EPS
UdG Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Tecnologies Industrials

Títol: Drone de suport a les tasques de rescat d'un gos ensinistrat i un robot terrestre

Document: Memòria i Annexos

Alumne:Joan Bataller Quiñones

Director/tutor: Xavier Cufí Solé
Departament: Arquitectura i Tecnologia de Computadors
Àrea: ATC

Convocatòria (mes/any) Setembre 2015

ÍNDEX

I-MEMÒRIA

1.Introducció

En la següent memòria es durà a terme una descripció de totes les parts, tant físiques com de software, que conformen l'elecció i adequació d'un drone capaç de seguir un objectiu a través de tècniques de reconeixement visual. Aquest projecte s'emmarca dins d'un projecte d'àmbit europeu de robòtica de rescat conformat per robots i gossos ensinistrats.

1.1.Objecte

L'objectiu del projecte és adequar una aeronau de ràdio control de tipus drone per a què pugui fer el seguiment dels moviments d'un gos ensinistrat sobre l'entorn d'operació mitjançant tècniques de visió per computador. Amb aquest sistema es pot tenir localitzat el gos ensinistrat en qualsevol moment i tenir la possibilitat de disposar d'un canal de comunicació per a l'intercanvi de dades en tot moment entre tots els elements del conjunt.

En aquest projecte per tant podem distingir dos objectius diferents. Per una banda, com a objectiu principal del projecte, tenim l'adequació de l'aeronau per a que sigui capaç de realitzar un seguiment de qualitat d'un objectiu i per altre banda tenim l'estudi de la implementació d'algun sistema per a generar una comunicació per a la transferència d'informació en temps real. Aquests dos objectius seran tractats de forma independent en la resolució d'aquest projecte per tal de facilitar-ne el seu compliment.

1.2.Antecedents

Aquest projecte queda emmarcat dins d'un projecte de recerca de robòtica de rescat que té com a objectiu donar suport operacional a un gos ensinistrat de rescat per mitjà d'un vehicle terrestre i un vehicle aeri amb els quals ha de poder intercanviar dades sense necessitat de la presència de l'ensinistrador. Aquest projecte de robòtica de rescat anomenat MATE és un projecte on participen vaires universitats europees.

1.3.Abast

L'abast d'aquest projecte serà aconseguir una teleoperació de qualitat del drone, així com un comportament en navegació consistent en el seguiment des de l'aire del gos

ensinistrat utilitzant tècniques de visió computacional. També s'estudiarà com establir canals de comunicació eficients amb el gos ensinistrat i un robot terrestre.

L'estudi de la comunicació entre el drone i un altre dels elements que conformen el sistema el duu a terme un altre alumne de la Universitat de Girona en el seu projecte de final de grau. És per aquest motiu que l'objectiu principal del projecte serà l'adequació de l'aeronau per a la realització del seguiment del gos sense deixar de banda en cap de les decisions que es preguin en el projecte l'interés que hi ha en la comunicació del drone amb altres aparells.

2. El Projecte MATE

Aquest treball de final de grau està ambientat en una proposta de projecte europeu anomenat Companions/MATE. Aquest neix de la necessitat de buscar una solució en el camp de la robòtica de rescat on els grups de recerca de la UdG VICOROB i ARLab hi estan involucrats [1].

Després d'un desastre natural com podria ser un terratrèmol o bé un tsunami, l'índex de mortalitat s'incrementa considerablement després de 48 hores. Una ràpida i efectiva gestió dels equips de rescat és de màxima importància. Avui en dia els equips de rescat i d'exploració normalment utilitzen gossos ensinistrats com a companys per trobar les víctimes. Els gossos ensinistrats són molt útils en aquestes situacions degut a la seva alta mobilitat, velocitat i capacitat de detecció mitjançant l'olfacte. No obstant necessiten instruccions i supervisió constant. D'altre banda, es poden trobar en algunes situacions perilloses i no són capaços de recollir dades precises del terreny per on es mouen. En comptes d'intentar construir aparells que puguin substituir el gos, el projecte Companions/MATE busca la cooperació entre gossos ensinistrats i robots. Aquest és un camp bastant nou en la recerca on els gossos poden ser complementats amb robots autònoms amb habilitats cognitives i capaços de cooperar amb humans en ambients de cerca i rescat.

L'objectiu d'aquest projecte és analitzar com un equip de gossos, robots i humans poden cooperar i interactuar en diferents escenaris de rescat.

Per tal de poder-lo dur a terme es necessitaran diferents parts. A continuació detallarem quines són aquestes parts i quina funció desenvolupen dins d'aquest projecte Europeu.

El primer és un robot que farà de nexe entre l'home i el gos, aquest s'anomenarà BIGBOT. El robot haurà d'actuar autònomament de dues maneres diferents: la primera serà seguint el gos de rescat per mitjà de tècniques de visió per computador i interactuant amb ell i la segona serà explorant l'entorn per si sol. Un equip de persones monitoritzaran en tot moment el robot per tal de poder veure com actua i recollir dades de l'entorn. A més a més l'equip humà podrà controlar el robot remotament si és necessari. El robot comptarà amb sensors per tal de poder extreure

el màxim de dades possibles del lloc on es troba i així poder detectar situacions perilloses. D'aquesta manera anirà creant un mapa al seu voltant amb totes aquestes dades que permetran veure a l'equip de control si es tracta d'una zona segura, si hi ha víctimes i recollir altres dades d'interès com podrien ser temperatures, percentatges de gasos, etc.

