Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Estudi sobre la viabilitat de diferents tecnologies per un ecosistema de drons

TITULACIÓ: Grau en Enginyeria d'Aeroports

AUTOR: Guillem Lloveras Ortega

DIRECTOR: Miguel Valero Garcia

DATA: 8 de setembre del 2022

Resum

Aquest document conté tota la informació rellevant sobre el treball portat a terme per mi, amb l'ajuda del meu tutor; Miguel Valero.

Cal tenir en compte que es tracta d'un estudi sobre diferents tecnologies que engloben un ecosistema en formació, per "drons".

L'objectiu principal ha sigut poder executar un pla de vol, definint per quins punts es vol passar, i en quins es vol fer una fotografia per reconèixer objectes. S'ha desenvolupat tot el projecte amb Python. Primerament, s'ha definit, un sistema de comunicació entre mòduls de l'ecosistema mitjançant la tecnologia proporcionada per broker MQTT, servidor que distribueix la informació entre els clients subscrits a tòpics determinats. Per poder mostrar el mapa i marcar els punts per on es vol fer volar el dron s'ha fet servir la llibreria "Tkinter".

Finalment, per la detecció d'objectes s'ha estudiat diferents tecnologies, entre elles, els classificadors proporcionats per OpenCV, una xarxa neuronal convolucional programada des de 0 basada en la llibreria Keras, un entrenador de classificadors fet per OpenCV i un detector de colors.

Com a conclusió final, destacar que s'ha assolit la majoria d'objectius marcats, tot i que el desenvolupament de l'aplicació mòbil no ha sigut un èxit, s'ha pogut completar la missió de vol del dron, creant un pla de vol totalment a gust, marcant en quins punts s'ha volgut fer una fotografia, detectant l'objecte desitjat i executant el vol sense problemes.

PARAULES CLAU: drons, ecosistema, Python, intel·ligència artificial, xarxes neuronals, programació

ÍNDEX

ÍNDEX DE FIGURES

0. INTRODUCCIÓ

L'objectiu d'aquest document és explicar i donar a conèixer tota la informació rellevant del projecte així com els passos i processos que s'han anat seguint.

En primer lloc, cal dir que el treball tracta sobre l'estudi de diferents tecnologies o eines per poder ser aplicades en un ecosistema per drons, és a dir, en un entorn construït per un seguit d'aplicacions, on cadascuna d'elles té una funció diferent, i totes juntes constitueixen un sistema de relacions entre si.

Breument, s'ha definit un estàndard de sistema de comunicació entre els diferents mòduls de l'ecosistema del projecte. Addicionalment, s'ha programat una nova opció dins del mòdul del Dashboard (aplicació per ordinador) per poder definir un pla de vol, és a dir, marcar per quins punts vols que voli el dron ("waypoints") i que s'executi automàticament. A part, s'ha estudiat diferents models per poder detectar objectes, donant així, l'opció de triar en quins punts de la ruta del dron es vol fer una fotografia per poder detectar, per exemple, pilotes de tenis.

Aquest treball de final de grau bé motivat per l'admiració al gran potencial de la intel·ligència artificial, intentant així, adquirir coneixements bàsics sobre aquest camp totalment nou per mi. També, gran part de la motivació ha sigut per la meva afició als drons, ja que per exemple, el treball de recerca que vaig presentar a l'acabar batxillerat va ser construir des de 0 i fer volar un dron de carreres. Cal dir que l'empenta final per portar endavant un treball com aquest, on totes les tecnologies amb què s'han treballat eren noves per mi, ha sigut la base sòlida adquirida de Python i la seguretat que m'ha proporcionat l'any que porto treballant com a enginyer de software.

Addicionalment, cal aclarir per sobre com està estructurat aquest treball. S'ha començat, en el capítol 1, explicant com era l'ecosistema en un principi, seguit en l'apartat 2, dels objectius a assolir en el projecte. En el capítol 3, es defineix el sistema de comunicació. Pel que fa al capítol 4, detallaa el desenvolupament portat a terme per poder definir un pla de vol. En l'apartat 5 es conta com poder reconèixer diferents objectes. En el capítol 6 es parla del conjunt d'elements o tecnologies escollides per poder realitzar la missió. La prova final executada s'explica en l'apartat 7, els problemes principals trobats al llarg del projecte es mostren en el capítol 8 i les conclusions arribades formen part del capítol 9.

Finalment, si es desitja consultar el codi sencer del projecte cal visitar el següent enllaç:
https://github.com/glloveras/DroneEcosystem

1. DESCRIPCIÓ DEL ECOSISTEMA INICIAL

L'ecosistema del dron és una eina de software que ens permet, amb diferents tipus de tecnologies, controlar el funcionament d'un o més vehicles no tripulats, també ens permet poder definir i comandar diferents funcionalitats explicades amb més detall al capítol següent.

L'ecosistema, inicialment estava constituït per diferents mòduls, tal com es pot veure en l'esquema de la Figura 1. Cada un d'aquest mòdul està detallat breument a continuació.

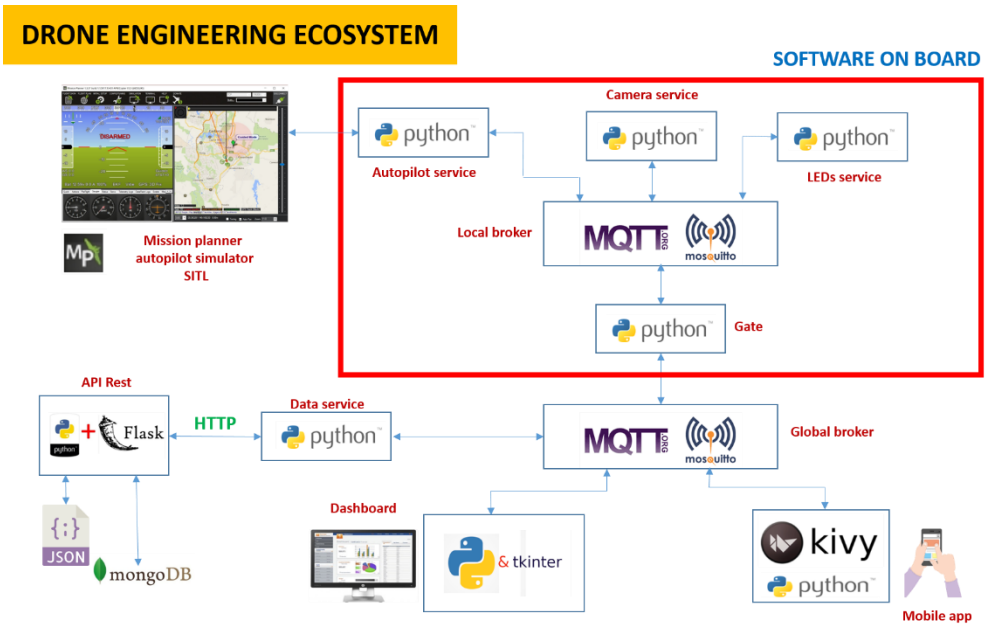


Figura 1: Esquema del ecosistema en el punt inicial. [8]

1.1. Mòduls del ecosistema

1.1.1. Tauler "Dashboard"

És l'aplicació de l'escriptori de l'ordinador amb la qual es pot controlar globalment el sistema. Permet connectar-te al don per tal d'executar les diferents funcionalitats, així com també et permet gestionar les dades implicades en la missió de vol.

El mòdul està implementat en el llenguatge de programació Python. Fa servir una llibreria que es diu "Tkinter" la qual serveix per al kit d'eines de GUI ("graphical user interface"), és a dir, permet el desenvolupament d'interfícies gràfiques per l'usuari, utilitzant un conjunt d'objectes gràfics, com per exemple, un icona, per poder representar les accions i la informació de la interfície.

1.1.2. Aplicació per el mòbil

És una aplicació per “smartphones”(mòbils intel·ligents) que implementa, alguna de les funcionalitats del “dashboard”.
En aquest cas, el mòdul està implementat en llenguatge Python, però fent servir una altra framework[1] anomenat Kivy, la qual et permet desenvolupar aplicacions amb propietats multitàctils, interfícies d’usuari i moltes altres funcionalitats.

1.1.3. Serveis

Pel que fa al software a bord, existeixen 3 serveis bàsics, cada un independent de l’altre.

1.1.3.1. Auto-pilot

Es pot definir com un mòdul a bord del dron que executa les ordres que li arriben des de el dashboard o l’aplicació mòbil. Aquestes comandes, en un inici son: l’enlairament o aterrar, anar a la posició especificada i obtenir dades del dron (velocitat, direcció, coordenades actuals, etc).

1.1.3.2. Servei de càmera

És el següent mòdul integrat al dron que també executar les ordres com fer una foto o obtenir la imatge en directe en format de vídeo. Aquestes comandes, com l’autopilot li arriben, tant del dashboard com de l’aplicació mòbil.

1.1.3.3. Servei de LEDs

Mòdul instal·lat a bord, amb l’objectiu de controlar les llums LEDs del dron per informar el seu estat i que es puguin fer servir per a altres mòduls en un futur. Igual que els altres dos serveis, rep les ordres del dashboard o de l’aplicació mòbil.

1.1.4. Porta “Gate”

Bàsicament, fa el que diu el nom, és a dir connecta, tot i que també és podria dir que obre les portes, entre els mòduls externs i els que estan a bord del dron. És qui connecta el “broker” (s’explica amb més detall seguidament) global amb el local que es troba instal·lat al dron.

1.1.5. Corredors “Brokers”

Existeixen dos brokers en l’ecosistema, el local i el global. Els dos són middleware[2] que permeten la comunicació entre sistemes i utilitzen el protocol

[1] Es podria dir que un framework és una mena de plantilla que serveix com a punt de partida per a l'organització i desenvolupament d’un software [1].
[2] Software que funciona com una capa de traducció oculta per permetre la comunicació i la gestió de dades entre aplicacions [2].

"Machine to Machine" MQTT amb el servidor Mosquitto, ambdues coses explicades seguidament.

1.1.5.1. Protocol MQTT

Tal com ens explica la pròpia pàgina web d'MQTT [3], és un dels protocols principals dels dispositius IoT[3] el qual basa el seu servei de missatgeria amb un model publicador-subscriptor (pub-sub), és a dir, els clients es connecten a un servidor central, els denominats brokers, encarregats de filtrar els missatges els quals es disposen en tòpics. Això vol dir que un client, pot fer una publicació amb el contingut que vulgui a un tòpic, i tots els altres clients que estiguin subscrits aquell tòpic, el broker els hi farà arribar el missatge (estructura del missatge explicada amb més detall al següent capítol). Veure Figura 2 sobre l'esquema del model pub-sub.

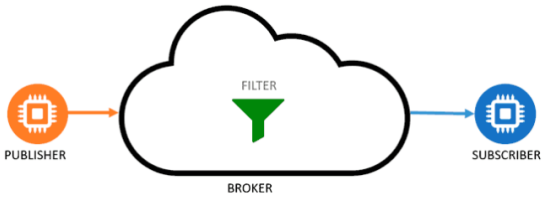


Figura 2: Model MQTT. Font [5]

1.1.5.2. Servidor Mosquitto

Mosquitto veure [6], és un broker MQTT OpenSource, és a dir, d'accés públic molt utilitzat. És ideal perquè es pot fer servir amb tota mena de dispositius, des dels més bàsics, fins a servidors complets. Tal com s'ha comentat anteriorment, funciona molt bé per la missatgeria en sistemes IoT, ja que fa servir un mètode molt lleuger, utilitzant el model pub-sub. Veure Figura 3.

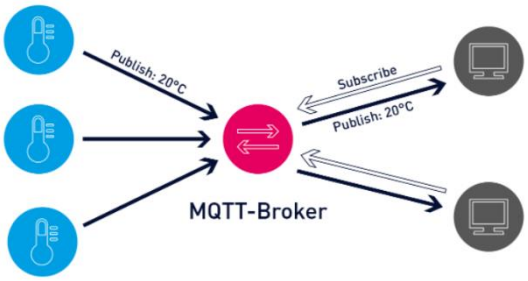


Figura 3: Exemple de sistema IoT amb broker MQTT. Font [7].

[3] És l'agrupació i la interconnexió de dispositius i objectes a través d'una xarxa (bé sigui privada o Internet), on tots podrien ser visibles i interaccionar entre ells, màquina a màquina [4].

1.2. Definició de la instal·lació i requisits de l’ecosistema

Addicionalment, en una primera instància i per tota aquella gent, tant estudiants, com persones externes, que volguessin treballar en l'ecosistema, s'ha definit una guia bàsica, per la correcta instal·lació de totes les eines necessàries per poder fer servir l'entorn de treball. Així com també s'ha definit un procediment per poder fer contribucions al projecte. Tota aquesta guia es pot trobar en el següent enllaç [8].

