Dron de vuelo autónomo con reconocimiento basado en inteligencia artificial.

Autonomous flight drone with recognition based on artificial intelligence.



Trabajo de fin de grado del grado en ingeniería informática.

Curso académico 2019-2020.

Realizado por Juan Carlos De Alfonso Juliá.

Dirigido por Juan Jiménez Castellanos.

Índice

[Índice 2](#_Toc33428962)

[Resumen. 4](#_Toc33428963)

[Palabras clave. 4](#_Toc33428964)

[Introducción. 4](#_Toc33428965)

[Estructura del repositorio 4](#_Toc33428966)

[Etapa 1: Diseño y construcción. 5](#_Toc33428967)

[Diseño hardware 5](#_Toc33428968)

[Primer acercamiento: elementos que podemos necesitar. 5](#_Toc33428969)

[Segundo acercamiento: Selección concreta de componentes. 5](#_Toc33428970)

[Construcción final del prototipo. 6](#_Toc33428971)

[Estructura adicional impresa en 3d 6](#_Toc33428972)

[Integración de los componentes. 7](#_Toc33428973)

[Etapa 2: software del prototipo 8](#_Toc33428974)

[Sistema operativo 8](#_Toc33428975)

[Mpu9250 IMU 8](#_Toc33428976)

[MS5611 9](#_Toc33428977)

[RGB LED 9](#_Toc33428978)

[Módulo GPS 9](#_Toc33428979)

[Controlador PWM 9](#_Toc33428980)

[Etapa 4: Diseño de una aplicación de control. 9](#_Toc33428981)

[Introducción. 9](#_Toc33428982)

[Protocolo de comunicación. 10](#_Toc33428983)

[Modelo cliente servidor. 10](#_Toc33428984)

[Protocolo TCP. 10](#_Toc33428985)

[Funcionamiento interno. 10](#_Toc33428986)

[Estructura de la aplicación. 11](#_Toc33428987)

[Logo de la aplicación. 11](#_Toc33428988)

[Ventanas de la aplicación. 11](#_Toc33428989)

[Estructura interna. 14](#_Toc33428990)

[Etapa 5: Diseño del reconocedor basado en IA. 16](#_Toc33428991)

[Introducción 16](#_Toc33428992)

[Vista teórica 16](#_Toc33428993)

[Classificación vs detección 16](#_Toc33428994)

[SSD – Single shot detector 17](#_Toc33428995)

[TensorFlow API 18](#_Toc33428996)

[Comparación de modelos pre-entrenados 18](#_Toc33428997)

[Funcionamiento interno. 19](#_Toc33428998)

[Etapa 6: Controlador y flujo de funcionamiento. 20](#_Toc33428999)

[Ejecución del controlador 20](#_Toc33429000)

[Estructura interna 21](#_Toc33429001)

[Etapa 7: Pruebas finales. 21](#_Toc33429002)

[Conclusiones. 21](#_Toc33429003)

[Bibliografía. 21](#_Toc33429004)

Resumen.

Palabras clave.

Dron, pid, tcp

Introducción.

# Estructura del repositorio

Con el propósito de dar una explicación más detallada durante la realización de esta memoria, en ocasiones se hace referencia al código fuente del trabajo que se encuentra alojado en el repositorio de GitHub juancalf/TFG (1).

El repositorio está subdividido en las siguientes carpetas:

Android\_app: Dentro de esta carpeta se encuentran todos los códigos referentes a la aplicación de control, en concreto está la aplicación completa comprimida “app.zip” y dos carpetas “archivos\_java” y “archivos\_xml” con los archivos fuente más importantes para poder consultarlos sin necesidad de descargar la aplicación ni tener que buscar en esta los distintos archivos.

Reconocimiento: En esta carpeta está alojado todo el software usado para la detección de los objetivos. Está subdividida de la siguiente manera:

* Reconocedor.py: archivo fuente con el algoritmo de reconocimiento.
* Labels: contiene el archivo .pbtxt con la información del id y el nombre usado para identificar a las personas.
* Utils: contiene todos los archivos y librerías que usa el algoritmo para funcionar.
* Capturas: contiene las imágenes de los individuos detectados.
* Modelos: contiene los modelos usados para la detección.
* TestVideos: contiene una serie de videos usados durante la depuración del algoritmo para comprobar su correcto funcionamiento.

