargado de ejecutar esta función se le va a denominar '**Hilo Procesador'**.

Todas las instrucciones de este hilo se van a encontrar dentro de un bucle 'while' infinito ya que se van a estar repitiendo continuamente hasta que se finalice manualmente el programa. En primer lugar, como ya se comentó en la función '*capturador(),* esta función no va a comenzar su labor hasta que se active el evento

‘ev1’, por lo que la primera línea de código consiste en esperar a que este evento se active.

```
ev1.wait()
```

Una vez se activa dicho evento, significa que el ‘Hilo Capturador’ ha añadido algún fotograma a la cola *‘frames’,* por lo que se comprueba que no ha habido ningún error y si la cola no está vacía. En caso de no estar vacía (lo lógico), se lee el fotograma que ha sido añadido a la cola, el cual va a servir para realizar la llamada a la función ‘*getObjects()’*.

Como argumentos de entrada a la función, se va a pasar el fotograma, el umbral de tasa de confianza establecido como constante global del programa, un valor de 0.2 nms y una lista de nombres de la clase de objetos que se quieren detectar. Esta última lista de los objetos que se quieren detectar va a estar compuesta por aquellos objetos que se han considerado potenciales obstáculos de la lista global de 91 objetos que el modelo es capaz de detectar, ya que dicha lista global tiene objetos como animales salvajes o utensilios de cocina los cuales no son obstáculos que uno se pueda encontrar por la calle. La lista de nombres de estos obstáculos se lee del fichero ‘*obstacles.names’* y se almacenan en la constante global ‘*obs’*.

```
result, names, inside, size = getObjects(img,thres,0.2,objects=obs)
```

La propia función ‘*getObjects()’* va a devolver el fotograma procesado con los objetos detectados, los nombres de dichos objetos, la lista que contiene si los objetos están dentro del marco de proyección o no y la lista con las áreas que ocupan los objetos en la imagen.

Inmediatamente después de realizar la detección, se muestra por pantalla el fotograma procesado. Después, se recorren simultáneamente las listas ‘names’, ‘inside’ y ‘size’, mediante un bucle ‘for’, para poder determinar cuál de los objetos detectados es el que mayor área ocupa en la imagen de los que se sitúan dentro del marco de proyección del sensor. El objeto elegido será el que el programa considere como el obstáculo contra el que están rebotando los pulsos emitidos por el sensor de ultrasonidos. Esto va a ser considerado así debido a que cuanto mayor área ocupe en la imagen un objeto significará que más cerca está de la cámara y por ende, del sensor. Este método no es exacto, ya que habrá alguna situación en la que exista un objeto pequeño que tenga justo detrás otro objeto de mayor tamaño, ambos en el marco de proyección del sensor, de forma que el sistema detectará el objeto de mayor tamaño como el obstáculo.

```
for i, s, o in zip(inside, size, objectInfo):
        if (i == 1) and s > size_max:
                size_max = s
                obj = o
```

Una vez se ha decretado el obstáculo de entre todos los objetos detectados en la imagen, se procede a realizar una llamada a la función '*measurement()*' para obtener la distancia al obstáculo más próximo en ese preciso instante. Esta distancia, la cual viene en centímetros, va a ser convertida a metros y redondeada sobre un decimal, de forma que si se obtienen 312 centímetros, serán convertidos en 3'1 metros.

```
dist = measurement()
meters = round(int(dist) / 100, 1)
```

Con la distancia obtenida y el tipo de obstáculo decretado, se van a enviar ambos datos al tercer y último hilo, el '**Hilo Reproductor'**, el cual ejecuta la función '*reproductor()*', la cual va a ser encargada de emitir un mensaje de audio con la información recibida del 'Hilo Procesador'. Para realizar este envío, se van a hacer uso de una cola que almacena las distancias '*distances'* y una cola que almacena el nombre del obstáculo '*obstacles'*.

```
distances.put(meters)
obstacles.put(obj)
```

Una vez se añade la distancia calculada a la cola '*distances'* y el nombre del obstáculo a la cola '*obstacles'*, se avisa al 'Hilo Reproductor' mediante la activación de un segundo evento 'ev2', y se espera a que este acabe de emitir el mensaje de audio para continuar con su función. Para esperar a que acabe, se espera a que se active un tercer evento 'ev3', el cual será activado por el 'Hilo Reproductor' al finalizar la emisión del mensaje de audio.

Una vez se activa dicho evento que indica que ha finalizado la reproducción, se limpia la cola '*frames'* para eliminar todos los frames que han estado siendo añadidos por el 'Hilo Capturador' durante la ejecución de las tareas de procesamiento y reproducción, y se vuelve de nuevo al comienzo del bucle 'while', donde se espera a que el 'Hilo Capturador' active el evento 'ev1' que indica que vuelve a añadir otro fotograma para ser procesado y repetir el mismo procedimiento.

4.5.5. reproductor()

Finalmente, esta función compone la tercera y última tarea que va a ser ejecutada por el '**Hilo Reproductor'**, el cual ya ha sido mencionado en la función 'procesador()'. El objetivo principal de esta función es obtener el nombre y la distancia al obstáculo, construir un mensaje y emitirlo vía audio por el altavoz.

Debido a que el proceso de esta función se va a estar repitiendo continuamente hasta que finalice la ejecución del programa, todas sus instrucciones también irán dentro de un bucle 'while' infinito.

Como ya se explicó en el anterior apartado, esta función necesita que la función '*reproductor()'* calcule la distancia y el obstáculo, los añada a sus respectivas colas y active el evento 'ev2', por lo que la primera instrucción debe ser una espera a dicho evento.

```
ev2.wait()
```

A continuación, se corrobora que la cola '*distance'* no está vacía, y se realiza una lectura sobre ambas colas para obtener el valor de la distancia y el nombre del tipo de obstáculo más próximo.

```
distancia = distances.get()
obs = obstacles.get()
```

En caso de que no haya ningún obstáculo detectado dentro del marco de proyección del sensor, es decir, que la cola '*obstacles'* se encuentre vacía, el mensaje que se emitirá se va a referir al obstáculo como 'un obstáculo' de manera genérica. En caso de que esta cola no esté vacía y contenga el nombre del obstáculo más próximo dentro del marco de proyección del sensor, el mensaje emitido se referirá a dicho obstáculo con el nombre de su clase.

```
if not obs:
        objectName = "un obstáculo"
else:
        objectName = obs
```

Finalmente, en caso de que la distancia sea menor a 5 metros, lo cual se va a considerar como **urgente**, se va a construir un mensaje que contenga la distancia hasta el obstáculo más próximo y el tipo de obstáculo que se trata. En caso de que la distancia sea superior a 5 metros, no es necesario notificar al usuario de ningún obstáculo.

El mensaje se construye haciendo uso de la librería ‘gTTS’, y contendrá el siguiente formato, donde ‘distancia’ es el valor en metros de la distancia, ‘objectName’ es el nombre del obstáculo y donde se especifica el idioma español (lang=’es’)

```
m = gTTS(f"{distancia} metros de distancia a {objectName}", lang='es')
```

Con el mensaje construido, con la librería ‘gTTS’ se va a crear un archivo de audio ‘.mp3’ que contenga el mensaje en español transformado en voz, el cual va a ser reproducido utilizando la librería ‘*playsound’*.

```
m.save('audio.mp3')
playsound('audio.mp3')
```

Una vez se reproduce el audio, se activa el evento ‘ev3’ para que el ‘Hilo Procesador’ continua su funcionamiento y pueda pasar a procesar otro fotograma.

4.6. Programa Principal

El programa principal que va a ser ejecutado va a contener la creación de cada uno de los hilos, colas y eventos necesarios para establecer la comunicación entre todas las funcionalidades ya explicadas en las anteriores funciones.

En primer lugar se crean las instancias de las colas, haciendo uso de la librería ‘*Queue’*:

```
frames = queue.Queue()
distances = queue.Queue()
obstacles = queue.Queue()
```

La primera de estas colas contendrá los frames del video capturado por la Camera Pi en tiempo real y que va a servir de comunicación entre el ‘Hilo Capturador’ y el ‘Hilo Procesador’, la segunda abarcará las distancias medidas por el sensor de ultrasonidos y que va a servir de comunicación entre el ‘Hilo Procesador’ y el ‘Hilo Reproductor’, y la tercera contendrá el nombre de los obstáculos que van siendo detectados y que también comunicará el ‘Hilo Procesador’ y el ‘Hilo Reproductor’.