2. OBJECTIUS

L'objectiu principal del projecte és poder crear un ecosistema per drons estable, on es pugui fer servir diferents tecnologies o softwares capaços de comunicar-se entre si.
Per tal d'assolir aquesta fita, primer de tot s'ha definit les funcionalitats principals que es volen tenir, és a dir, les accions principals que ha de fer el dron.

2.1. Dasboard

Així doncs, pel que fa al dashboard s'ha marcat com a objectiu les següents funcionalitats:

- Poder definir un pla de vol punt a punt, és a dir, marcar una ruta amb tots els punts per on vols que el dron sobrevoli.
- Poder indicar en quin, de tots els punts definits, vols que el dron faci una fotografia
- Processar les fotografies del dron i reconèixer algun objecte definit anteriorment. Per exemple, si en un dels punts hi ha una persona.

2.2. Aplicació mòbil

Altrament, pel que fa a la part de l'aplicació mòbil, s'ha especificat les metes següents:

- Poder posar en marxa el dron, juntament amb el pla de vol definit al dashboard.
- Poder rebre les fotos i les coordenades dels punts on s'ha reconegut l'objecte desitjat.

2.3. Altres objectius

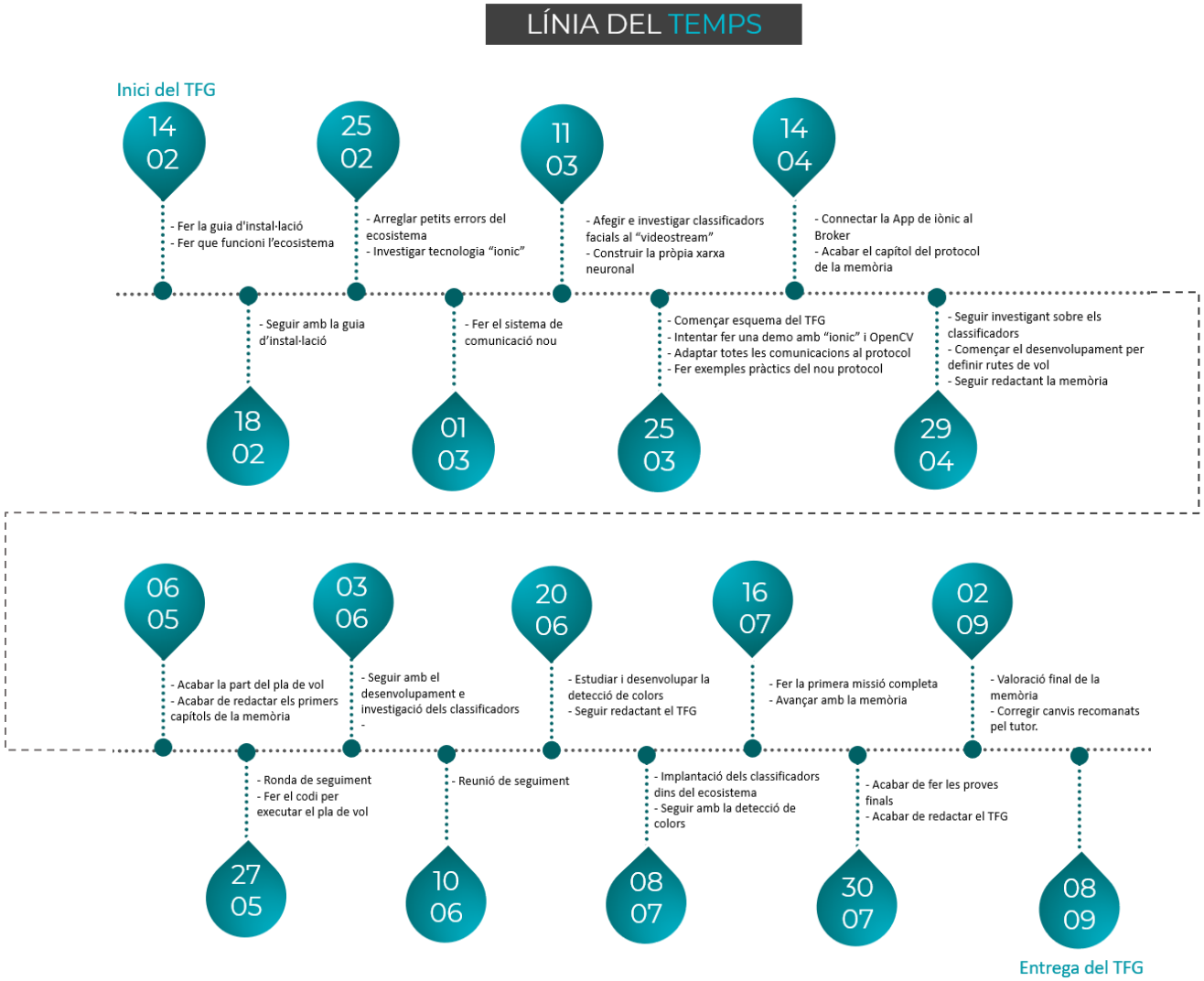
A part, com a objectius complementaris s'han marcat els següents:

- Poder definir un protocol de comunicació estàndard per tot el projecte, clar i concís.
- Poder analitzar diferents alternatives per dibuixar el pla de vol, és a dir, definir els punts per on vols que el dron passi.
- Poder fer un estudi de diversos mètodes per implementar xarxes neuronals, capaces de reconèixer objectes.
- Poder investigar diferents tecnologies per crear una aplicació mòbil que permeti complir les funcionalitats especificades en l'apartat 2.2.

2.4. Organització del treball

Per tal d'assolir els objectius fixats, s'ha fet un seguiment fluït i constant de les tasques a desenvolupar. Tal com es mostra a continuació, en el diagrama temporal, s'ha mantingut una comunicació fluida i propera amb el tutor.

Cal agrair al Sr. Miguel que sempre ha estat disponible per qualsevol dubte i per ajudar-me en qualsevol moment.



3. DEFINICIÓ DEL SISTEMA DE COMUNICACIÓ

Tal com s'ha explicat en l'apartat 1.1.5. Corredors "brokers", entre la comunicació es fa servir el servei de missatgeria del model publicador-subscriptor.

És a dir, un servidor central, que en MQTT es denomina broker, rep els missatges, els filtra i els distribueix. Aquest filtre discriminador és el que es coneix com a tòpic, el qual simplement és una cadena de text. Cada un d'aquests tòpics pot estar format per un o més nivells separats entre si per una barra inclinada '/'. Per exemple, un tòpic vàlid podria ser:

- Escola/Classe/Taules

3.1. Wildcards

A més a més, existeix l'opció de subscriure't a diversos tòpics, és a dir, escoltar (fent referència a rebre) tots els missatges que es publiquen a cada un dels tòpics subscrits. Això s'aconsegueix usant els denominats "wildcards", per MQTT existeixen només dos.

3.1.1. Wildcard d'un únic nivell

És el caràcter **+**, i només pot substituir un únic subnivell, per exemple:

- Escola/Classe/+

Serviria per rebre els missatges de:

- Escola/Classe/Cadira
- Escola/Classe/Taula
- Escola/Classe/Pissarra

Però no serviria per:

- Escola/Classe/Taula/Verda
- Escola/Classe/Cadira/Trencada

Ja que en aquests casos, estaria substituït més d'un nivell i per poder fer això, existeix el wildcard explicat tot seguit.

3.1.2. Wildcard de diversos nivells

És el caràcter # i serveix per reemplaçar uns quants nivells. Per exemple:

- Escola/#

Serviria per escoltar missatges de tòpics com:

- Escola/Classe/Cadira/Trencada
- Escola/Professors/Masculins
- Escola/Alumnes
- Escola/Alumnes/Ulleres

3.2. Flux dels missatges dins de l'ecosistema

3.2.1. Estructura del tòpic

En aquest projecte, s'ha definit un protocol d'estructura del tòpic, per tal de poder facilitar als altres contribuïdors de l'ecosistema, una homogeneïtat a l'hora de treballar en desenvolupaments posteriors.

El missatge de publicació s'ha dividit en tres parts:

- Des d'on s'envia el missatge.
- A qui es dirigeix el missatge.
- Acció que es vol dur a terme.

És a dir, en general, és quelcom tal que així:

- client.publish(**"OnVaig/OnEstic/Acció"**)

Per posar un exemple pràctic de l'ecosistema, existeix dins del mòdul dels LEDs del dron, una opció per engegar un a seqüència de LEDS.

Així doncs, des del dashboard, en el codi del botó on es vol clicar per donar l'ordre, es posa la següent publicació.

- client.publish(**"dashBoard/LEDsService/startLEDsSequence"**)

3.2.2. Connexió i camins de les publicacions

El primer que es fa és connectar els mòduls externs amb els que estan a bord. Cal entendre que no es connecten directament, per això esta la gate, ja que uns estan connectats al broker global, mentre que el dron està connectat al broker local.

Quan des del dashboard es dona l'ordre de connectar, internament s'està seguint el següent flux:

- Publicació des del dasboard a la gate ("dashBoard/gate/connectPlatform").
- Subscripció a tots els tòpics que tinguin com a destí el dashboard ("+/dashBoard/#").
- La gate es connecta al broker global i es subscriu als tòpics que estigui ella com a destí i tingui l'acció de connectar-se a la plataforma ("+/gate/connectPlatform").
- A part, quan li arriba el missatge de connectar del dashboard, es connecta al broker local i fa les publicacions de connectar als mòduls interns:
    - ("gate/LEDsService/connectPlatform")
    - ("gate/cameraService/connectPlatform")
    - ("gate/autopilotService/connectPlatform")
- A més a més es subscriu localment a tots els tòpics que tinguin com a destí el dashboard o la app del movil.

- Finalment subscriu el broker global a tots els tòpics de cada un dels mòduls:
    - (dashBoard/LEDsService/#")
    - (dashBoard/cameraService/#")
    - (dashBoard/autopilotService/#")
- Pel que fa a, cada un dels mòduls a bord, es connecten al broker local, i quan reben el missatge de connectar-se, es subscriuen a tots els tòpics del seu servei, per així poder rebre, els missatges de tornada del dashboard ('+/autopilotService/#')

3.3. Exemple pràctic

Aquest exemple tracta d'afegir un nou servei en l'ecosistema. Aquest nou servei es molt simple, mostrar la radiació de l'aire quan es demana en el dashboard. A part, també serveix per acabar d'entre el funcionament bàsic de la comunicació entre mòduls.

Primer de tot, tal com mostra la Figura 4, s'ha creat la part del dashboard, és a dir el botó que en clicar, t'ha de mostrar la radiació. Tal com ens indica la fletxa, la comanda que executa el botó és una publicació dirigida al servei de radiació ("dashBoard/radiationService/getRadiation").

23 ■■■■■ Dash&gate&Camera&LED/Dashboard.py

```
376 + # Radiation control frame ----------------------
377 + radiationControlFrame = tk.LabelFrame(topFrame, text="Radiation control", padx=5, pady=5)
378 + radiationControlFrame.pack(padx=20, pady=20);
379 +
380 + v4 = tk.StringVar()
381 + s1r9= tk.Radiobutton(radiationControlFrame,text="Radiation value", variable=v1, value=4).grid(column=0, row=1, columnspan = 5, sticky=tk.W)
382 + v4.set(1)
383 +
384 + def radiationGetButtonClicked():
385 +     if v4.get() == "1":
386 +         client.publish("dashBoard/radiationService/getRadiation")
387 +
388 + radiationGetButton = tk.Button(radiationControlFrame, text="Get", bg='red', fg="white", width = 10, height=3, command=radiationGetButtonClicked)
389 + radiationGetButton.grid(column=5, row=0, columnspan=2, rowspan = 3, padx=10)
390 +
391 + lbl = tk.Label(radiationControlFrame, text=" ", width = 10, borderwidth=2, relief="sunken")
392 + lbl.grid(column=7, row=1,  columnspan=2 )
393 +
```

Figura 4: Codi del dashboard de la part de radiació. Font pròpia

Tot seguit, tal com mostra la Figura 5, s'ha creat el codi del servei de radiació. S'ha tingut en compte que primer de tot és necessari connectar-se al broker local i subscriure's al tòpic de connectar la plataforma, per poder rebre els missatges de la gate.

A més a més, s'ha subscrit a tots els tòpics que tinguin a destí el mateix servei, perquè així, pugui rebre la publicació del dashboard de "getRadiation", pugui calcular la radiació i fer una publicació cap al dashboard informant d'aquesta radiació ("radiationService/dashBoard/Radiation").