Control: contiene el archivo “controlador.py” que se encarga de controlar y gestionar todo el funcionamiento del aparato.

Navegación:

Etapa 1: Diseño y construcción.

# Diseño hardware

## Primer acercamiento: elementos que podemos necesitar.

De forma abstracta resulta sencillo el saber que elementos podemos necesitar para construir un dron:

* Chasis o armazón.
* Ordenador que actúe como “cerebro” y sea capaz de al recibir información de los sensores y tomar decisiones durante el vuelo.
* Una cámara para la etapa de reconocimiento.
* Una batería.
* Cuatro motores con sus respectivos variadores (ESCs) y hélices.
* Sensores para poder conocer la inclinación del aparato, sus coordenadas, la altura o el nivel de batería.
* Un módulo de comunicación inalámbrica (wifi, bluetooth, etc.).

## Segundo acercamiento: Selección concreta de componentes.

Para el chasis se ha elegido el modelo F450 de la marca DJI dado que al tener una superficie amplía la colocación de los componentes será más sencilla, además cuenta con un tablero con una pdb[[1]](#footnote-1) integrada lo que hace mucho más fácil el soldar y colocar los distintos cables.

Para el ordenador de a bordo hay dos opciones claras (Arduino o Raspberry) en concreto dado que vamos a necesitar poder ejecutar un algoritmo de machine learning y alguna forma de conectarnos al ordenador lo más sencillo es usar una Raspberry, en concreto el modelo 3B+ que cuenta con bluetooth y wifi de serie.

Imagen que contiene electrónica, circuito

Descripción generada automáticamenteLos sensores son quizá la parte más importante pues deben estar colocados de manera correcta en el aparato para poder funcionar correctamente, por la cantidad de sensores necesarios se decidió comprar un HAT para la raspberry llamado NAVIO2 compatible con nuestro modelo.

Fig x.x. Placa Navio2 montada sobre Raspberry (1)

Esta shield cuenta con un barómetro de alta resolución (MS5611), dos sensores IMU (MPU9250 y LSM9DS1), la compatibilidad de, mediante un adaptador poder ser alimentado por una batería LIPO, un módulo gps (Ublox M8N) y 14 canales PWM para poder controlar los motores (solo se usarán 4). Esta hat lleva otros sensores y conectores que no se mencionan dado que no se usarán para este proyecto (3).

## Estructura adicional impresa en 3d

Para que la estructura fuera más sólida, algunos componentes se han unido entre si usando piezas impresas en 3d usando una impresora Prusa modelo mk3s (2). Todos los archivos, tanto los originales como los modificados se pueden encontrar dentro del repositorio de GitHub mencionado al principio de esta memoria (3).

La placa Navio2 y la Raspberry se han integrado en la parte superior usando una carcasa añadiendo huecos para atornillarla a nuestro armazón (4). Esta carcasa además cuenta con huecos para la colocación de bolas anti-vibración para que durante el vuelo sea más estable la lectura de los sensores (Fig x.x).

También se ha rediseñado una funda para la cámara basada en el siguiente diseño (4), añadiéndole un soporte con el que poder orientar el ángulo de esta de forma sencilla y unos agujeros adicionales para poder atornillarla (Fig x.x).

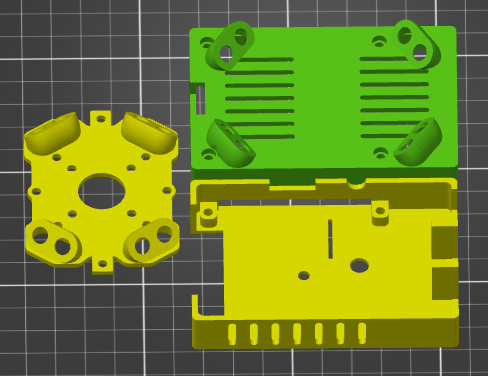


Imagen que contiene suelo

Descripción generada automáticamente

Fig x.x. Diseño 3d de la funda para la cámara

Fig x.x. Diseño 3d del armazón de la Raspberry.