El siguiente paso es crear los 3 eventos ya mencionados que sirven también para mandar señales entre los hilos y para la sincronización entre ellos. Para ello se hará uso de la clase ‘*Event’* de la librería ‘*threading’.*

```
ev1 = threading.Event()
```

```
ev2 = threading.Event()
ev3 = threading.Event()
```

A continuación, se crean las instancias de los tres hilos, indicando como argumento de entrada la función que van a ejecutar. Se hace uso de la clase '*Thread*' de la librería '*threading*'.

```
hilo_capturador = threading.Thread(target=capturador)
hilo_procesador = threading.Thread(target=procesador)
hilo_reproductor = threading.Thread(target=reproductor)
```

Finalmente, se inicializan los hilos haciendo llamadas a sus métodos '*start()*' y '*join()*'. Tras esto, los hilos comenzarán su ejecución realizando todas sus funcionalidades ya explicadas en los anteriores apartados, comenzando por la captura de vídeo, el posterior procesamiento de los fotogramas y la detección de objetos en ellos, y por último la emisión del mensaje de voz a través del altavoz para notificar al usuario.

5. Funcionamiento y Pruebas

En esta sección se detallan los pasos a seguir para poner en marcha el funcionamiento del sistema desarrollado y, a su vez, diferentes pruebas realizadas sobre el mismo donde se plantean hipotéticas situaciones que el sistema es capaz de resolver.

5.1. Despliegue

Para el despliegue y posterior arranque del sistema, se deben de seguir una serie de pasos indispensables, los cuales se pueden asociar en dos tareas principales: incorporar una batería y configurar el arranque de la Raspberry Pi.

En primer lugar, debido a que el sistema va a ser desplegado dinámicamente en lugares sin acceso a una corriente eléctrica donde obtener una fuente de alimentación constante, es necesario añadir una batería como fuente de alimentación, de forma que se pueda acoplar al sistema cómodamente. Esta batería debe de ser lo suficientemente potente como para poder estar corriendo el sistema por un tiempo prolongado de tiempo. En este caso se ha hecho uso de la batería portátil 'Belkin BPB011'.

Figura 14. *Batería Externa Belkin BPB011 [25].*

Esta batería posee una capacidad nominal de 10000 mAh, utiliza ion de Litio y tiene un peso aproximado de 211 gramos.

Por otro lado, a la hora del despliegue no se van a poder utilizar ningún tipo de periféricos como ratón, teclado o monitor los cuales permitan acceder al entorno gráfico de la Raspberry Pi y poder ejecutar el software del sistema. De esta forma, es necesario configurar la Raspberry Pi para que en cuanto se arranque, sea capaz de ejecutar el programa de manera autónoma sin necesidad de interactuar con la misma [26].

Para ello, se necesita una tarjeta MicroSD que contenga un sistema operativo compatible con la Raspberry. En este caso, se ha utilizado la misma tarjeta MicroSD durante todo el proceso de desarrollo del sistema, la cual contiene el sistema operativo 'Raspbian'. A continuación se explican los pasos para configurar la tarjeta MicroSD para que cada vez que se introduzca sobre la Raspberry Pi o cada vez que se encienda la misma, se comience a ejecutar el software inmediatamente después de inicializarse el propio sistema operativo:

- Se crea un Script de Python que contenga el código desarrollado, en este caso se va a denominar '*programa_final.py*'
- Se guarda el Script en el escritorio.

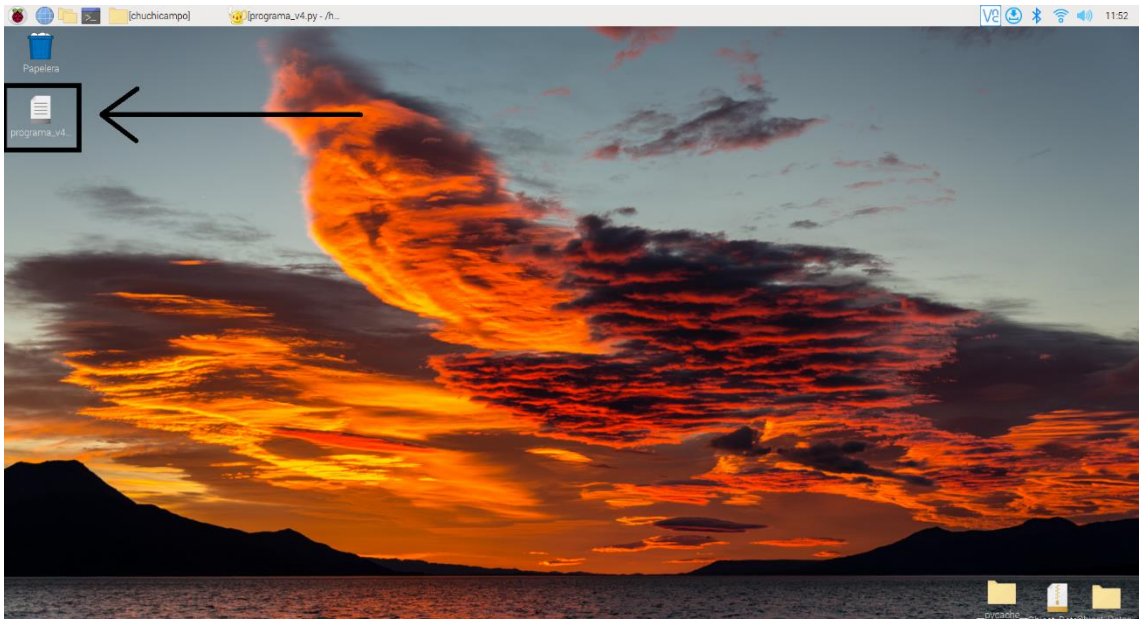

Figura 15. *Script del programa principal guardado en el escritorio.*

- Dentro del directorio '*home/user*' del sistema operativo, se accede al directorio oculto '*.config*', donde se va a crear un subdirectorio bajo el nombre '*autostart*'.

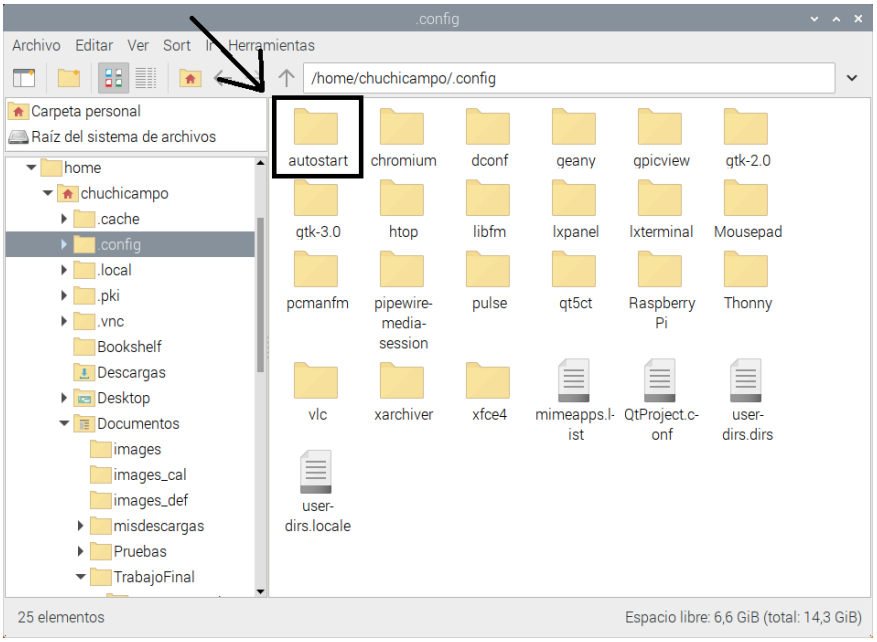

Figura 16. *Creación del subdirectorio 'autostart' en el directorio '.config'.*

- Se crea un fichero con la extensión '.desktop' dentro del directorio '*autostart*', que en este caso va a ser '*inicio.desktop.* Este fichero debe contener lo siguiente, donde '*ruta_completa*' equivale a la ruta donde se ubica el Script de Python que contiene el software del sistema.

```
inicio.desktop
1  [Desktop Entry]
2  Exec=python /ruta/completa/programa_final.py
3
```

Figura 17. *Contenido del fichero 'inicio.desktop' que configura la inicialización del programa principal. La primera línea indica el inicio de una entrada de escritorio. La segunda línea define el comando que se ejecutará al inicio de arranque del sistema operativo, en este caso, la ejecución del programa final.*

Con estos pasos completados, la Raspberry Pi esta correctamente configurada para que una vez se inicialice, lo primero que haga sea ejecutar el Script con el software desarrollado. De esta manera, al salir a la calle a utilizarlo, una vez se conecta la batería como fuente de alimentación y se introduce la tarjeta MicroSD, el programa comenzará a funcionar.