48 Dash&gate&Camera&LED/RadiationService.py

```
@@ -0,0 +1,48 @@
+ import random
+ import paho.mqtt.client as mqtt
+
+ def getRadiation():
+     rad = round(random.uniform(0, 1), 4)
+     return rad
+
+ def on_message(client, userdata, message):
+     splited = message.topic.split('/')
+     origin = splited[0]
+     destination = splited[1]
+     command = splited[2]
+
+     if command == 'connectPlatform':
+         print ('Radiation service connected by ' + origin)
+         client.subscribe('+/radiationService/#')
+
+     if command == 'getRadiation':
+         rad = getRadiation()
+         # The Equivalent Dose is the magnitude used to express the amount of energy deposited per unit mass (absorbed dose)
+         # and the type of radiation that supplies said energy.
+         # This magnitude is also measured in J/Kg, but is called Sievert ( Sw).
+         print('Radiation is: '+str(rad)+' mSv')
+         client.publish('radiationService/' + origin + '/Radiation', rad)
+
+ local_broker_address =  "127.0.0.1"
+ local_broker_port = 1883
+ LEDSequenceOn = False
+
+ client = mqtt.Client("Radiation service")
+ client.on_message = on_message
+ client.connect(local_broker_address, local_broker_port)
+ client.loop_start()
+ print ('Waiting connection from DASH...')
+ client.subscribe('gate/radiationService/connectPlatform')
+
```

Figura 5: Codi sencer del servei de radiació. Font pròpia.

A part, és necessari editar un tros de codi de la gate, perquè així el nou servei, pugui tenir la interacció entre el broker global i el broker local. Veure Figura 6.

```
@@ -38,6 +38,7 @@ def on_global_message(client, userdata, message):
        local_client.publish("gate/LEDsService/connectPlatform")
        local_client.publish("gate/cameraService/connectPlatform")
        local_client.publish("gate/autopilotService/connectPlatform")
+       local_client.publish("gate/radiationService/connectPlatform")

        # Subscribe to commands from services to the module (dash or App) that connected the platform
        local_client.subscribe('+/' + origin + '/#')
@@ -48,6 +49,7 @@ def on_global_message(client, userdata, message):
        global_client.subscribe(origin+'/LEDsService/#')
        global_client.subscribe(origin+'/cameraService/#')
        global_client.subscribe(origin+'/autopilotService/#')
+       global_client.subscribe(origin + '/radiationService/#')
```

Figura 6: Tros de codi editat de la gate. Font pròpia.

Finalment, s'ha d'afegir en el dashboard, dins de la funció de quan rep un missatge, el tros de codi per mostrar la radiació que envia el servei. Veure figura 7.

```
@@ -87,6 +91,11 @@ def on_message(client, userdata, message):
91
92              table.pack()
93
94  +   if origin == "radiationService":
95  +       if (command == "Radiation"):
96  +           answer = str(message.payload.decode("utf-8"))
97  +           lbl['text'] = answer[:5] + ' mSv'
98  +
```

Figura 7: Codi del missatge de tornada al dashboard. Font pròpia.

4. GENERACIÓ D'UN PLAN DE VOL

Per tal de poder definir un pla de vol s'ha utilitzat el paquet de python "Tkinter", el qual proporciona un conjunt d'eines per tal de poder desenvolupar aplicacions d'escriptori multiplataforma. És dels mòduls més coneguts i més utilitzats per crear interfícies gràfiques per usuaris, ja que és senzill i versàtil.

Per tal de poder dibuixar un pla de vol pel dron s'han seguit els següents passos:

- Poder mostrar un mapa didàctic.
- Poder definir punts en aquest mapa.
- Poder mostrar la distància entre els punts.
- Poder definir els punts on es volen fer les fotos.
- Poder enviar i executar la ruta dissenyada.

4.1. Mostrar el mapa

Després d'investigar diferents opcions, s'ha optat per fer servir la base d'una llibreria ja existent que et permet la interacció de mapes [9] a partir d'un servidor, ja que si no, l'altra opció era posar captures d'imatges de mapes, com per exemple, "Google maps", definint així una zona predeterminada amb la qual no seria possible moure'ns més enllà, és a dir, no ens podríem desplaçar cap als costats.

A més a més, i molt important, ja té la funció de càlcul implementada que et permet passar de número de píxel on el ratolí és clicat, a coordenades decimals del mapa real.

Així doncs, s'ha creat un quadre nou dins de l'aplicació inicial on es mostra el mapa tal com es veu a la Figura 7.

Figura 8: Mapa per definir un pla de vol. Font pròpia.

Tal com s'observa a la figura superior, en la part central hi ha el mapa, el qual es pot desplaçar tant com es vulgui, a la part inferior, una barra que permet buscar coordenades, així com també diferents noms de ciutats, a la dreta del tot, hi ha un quadre blanc, on es mostren les coordenades dels punts marcats i alguns botons amb diferents funcionalitats, com per exemple, triar en quins punts vols fer una foto, netejar el pla de vol que has marcat o enviar al "dron" els punts que ha de seguir. El codi es pot veure a l'Annex 1.

Veure el vídeo curt per entendre millor el funcionament:
https://drive.google.com/file/d/1DYPKQ2-Bp48OHO5f7o3pGnZE4eVmxFpv/view?usp=sharing

4.2. Definir els "Waypoints[4]"

Per poder definir els punts en el mapa s'ha associat una funció al clic dret del ratolí, ja que el clic esquerre està ocupat per la funció de desplaçar el mapa.
Exemple per Tkinter per associar funcions al ratolí:

```
- self.canvas = tkinter.Canvas(master=self, width=self.width, height=self.height)
- self.canvas.bind("<Button-3>", self.mouse_right_click)
```

La funció mencionada anteriorment (veure codi al Annex 2) obra un petit menú a la pantalla, veure Figura 9, on es mostra el píxel clicat, la coordenada corresponent, en decimals i l'ordre d'afegir el punt clicat.

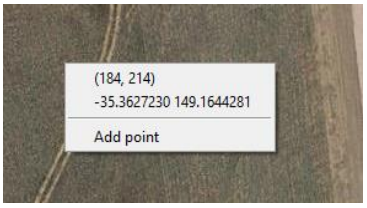


Figura 9: Menú desplegat en fer clic dret al ratolí. Font pròpia.

Addicionalment, s'ha creat una ordre per afegir el punt en el pla de vol. Bàsicament, comprova si és el primer punt de la ruta, si ho és, se'l guarda, ja que també és l'últim i ho mostra al quadre de punts, que es troba al costat dret del mapa. A més a més, afegeix una marca al mapa al punt clicat i, si no és el primer punt, dibuixa una línia que mostra el recorregut que seguirà el dron (Figura 9).

Veure codi a l'Annex 3.

[4] Un "waypoint" és un punt o lloc intermedi d'una ruta o línia de viatge, un punt d'aturada en què es canvia el rumb

4.3. Mostrar distància entre punts

Per tal de mostrar la distància real en metres, s'ha procedit a fer el següent càlcul:

1. Agafar dos punts de referència. Veure Figura següent.

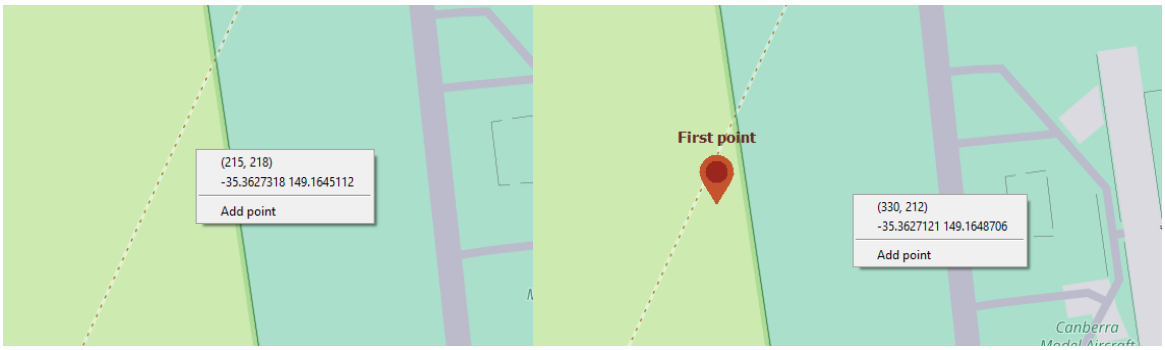


Figura 10: Punts de referència seleccionats al mapa. Font pròpia.

2. Posar els punts al "Google maps" i mostrar la distància en metres. Veure Figura 11.

Figura 11: Distància en metres posant els punts al google maps. Font pròpia.

3. Fer l'operació matemàtica. Calcular la distància entre els punts en píxels. Un cop s'ha obtingut la distància en píxels, fer la regla de tres per saber quants metres equival 1 píxel.

   1) $d = \sqrt{(330-215)^2 + (212-218)^2} = 114{,}84 \cong 115\ píxels$

   2) $d_m = \frac{33m \cdot 1p}{115p} = 0{,}28\ m$

Un cop s'ha obtingut el factor de conversió, s'ha procedit a definir la funció que a l'arrastrar el ratolí amb el botó del mig pres, es mostra una línia amb la distància en metres. Per poder-ho fer, anteriorment s'ha hagut de modificar la funció del

clic dret del ratolí, perquè en clicar, et creí una línia d'un píxel, com si fos un punt. Així doncs, a l'arrastrar el ratolí amb el botó del mig clicat, es modifica la línia ja existent, així pots visualitzar la distància mentre es mou el ratolí. Si no, el que passaria és que s'anirien creant línies infinites.

Visitar l'enllaç per veure un exemple pràctic:
https://drive.google.com/file/d/1bIHl9RdwZrAyRIzxL0J2ruiow9XVQBpo/view?usp=sharing

Veure el codi visitar Annex 4.

4.4. Definir els punts on es vol fer una fotografia

Primerament, s'han definit dos botons al quadre del mapa (Figura 12) els quals tenen la funció d'afegir o eliminar les coordenades seleccionades a un "set[5]" de python.

S'ha triat fer servir un set, en comptes d'una llista, ja que l'únic objectiu és guardar els punts on es volen fer les fotos, i el set té l'avantatge que no es poden afegir ni eliminar elements duplicats o que no existeixen.

Figura 12: Captura dels botons per afegir o eliminar els punts on es fan fotos. Font pròpia.

El funcionament d'aquests botons és bàsicament, obtenir les coordenades seleccionades del quadre dret del mapa, i afegir-les o eliminar-les de la variable definida com un set, per guardar els "waypoints" on es vol fer la fotografia. Enllaç al vídeo d'exemple:
https://drive.google.com/file/d/1U9cQpGvYhoGhhgcIClipC_0UWPvCDeJ6/view?usp=sharing

Veure el codi dels botons i les funcions a l'Annex 5.

[5] Un set a Python es fa servir per representar un conjunt de dades. Aquest conjunt es una col·lecció desordenada d'elements únics, és a dir, que no es repeteixen [10].

4.5. Executar i enviar el pla de vol

S'ha creat un botó a la part superior de quadre, veure Figura 13, que al ser clicat, recorre la llista on es troben les coordenades dels punts triats, si algun d'aquests punts es troba dins de la llista dels "waypoints" on es vol fer la foto, afegeix el valor 1. Seguidament, transforma aquesta llista a un "string", és a dir, en format de text, fent servir el codificador de "json.dumps()[6]", ja que, finalment, s'envia la informació al autopilot (tal com s'explica a l'apartat anterior) publicant al tòpic següent:

Codi - `client.publish('flightplanService/autopilotService/flightplanpoints', position_string)`

La resta de codi es pot trobar a l'Annex 6.

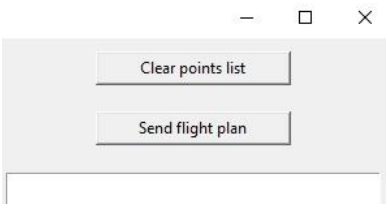


Figura 13: Captura del botó per enviar el pla de vol. Font pròpia.

A continuació es mostra el cos del missatge enviat, com a exemple d'un pla de vol amb 5 punts, on es vol fer una foto al tercer i al quart, ja que com es mostra, tenen un 1 de més:

[-35.36263988957226, 149.1642939453185],
[-35.36289143509654, 149.1640498642978],
[-35.36315829123077, 149.1646882300433, 1],
[-35.36287612365224, 149.1651120190677, 1],
[-35.36263988957226, 149.1642939453185]

Seguidament, s'ha definit la funció al servei d'autopilot per poder executar el pla de vol. Aquesta funció bàsicament descodifica el missatge que li arriba, obtenint així un llistat de punts, i recorre cada un d'aquests punts, tot i que no envia l'orde d'anar al punt següent fins que la distància entre el dron i el "waypoint" no és menor d'un metre.

A més a més, per cada punt comprova si s'ha de fer una fotografia, en cas afirmatiu, publica al tòpic mostrat a sota perquè la càmera faci l'acció.