# Construcción final del prototipo.

Etapa 2: software del prototipo

# Sistema operativo

Para poder utilizar la placa Navio2 es necesario utilizar una versión modificada del sistema operativo Raspbian, este software lo proporciona el fabricante y no lleva incorporada interfaz gráfica por lo que en mi caso lo más cómodo ha sido el conectarla a una red wifi modificando el archivo “/boot/wpa\_supplicant.conf” y a partir de ahí trabajar mediante ssh[[2]](#footnote-2) desde una terminal (7).

Para modificar dicho archivo basta con abrirlo con algún editor de texto y añadir las siguientes líneas:

network ={

ssid = “ssid de la red”

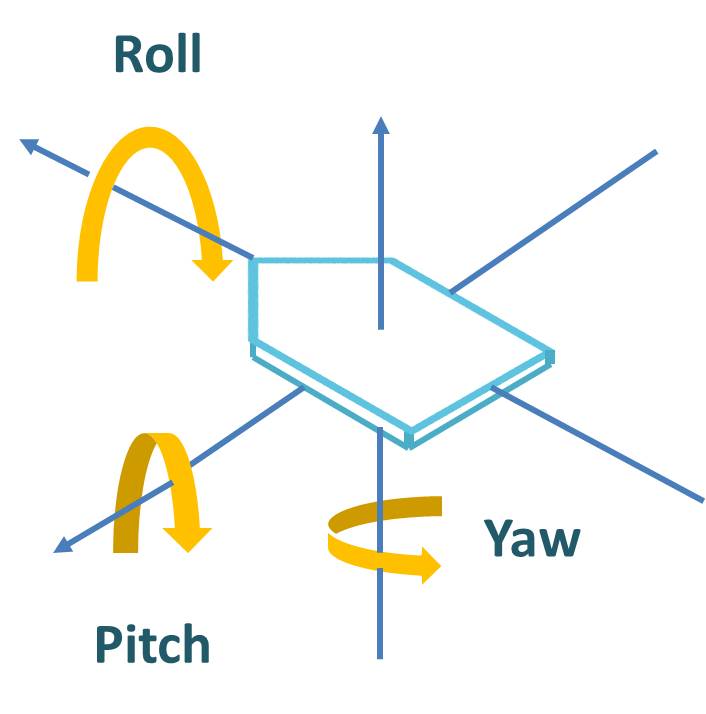
psk = “contraseña de la red”

key\_mgmt = “seguridad de la red (WPA-PSK, etc.)”

}

# Mpu9250 IMU

Es un conjunto de sensores que contiene un magnetómetro de 3 ejes, un acelerómetro de 3 ejes y un giroscopio de 3 ejes lo que va a permitir conocer la orientación del aparato en diversas posiciones, en este caso concreto lo que más interesa es conocer las posiciones del pitch, roll y yaw



Para la obtención de los valores del Pitch y el Roll

<https://www.amebaiot.com/en/ameba-arduino-quadcopter/>

<https://www.electrohobby.es/posicion/288-mpu9250-9-ejes-movimiento.html>

# MS5611

Es un sensor barométrico capaz de medir la presión ambiental con una precisión significativa, esto nos va a permitir con diversos cálculos conocer la altitud del aparato.

La programación de este sensor se realiza en el archivo barometro.py que esta subdividida en tres funciones principales y una de inicialización, en donde los valores de la presión atmosférica se obtienen dentro de la función getPress que realiza una llamada a los drivers del fabricante para proporcionar dicho dato, para la obtención de la altitud se utiliza la siguiente fórmula:

# RGB LED

Módulo implementado en el archivo led.py permite mediante dos funciones establecer el color del led superior de la placa Navio2 o bien desconectarlo en función del estado de este.

# Módulo GPS

Va a permitir conocer la latitud y la longitud del aparato y permitirá corregir el rumbo usando estos datos, además también se usará esta información durante la fase de reconocimiento para guardar donde ha encontrado los objetivos.

La implementación de la acción de leer las coordenadas gps se realiza

# Controlador PWM

Quizá uno de los módulos más importantes dado que es el que permite el controlar la velocidad de giro de los motores.