5.2. Detección de Objetos

En cuanto al modelo de detección de objetos, se han realizado diversas pruebas para comprobar el correcto funcionamiento del modelo a la hora de identificar los diferentes tipos de obstáculos en situaciones variadas. En concreto se han realizado pruebas para ver si es capaz de detectar 3 tipos de obstáculos comunes que se pueden dar en la vida real, como son personas, señales de stop o semáforos. Se han utilizado varias imágenes de cada uno de los tres tipos de obstáculos mencionados, en las que aparecen desde diferentes ángulos, posiciones y tamaños. Sobre estas imágenes se ha aplicado el modelo de detección de objetos implementado, obteniendo los resultados que se muestran en las **figuras 18-23**. En estas figuras aparecerán cajas delimitadoras sobre los objetos que se detecta, en este caso personas, señales de stop o semáforos, acompañados de un pequeño texto que indica que tipo de objeto es y un número que indica la tasa de confianza de que dicho objeto sea realmente el tipo de objeto detectado.

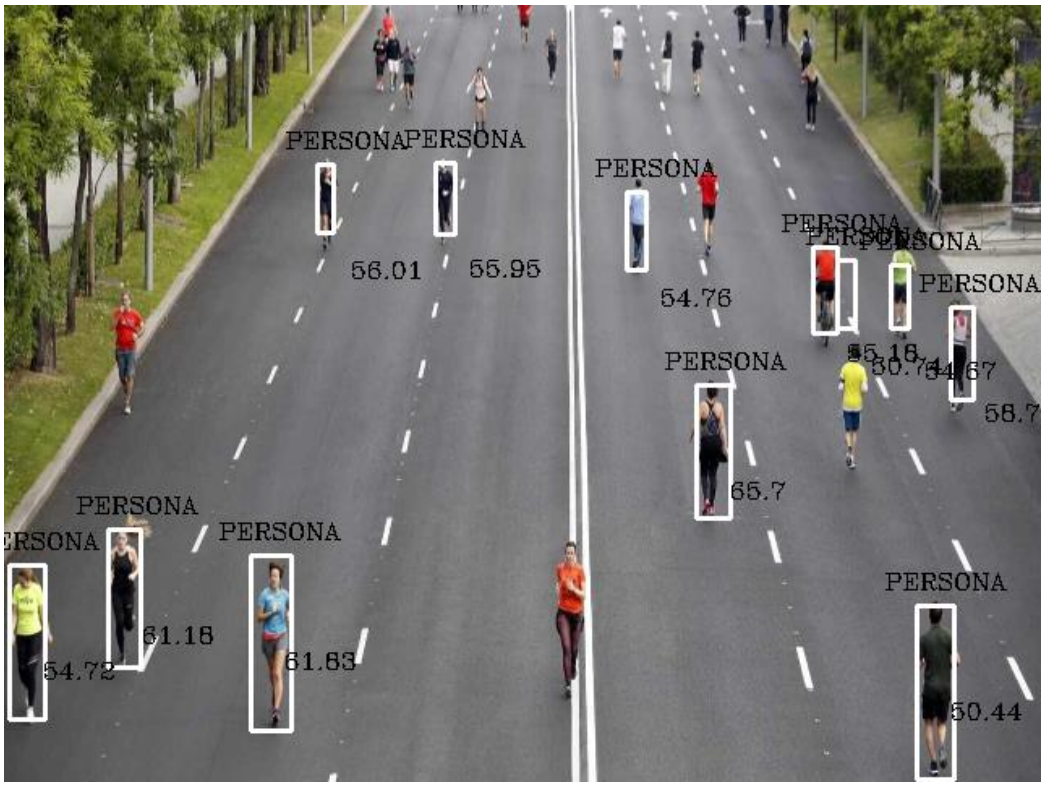

Figura 18. *Detección de personas desde vista aérea.*

Figura 19. *Detección de personas de espaldas.*

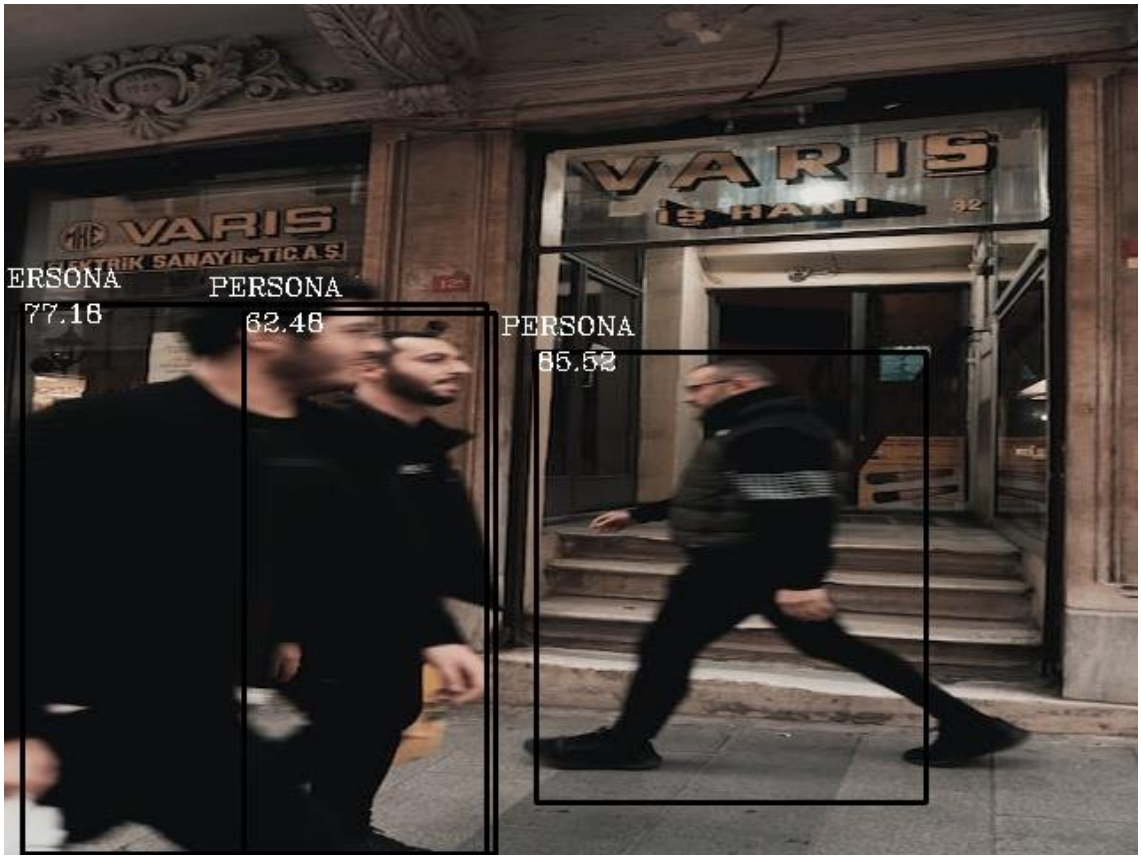

Figura 20. *Detección de personas en movimiento.*

En las **figuras 18**, **19** y **20**, aparecen los resultados de aplicar el modelo de detección de obstáculos utilizado en el sistema pero sobre imágenes de personas. Como se puede apreciar en la **figura 18** donde aparece una imagen con vista aérea de personas corriendo, el modelo es capaz de detectar y dibujar la caja delimitadora de casi todas las personas que aparecen en la imagen, a excepción de las personas situadas al fondo de la carretera. La **figura 19** corresponde con la detección de personas de espaldas, donde se detectan todas las personas a la perfección, encontrando solo un 'error', ya que la niña pequeña de la izquierda de la imagen se detecta dos veces, es decir, hay un solapamiento el cual no afectaría en gran medida al funcionamiento del sistema. Por otro lado, se ha querido probar también la detección de personas en movimiento, ya que son situaciones muy corrientes que se pueden dar por la calle en la que vienen personas a corriendo o caminando muy deprisa y la cámara puede tomar un fotograma borroso como el mostrado en la **figura 20**. En este caso, la detección es correcta.