Tòpic - `client.publish('autopilotService/cameraService/takePicture')`

[6] JSON es un format independent del llenguatge de programació que es fa servir per crear i emmagatzemar estructures de dades. Per crear una cadena de caràcters en format JSON es fa servir la funció "dumps". [11]

Cal comentar que el procés d'execució del pla de vol, es fa mitjançant un "threat" de python, és a dir, un subprocés independent dins del procés principal, per tal de no bloquejar-lo, mentre s'executa. Això es defineix quan al autopilot li arriba el missatge del pla de vol, veure codi següent:

```
if command == "flightplanpoints":
    if not recived:
        recived = True
        print("flight plan recived")
        points_list_json = message.payload.decode("utf-8")
        print ('Arming the drone')
        arm()
        if vehicle.armed == True:
            print ('Drone armed')
            w = threading.Thread(target=executeFlightPlan, args=[points_list_json, origin])
            w.start()
            timer = threading.Timer(1, sendPosition)
            sendPosition()
```

Veure Annex 7 pel codi del autopilot que executa el pla de vol.

5. RECONEIXEMENT D'OBJECTES

Tal com s'explica a la web de l'empresa SAS [12] i el projecte de final de grau de la universitat de Sevilla sobre xarxes de convolucions [13], la intel·ligència artificial és la replicació de les màquines al comportament i el raonament humà, és a dir, robots o sistemes que intenten aprendre de la seva pròpia experiència, millorar i ajustar-se a noves aportacions.

Aquest camp engloba l'aprenentatge automàtic ("Machine Learning"), la qual estudia com dotar a les màquines la capacitat d'aprendre per si soles, i l'aprenentatge profund ("Deep Learning"), algoritmes automàtics que emulen xarxes neuronals que aprenen de grans quantitats de dades. Veure Figura 14.

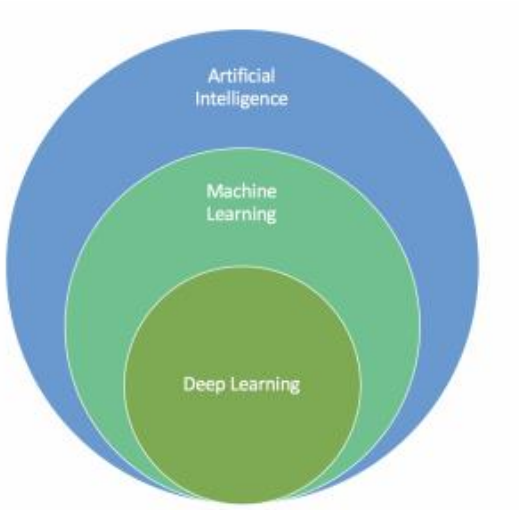


Figura 14: Esquema jeràrquic sobre intel·ligència artificial. Font: [13]

5.1. Xarxes Neuronals Artificials

Les xarxes neuronals, referència [14], són models matemàtics que simulen la interconnexió i el funcionament de neurones biològiques, les quals basen el seu aprenentatge a diferents senyals generats a la sortida, segons els estímuls que reben al llarg del temps de diferents senyals d'entrada.

Aquestes xarxes neuronals artificials es fan servir per aproximar funcions que poden dependre d'una gran quantitat de dades a l'entrada, les quals no tenen res a veure entre si.

La seva distribució de neurones dins de la xarxa és molt bàsica, tal com es mostra a la Figura 15, tenen tres capes que es poden distingir perfectament:

- **Entrada:** És la capa que rep directament la informació de l'exterior.

- **Neurones ocultes:** Són les capes internes a la xarxa, les quals estan interconnectades entre elles de diferents maneres, aquest tipus de connexió, juntament amb el seu número, determinen la tipologia de la xarxa neuronal. També són les que no tenen contacte directe amb l'exterior.
- **Sortida:** Son les que reben la informació i la traspassen l'exterior.

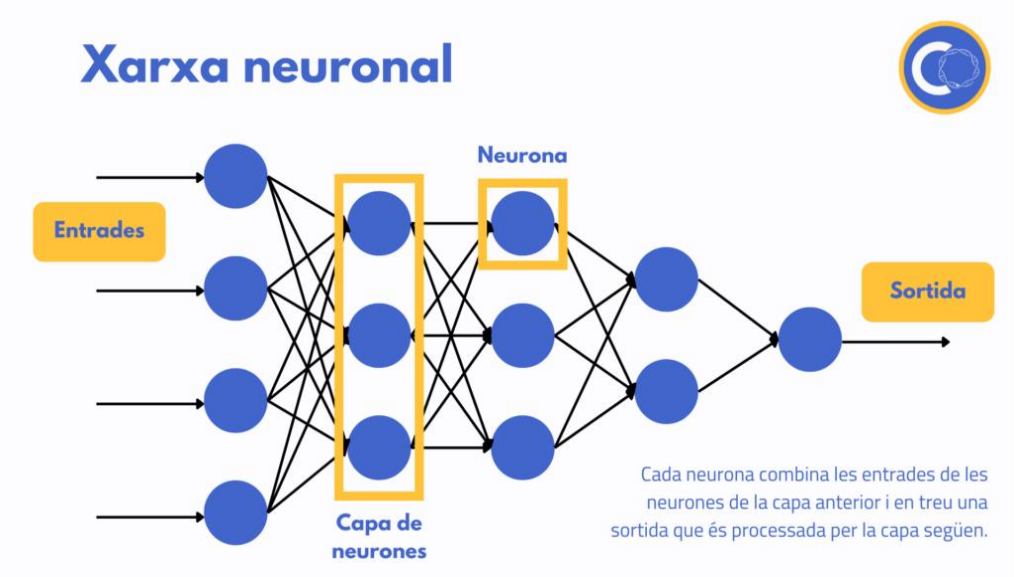


Figura 15: Esquema d'una xarxa neuronal. Font: [15]

5.1.1. Tipologia de xarxes neuronals

Les següents arquitectures venen definides segons com s'interconnecten les neurones neuronals entre si [13]:

- **Perceptrón:** És la més simple i la més bàsica. Sol ser la base pels models més avançats del "deep learning". S'utilitza en problemes de classificació, ja que es necessita donar etiquetes de referència a les dades d'entrada. Aquests valors d'entrada es multipliquen per un pes, es sumen i s'envien directament a la sortida, on es mira si està per sota o per sobre d'un cert llindar, que determina la sortida final.

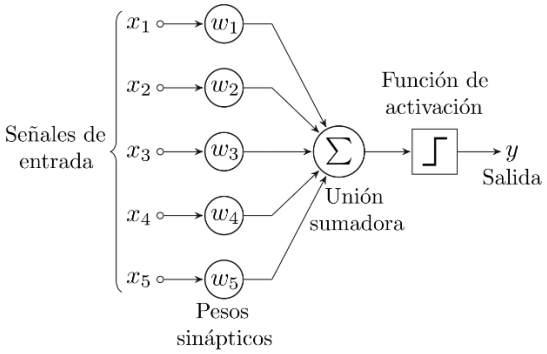


Figura 16: Xarxa neuronal Perceptrón. Font:[16]

- **Perceptrón multicapa:** Similar a l'anterior, però aquest cas, té multicapes interiors que es connecten entre si formant una xarxa neuronal que es retro alimenta.

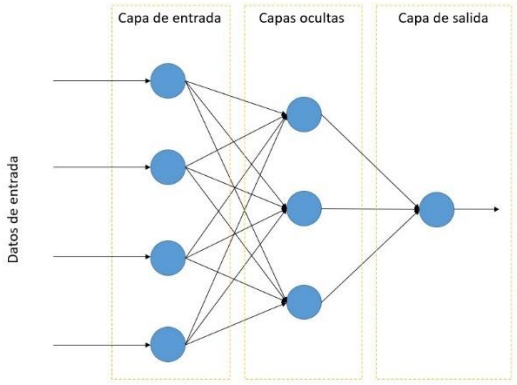


Figura 17: Xarxa Perceptrón multicapa. Font: [17]

- **Xarxes Recurrents (RNN):** Són les més utilitzades pel reconeixement de veu i d'escriptura. Això és gràcies al fet que les connexions entre les neurones formen un cicle dirigit, és a dir, cada seqüència segueix un camí que esta completament determinat pel conjunt d'arestes que s'ha usat, donant una aparença de seguir un ordre específic.

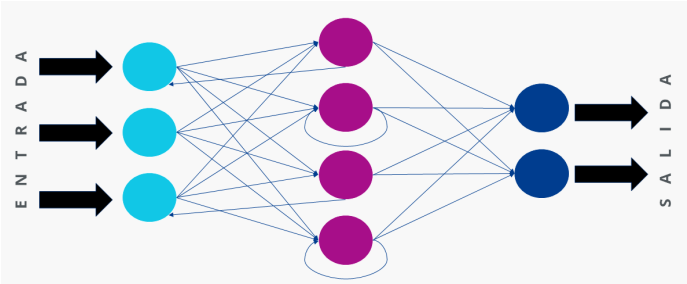


Figura 18: Xarxa Neuronal Recurrent. Font: [18]

- **Xarxes de Convolucions (CNN):** Són les principals a l'hora de processar imatges per ordinadors. Aquestes intenten imitar l'estructura de l'escorça visual primària del cervell, la qual s'encarrega de descodificar la percepció i convertir-la a visió. La xarxa està formada per multicapes de filtres de convolucions d'una o més dimensions, per tant, les neurones només estan connectades a una petita regió de la capa anterior.

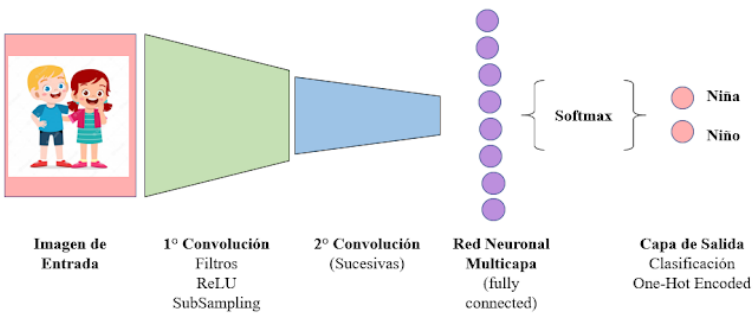


Figura 19: Xarxa Neuronal de Convolucions. Font: [19]

5.2. Programació d'una CNN en "Python"

Les CNN són les més adequades per obtenir resultats en el reconeixement d'imatges. Tant com per processar imatges, com per extreure la informació, la tècnica més utilitzada és la convolució[7].

Així doncs, una CNN no és res més que un tipus de xarxa neuronal que es caracteritza per aplicar una capa basada en la convolució aplicada a una imatge.

5.2.1. Arquitectura d'una CNN

5.2.1.1. Capa convolucional "Convolutional layer"

És la capa principal, tal com indica el nom. Bàsicament es un conjunt de filtres, que es poden entrenar, que es propaguen al llarg de tota la profunditat del volum d'entrada i va generant un mapa de característiques [13].

Per tant, cada píxel nou que es vagi a generar es calcularà col·locant una matriu de números, el filtre o "Kernel" sobre el píxel original, es multiplicarà i sumarà també el valor dels píxels veïns al principal. Veure Figura 20.

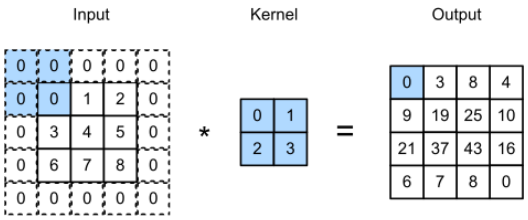


Figura 20: Exemple amb números d'una capa convolucional. Font: [21]

5.2.1.2. Capa d'agrupació "Pooling layer"

Segons la font [13], entre cada capa de convolució, és comú posar una capa d'agrupació. Així doncs, el que fa és agrupar en una imatge el mapa de característiques que ha generat la capa de convolució i reduir la dimensió per evitar un sobre ajust del model.

Bàsicament, simplifica la informació a la sortida de la capa convolucional. Per poder reduir la quantitat de paràmetres a una xarxa es fa servir l'operació "pooling", precisament, la que ha donat millors resultats és la "max-pooling" [13], la qual fa a l'agrupació agafant el valor màxim. Veure Figura 21.

[7] Es un operador matemàtic que transforma dues funcions f i g en una tercera funció que en cert sentit representa la magnitud en què se superposen f i una versió traslladada i invertida de g [20].