Etapa 4: Diseño de una aplicación de control.

# Introducción.

La aplicación de nombre Pathfinder en honor a la sonda enviada a explorar Marte la he diseñado desde cero en los lenguajes java y XML usando Android Studio como entorno de desarrollo. El propósito de dicha aplicación es poder gestionar el recorrido del dron de forma integra permitiendo la selección de un plan de vuelo incluyendo el recorrido, la altitud o el lugar de aterrizaje.

La conexión entre el dispositivo móvil y el prototipo se ha realizado mediante un modelo cliente-servidor a través de un protocolo TCP explicado más adelante.

Nota: la aplicación ha sido testeada en un móvil Xiaomi MI A2 con Android10 como sistema operativo por lo que en versiones anteriores y posteriores no se asegura el funcionamiento de esta, además es necesario conceder los permisos de ubicación y almacenamiento a esta aplicación para poder usarla.

# Protocolo de comunicación.

## Modelo cliente servidor.

El modelo cliente servidor es una estructura de comunicación entre dos dispositivos (cliente y servidor) a través de internet en donde por lo general el servidor es un proceso activo que espera la conexión de un cliente que se conectará únicamente cuando este necesite algo del servidor.

Dependiendo de las necesidades es posible que el servidor sea capaz de trabajar de forma concurrente admitiendo varios clientes conectados de forma simultanea

Cabe mencionar que el modelo cliente servidor proporciona la conexión entre los dispositivos sin embargo se necesita algún tipo de protocolo para poder mandar y recibir información de tal forma que esta sea legible por ambas partes, para esto tenemos varias alternativas (protocolos TCP, UDP, etc.).

## Protocolo TCP.

## Funcionamiento interno.

En el caso concreto del dron y el teléfono el primero actuará como servidor ejecutando un bucle de espera hasta que un cliente (el teléfono) se conecte y transmita la información.

Únicamente se transmitirán dos mensajes, el primero por parte del teléfono que contendrá una línea del estilo siguiente:

6.3|40.34523,-3.43463|42.343243,-3.3532…

El primer valor corresponde a la altura que debe llevar el dron durante todo su recorrido, y el resto de valores corresponden a pares de coordenadas por donde debe pasar el aparato.

El segundo mensaje corresponde a una confirmación de recibido (ACK) que pondrá fin a la comunicación.

El código del cliente se encuentra alojado en la raspberry

# Estructura de la aplicación.

## Logo de la aplicación.

El logo se ha creado usando la herramienta Freelogodesign donde el servicio permite el uso de los logotipos diseñados con fines personales o comerciales (2).

## Ventanas de la aplicación.

La aplicación la conforman cinco ventanas.

La primera de estas corresponde a la que se ve al ejecutar la aplicación (fig x.x), y corresponde a un menú en donde se pueden seleccionar las opciones de iniciar vuelo, ajustes e información junto con el logo ampliado de la aplicación.

Si seleccionamos el botón de información se abrirá una nueva ventana meramente informativa con información acerca de la app (instrucciones, autoría…) (Fig .).

En el caso de seleccionar ajustes, la interfaz mostrará una serie de parámetros configurables que se aplicarán durante el vuelo del dron o en la conexión con este (Fig x.x), en concreto encontramos dos submenús, el primero subtitulado “Protocolo TCP” permite configurar dos parámetros relativos a la comunicación:

* Dirección IP: corresponde a la dirección ip asignada a la Raspberry dentro de la red.
* Puerto: identificador único del puerto del servidor de la Raspberry, por defecto usaremos el puerto 12345.

El segundo submenú referente a la navegación permite configurar lo siguiente:

* Altura: la altura en metros a la que debe volar el aparato.
* Vuelta al origen: representa un switch que en el caso de estar seleccionado indicará al dron que al finalizar el recorrido debe volver al punto del que partió y aterrizar ahí, en caso de no seleccionarse aterrizará en el último punto establecido por el usuario.

La tercera de las ventanas a la que se llega si se selecciona el botón “iniciar vuelo” abrirá una interfaz más compleja formada por un mapa en donde aparecerá la ubicación de nuestro aparato así como dos botones “iniciar” y “reset”.