Figura 21. *Detección de señales de stop de agrupadas y de perfil.*

Figura 22. *Detección de señales de stop con diferentes textos.*

Figura 23. *Detección de señal de stop en baja altura.*

En las **figuras 21**, **22** y **23**, aparecen los resultados de aplicar el modelo de detección de obstáculos sobre señales de stop. Se han querido probar diferentes situaciones para verificar la eficacia del modelo, como puede ser la imagen de la **figura 21**, donde aparecen varias señales de stop agrupadas y vistas de perfil, donde el modelo es capaz de detectar casi todas ellas de manera correcta, a excepción de un solapamiento en una de ellas. Por otro lado, en la **figura 22** se comprueba la correcta detección en caso de que el contenido de la señal no sea 'STOP' y finalmente en la **figura 23** se presenta un caso en el que se encuentra una señal de stop desplegada en la acera, lo cual puede ser un potencial obstáculo para el viandante.

Figura 24. *Detección de semáforo visto desde abajo.*

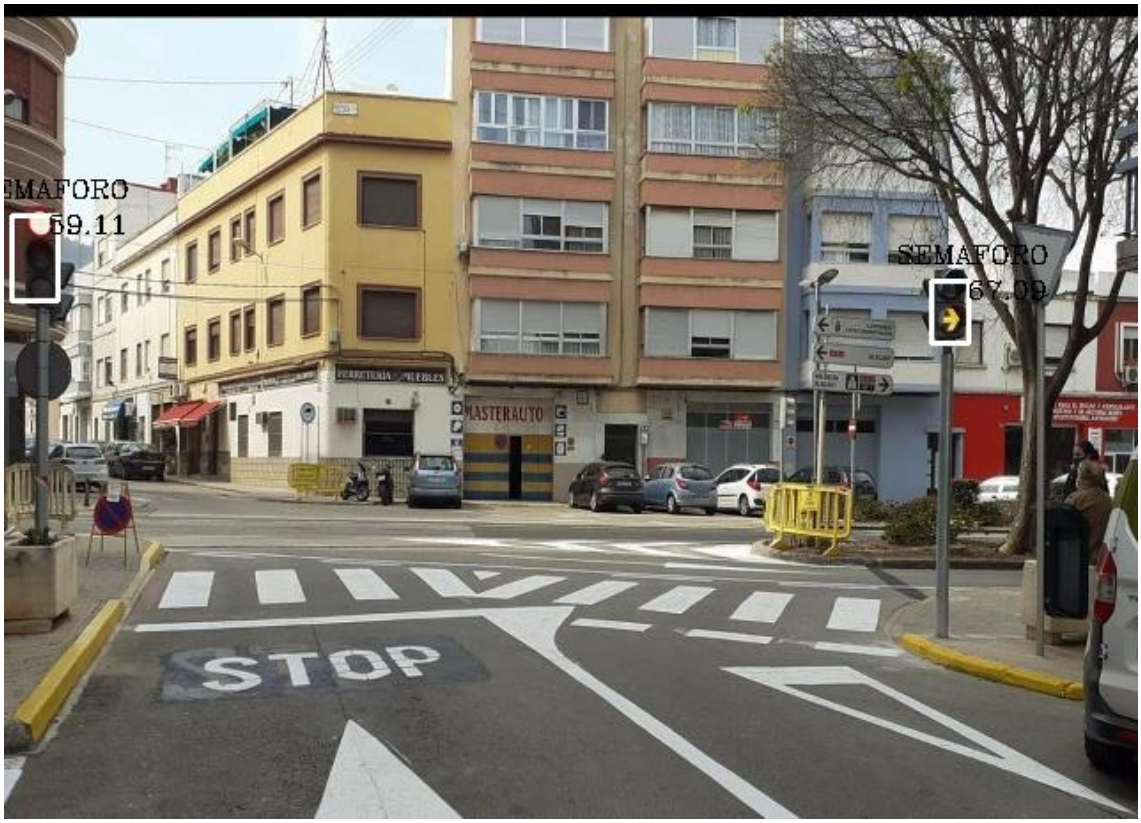

Figura 25. *Detección de semáforos en paso de cebra.*

Por último, en el caso de la detección de semáforos, tal y como se puede ver en las **figuras 24** y **25**, el resultado obtenido por el modelo no es del todo útil, ya que no detecta la estructura completa del semáforo, es decir, solo detecta la 'cabeza' y no detecta la estructura sobre la que se sostiene. De esta forma, la detección de semáforos sería útil para los casos en los que se ubican a una altura cercana a la cabeza, como puede ser el caso de la **figura 24**, en la que se presenta un semáforo saliente que podría estar a baja altura.

Por lo tanto, se puede observar que el modelo funciona apropiadamente en diferentes situaciones y para diferentes obstáculos, aunque pueden existir algunas confusiones o solapamientos pero que no afectan gravemente al funcionamiento del sistema.

5.3. Sensor de Ultrasonidos

Otro de los factores más determinantes para el correcto funcionamiento del sistema es la eficacia del sensor de ultrasonidos y la precisión que este posee en diferentes situaciones y distancias. Es por ello por lo que se han realizado pruebas sobre el sensor donde se miden distancias cortas, medias y largas, donde para cada una de las distancias se realizan varias mediciones.

Para realizar estas pruebas, se ha montado el sistema sobre un trozo de cartón y un trozo de metal como sostén, tal y como se observa en la **figura 26**.

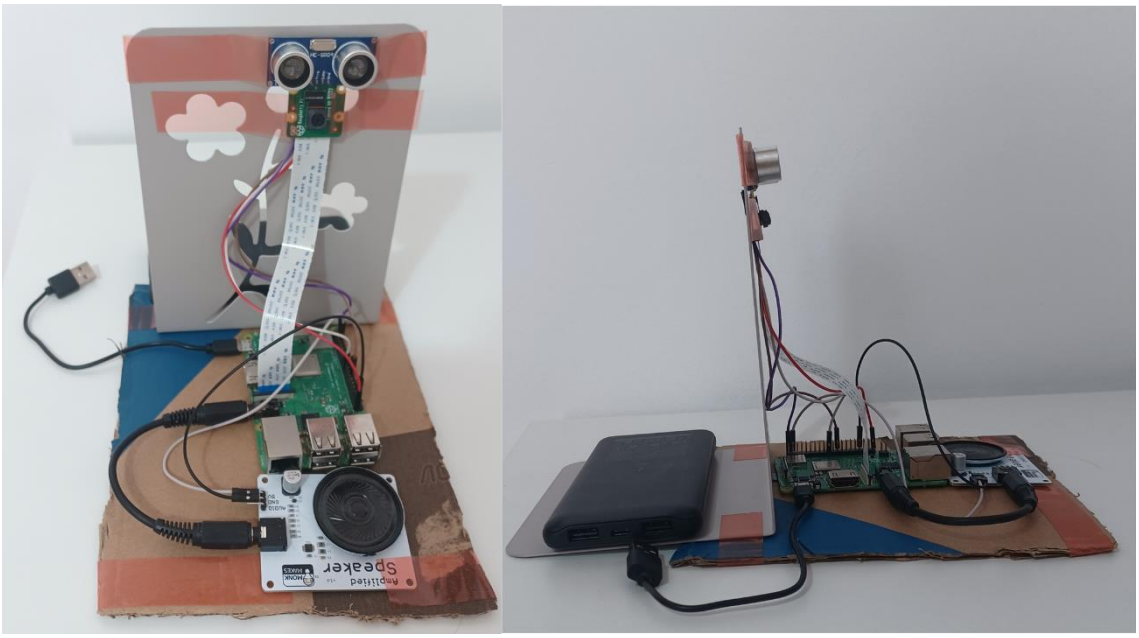

Figura 26. *Sistema montado con la Raspberry Pi 3B+, sensor HC-SR04, Amplified Speaker y Camera Pi. Alimentado con batería Belkin BPB011.*

El sistema montado se ha sacado a una terraza y se ha puesto a prueba la capacidad del sensor de ultrasonidos colocándole un obstáculo a distancias de 1, 4 y 8 metros. Sobre el sistema se ha corrido un programa simple el cual cada 3 segundos calcula la distancia haciendo uso de la función '***measurement()'*** explicada en el apartado de '*Descripción del Software'*. La distancia calculada se almacena en un fichero '.txt' y a su vez se hace una captura inicial con la cámara de la visión que tiene el sistema.