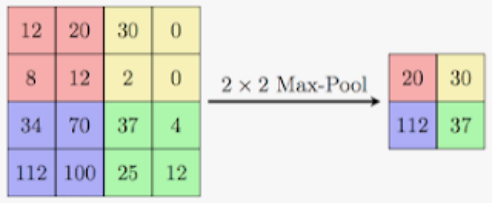


Figura 21: Max-pooling filtre 2x2. Font: [22]

5.2.1.3. Capa densa o completament connectada ("Dense layer")

Aquesta capa és igual a qualsevol altra d'una xarxa neuronal clàssica. La funció principal és agrupar completament que s'ha obtingut fins al moment, perquè serveixi com a càlculs per classificacions posteriors.

L'únic paràmetre que es pot configurar és el nombre de neurones que la formen [13].

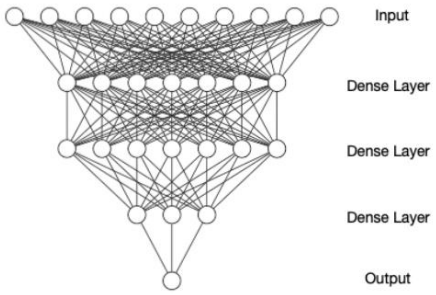


Figura 22: Conjunt de capes denses. Font: [23]

5.2.2. Explicació del codi de la CNN en python

Primer de tot cal mencionar que s'ha fet servir la llibreria Keras que proporciona accés a les funcionalitats de la llibreria Tensorflow [24], que es fa servir per crear a un nivell molt senzill, models de deep learning.

Fent una combinació de les dues es pot definir tots els elements que componen una xarxa neuronal.

Tot seguit es procedeix a fer una numeració dels passos seguits:

1. Importar les llibreries necessàries. Codi:

```
import tensorflow as tf
# Opencv per processar imatges
import cv2 as cv
# Numpy per fer operacions matemàtiques
import numpy as np
# OS per accedir a les imatges del dataset
import os
# TensorBoard per veure el funcionamient de la xarxa
from tensorflow.keras.callbacks import TensorBoard
# ImageDataGenerator per poder modificar las imatges
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2. Aconseguir les direccions de les imatges guardades i definir els paràmetres de les fotos. Codi:

```
## Obtenir i guardar les direccions de les imatges
train_path = 'C:/RedNeuronal/data/train'
vali_path = 'C:/RedNeuronal/data/val'
train_list = os.listdir(train_path)
vali_list = os.listdir(vali_path)

## Altura i amplada de les imatges
width = 200
heigth = 200
```

3. Definir les llistes on s'extreuen les imatges i es guarden juntament amb l'etiqueta respectiva, tant per les fotos d'entrenament com per les de validació. Codi:

```
# Entrenament
labels_train = []
imag_train = []
data_train = []
cont_train = 0
# Validacion
labels_val = []
imag_val = []
data_val = []
cont_val = 0

## Extreure les fotos i guardar-les
for folder_name in train_list:
    name_path = train_path + '/' + folder_name
    folder = os.listdir(name_path)
    for file in folder:
        labels_train.append(cont_train) # Guardar valor de la etiqueta (0 pel primer folder i
1 pel segon)
        img = cv.imread(name_path + '/' + file, 0) # Llegir la imatge
        img = cv.resize(img, (width, heigth), interpolation=cv.INTER_CUBIC) #
Redimensionar la imatge
        img = img.reshape(width, heigth, 1) # Deixar un sol canal
        data_train.append([img, cont_train])
        imag_train.append(img) # Afegir les imatges
    cont_train += 1

for folder_name in vali_list:
    name_path = vali_path + '/' + folder_name
    folder = os.listdir(name_path)
    for file in folder:
        labels_val.append(cont_val)
        img = cv.imread(name_path + '/' + file, 0)
        img = cv.resize(img, (width, heigth), interpolation=cv.INTER_CUBIC)
        img = img.reshape(width, heigth, 1)
        data_val.append([img, cont_val])
        imag_val.append(img)
    cont_val += 1
```

4. Normalitzar les imatges, passar-les a matrius, ja que les xarxes neuronals no treballen amb llistes, i modificar-les aleatòriament per donar un punt de

realisme, és a dir, deformar-les una mica. En la Figura després del codi es mostra un exemple. Codi:

```
## Normalizar les imatges
imag_train = np.array(imag_train).astype(float) / 255
print(imag_train.shape)
imag_val = np.array(imag_val).astype(float) / 255
print(imag_val.shape)
# Pasar les llistas a "arrays" matriu (requisit per les xarxes neuronals)
labels_train = np.array(labels_train)
labels_val = np.array(labels_val)

## Donar realisme a les imatges
# Modificar les imatges aleatoriamente
image_random_gen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=10,
    zoom_range=[0.2, 1],
    horizontal_flip=True,
    vertical_flip=True,
)
image_train_gen = image_random_gen.flow(imag_train, labels_train, batch_size=32)
```

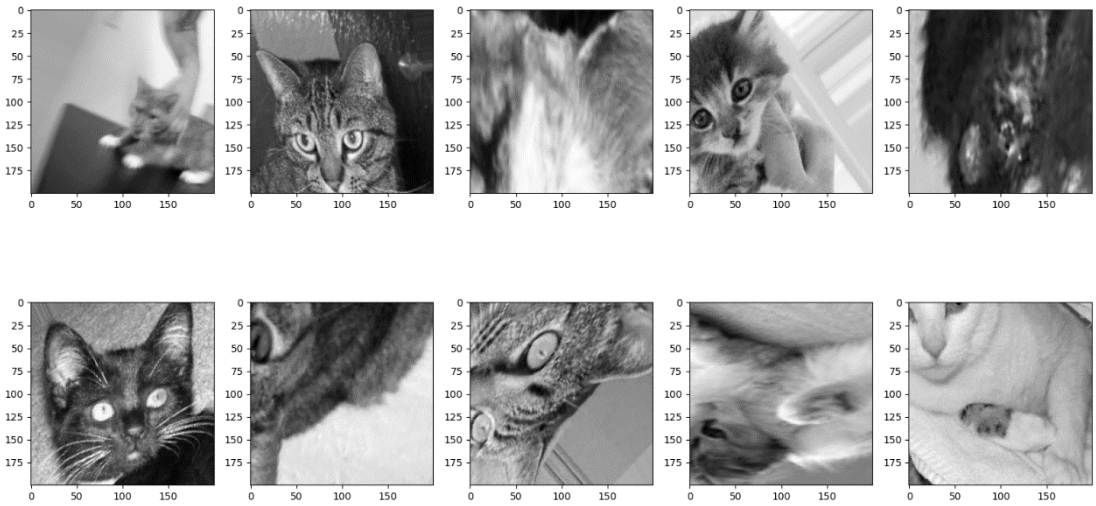


Figura 23: Exemples de fotografies modificades. Font pròpia.

5. Definir la xarxa neuronal amb tres capes de convolució i d'agrupació. Cada una d'elles té activació del tipus "Relu", és a dir, si el valor la sortida és menor que 0, es deixa el valor 0, si no, es deixa el seu valor, això ajuda a una millor classificació. A més a més, s'afegeix una capa de "drop out", perquè per cada cicle d'entrenament nou apagui la meitat de les connexions, evitant així que sempre aprengui el mateix camí. Codi:

```
## Creem un model de xarxa convolucional amb drop out
cnn_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 1)),  #
capa convolucional amb 32 Kernel
    tf.keras.layers.MaxPool2D(2, 2),   # primera capa d'agrupació
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),  # capa convolucional amb 64
```

```
Kernel
    tf.keras.layers.MaxPool2D(2, 2),    # segona capa d'agrupacio
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  # capa convolucional amb 128
Kernel
    tf.keras.layers.MaxPool2D(2, 2),    # tercera capa d'agrupació

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid') # capa de sortida
])
```

6. Finalment es compila el model, afegint una funció d'optimització i una funció de pèrdues i s'entrena la xarxa neuronal. Codi:

```
## Compilem el model
cnn_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

## Entrenar la xarxa neuronal
# tensorboard --logdir="C:/RedNeuronal/board"

board = TensorBoard(log_dir='C:/RedNeuronal/board')
cnn_model.fit(
    image_train_gen,
    batch_size=32,
    validation_data=(imag_val, labels_val),
    epochs=10,
    callbacks=[board],
    steps_per_epoch=int(np.ceil(len(imag_train) / float(32))),
    validation_steps=int(np.ceil(len(imag_val) / float(32)))
)

cnn_model.save('clasificador1.h5')
print(Xarxa Neuronal acabada)
```

7. Un cop obtingut el classificador s'ha fet la prova, en aquest cas, per detectar un gat. S'ha creat un "script" que captura el vídeo de la càmera i el passa pel classificador. El codi es pot trobar a l'Annex 8 i la demostració al següent enllaç: https://drive.google.com/file/d/15kGo9bZ0FwbZlr-KiT5dAF1j21Su6bI3/view?usp=sharing

5.3. Altres mètodes per classificar i detectar objectes

La classificació d'imatges és un tema molt recurrent avui en dia, és per això que existeix multitud d'informació. Pel projecte s'han explorat diverses tecnologies per tenir altres opcions per identificar objectes.

5.3.1. Classificador per colors

Gràcies al meu tutor Miguel, qui m'ha proporcionat la informació necessària per poder estudiar la detecció de colors mitjançant la llibreria Opencv, s'ha fet una prova per detectar pilotes de tenis mitjançant el seu color.

Primer cal esmentar que la detecció de colors es basa en el model HSV, en comptes de fer servir el tradicional RGB. Aquest nou sistema és útil a l'hora de treballar amb informació dels colors. El nom ve de "Hue"(Matriu), "Saturation"(Saturació) i "Value"(valor/brillantor) [25].

- Matriu: fa referència al que normalment es diu color.
- Saturació: representa la puresa de color, és a dir, com més saturació, menys presencia d'altes colors.
- Valor: és la intensitat de la llum, 0% és negre i 100% és el color en màxima esplendor.

Tal com es mostra a la Figura 24, defineix un espai cilíndric de colors.

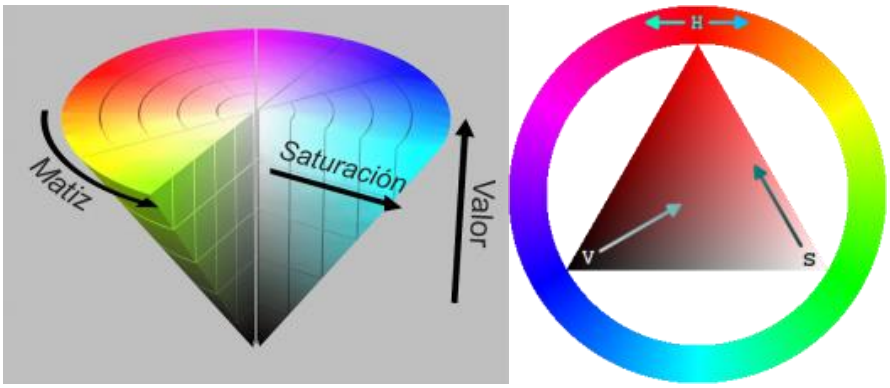


Figura 24: Model de colors HSV. Font: [25]

La prova programada s'ha basat en la segmentació de la imatge amb els seus contorns. Cada un dels passos ha sigut el següent:

1- Fer la imatge més petita i convertir-la en el model de colors HSV. Codi:

```
path_list = ['C:/RedNeuronal/tball_court.jpg', 'C:/RedNeuronal/tballs.jpg']
image_list = [cv.imread(path) for path in path_list]

for img in image_list:
   print('Original Dimensions : ', img.shape)

   scale_percent = 40  # percent of original size
   if img.shape[0] < 600:
      scale_percent = 100
   width = int(img.shape[1] * scale_percent / 100)
   height = int(img.shape[0] * scale_percent / 100)
   dim = (width, height)

   # resize image
   resized = cv.resize(img, dim, interpolation=cv.INTER_AREA)
```

```
print('Resized Dimensions : ', resized.shape)

cv.imshow("Resized image", resized)

copy = resized.copy()
image = cv.cvtColor(copy, cv.COLOR_BGR2HSV)
```

2- Definir el rang pel color groc. Fer servir la funció “cv.InRange()”, la qual retorna una imatge binària (màscara) on els píxels en blanc representen els píxels que es troben dins del rang del groc i els de color negre, els que es troben fora. Codi:

```
yellow_lower_1 = np.array([20, 10, 10])
yellow_upper_1 = np.array([50, 255, 255])
y_mask = cv.inRange(image, yellow_lower_1, yellow_upper_1)
```

Veure Figura 25 com a exemple.

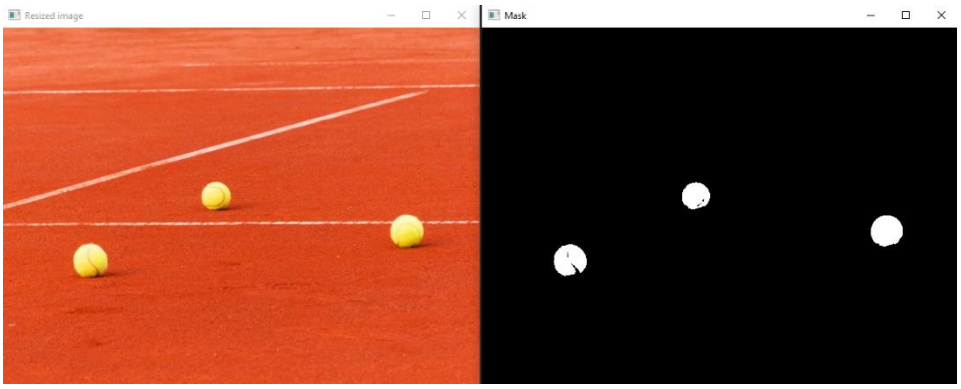


Figura 25: Comparació entre la foto original i la foto filtrada. Font pròpia.

3- Filtrar la màscara fent servir, primer, el que es coneix com a obertura, una erosió, la qual elimina els sorolls blancs petits, i després una dilatació, per corregir la reducció que ha creat l’erosió, segon, un tancament, és a dir, la funció inversa, una dilatació, seguida d’una erosió. Veure Figura 26.

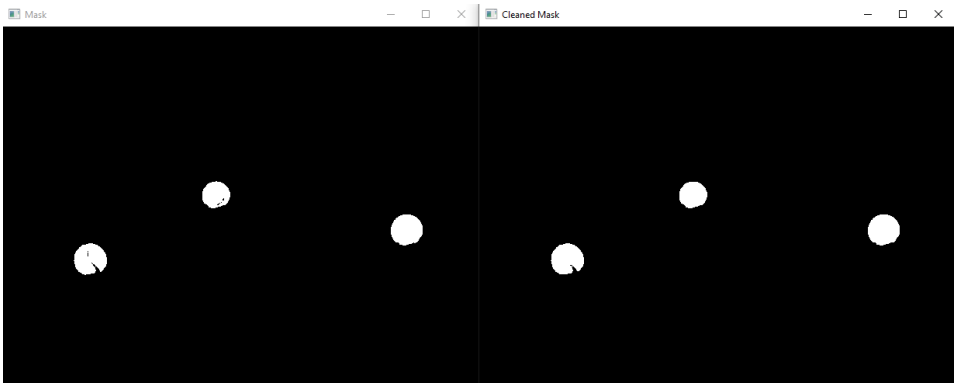


Figura 26: Comparació entre la màscara i la màscara filtrada.

Codi:

```
kernel = np.ones((1, 3), np.uint8)
# this is opening
y_opening = cv.morphologyEx(y_mask, cv.MORPH_OPEN, kernel)
# this is closing
y_clean_mask = cv.morphologyEx(y_opening, cv.MORPH_CLOSE, kernel)
cv.imshow('Mask', y_mask)
cv.imshow("Cleaned Mask", y_clean_mask)
```

4- Finalment, s'ha fet servir la funció "cv.findContours" de la màscara neta, la qual torna els contorns clarament definits i s'ha dibuixat aquests contorns a la imatge original. Veure Figura després del codi.

```
contours, hierarchy = cv.findContours(y_clean_mask, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_NONE)

for contour in contours:
    area = cv.contourArea(contour)
    if area > 300:
        perimeter = cv.arcLength(contour, True)
        approx = cv.approxPolyDP(contour, 0.02 * perimeter, True)
        print(len(approx))
        if len(approx) in (8, 9, 10, 11):
            cv.drawContours(resized, contour, -1, (255, 0, 0), 2)