En esta ventana debemos seleccionar el recorrido que queremos que haga el aparato pulsando en el mapa, esto hará que se creen diferentes puntos (con un máximo de diez) que se recorrerán en orden de selección, el botón reset permite borrar todos los puntos y volver a crear el recorrido, y el botón iniciar nos lleva a la última ventana explicada a continuación.

La última ventana es meramente informativa pues dado que cuando se inicia, gráficamente no realiza ninguna acción, el código de esta en cambio es muy importante pues realizará la comunicación inalámbrica enviando todos los datos necesarios y usando la dirección ip y el puerto de los ajustes para dicha acción.

Imagen que contiene texto, mapa

Descripción generada automáticamenteImagen que contiene captura de pantalla

Descripción generada automáticamenteImagen que contiene captura de pantalla

Descripción generada automáticamenteImagen que contiene captura de pantalla

Descripción generada automáticamente

Fig x.x. Pantalla de ajustes.

Fig x.x. Pantalla principal de la aplicación.

Fig x.x. Pantalla de información.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Fig x.x. Pantalla informátiva final.

Fig x.x. Pantalla de creación de recorrido

## Estructura interna.

Archivos .java

Corresponden a clases usadas..

Java//AjustesActivity.java

Java//InfoActivity.java

Java//MainActivity.java

Es la clase principal llamada al inicio de la aplicación, está formada por varios métodos que actúan como liseners de los botones de la interfaz.

Es la encargada de pasar los parámetros de configuración de vuelo y tcp a la siguiente clase teniendo en cuenta que los parámetros por defecto han podido ser modificados, esta acción se realiza recogiendo los datos enviados por los ajustes con la siguiente porción de código:

Java//VueloActivity.java

Java//VueloActivity.java

Archivos.xml

Manifest/AndroidManifest.xml

<https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es>

Es un archivo que proporciona al teléfono información acerca de la aplicación, dentro de este archivo debemos incluir lo siguiente:

* Token para poder usar el sdk que proporciona Here.
* Permisos: funciones sensibles que la aplicación realizará, en este caso concreto necesitamos acceso a la ubicación, internet y almacenamiento. La sintaxis para reclamar estos permisos tiene una estructura similar a la descrita a continuación.

<uses-permission android:name="android.permission. ACCESS\_FINE\_LOCATION" />

En este ejemplo estamos reclamando permiso de acceso a la ubicación del teléfono, en el archivo del repositorio estos permisos están incluidos entre las líneas 5 y 10.

* Actividades: cada una de las vistas descritas en el apartado anterior lleva asociada una etiqueta en este archivo con información acerca de diversos aspectos (etiqueta, actividad padre o nombre).

res/layout/activity\_ajustes.xml

res/layout/activity\_info.xml

res/layout/activity\_main.xml

res/layout/activity\_vuelo\_msg.xml

res/layout/activity\_vuelo\_map.xml

Etapa 5: Diseño del reconocedor basado en IA.

# Introducción

Una vez realizada la tarea de tener un aparato funcional queda el que se puedan localizar los objetivos buscados de una forma rápida y efectiva, para dicha tarea hay diversas alternativas como por ejemplo el envío de video en tiempo real o el análisis de este una vez que el aparato haya regresado del recorrido establecido, sin embargo dado que buscamos que el conjunto de tareas tengan la menor interacción humana lo lógico es que el propio ordenador de abordo sea capaz de realizar la tarea mencionada.