Como obstáculo se ha utilizado una caja de cartón rectangular, donde en primera instancia se ha situado a una distancia de 1 metro, como se puede observar en las **figuras 27** y **28**, obteniendo los resultados mostrados en la **figura 29**.

Figura 27. *Escenario de prueba con un obstáculo a 1 metro del sistema.*

Figura 28. *Captura con Camera Pi del obstáculo a 1 metro.*

```
La distancia calculada es: 108.2cm
La distancia calculada es: 102.2cm
La distancia calculada es: 100.1cm
La distancia calculada es: 99.6cm
La distancia calculada es: 101.8cm
La distancia calculada es: 102.7cm
La distancia calculada es: 103.1cm
La distancia calculada es: 100.6cm
La distancia calculada es: 100.8cm
La distancia calculada es: 102.3cm
La distancia calculada es: 100.1cm
```

Figura 29. Fichero ‘txt’ con las *distancias calculadas sobre el obstáculo a 1 metro.*

Como se puede observar en la **figura 29**, a excepción de la primera iteración donde se produce un desvío notable de alrededor de 8 centímetros, en el resto de las iteraciones la precisión es bastante elevada, proporcionando una desviación máxima de alrededor de 2 centímetros. De cara al rendimiento del sistema, esta desviación no produce ningún tipo de problema ya que la información que se notifica al usuario acerca de la distancia se realiza en metros y se redondea sobre un solo decimal. Por lo tanto, se puede decir que a cortas distancias el sensor es muy preciso y fiable.

Para la siguiente prueba, se va a repetir el mismo proceso, pero esta vez colocando la caja de cartón a una distancia de 4 metros, lo cual se considera como distancia media.

Figura 30. *Escenario de prueba con un obstáculo a 4 metros del sistema.*

Figura 31. *Captura con Camera Pi del obstáculo a 4 metros.*

```
La distancia calculada es: 408.2cm
La distancia calculada es: 400.2cm
La distancia calculada es: 401.4cm
La distancia calculada es: 400.0cm
La distancia calculada es: 400.1cm
La distancia calculada es: 400.3cm
La distancia calculada es: 399.9cm
La distancia calculada es: 400.6cm
La distancia calculada es: 401.1cm
La distancia calculada es: 399.8cm
La distancia calculada es: 401.4cm
```

Figura 32. Fichero 'txt' con las *distancias calculadas sobre el obstáculo a 4 metros.*

En la **figura 32** aparecen las distancias en centímetros calculadas sobre el obstáculo situado a 4 metros, donde se aprecia que la precisión de nuevo es muy elevada, presentando una desviación de apenas 1 centímetro. Se puede concluir de esta forma que para distancias medias el sensor también es bastante preciso.

Finalmente, se va a probar el rendimiento del sensor sobre un obstáculo situado a una distancia lejana, en concreto a 8 metros.

Figura 33. *Escenario de prueba con un obstáculo a 8 metros del sistema.*

Figura 34. *Captura con Camera Pi del obstáculo a 8 metros.*

```
La distancia calculada es: 1655.1cm
La distancia calculada es: 2354.2cm
La distancia calculada es: 2348.9cm
La distancia calculada es: 2350.0cm
La distancia calculada es: 2348.9cm
La distancia calculada es: 2346.1cm
La distancia calculada es: 2347.5cm
La distancia calculada es: 2353.3cm
La distancia calculada es: 2349.8cm
La distancia calculada es: 2349.9cm
La distancia calculada es: 2342.2cm
```

Figura 35. Fichero 'txt' con las *distancias calculadas sobre el obstáculo a 8 metros.*

En la **figura 35** aparecen las distancias calculadas por el sistema tras 11 iteraciones sobre el obstáculo situado a 8 metros, donde como se puede observar claramente, los resultados no se aproximan en nada a los esperados, ya que el sensor indica que la distancia es de alrededor de 23'5 metros. Esto se debe a que la capacidad máxima de detección que tiene el sensor de manera precisa es entre 2 y 450 cm [27], por lo que al situar obstáculos a distancias superiores a esa franja, como puede ser 8 metros, el sensor no es capaz de determinar la distancia. Su limitación viene producida porque el temporizador que utiliza para medir el tiempo entre el envío y recibo de los pulsos ultrasónicos, es de tan sólo 16 bits, por lo que al superar dicho tiempo el sensor de vuelve menos preciso. Además, el hecho de colocar un obstáculo a distancias lejanas hace que la onda tenga que viajar más distancia y de esta manera se atenúa y pierde intensidad.

Por lo tanto, se puede concluir con que el sensor puede medir distancias con precisión hasta los 4-5 metros, lo cual es justamente el rango que ha sido designado en el sistema como alerta, es decir, en el momento en el que un obstáculo se encuentra a

menos de 5 metros, se debe notificar al usuario del peligro. De esta manera, el hecho de que el sensor no pueda calcular distancias superiores a 5 metros hace que el sistema no tenga tanta sensibilidad pero no afecta al funcionamiento de este.

5.4. Rendimiento

Finalmente, se han realizado unos sencillo cálculos para estimar el rendimiento y los tiempos de procesamiento de algunas partes del sistema.

En primer lugar se han realizado mediciones para calcular el tiempo que tarda el sistema en realizar la fase de procesamiento de cada fotograma, es decir, el tiempo desde que el sistema obtiene un fotograma, le aplica el modelo de detección de objetos, determina qué objeto es el obstáculo más cercano y calcula la distancia con el sensor de ultrasonidos sobre dicho obstáculo. Para ello se ha hecho uso de la librería '*time*', donde se mide el tiempo antes y después del procesamiento, y se obtiene el tiempo total de procesamiento. En la **figura 36** se pueden observar los resultados tras varias iteraciones del sistema.

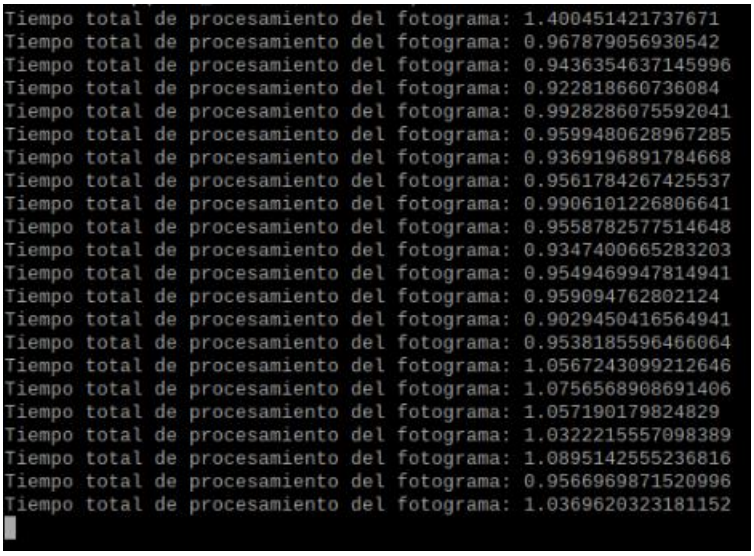

Figura 36. *Tiempos de procesamiento de varios fotogramas.*

El tiempo de procesamiento de cada fotograma, a excepción del primer que es algo más lento al tratarse del arranque del programa, ronda entre 0.9 y 1 segundos. Haciendo una media de los tiempos obtenidos, hay un tiempo medio de 0.964 segundos.

Se ha realizado el mismo procedimiento sobre el tiempo de reproducción de los mensajes emitidos por el altavoz, donde se ha calculado el tiempo estimado de media que trascurre en la creación del mensaje de voz que va a ser emitido y en la reproducción del propio mensaje. Se utiliza la misma metodología que con el tiempo de procesamiento de los fotogramas, donde se realiza una medición del tiempo inmediatamente antes y después de la creación del mensaje de voz y otra medición inmediatamente antes y después de la reproducción del mensaje por el altavoz. En la **figura 37** se pueden observar los resultados tras 10 iteraciones del sistema.