cv.imshow("Original with contours", resized)
```

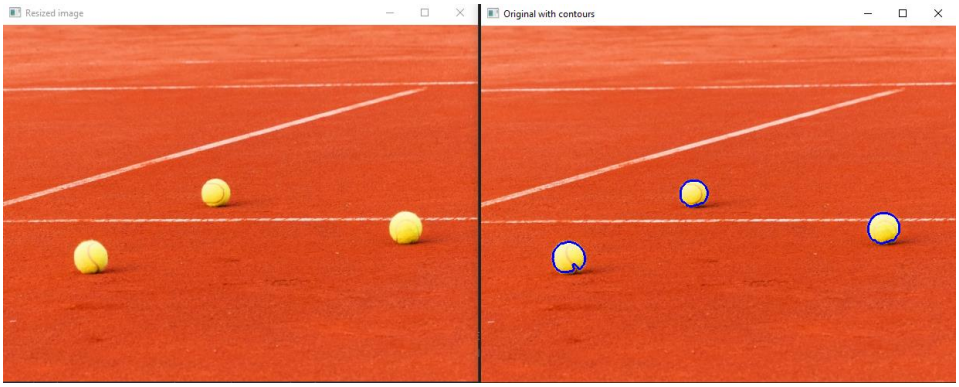


Figura 27: Comparació entre la foto original i la detecció de color groc. Font pròpia.

Per veure el vídeo de la simulació completa, visitar el següent enllaç: https://drive.google.com/file/d/1gfR8zRkyel1d0TV6Gl312iNL9l7hHatD/view?usp=sharing

5.3.2. Classificadors ja existents

Tal com s'explica a la llibreria d'OpenCV [26], han creat els seus propis detectors de cares mitjançant una tècnica proposada per Paul Viola i Michael Jones en el seu projecte el 2001 "Rapid Object Detection using a Boosted Cascade of Simple Features".

Aquest mètode requereix moltes imatges positives, és a dir, cares, i moltes imatges negatives, tot el que no sigui rostres. Internament, funciona com una xarxa neuronal convolucional, que es guarda cada una de les característiques

que més representen una cara i que tenen una taxa d'error mínim i continua calculant fins a arribar a una precisió adequada.

Lògicament, aplicar totes aquestes característiques a totes les zones d'una imatge, fa que el procés sigui molt robust i lent, per això han introduït el concepte de "Cascada de classificadors", ja que cada una de les funcions s'agrupen en diferents etapes de classificadors i que s'apliquen una per una, per tant, si en una de les primeres etapes ja falla, no fa falta aplicar les següents.

El detector de cares dels autors mencionats anteriorment, tenia més de 6000 característiques en 38 etapes.

Per veure la prova del detector facial, visitar l'enllaç a continuació. El codi es pot trobar a l'Annex 9.
https://drive.google.com/file/d/1OnHdvJv11bUq_aysXrct6xjqmCjCWW2W/view?usp=sharing

5.3.3. Aplicació per crear classificadors

En últim lloc, s'ha fet servir una aplicació ja existent, Cascade Trainer GUI [26] és un programa que es pot fer servir per entrenar, provar i millorar els classificadors, en format cascada i que fa servir les eines d'OpenCV.
Fa servir una interfície gràfica molt senzilla per entrenar i provar classificadors.

5.3.3.1. Com fer-lo servir

Quan s'inicia GUI de Cascade Trainer, es mostra la pantalla inicial que es pot utilitzar per entrenar classificadors, veure Figura 28. Per entrenar els classificadors, es necessita proporcionar milers de mostres d'imatges positives i negatives.

Per començar l'entrenament, s'ha de crear una carpeta per al vostre classificador amb dues carpetes dins seu. Una anomenada "p" (per a imatges positives) i l'altre ha de ser "n" (per a imatges negatives).
Per exemple, una carpeta anomenada "Cotxe" que tingui les carpetes "p" i "n" dins, amb fotografies de cotxes i l'altra amb fotos de qualsevol objecte, respectivament.

Cal esmentar que les imatges negatives mai han d'incloure imatges positives. Ni tan sols parcialment. Les imatges negatives poden ser qualsevol imatge que no sigui la imatge positiva, però a la pràctica, les imatges negatives han de ser rellevants. Per exemple, utilitzar imatges del cel com a imatges negatives és una mala elecció per entrenar un bon classificador.

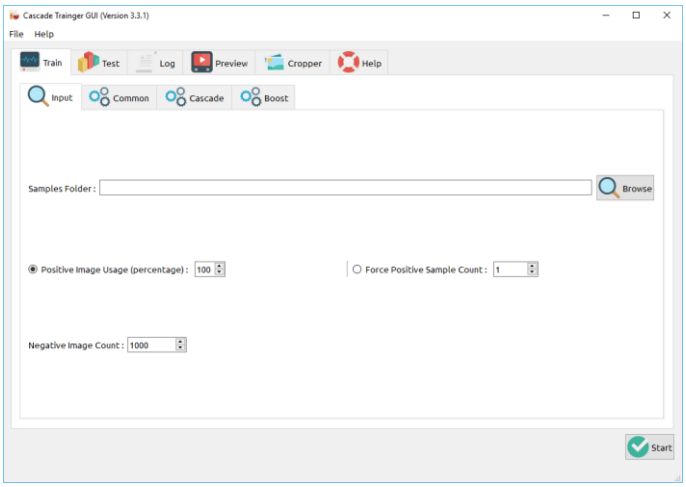


Figura 28: Pantalla inicial del Cascade Trainer. Font pròpia.

Un cop creada la carpeta només cal seleccionar-la, ajustar els paràmetres que es vulgui, com per exemple, el nomnre d'imatges negatives, la mida de les imatges, el tipus de classificador, es recomana fer servir HAAR, ja que són més precisos, la quantitat de memòria RAM "buffers" que assignes per emmagatzemar les dades, etc. Tot es pot trobar a les 3 pestanyes adjacents, que tenen una icona d'engranatge, veure Figura 28.

Finalment, cal clicar el botó "Start" i esperar que el classificador acabi. Un cop acabat, a la carpeta inicial, s'ha creat una nova carpeta "classifier" amb el fitxer del classificador, un fitxer .txt amb els "logs" del procés (tot el registre) i un fitxer per cada etapa.

Tal com es mostra a la Figura 29, el classificador creat per detectar pilotes de tennis ha tardat 2 dies, 7 hores i 37 minuts. S'ha agafat 1300 imatges positives i 10000 imatges negatives.

Figura 29: Captura del resultat final del Cascade Trainer. Font pròpia.

6. ARQUITECTURA DE PROVA

Per tal de fer la prova “final”, s’ha triat les següents funcionalitats:

- Ús del protocol de comunicació definit a l’apartat 3 “Definició del sistema de comunicació”. Basat en el “Broker” MQTT amb un esquema molt senzill, sobre publicar i escoltar a tòpics definits pel següent model:
    - Origen del servei al qual es publica el missatge.
    - Cap a quin servei es dirigeix el missatge.
    - Acció que es vol dur a terme al servei de destí.
- Ús de l’editor de pla de vols definit a l’apartat 4 “Generació d’un pla de vol”, capaç de mostrar i navegar per un mapa a temps real, definir els “waypoints” per on es vol fer passar el dron, triar en quins d’aquests punts es vol fer una fotografia, mostrar la distància entre els punts, tant a temps real com un cop establerts, eliminar tantes vegades com es vulgui el pla de vol si encara no s’ha enviat i finalment, enviar el pla de vol al dron perquè l’executi.
- Ús del classificador de pilotes de tennis obtingut per l’aplicació Cascade Trainer, explicat a l’apartat 5.5.3. “Aplicació per crear classificadors”. S’ha triat aquesta opció davant les altres per les següents raons lògiques:
    - El detector de cares i persones ja creat, funcionen a la perfecció i estan molt estudiats, per tant, no té cap gràcia. A més a més, en una prova amb el dron real, no es respectaria la seguretat de la persona dins del camp de vol del dron.
    - Enfront el detector de gats, per sentit comú, ja que no és viable ni segur mantenir un gat quiet, tant en la prova local com al camp de vol del dron.
    - El detector de colors, perquè com indica el nom, només detecta colors i no formes, per tant, s’hauria d’estudiar la manera de definir formes o filtres per tal de definir només els contorns desitjats.

7. IMPLANTACIÓ

Finalment, per raons de temps i disponibilitat només s'ha pogut fer la prova completa en local posant la càmera a una distància d'uns 3 metres de l'objecte a reconèixer.
Aquesta prova consta de les següents parts:

- Connectar el dron amb tots els serveis definits i amb el "mission planner", programa el qual permet simular el vol d'un dron.
- Obrir el quadre de control i tot seguit el widget per definir el pla de vol.
- Definir un pla de vol qualsevol i triar que es faci una foto al segon "waypoint".
- Enviar i executar el pla de vol.
- Detectar la pilota de tenis en fer la foto.
- Completar el pla de vol.

Per poder observar l'èxit de la missió cal visitar el següent enllaç:
https://drive.google.com/file/d/17mH2f_CSnBiagpoTzMGgs1vXrG83KAX4/view?usp=sharing

8. PROBLEMES TROBATS

Com és comú en aquest tipus de projecte, s'avança a poc a poc, anar provant i anar fracassant, és com anar xocant contra un mur constantment. En aquest apartat s'ha volgut explicar de manera sintetitzada alguns dels problemes trobats i les seves solucions.

8.1. Repte inicial. Instal·lació de l'ecosistema

Només començar el projecte va ser tot un repte, aconseguir instal·lar tots els mòduls i tenir tot l'entorn de treball preparat per fer funcionar l'ecosistema inicialment. Per això, tal com s'explica a l'apartat 1.2 "Definició de la instal·lació i requisits de l'ecosistema", es va definir un guia d'instal·lació i de treball.

8.2. Dificultats amb l'aplicació mòbil

Tot seguit, es va procedir a provar un nou framework, en aquest cas, Ionic, per desenvolupar l'aplicació mòbil, ja que la tecnologia feta sevir inicialment (Kivy) té problemes a l'hora d'implementar la llibreria Opencv, per tant, problemes amb la càmera.

Malauradament, no es va aconseguir avançar, ja que, el primer que s'ha de fer es connectar-te al broker MQTT per poder comunicar-te amb els altres serveis ja definits. No es va assolir, demostrant així que Ionic no es compatible amb MQTT. La solució és fer el desenvolupament en Android pur, però es va deixar de costat, ja que no era viable començar des de 0 l'aplicació mòbil, aprenent tot un llenguatge nou i fer millores a l'ecosistema explicades en aquest projecte.

8.3. Zoom en el mapa del pla de vol

Seguidament es va trobar que a l'hora de fer servir la llibreria ja existent "tkintermapview" per poder mostrar el mapa per definir un pla de vol, hi havia implementat el zoom, cosa no desitjada, ja que si es vol mostrar la distància entre dos punts, en metres, això és relatiu al zoom que hi hagi definit en aquell moment. La solució aplicada ha sigut copiar el codi base de la llibreria a un fitxer dins del projecte i eliminar totes les opcions del zoom. A l'hora de mostrar el mapa s'ha fet instància del nou fitxer creat en comptes de la llibreria original.

8.4. Mal de cap amb els classificadors d'imatges

Addicionalment, per les xarxes neuronals es van trobar diversos problemes, el primer de tots va ser com poder obtenir milers i milers de fotografies de l'objecte desitjat, el que es coneix com a "dataset". L'única solució trobada ha sigut

navegar i navegar per internet, trobant així un conjunt de pàgines web, fetes per la recerca de “datasets”. Algunes d’elles són les següents:

- Google Dataset Search
- Kaggle
- Data.Gov
- Datahub.io
- UCI Machine Learning Repository

A més a més, l’altre problema que s’ha trobat és la gran dedicació de temps i de memòria de l’ordinador que es necessita per poder entrenar les xarxes neuronals.

Cal esmentar que aquests “datasets” conten milers i fins i tot milions d’imatges, cosa normal, ja que és la quantitat necessària per poder entrenar adequadament una xarxa neuronal. El punt és que això implica moltíssim temps i moltíssima memòria. El meu ordinador no era capaç de processar sets de centenars de milers d’imatges. La solució aplicada ha sigut entrenar la xarxa neuronal durant menys temps, aproximadament un parell de dies, i amb menys imatges, unes 10000 com a màxim.

També s’ha trobat un gran entrebanc, i es que un cop fet el classificador, hi ha un parell de paràmetres que juguen en contra, la llum i la qualitat de la imatge capturada. Lògicament, com menys llum, pitjor eficiència reconeixent objectes, el mateix si es fa servir una qualitat d’imatge pitjor.

La solució aplicada ha sigut fer les proves amb la màxima llum natural possible i assumir una distància màxima, en funció de la qualitat d’imatge de la meva càmera, on l’objecte es pugui reconèixer bé.

8.5. Poca disponibilitat

Finalment, com a últim problema a destacar poso la falta de temps i de disponibilitat, ja que he fet el TFG mentre treballo, per poder fer una prova amb un dron real.

La solució aportada ha sigut fer la missió final en local, tot i que ha sigut un èxit, no és del tot realista, puix que a casa, amb tot l’entorn preparat i ben configurat, després de moltes proves unitàries, existeix molt menys “soroll”, entenent com a soroll a tots aquells afers externs al teu projecte.

9. CONCLUSIONS

Cal comunicar que l'objectiu principal no s'ha acabat d'assolir del tot, ja que com s'ha comentat en l'apartat anterior, la part de l'aplicació mòbil de l'ecosistema no s'ha pogut desenvolupar tal com s'esperava en un inici.

No obstant això, mencionar que la resta de funcionalitats marcades en l'apartat dels objectius han sigut tot un èxit, degut a que s'ha pogut completar el vol de missió del dron, cosa que implica, poder definir un pla de vol, marcar en quins punts és vol fer una fotografia, executar la ruta, reconèixer l'objecte desitjat i la comunicació, mitjançant el protocol definit, entre els diferents mòduls de l'ecosistema.

Cal destacar que fer aquest projecte m'ha ajudat en augmentar les meves capacitats en alguns valors, molt importants, dins del camp de l'enginyeria. Recalcar algun d'ells, com per exemple, l'autoaprenentatge, gràcies al fet que tots els coneixements adquirits, en la gran majora, han sigut a través dels meus propis medis, la gestió de canvis constants, ja que cada setmana s'han trobat obstacles i problemes que han fet derivar el projecte cap a camins alterns, la intel·ligència emocional, perquè s'ha sigut constant, i s'ha treballat en situacions d'esgotament mental (després de treballar cada dia) o en mals moments personals, o la creativitat, gràcies a poder dissenyar i trobar amb les formes de resoldre els conflictes sorgits pels intents de crear allò que s'havia imaginat.

Cal comentar que sí tornés a començar, l'única i gran diferència que faria és intentar atacar una idea més concreta, és a dir, en comptes d'arrancar amb la idea inicial de crear una aplicació per "tauletes" on es pugui dissenyar un pla de vol i a la vegada es pugui reconèixer un objecte, buscaria un objectiu més concret on poder profunditzar més.

Finalment, explicar una de les línies obertes que tinc present i que intentaria desenvolupar si tingués uns quants mesos més per seguir amb el projecte. Concretament, seria intentar implementar i controlar mòduls extensibles al dron, per exemple, implementar un petit braç robòtic o una mena de mà, capaç de poder agafar un objecte i transportar-lo a un altre punt. Això neix de la idea, ja aplicada actualment, de poder facilitar l'accés a medicines a poblacions apartades de la societat.

BIBLIOGRAFIA

[1] EDIX. *Framework*. 2021 [en línia]. Disponible en: https://www.edix.com/es/instituto/framework/

[2] AZURE. *¿Que es un middleware?*. 2022. [en línia]. Disponible en: https://azure.microsoft.com/es-es/overview/what-is-middleware/

[3] MQTT. *Getting started*. 2022. [en línia]. Disponible en: https://mqtt.org/getting-started/

[4] DELOITTE. *IoT. Internet Of Things.* 2022 [en linia]. Disponible en: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

[5] LUIS LLAMAS. *¿Que es MQTT? Su importància como protocolo IoT.* 2022. [en línia]. Disponible en: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[6] MOSQUITTO. *An open source MQTT broker.* 2022. [en línia]. Disponible en: https://mosquitto.org/

[7] PAESSLER. *IT explained: MQTT.* 2022. [en línia]. Disponible en: https://www.paessler.com/es/it-explained/mqtt

[8] GITHUB. *Drone Engineering Ecosystem.* 2022. [en línia]. Disponible en: https://github.com/miguelvalero/DroneEngineeringEcosystem

[9] GITHUB. *TkinterMapView.* 2022. [en línia]. Disponible en: https://github.com/TomSchimansky/TkinterMapView

[10] W3 SCHOOLS. *Python Sets.* 2022. [en línia]. Disponible en: https://www.w3schools.com/python/python_sets.asp

[11] FREE CODE CAMP. *Python leer archivo JASON – Como cargar JSON desde un archivo y processar dumps.* 2022. [en línia]. Disponible en: https://www.freecodecamp.org/espanol/news/python-leer-archivo-json-como-cargar-json-desde-un-archivo-y-procesar-dumps/

[12] SAS. *Inteligencia Artificial. Que es IA y porque improta.* 2022. [en línia]. Disponible en: https://www.sas.com/es_es/insights/analytics/what-is-artificial-intelligence.html

[13] Carrión, C.B. 2020. Redes Convolucionales. Universidad de Sevilla, Sevilla.

[14] Matich, D.J. 2001. Redes Neuronales: Conceptos básicos y aplicaciones. Universidad Tecnológica Nacional, México.

[15] CIÈNCIA OBERTA. *És la intel·ligència artificial tal misteriós com sembla?.* 2022. [en línia]. Disponible en: https://www.cienciaoberta.cat/intelligencia-artificial/

[16] WIKIPEDIA. *Perceptron.* 2022. [en línia]. Disponible en: https://es.wikipedia.org/wiki/Perceptr%C3%B3n

[17] INTERACTIVE CHAOS. *El perceptron Multicapa.* 2022. [en línia]. Disponible en: https://interactivechaos.com/es/manual/tutorial-de-deep-learning/el-perceptron-multicapa

[18] IONOS. *Neutral Networks ¿DE que son capces las redes neuronales?* 2022. [en línia]. Disponible en: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-una-neural-network/

[19] HANDSONPARDES. *Desarolla una red neuronal convolucional en keras con Python.* 2022. [en línia]. Disponible en: http://blog.hadsonpar.com/2021/08/crear-una-red-neuronal-convolucional-en.html

[20] WIKIPEDIA. *Convolución.* 2022. [en línia]. Disponible en: https://es.wikipedia.org/wiki/Convoluci%C3%B3n

[21] MEDIUM CODEX. *Kernels (Filters) in convolutional neural network (CNN), Let's talk about them.* 2021. [en línia]. Disponible en: https://medium.com/codex/kernels-filters-in-convolutional-neural-network-cnn-lets-talk-about-them-ee4e94f3319

[22] BLOGSPOT. *Machine learning and statistics with python.* 2020. [en línia]. Disponible en: https://shyambhu20.blogspot.com/2020/12/convolutional-neural-network-concepts-definition-howto.html

[23] AMAZON AWS. *Dense Lawyer. Introduction to TensorFlow in python.* 2022. [en línia]. Disponible en: https://s3.amazonaws.com/assets.datacamp.com/production/course_15108/slides/chapter3.pdf

[24] DATASCIENE RECURSOS. UOC. *Keras + Tensorflow.* 2022. [en línia]. Disponible en: http://datascience.recursos.uoc.edu/es/keras-tensorflow/

[25] LLEDO ENERGIA. *Colorimetría lll: espacio de color RGB, HSV y HSL.* 2020. [en línia]. Disponible en: https://lledoenergia.es/colorimetria-iii-espacios-de-color-hsl-hsv-y-rgb/

[26] OPENCV. *Cascade Classifier.* 2022. [en línia]. Disponible en: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[27] RESEARCHGATE. *Building Custom HAAR-Cascade Classifier for face Detection.* 2020. [en línia]. Disponible en: https://www.researchgate.net/publication/338680205_Building_Custom_HAAR-Cascade_Classifier_for_face_Detection/figures?lo=1

ANNEXOS

Annex 1: Definir mapa i funcionalitats

```
class Flightplan(tkinter.Tk):

  def __init__(self, *args, **kwargs):
    tkinter.Tk.__init__(self, *args, **kwargs)

    self.geometry(f"{1200}x{800}")
    self.title("Flightplan Map")
    self.protocol("WM_DELETE_WINDOW", self.closing)
    self.bind("<Return>", self.search)

    # ============ create the general grid ============

    self.grid_columnconfigure(0, weight=1)
    self.grid_columnconfigure(1, weight=0)

    self.grid_rowconfigure(0, weight=1)
    self.grid_rowconfigure(1, weight=0)

    # ============ search configuration ============

    self.search_bar = tkinter.Entry(self, width=50)
    self.search_bar.grid(row=2, column=0, pady=10, padx=10, sticky="nsew")
    self.search_bar.focus()

    self.button_1 = tkinter.Button(master=self, text="Search", command=self.search, width=15)
    self.button_1.grid(pady=10, padx=20, row=2, column=1)

    self.button_2 = tkinter.Button(master=self, text="Clear", command=self.clear, width=15)
    self.button_2.grid(pady=10, padx=20, row=2, column=2)

    # ============ map ============

    self.map_widget = TkinterMap(self, width=800, height=600, corner_radius=0, max_zoom=19)
    self.map_widget.grid(row=0, column=0, columnspan=3, sticky="nsew", padx=20, pady=20)
    self.map_widget.set_address("-35.363 149.165")
    self.map_widget.add_right_click_menu_command(label="Add point", command=self.set_point,
pass_coords=True, pass_pixels=True)