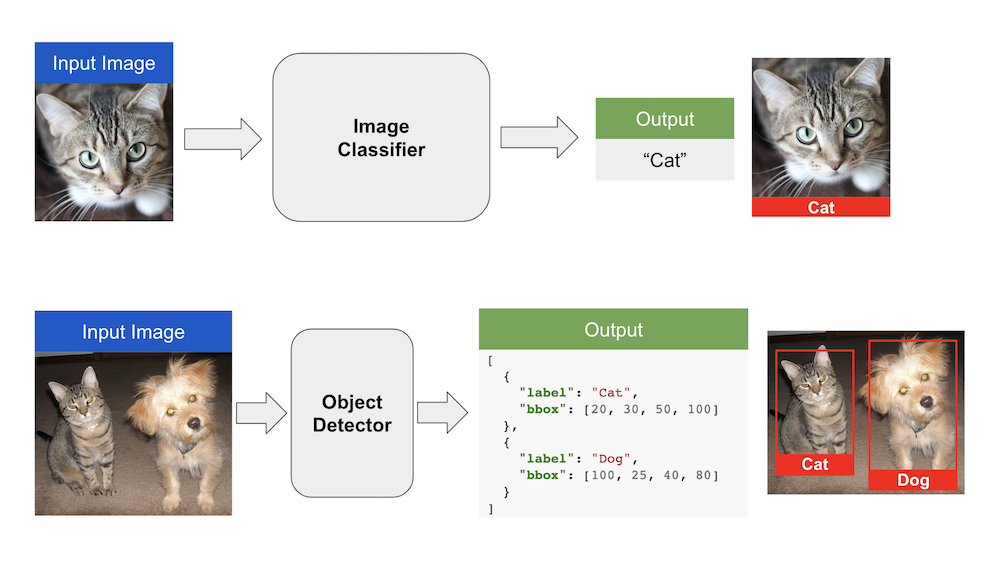
Esto es posible usando algoritmos de inteligencia artificial que son capaces de detectar patrones durante el análisis de video y por tanto reconocimiento de objetivos con una determinada certeza, en concreto en este proyecto se usa una técnica de Deep learning llamada SSD explicada más adelante.

# Vista teórica

## Clasificación vs detección

Dentro del paradigma de la identificación de objetos tenemos dos alternativas, la clasificación y la detección. El primero como su nombre indica permite dada una imagen saber que contiene, por ejemplo, si diseñamos un clasificador de aves nuestro algoritmo debería ser capaz de dada una imagen decirnos la especie del ave.

En cambio, en un algoritmo de detección no solo buscamos clasificar una imagen según el tipo de esta según algún parámetro, sino además saber su posición en la imagen acotándola normalmente con algún recuadro.



<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.learnopencv.com%2Ffaster-r-cnn-object-detection-with-pytorch%2F&psig=AOvVaw0aQfrs2nwoYoSFawxtoTUI&ust=1581582258583000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLDB36HLy-cCFQAAAAAdAAAAABAD>

https://developers.arcgis.com/python/guide/how-ssd-works/

## SSD – Single shot detector

Es un algoritmo de Deep learning considerado uno de los más rápidos que para poder realizar la detección de elementos en una imagen, combina la detección de regiones de interés mediante una red RPN y de forma simultanea la clasificación de estas regiones para saber a que corresponden.

En profundidad este algoritmo realiza una descomposición de la imagen usada como entrada en distintas partes de distintos tamaños para que por cada sub-imagen obtenida, se le aplique un algoritmo de clasificación y si se ha obtenido un resultado concluyente se guardará la posición de la sub-imagen para luego añadirle un recuadro con el nombre asociado a la clase detectada, así como su porcentaje de certeza.

https://www.deeplearningitalia.com/uso-del-aprendizaje-profundo-para-el-reconocimiento-de-objetos/

# TensorFlow API

Con el propósito de simplificar la tarea de implementar el reconocedor se va a utilizar una api proporcionada por Tensorflow que permite reconocer imágenes usando modelos pre-entrenados de su página web.

# Comparación de modelos pre-entrenados

Tensorflow proporciona varios modelos entrenados y validados con COCO que es un dataset ampliamente usado en problemas de detección de objetos por su gran tamaño y sus 91 categorías, además proporciona por cada imagen multiples objetos con lo que su eficacia es notable.

<https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/>

Estos modelos listados en la imagen x.x se encuentran alojados en el repositorio de Tensorflow subdivididos en velocidad(ms) y eficacia de reconocimiento(mAP), para el uso en aparatos con poca poteancia recomienda el uso de los modelos especificados como “mobilenet” que son modelos cuyos requisitos de ejecución son notablemente más bajos en relación al resto.



Fig x.x. Modelos ssd mobilenet proporcionados por tensorFlow.