El robot també portarà una pantalla i sistemes d'àudio per tal de poder comunicar-se amb la víctima i el gos. Això serà de vital importància sobretot per poder estar donant ordres al gos. El gos necessita constantment ordres de l'ensinistrador per saber si va pel camí correcte o no i al suprimir la figura de l'entrenador aquesta tasca l'ha de suplir el robot. A més a més aquests sistemes de comunicació, també servirien per fer un primer anàlisi de la víctima en cas de ser trobada ja que s'hi podria interactuar de manera directa.

A part d'aquest robot se'n crearà un altre de més petit anomenat CUB (Canine Unleashed roBot). Aquest serà desplegat pel gos, que el durà a sobre durant la recerca. Aquest petit robot tindrà la peculiaritat de poder introduir-se en forats molt petits per on el gos no pugui anar. També comptarà amb sensors de navegació, temperatura i gasos així com una càmera i aparells d'àudio.

A través del robot BIGBOT es podran enviar dades a l'equip de supervisió de la missió, que es podran comunicar amb la víctima. No obstant el CUB no durà pantalla ja que la idea és que sigui un robot petit que li permeti una alta mobilitat en espais reduïts. El robot BIGBOT actuarà com a pont entre les comunicacions del robot CUB i l'estació de control de la missió.

El gos entrenat comptarà amb una càmera, àudio i wireless. A més a més durà a sobre el robot CUB, que podrà deixar-lo allà on el BIGBOT li indiqui. El gos haurà de ser capaç de respondre a diferents situacions per tal de que el robot ho pugui interpretar.

Finalment es treballarà amb un equip de persones que duran a terme el control de la missió. Aquest tindrà accés a l'àudio i el vídeo del gos, el BIGBOT i el CUB. Cal recordar que tota aquesta informació serà enviada a través del BIGBOT. L'equip també tindrà

accés al mapa que construeixi el robot BIGBOT a través dels seus sensors (visuals, termal i de gasos) i també dels sensors del CUB.

2.1.Encaix d'aquest projecte dins del Projecte MATE

En l'actualitat el departament VICROB de la UdG està duent a terme la creació i la continua millora dels sistemes de navegació del robot BIGBOT. El problema actual de la navegació del robot BIGBOT s'ha trobat en el moment en el qual el robot perd de vista el gos i per tant no pot dur a terme un seguiment per mitjà de tècniques de visió per computadora. En aquest moment el robot es troba perdut i no sap quina és la següent ubicació a la que s'ha de destinar. Fins ara, la manera d'intentar solucionar aquest problema era per mitjà de rotacions de 360º sobre si mateix per intentar tornar a localitzar en imatge al gos. S'ha vist que aquest recurs no és suficient per a realitzar un seguiment de qualitat del gos sense tenir problemes d'aquest tipus.

La idea per a solucionar aquest problema neix de la voluntat de l'autor d'aquesta memòria per a realitzar qualsevol tipus de projecte de final de grau relacionat amb drones. És llavors quan un dels professors responsable de la gestió del projecte MATE i tutor d'aquest projecte Xevi Cufí, va tenir l'idea d'implementar un drone que també realitzés un seguiment aeri del gos per així poder-lo tenir localitzat en tot moment.

Aquest drone, igual que la resta de robots del projecte MATE, funcionarà de forma autònoma, tindrà una comunicació amb l'estació terrestre de control i comptarà amb un GPS que indicarà la seva posició en temps real.

La implementació d'un sistema de posicionament per GPS genera un nou problema: la necessitat de la implementació d'un sistema GPS en el robot BIGBOT. Amb aquesta implementació també seran necessaris canvis en el comportament del robot. En el moment que perdi de vista el gos, el robot BIGBOT demanarà la posició GPS del drone i es dirigirà cap a ella fins a retrobar el gos. Aquesta actualització del robot BIGBOT es duu a terme en un altre projecte de final de grau d'un alumne de l'UdG.

Al tractar-se d'una fase molt primària del desenvolupament, l'objectiu a seguir no serà un gos de rescat amb una armilla, sinò una taca d'un color que destaqui respecte l'entorn.

3. Computació visual

El terme computació visual fa referència a qualsevol ciència de computadors que tracti amb imatges i models 3D, gràfics d'ordinador, processament d'imatges, visualització, visió per computador, realitat virtual o realitat augmentada, processament de vídeo, auto aprenentatge i el reconeixement de patrons. Els principals objectius de la computació visual són l'adquisició, el processat, l'anàlisi i renderitzat d'informació visual [2].

Tal i com hem dit, la computació visual té moltes disciplines diferents, però en aquest treball ens centrarem bàsicament en la visió per computador o visió artificial. La visió per computador són tècniques informàtiques capaces d'extreure informació provinent d'una imatge. De totes les possibles aplicacions que té la visió per computador en aquest projecte es treballarà amb la detecció, segmentació localització i reconeixement de certs objectes en imatges i en el seguiment d'un objecte en una seqüència d'imatges.