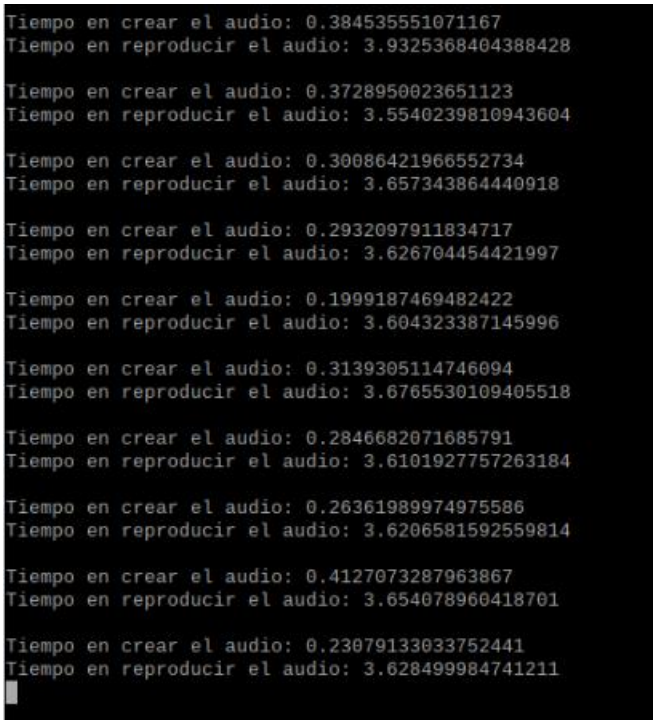

Figura 37. *Tiempos de reproducción de mensajes de voz tras varias iteraciones.*

Tras 10 iteraciones, tanto los tiempos de creación del audio como los tiempos de reproducción no varían en demasía, obteniendo un tiempo medio de creación de 0.302 segundos y un tiempo medio de reproducción de 3.651 segundos. Por tanto, el proceso total entre creación y reproducción consume alrededor de 4 segundos.

Teniendo en cuenta los resultados obtenidos en cuanto a procesamiento de los fotogramas y de reproducción de los mensajes de voz, se pueden sacar varias conclusiones. En primer lugar, desde que el sistema obtiene un fotograma capturado en video, hasta que detecta objetos sobre él y reproduce un mensaje de voz notificando al usuario sobre los posibles obstáculos, trascurre una media de 5 segundos. En ese periodo de tiempo, el sistema no puede procesar más fotogramas y detectar más obstáculos que puedan ir apareciendo ya que tiene que avisar al usuario de los obstáculos detectados previamente. Esto es algo inevitable, ya que como se ha

observado, el tiempo de procesamiento es bastante bajo (~0.9s) y el tiempo de reproducción del audio es lo más bajo que se ha podido (~4s), ya que hay que informar de la distancia y del tipo de objeto lo más abreviado posible.

Por tanto, se obtiene que la cadencia máxima de procesamiento de fotogramas es aproximadamente cada 5 segundos.

5.5. Consumo

Para terminar, se han medido el uso de recursos que realiza el sistema sobre la Raspberry Pi, en concreto en términos de uso de la CPU y memoria. Los resultados obtenidos, haciendo uso de la librería '*psutil'* para medir el tiempo medio de uso de CPU del programa, indica que se utiliza alrededor del 80% de la CPU en cada iteración de este.

El modelo de Raspberry Pi utilizado, el 3B +, realiza un **consumo** en **reposo** de aproximadamente **1.75W (Vatios)** y un **consumo máximo** de **4.9W** [28]. De esta información se puede derivar a que el consumo en reposo de la Raspberry Pi se produce cuando el uso de la CPU es próximo al 0%, mientras que el consumo máximo se produce cuando el uso de la CPU es próximo al 100%.

De esta manera, al saber que la media de uso de CPU del programa desarrollado es del 80%, haciendo una regla de tres se obtiene una estimación acerca del **consumo medio** del **programa**, el cual es de **4.27W**.

Por lo tanto, teniendo en cuenta que se usa una batería externa con una capacidad de **10000 mAH (mili-amperios/hora)** y el **voltaje** de la Raspberry Pi es de **5V**, la **capacidad** de la batería en Vatios/Hora es de **50 Wh**. De esta forma, el programa será capaz de mantenerse ejecutado durante: **50 Wh / 4.27 W = 11.7 horas**.

6. Trabajo Futuro

Como trabajo futuro, el sistema implementado en este proyecto tiene un amplio margen de mejora en diversos sentidos, sobre todo teniendo en cuenta las limitaciones de material y de presupuesto que puede tener un trabajo de fin de Máster. Generalmente, se van a desarrollar algunas de las ideas que más lógicas y funcionales que se me han ocurrido las cuales harían de este proyecto una herramienta aún más útil y funcional.

En primera instancia, la idea original de este trabajo traía consigo el **despliegue** de este sistema sobre unas **gafas** o un **gorro**, de forma que la cámara y el sensor apuntasen a una altura considerable superior al pecho del usuario. Finalmente, por falta de tiempo no ha podido realizarse este despliegue, lo cual haría que este sistema fuera funcional y pudiera ser usado realmente por una persona con discapacidad visual en su día a día.

Por otro lado, otra de las ideas que se me ocurrieron en un principio pero que por falta de recursos y de tiempo no pudo ser implementada, es la de elaborar un **modelo propio** de **detección** de **obstáculos**, lo cual sería ideal para utilizar en este sistema. Para poder desarrollar un modelo de detección de obstáculos, lo más fundamental es obtener un extenso conjunto de imágenes para poder entrenar el modelo, donde existan miles de imágenes de cada tipo de obstáculo que se quiera detectar con el modelo, es decir, miles de imágenes de farolas, miles de imágenes de señales de tráfico, de coches, de postes de la electricidad, etc.… Para ello, se tendrían que sacar todas estas fotos a mano y en contextos similares a donde se quieren detectar dichos obstáculos, es decir, sacar fotos de dichos obstáculos en la calle, desde diferentes ángulos y en diferentes situaciones, lo cual requeriría mucho tiempo de obtener tal conjunto de imágenes. A parte, suponiendo que se obtuviera tal conjunto de datos sobre diferentes tipos de obstáculos clasificados, haría falta entrenar dicho modelo. En el supuesto caso de que nuestro modelo tuviera 100 clases de obstáculos distintos que una persona se puede encontrar en la calle, el conjunto de imágenes totales con las que se entrenaría el modelo sería de alrededor de 500mil o 600mil imágenes. Para realizar un entrenamiento con semejante cantidad de imágenes, harían falta GPUs de gran potencia y mucho tiempo para procesar cada iteración del entrenamiento, por lo cual se escapa del plazo de presupuesto y tiempo destinado a este proyecto.

Si se consiguiera elaborar este modelo propio, sería una mejora considerable, ya que estaría entrenado exclusivamente con imágenes de obstáculos en entornos reales y

con un fin concreto, ya que el modelo utilizado en este proyecto ha sido entrenado para detectar diversos tipos de objetos en situaciones más generales y cotidianas, y no para detectar obstáculos para personas con discapacidad visual.

En el caso de que se tuviera un presupuesto muy elevado, una de las mejoras a futuro que se podrían incluir sería implementar el sistema sobre una **placa** de **desarrollo** más **potente** y con **mayor capacidad** de **cómputo**, como podría ser la placa 'Coral' desarrollada por Google. Esta placa está diseñada específicamente para aplicaciones de inteligencia artificial y de Machine Learning, lo cual permitiría por ejemplo que los fotogramas capturados por vídeo fueran procesados a mayor velocidad, lo cual permitiría una detección de obstáculos más precisa y en tiempo real.

7. Conclusiones

En líneas generales, este proyecto de fin de Máster me ha permitido aprender y explorar acerca de un mundo como el Machine Learning y la detección de obstáculos lo cual abarca un gran campo de posibilidades y en donde se pueden construir proyectos y herramientas tan interesantes como la desarrollada en este trabajo.

Por un lado, desde un punto de vista personal, este proyecto me ha hecho ver la verdadera utilidad que tiene combinar las diferentes técnicas y tecnologías estudiadas durante el Máster aplicadas a la vida real. Desde desarrollar un software desde cero combinando paralelismo, creación de modelos de redes neuronales para la detección de objetos, procesamiento de señales recibidas por pines GPIO… hasta la construcción física sobre una Raspberry Pi de un sistema con cámara, sensor de ultrasonidos, altavoz, batería…

En cuanto a la parte técnica, al venir de un grado de Ingeniería Informática, la parte del desarrollo del software no me ha resultado excesivamente compleja al estar ya familiarizado con el lenguaje Python y con muchas de las librerías utilizadas como OpenCV, Thread… En cambio, los aspectos de montaje, de uso de los pines GPIO y de configuración de la arquitectura hardware me han resultado algo más novedoso ya que no había utilizado estos tipos de módulos como la Camera Pi o el altavoz sobre una Raspberry Pi.