[[3]](#footnote-3)

Para este caso en particular se ha elegido el modelo “ssdlite\_mobilenet\_v2\_coco” que es uno de los modelos más modernos proporcionados por TensorFlow con una buena relación entre eficacia y tiempo de detección.

# Funcionamiento interno.

El funcionamiento de esta parte del proyecto como se ejemplifica en la () realiza las siguientes tareas:

1. Inicialización y carga:
2. Ejecución en bucle del algoritmo:
   1. Captura de un fotograma de la webcam.
   2. Análisis de ese fotograma usando el algoritmo ssd.
   3. Comprobación de requisitos de detección.
      1. Requisitos temporales.
      2. Requisitos de umbra.
   4. Guardado del fotograma: en el caso de que los requisitos anteriores se cumplan la imagen es considerada válida y un posible resultado positivo

Los requisitos a los que se refiere el esquema en el caso de los temporales especifican que solo se guardarán imágenes cada diez segundos como mínimo para evitar que se guarden fotogramas prácticamente idénticos, en cuanto al requisito de umbral hace referencia a la certeza con la que la red detecta un objetivo, en un principio se ha configurado un nivel de certeza del 75% pero esto es fácilmente configurable dentro del archivo “reconocedor.py” modificando la variable umbralCaptura.

Por último, es importante destacar que pese a que el bucle es infinito, el algoritmo no se ejecutará siempre, dado que intentamos el mayor paralelismo posible, esta fase se realizará en un thread separado del resto de la ejecución por lo que cuando no se necesite usar el reconocedor (finalización del recorrido) el coordinador que se explicará en la siguiente fase parará el hilo finalizando de esta manera la ejecución del algoritmo.

Imagen que contiene árbol

Descripción generada automáticamente

Fig x.x. Ejemplo de captura proporcionada por el reconocedor.

Etapa 6: Controlador y flujo de funcionamiento.

Una vez explicados los tres bloques principales en los que se divide el funcionamiento del prototipo es necesario hacer mención a como se conectan para poder llegar a funcionar, esta medida la lleva a cabo el controlador.

Este controlador se encuentra implementado dentro del repositorio en la carpeta control con el nombre “controlador.py”

## Ejecución del controlador

Dado que este script va a ser el encargado de ejecutar el resto de archivos es esperable que sea el primero en ejecutarse, para poder realizar esta tarea se ha especificado dicho programa como daemon[[4]](#footnote-4) que es un proceso el cual se ejecuta de forma autónoma en nuestro caso al encenderse la Raspberry y que no requiere una interacción humana para funcionar.

Para poder ejecutar este archivo se ha reconfigurado…

## Estructura interna

Declaración de. Variables:…

La estructura interna de el programa comienza por la llamada a la función principal “autorun” cuyas funciones son las siguientes:

* Inicialización del módulo gps: dado que este módulo tarda en conectarse a un satélite unos minutos es lo primero que inicializamos para que mientras esta conexión ocurre podamos hacer otras tareas (punto).
* Llama a la creación del servidor y permanece a la espera hasta la conexión y desconexión del cliente tras recibir los parámetros del vuelo (punto 1).

Etapa 7: Pruebas finales.

Conclusiones.

Bibliografía.

1. <https://github.com/juancalf/TFG>
2. <https://github.com/emlid/emlid-docs/blob/master/docs/autopilots/navio2/img/Navio2WithPaspberryPi.png?raw=true>
3. <https://emlid.com/navio/#navio-specs>
4. <https://www.prusa3d.es/original-prusa-i3-mk3-spa/>
5. <https://github.com/juancalf/TFG/tree/master/archivos%203d>
6. <https://www.thingiverse.com/thing:3723178>
7. <https://www.thingiverse.com/thing:92208>
8. <https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/>
9. <https://es.freelogodesign.org/terms-of-use>

\*para añadir a la bibliografía usar referencias cruzadas.

1. Power distribution board [↑](#footnote-ref-1)
2. Secure Shell. [↑](#footnote-ref-2)
3. mAP: mean average Precision. [↑](#footnote-ref-3)
4. Disk and execution monitor. [↑](#footnote-ref-4)