    # ============ coordinates menu ============

    self.listbox_button_frame = tkinter.Frame(master=self)
    self.listbox_button_frame.grid(row=0, column=1, sticky="nsew", columnspan=3)
    self.listbox_button_frame.grid_columnconfigure(0, weight=1)
    self.listbox_button_frame.grid_rowconfigure(4, weight=1)

    self.clear_marker_button = tkinter.Button(
      master=self.listbox_button_frame, width=20, text="Clear points list",
command=self.clear_points_list
    )
    self.clear_marker_button.grid(row=0, column=0, pady=10, padx=10)

    self.send_points_button = tkinter.Button(
      master=self.listbox_button_frame, width=20, text="Send flight plan",
```

```
command=self.send_points
        )
        self.send_points_button.grid(row=3, column=0, pady=10, padx=10)

        self.point_list_box = tkinter.Listbox(master=self.listbox_button_frame, height=20,
selectmode="extended")
        self.point_list_box.grid(row=4, column=0, sticky="nsew", padx=10, pady=10)

        #self.take_photo_frame = tkinter.Frame(master=self)
        photo_button_add = tkinter.Button(
            master=self.listbox_button_frame, width=20, text="Add point to Take photo list",
command=self.selected_items
        )
        photo_button_add.grid(row=5, column=0, sticky="nsew", padx=10, pady=10)

        photo_button_rem = tkinter.Button(
            master=self.listbox_button_frame, width=20, text="Remove point from Take photo list",
command=self.delete_items
        )
        photo_button_rem.grid(row=6, column=0, sticky="nsew", padx=10, pady=10)
```

Annex 2: Clicar el botó dret del ratolí

```
def mouse_right_click(self, event):
    mouse_pixel = (event.x, event.y)
    self.last_mouse_right_down_position = (event.x, event.y)

    if self.first_click is True:
        self.fist_mouse_right_position = (event.x, event.y)
        self.first_click = False

    if self.line:
        self.canvas.delete(self.line)

    self.line = self.canvas.create_line(
        self.last_mouse_right_down_position[0],
        self.last_mouse_right_down_position[1],
        self.last_mouse_right_down_position[0],
        self.last_mouse_right_down_position[1],
        width=9, fill="#3E69CB"
    )

    if self.dist:
        self.canvas.delete(self.dist)

    self.dist = self.canvas.create_text(
        self.last_mouse_right_down_position[0] + (event.x - self.last_mouse_right_down_position[0]) /
2,
        self.last_mouse_right_down_position[1] + (event.y - self.last_mouse_right_down_position[1]) /
2,
        text=""
    )

    coordinate_mouse_pos = self.convert_canvas_coords_to_decimal_coords(event.x, event.y)

    def click_coordinates_event():
        try:
            pyperclip.copy(f"{coordinate_mouse_pos[0]:.7f} {coordinate_mouse_pos[1]:.7f}")
            tkinter.messagebox.showinfo(title="", message="Coordinates copied to clipboard!")

        except Exception as err:
            if sys.platform.startswith("linux"):
                tkinter.messagebox.showinfo(title="", message="Error copying to clipboard.\n" +
str(err) + "\n\nTry to install xclip:\n'sudo apt-get install xclip'")
            else:
                tkinter.messagebox.showinfo(title="", message="Error copying to clipboard.\n" +
str(err))

    def click_pixel_event():
        try:
            pyperclip.copy(f"{mouse_pixel}")
            tkinter.messagebox.showinfo(title="", message="Pixel copied to clipboard!")

        except Exception as e:
            tkinter.messagebox.showinfo(title="", message="Error copying to clipboard.\n" + str(e))

    m = tkinter.Menu(self, tearoff=0)
    m.add_command(label=f"{mouse_pixel}", command=click_pixel_event)
    m.add_command(label=f"{coordinate_mouse_pos[0]:.7f} {coordinate_mouse_pos[1]:.7f}",
command=click_coordinates_event)
```

```
    if len(self.right_click_menu_commands) > 0:
        m.add_separator()

    for command in self.right_click_menu_commands:
        if command["pass_coords"] and command['pass_pixels']:
            m.add_command(label=command["label"], command=partial(command["command"],
coordinate_mouse_pos, mouse_pixel))
        else:
            m.add_command(label=command["label"], command=command["command"])

    m.tk_popup(event.x_root, event.y_root)
```

Annex 3: Funció per afegir punts al pla de vol

```
def set_point(self, coords, mouse_pixels):

    print("Add point:", coords)
    print('pixel', mouse_pixels)
    positions_list = []

    if self.first_point is False:
        self.new_point = self.map_widget.set_marker(coords[0], coords[1], text="")
        self.point_list_box.insert(tkinter.END, f"{coords}; {'First'}")
        self.point_list_box.see(tkinter.END)
        self.points_list.append(self.new_point)
        self.pixels_list.append(mouse_pixels)
        self.map_widget.delete(self.last_point)
        self.last_point = False
        self.first_point = True
    else:
        if self.last_tram:
            self.map_widget.delete(self.last_tram)
        if self.last_point:
            self.map_widget.delete(self.last_point)
            self.point_list_box.delete(tkinter.END, tkinter.END)
            self.points_list.pop(-1)
        self.pixels_list.append(mouse_pixels)
        pixels = calc_pixels(self.pixels_list[-1], self.pixels_list[-2])
        dist = pixels * self.factor_p_to_m
        self.new_point = self.map_widget.set_marker(coords[0], coords[1], text="{}
m".format(round(dist, 2)))
        self.point_list_box.insert(tkinter.END, f"{coords}; {round(dist, 2)}")
        self.point_list_box.see(tkinter.END)
        self.points_list.append(self.new_point)
        self.previous_path.append(self.ptp_path)

        for point in self.points_list:
            positions_list.append(point.position)

        self.ptp_path = self.map_widget.set_path(positions_list)

        pixels = calc_pixels(self.pixels_list[0], self.pixels_list[-1])
        dist = pixels * self.factor_p_to_m

        first = positions_list[0]
        last = positions_list[-1]
        last_tram = [first, last]
        self.last_tram = self.map_widget.set_path(last_tram)
        self.last_point = self.map_widget.set_marker(first[0], first[1], text="{} m".format(round(dist,
2)))
        self.points_list.append(self.last_point)
        self.point_list_box.insert(tkinter.END, f"{first}; {'Last'}")
```

Annex 4: Funció per mostrar els metres entre punts a temps real

```
def mouse_middle_move(self, event):
    if not self.line:
        raise Exception("No line to edit")

    mouse_pixel = (event.x, event.y)

    self.canvas.coords(
        self.line, self.last_mouse_right_down_position[0], self.last_mouse_right_down_position[1],
event.x, event.y
    )
    dist_pixel = calc_pixels(self.last_mouse_right_down_position, mouse_pixel)
    dist_m = dist_pixel * self.factor_p_to_m

    self.canvas.coords(
        self.dist,
        self.last_mouse_right_down_position[0] + (event.x - self.last_mouse_right_down_position[0]) /
2,
        self.last_mouse_right_down_position[1] + (event.y - self.last_mouse_right_down_position[1]) /
2,
    )
    self.canvas.itemconfig(
        self.dist, text="{} m.".format(round(dist_m, 2))
    )
```

Annex 5: Definir botons pels punts on fer les fotos i les seves funcions

```
photo_button_add = tkinter.Button(
   master=self.listbox_button_frame, width=20, text="Add point to Take photo list",
command=self.selected_items
)
photo_button_add.grid(row=5, column=0, sticky="nsew", padx=10, pady=10)

photo_button_rem = tkinter.Button(
   master=self.listbox_button_frame, width=20, text="Remove point from Take photo list",
command=self.delete_items
)
photo_button_rem.grid(row=6, column=0, sticky="nsew", padx=10, pady=10)

def selected_items(self):
   for i in self.point_list_box.curselection():
      info = self.point_list_box.get(i)
      splited = info.split(";")
      coord_tuple = ast.literal_eval(splited[0])
      self.photo_coord_set.add(coord_tuple)

   print(self.photo_coord_set)

def delete_items(self):
   for i in self.point_list_box.curselection():
      info = self.point_list_box.get(i)
      splited = info.split(";")
      coord_tuple = ast.literal_eval(splited[0])
      self.photo_coord_set.discard(coord_tuple)

   print(self.photo_coord_set)
```

Annex 6: Codi per enviar el pla de vol al auto-pilot

```
def send_points(self):
    position_list = []
    position_list_photo = []
    new_list_points = []

    for point in self.points_list:
        position_list.append(point.position)

    for coord in self.photo_coord_set:
        position_list_photo.append(coord)

    for x in range(len(position_list)):
        if position_list[x] in position_list_photo:
            tuple_position = position_list[x]
            photo_tuple = tuple_position + (1,)
            new_list_points.append(photo_tuple)
        else:
            new_list_points.append(position_list[x])

    position_string = json.dumps(new_list_points)

    # Enviar al autoplioto
    client.publish('flightplanService/autopilotService/flightplanpoints', position_string)
    print('flightplanService/autopilotService/flightplanpoints', position_string)
```

Annex 7: Codi per executar el pla de vol

```
def distanceInMeters(aLocation1, aLocation2):
    """
    Returns the ground distance in metres between two LocationGlobal objects.

    This method is an approximation, and will not be accurate over large distances and close to the
    earth's poles. It comes from the ArduPilot test code:
    https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py
    """
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5

def executeFlightPlan(list_points_json, origin):
    global vehicle
    global timer
    print('Executing Flight plan')
    points_list = json.loads(list_points_json)
    originPoint = vehicle.location.global_frame
    takeOff(20)
    distanceThreshold = 1
    for wp in points_list[0:]:
        print ('Siguiente punto del flight plan: ', wp)

        # ir al siguiente punto
        destinationPoint = dronekit.LocationGlobalRelative(wp[0], wp[1], 20)
        vehicle.simple_goto(destinationPoint)
        currentLocation = vehicle.location.global_frame
        dist = distanceInMeters(destinationPoint, currentLocation)
        while dist > distanceThreshold:
            time.sleep(0.2)
            currentLocation = vehicle.location.global_frame
            dist = distanceInMeters(destinationPoint, currentLocation)
            print(dist)
        print ('Arrived to the point')
        if len(wp) == 3:
            print('order take picture')
            client.publish("autopilotService/cameraService/takePicture")
        position = str(wp[0]) + '*' + str(wp[0])
        client.publish('autopilotService/' + origin + '/dronePosition', position)

    vehicle.mode = dronekit.VehicleMode("RTL")
    currentLocation = vehicle.location.global_frame
    dist = distanceInMeters(originPoint, currentLocation)

    while dist > distanceThreshold:
        time.sleep(0.25)
        currentLocation = vehicle.location.global_frame
        dist = distanceInMeters(originPoint, currentLocation)
    print ('Arrived at home')
    time.sleep(1)
    timer.cancel()
    last_point = points_list[-1]
    position = str(last_point[0]) + '*' + str(last_point[0])
    client.publish('autopilotService/' + origin + '/dronePosition', position)
```

Annex 8: Codi per provar el classificador de gats

```
import tensorflow as tf
import numpy as np
import cv2 as cv
from keras_preprocessing.image import img_to_array


## path de nuestro modelo
cnn_model = 'C:/RedNeuronal/CatsDogs.h5'

## Leer la red neuronal
cnn = tf.keras.models.load_model(cnn_model)

## Empezar la videocaptura
video = cv.VideoCapture(0)
while True:
  _, frame = video.read()
  # change to one color
  gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
  gray = cv.resize(gray, (200, 200), interpolation=cv.INTER_CUBIC)
  gray = np.array(gray).astype(float) / 255
  img = img_to_array(gray)
  img = np.expand_dims(img, axis=0)

  predict = cnn.predict(img)
  predict = predict[0][0]
  print(predict)
  if predict > 0.85:
    cv.putText(frame, "Cat detected", (200, 70), cv.FONT_HERSHEY_PLAIN, 3, (0, 0, 255))

  cv.imshow('CAM', frame)
  a = cv.waitKey(1)
  if a > 0:
    break

cv.destroyAllWindows()
video.release()
```

Annex 9: Codi per provar el detector facial Haar Cascade

```
## Empezar la videocaptura
video = cv.VideoCapture(0)
detect = cv.CascadeClassifier(cv.data.haarcascade + 'haarcascade_frontalface_default.xml')
while True:
    _, frame = video.read()
    # change to one color
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # # detect, if detected will save x,y,w,h
    object = detect.detectMultiScale(gray, 1.2, 5)
    # Drawing the rectangle
    for (x, y, w, h) in object:
        cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv.imshow('CAM', frame)
    a = cv.waitKey(1)
    if a > 0:
        break

cv.destroyAllWindows()
video.release()
```