Finalmente, este trabajo lo considero una herramienta bastante interesante y que puede servir de gran utilidad para las personas con discapacidad visual, y que aún contiene un amplio margen de mejora y de perfeccionamiento que puede hacer de este proyecto un gran producto en un futuro.

8. Referencias

[1] *Raspberry Pi 3 Model B+*. (s/f). Raspberrypi.com. Recuperado el 15 de septiembre de 2023, de https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf

[2] *Estudio comparativo de tecnologías para la detección de obstáculos en pasos a nivel*. (s/f). Begiralerailway.com. Recuperado el 15 de septiembre de 2023, de http://begiralerailway.com/wp-content/uploads/2019/06/Estudio-comparativo-de-tecnolog%C3%ADas-para-la-detecci%C3%B3n-de-obst%C3%A1culos-en-pasos-a-nivel-v2.pdf

[3] *Radar de detección de obstáculos para drones DJI M300 RTK*. (s/f). Shop | SPH Engineering. Recuperado el 16 de septiembre de 2023, de https://shop.ugcs.com/es/products/obstacle-detection-radar

[4] *Computer Vision* (S/f). Microsoft.com. Recuperado el 15 de septiembre de 2023, de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-computer-vision/

[5] Soto, J. (2021, enero 28). *¿Qué es un sistema LiDAR?* Geoinnova; Asociación Geoinnova. https://geoinnova.org/blog-territorio/que-es-un-sistema-lidar/

[6] *Sensor IR Módulo de Sensor Infrarrojo de Prevención de Obstáculos por Infrarrojos para Robot de Coche Inteligente de 2 A 30 cm, Distancia de Detección Ajustable*. (s/f). https://www.amazon.es/sensores-infrarrojos-inteligentes-distancia-detecci%C3%B3n/dp/B07D3Q5B95

[7] Veloso, F. (2019, septiembre 30). *Pasos involucrados en el Machine Learning*. Feeding The Machine; Felipe Veloso. https://www.feedingthemachine.ai/pasos-involucrados-en-el-machine-learning/

[8] López, R. F., & Fernández, J. M. F. (2008). *Las Redes Neuronales Artificiales*. Netbiblo.

[9] *¿Qué son las redes neuronales convolucionales?* (s/f). Ibm.com. Recuperado el 15 de septiembre de 2023, de https://www.ibm.com/es-es/topics/convolutional-neural-networks

[10] *Google Books*. (s/f). Google.es. Recuperado el 16 de septiembre de 2023, de https://www.google.es/books/edition/Inteligencia_Artificial/Fc64EAAAQBAJ?hl=es&gbpv=0

[11] Wikipedia contributors. (s/f). *Algoritmo You Only Look Once (YOLO)*. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Algoritmo_You_Only_Look_Once_(YOLO)&oldid=153385642

[12] Graetz, F. M. (2018, Noviembre 25). *RetinaNet: how Focal Loss fixes Single-Shot Detection*. Towards Data Science. https://towardsdatascience.com/retinanet-how-focal-loss-fixes-single-shot-detection-cb320e3bb0de

[13] Ostos, E. F. (2021, mayo 3). *Detección de objetos MobileNet SSD mediante el módulo OpenCV 3.4.1 DNN*. Ebenezer Technologies - Artificial Intelligence; Ebenezer technologies - alice security. https://ebenezertechs.com/como-utilizar-opencv-mobilenet-ssd-caffe-ssd-deteccion-de-objetos/

[14] Pherkad, P. P. (s/f). *Python 3 para impacientes*. Blogspot.com. Recuperado el 16 de septiembre de 2023, de https://python-para-impacientes.blogspot.com/2016/12/threading-programacion-con-hilos-i.html

[15] Alonso, R. (2020, abril 3). *Raspberry Pi vs Arduino, ¿en qué se diferencian y para qué se usan?* HardZone. https://hardzone.es/reportajes/comparativas/raspberry-pi-vs-arduino/

[16] *Dev Board Datasheet*. (s/f). Rs-online.com. Recuperado el 16 de septiembre de 2023, de https://docs.rs-online.com/f811/A700000006921329.pdf

[17] *Arduino® MEGA 2560 Rev3*. (s/f). Arduino.cc. Recuperado el 16 de septiembre de 2023, de https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf

[18] *Raspberry Pi 3B+*. (2019, febrero 18). Raspberry Pi; MCI Electronics. https://raspberrypi.cl/raspberry-pi-3b-2/

[19] Product Description. (s/f). *Raspberry Pi Camera Module*. Rs-online.com. Recuperado el 16 de septiembre de 2023, de https://docs.rs-online.com/3b9b/0900766b814db308.pdf

[20] *Ultrasonic Ranging Module*. (s/f). Rs-online.com. Recuperado el 16 de septiembre de 2023, de https://docs.rs-online.com/8bc5/A700000007388293.pdf

[21] Diosdado, R. (2014, noviembre 26). *Sensor de ultrasonidos HC-SR04*. Zona Maker. https://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04

[22] Monk, S. (s/f). *Micro:Bit - electronic kits*. Monkmakes.com. Recuperado el 16 de septiembre de 2023, de https://www.monkmakes.com/pi_speaker_kit.html

[23] *OpenCV: Object Detection*. (s/f). Opencv.org. Recuperado el 16 de septiembre de 2023, de https://docs.opencv.org/4.x/d5/d54/group__objdetect.html

[24] *GTTS*. (s/f). PyPI. Recuperado el 16 de septiembre de 2023, de https://pypi.org/project/gTTS/

[25] *Power Bank 10K*. (s/f). Belkin ES. Recuperado el 16 de septiembre de 2023, de https://www.belkin.com/es/bater%C3%ADa-externa-port%C3%A1til-de-10-000-mah/P-BPB011.html

[26] *How to run a raspberry pi program on Startup*. (s/f). Sparkfun.com. Recuperado el 16 de septiembre de 2023, de https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup

[27] *Sensor Ultrasonido HC-SR04*. (s/f). Naylamp Mechatronics - Perú. Recuperado el 16 de septiembre de 2023, de https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

[28] Sole, R. (2022, septiembre 27). *Ahorra en tu factura de la luz gracias a la Raspberry Pi y su bajísimo consumo*. HardZone. https://hardzone.es/noticias/componentes/consumo-raspberry-pi/

ANEXOS

Script 'programa_final.py':

```
# Importe de Librerías
import cv2
import RPi.GPIO as GPIO
import time

from gtts import gTTS           # Convertir texto a Voz / Modulos
from playsound import playsound # Reproducción de audio

import threading    # Paralelismo
import queue
import random



# DETECCIÓN DE OBJETOS

# Se almacena el nombre de todas las clases que el modelo es capaz de detectar
classNames = []
classFile = "/home/pi/Documentos/TrabajoFinal/spanish.names"   # Se obtienen del fichero
'spanish.names'

with open(classFile,"rt") as f:
    classNames = f.read().rstrip("\n").split("\n")

# Se almacenan las clases de los objetos que quieren ser detectados, es este caso, los
potenciales obstáculos
obs = []
obsFile = "/home/pi/Documentos/TrabajoFinal/obstacles.names"   # Se obtienen del fichero
'obstacles.names'

with open(obsFile,"rt") as f:
    obs = f.read().rstrip("\n").split("\n")


# Rutas de los ficheros de configuración y pesos
configPath =
"/home/pi/Documentos/TrabajoFinal/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath = "/home/pi/Documentos/TrabajoFinal/frozen_inference_graph.pb"

# Creación de la red neuronal en base a la configuración y pesos definidos
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,240)       # Tamaño de entrada de la imagen en la red neuronal
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)

# Umbral de coincidencia para detectar un objeto
thres = 0.5

# Coordenadas de la caja que delimita el marco de proyección del sensor de ultrasonidos
sobre la vista de la cámara
limit = [180, 135, 280, 210]




# ESTABLECIMIENTO DE PARAMETROS DEL SENSOR DE ULTRASONIDOS
# Se establece el modo del GPIO
GPIO.setmode(GPIO.BCM)

# Número de los Pines Trigger y Echo
GPIO_TRIGGER = 8
GPIO_ECHO = 10

# Se establece el pin Trigger como salida y Echo como entrada
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
```

```python
def getObjects(img, thres, nms, objects=[]):

    '''
    Esta función recibe como entrada un fotograma capturado por video a través de la cámara y
    aplica sobre la misma el modelo de detección de objetos, donde se identifican todos los objetos
    existentes en la imagen que superen el umbral de confianza. A su vez, se dibuja una caja que delimita
    el espacio que ocupa dicho objeto junto al nombre del objeto y un punto que marca el centro de dicha caja.

    Args:
        img: Imagen que va a ser procesada por el modelo de detección
        thres: Indica el valor mínimo de confianza de un objeto para que sea detectado
        nms: Indica cuanta supresión de no máximos se aplicará a las detecciones
        objects: Lista de clases de objetos que se pueden detectar

    Returns:
        img: Imagen ya procesada con cada objeto detectado con sus respectivas cajas delimitadoras y su nombre dibujados
        objectName: Lista con los nombres de los objetos detectados
        inside: Lista con valores 1 y 0, que indican si el objeto esta dentro de la caja limite
                de detección (1) o no (0), donde el orden de los valores corresponde con el orden
                de los objetos en la lista 'objectName'
        size: Lista con las áreas que ocupan en la imagen cada objeto, siguiendo el mismo orden que
              las listas 'objectName' y 'inside'
    '''


    idsClases, tasasConf, cajas = net.detect(img,confThreshold=thres,nmsThreshold=nms)
    if len(objects) == 0: objects = classNames
    objectName =[]
    inside = []
    size = []
    if len(idsClases) != 0:
        for idClase, conf,caja in zip(idsClases.flatten(),tasasConf.flatten(),cajas):
            className = classNames[idClase - 1]
            if className in objects:
                objectName.append(className)

                # Coordenadas del centro del objeto
                x = (2*caja[0] + caja[2]) / 2
                y = (2*caja[1] + caja[3]) / 2

                # Caja delimitadora del objeto
                cv2.rectangle(img,caja,color=(0,255,0),thickness=2)
                # Punto central del objeto
                cv2.circle(img,(int(x), int(y)),1,color=(255,0,0),thickness=2)
                # Caja límite de detección
                cv2.rectangle(img,limit,color=(0,0,255),thickness=2)

                # Se escribe el nombre del objeto y el porcentaje de coincidencia
                cv2.putText(img,classNames[idClase-1].upper(),(caja[0]+10,caja[1]+30),
                cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
                cv2.putText(img,str(round(conf*100,2)),(caja[0]+200,caja[1]+30),
                cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

                # Se comprueba si el centro del objeto esta dentro de la caja limite de deteccion
                if x >= limit[0] and x <= (limit[0] + limit[2]) and y >= limit[1] and y<= (limit[1] + limit[3]):
                    inside.append(1)    # El 1 significa verdadero
                    size_obj = caja[2]*caja[3]    # En caso afirmativo, se calcula el área de ocupación aproximada
                    size.append(size_obj)
                else:
                    inside.append(0)    # El 0 significa falso
                    size.append(0)

    return img,objectName,inside,size
```

```
def measurement():

    '''
    Esta función realiza el cálculo de la distancia hasta el obstáculo más próximo en
    base a la información recibida del sensor HC-SR04 mediante los pines GPIO.

    Args: NONE

    Returns:
        distance: Distancia calculada en centímetros


    '''

   # Se establece el pin TRIGGER en estado ALTO
    GPIO.output(GPIO_TRIGGER, True)

   # Se espera 0.01ms y se establece el pin TRIGGER en estado BAJO, para enviar el pulso
ultrasónico
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    # Se inicializa el tiempo de salida y llegada del pulso ultrasónico
    StartTime = time.time()
    StopTime = time.time()

   # Cuando el pin ECHO pase a estado ALTO, se guarda el tiempo de salida,
    # ya que significará que el pulso ultrasónico ha salido
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

   # Cuando el pin ECHO pase a estado BAJO, se guarda el tiempo de llegada,
    # ya que significará que el pulso ultrasónico ha llegado
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

   # Se calcula el tiempo transcurrido entre salida y llegada del pulso ultrasónico
    TotalTime = StopTime - StartTime


   # Se calcula la distancia, multiplicando el tiempo por la velocidad del sonido (34300
cm/s)
    # y dividido entre dos, porque el tiempo es de ida y vuelta.
    distance = (TotalTime * 34300) / 2

    return distance


def capturador():

    '''
    Es la función principal del 'Hilo Capturador' y captura vídeo mediante la Camera PI
y añade los
    fotogramas sobre una cola pública sobre la cual tendrá acceso el 'Hilo Procesador'

    Args: NONE

    Returns: NONE

    '''
    cap = cv2.VideoCapture(0)   # Se comienza a capturar video
    cap.set(3,640)
    cap.set(4,480)

    while True:
        # Se lee un fotograma y se añade a la cola 'frames'
        success, img = cap.read()
        frames.put(img)
        ev1.set()   # Se notifica al 'Hilo Procesador'
```

```python
def procesador():

    '''
     Es la función principal del 'Hilo Procesador' y procesa cada fotograma que lee de la cola 'frames'
     aplicando la función 'getObjects', y a su vez calcula la distancia al objeto más próximo mediante el uso
     de la función 'measurement'. Determina que objeto es el obstáculo más próximo, y dicha información junto
     a la distancia en metros, se añade sobre una cola sobre la cual tendrá acceso el 'Hilo Reproductor'.

    Args: NONE

    Returns: NONE


    '''

    while True:
        ev1.wait()
        size_max = 0
        obj = None

        if not frames.empty():
            # Se lee el último fotograma añadido a la cola 'frames' y se aplica la detección de objetos
            img = frames.get()
            result, names, inside, size = getObjects(img,thres,0.2, objects=obs)

            cv2.imshow("Output", img)

            # De entre todos los objetos detectados, se determina el obstáculo más próximo dentro del rango del sensor
            for i, s, o in zip(inside, size, names):
                if (i == 1) and s > size_max:
                    size_max = s
                    obj = o

            # Se calcula la distancia y se redondea en metros
            dist = measurement()
            meters = round(int(dist) / 100, 1)

            # Se añaden la distancia y el nombre del obstáculo a sus respectivas colas
            distances.put(meters)
            obstacles.put(obj)

            # Se notifica al 'Hilo Reproductor' y se espera a que finalice
            ev2.set()
            ev3.wait()

            # Se limpia la cola de fotogramas
            frames.queue.clear()

            ev3.clear()
            cv2.waitKey(1)

        ev1.clear()
```

```python
def reproductor():

    '''
    Es la función principal del 'Hilo Reproductor' y lee de una cola la información relativa sobre la
    distancia al obstáculo más próximo. Crea un mensaje de voz con dicha información y lo reproduce
    a través de un altavoz.

    Args: NONE

    Returns: NONE
    '''

    while True:
        ev2.wait()
        if not distances.empty():
            # Se lee la última distancia calculada y el último obstáculo detectado
            distancia = distances.get()
            obs = obstacles.get()

            # En caso de no haberse detectado ningún objeto, se considerará un 'obstáculo' cualquiera
            if not obs:
                nombreObs = "un obstáculo"
            else:
                nombreObs = obs

            # En caso de que la distancia sea inferior a 5 metros, se notifica al usuario del obstáculo
            if distancia < 5:
                audio = gTTS(f"{distancia} metros a {nombreObs}", lang='es')
                audio.save('/home/chuchicampo/Documentos/TrabajoFinal/audio.mp3')
                playsound('/home/chuchicampo/Documentos/TrabajoFinal/audio.mp3')

            nombreObs = None

            ev3.set()

        ev2.clear()


if __name__ == "__main__":

    # Inicialización de las colas
    frames = queue.Queue()      # Almacena fotogramas capturados por la cámara
    distances = queue.Queue()   # Almacena distancias calculadas por el sensor
    obstacles = queue.Queue()   # Almacena nombres de los obstáculos detectados

    # Inicialización de eventos
    ev1 = threading.Event()
    ev2 = threading.Event()
    ev3 = threading.Event()

    # Inicialización de los hilos
    hilo_capturador = threading.Thread(target=capturador)
    hilo_procesador = threading.Thread(target=procesador)
    hilo_reproductor = threading.Thread(target=reproductor)

    # Comienza la ejecución de los hilos
    hilo_capturador.start()
    hilo_procesador.start()
    hilo_reproductor.start()

    # Sincronización de la ejecución de los hilos
    hilo_capturador.join()
    hilo_procesador.join()
    hilo_reproductor.join()
```