Existeixen llibreries de programació molt extenses per a dur a terme el reconeixement i seguiment d'un objecte per mitjà de tècniques de visió per computador però sens dubte la llibreria per excel·lència és l'anomenada OpenCV. Una altre de les llibreries que també s'utilitzarà en aquest projecte és sikit-learn, una llibreria d'aprenentatge automàtic.

En els següents sub-apartats s'explicaran alguns dels conceptes necessaris per poder entendre millor el sistema de seguiment autònom que s'ha utilitzat en la realització d'aquest projecte.

3.1.OpenCV

OpenCV és una biblioteca de visió artificial originalment desenvolupada per la companyia de software i hardware Intel. Des de l'aparició de la seva primera versió alfa al mes de gener de 1999, OpenCV s'ha utilitzat en infinitat d'aplicacions, des de sistemes de seguretat amb detecció de moviment fins a aplicacions de control de processos. La llicència BSD, de la qual disposa aquesta llibreria, permet que sigui utilitzada lliurement per propòsits comercials i d'investigació sota les condicions que imposa aquesta llicència.

OpenCV és una multiplataforma per als sistemes operatius Linux, Mac OS, Windows, Android i iOS. Conté més de 500 funcions que inclouen una gran gamma d'àrees en el procés de visió, com reconeixement d'objectes, reconeixement facial, calibrat de càmeres, visió estèreo i visió robòtica.

Aquesta llibreria pretén proporcionar un entorn de desenvolupament fàcil d'utilitzar i altament eficient. A més a més OpenCV pot utilitzar el sistema de primitives de rendiment integrades d'Intel, que són un conjunt de rutines de baix nivell per a processadors Intel.

3.2.Sikit-learn

Sikit-learn és una llibreria de codi obert d'aprenentatge automàtic programada en el llenguatge Python. Aquesta llibreria és capaç de fer varis tipus de classificacions, regressions i algoritmes de clusterització.

Per entendre millor el que fa aquesta llibreria, hem d'entendre una mica del que tracta l'aprenentatge automàtic. L'aprenentatge automàtic és un camp de la intel·ligència artificial que està dedicat al disseny, anàlisi i desenvolupament d'algorismes i tècniques que permeten que les màquines evolucionin. Es tracta de crear programes capaços de generalitzar comportaments a partir del reconeixement de patrons o classificació.

La funció de sikit-learn en el projecte serà la de generar un aprenentatge que condueixi la nostra aeronau cap a un comportament predictiu de millor precisió. Gràcies a l'aprenentatge automàtic el nostre sistema de visió per computador serà capaç de determinar si l'objecte a seguir es troba en la imatge capturada o no. D'aquesta manera s'evitaran possibles errors durant el seguiment.

3.3.L'espai de color HSV

Existeixen diverses maneres de definir un color, una d'elles és segons els components vermell, verd i blau de la imatge, conegut com a RGB de l'anglès "Red", "Green" i "Blue". L'HSV és un altre mètode de definir un model de color però en termes dels seus components de matís (Hue), saturació (Saturation) i brillantor (Value). En la Fig. 1 podem veure la representació gràfica de l'HSV [3].

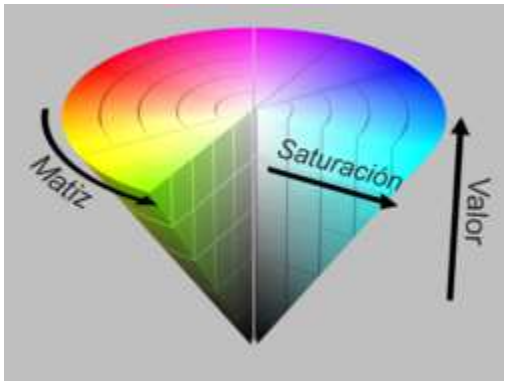

Figura 1 – Con representació HSV

- **Matís:** Es representa com un angle en graus el valor del qual pot oscil·lar entre 0º i 360º. En aquests 360º es divideixen els 3 colors RGB, per tant tenim 120º per a qualsevol dels 3 colors. Per exemple, el vermell és el 0, el verd és el 120 i el blau és el 240.
- **Saturació:** Es representa com al radi del con que té com a centre l'eix del valor o brillantor. Aquest valor es representa amb un percentatge que pot prendre valors entre el 0 al 100%. Com major sigui aquest valor, més "pur" serà el color, per altre banda, com més baix sigui, tindrem un color més grisos o popularment dit descolorit.
- **Valor:** El valor o brillantor serà l'altura del con. A un extrem trobem el blanc i en l'altre el negre. Els valors oscil·len entre el 0 i el 100%. Depenent de la saturació, el 100% podria representar el blanc.

Normalment les imatges que captura un ordinador tracta els colors com a RGB, en el cas que ens ocupa ens interessa que l'ordinador tracti els colors com a HSV degut a la metodologia que s'utilitza en el sistema de visió per computador. Així doncs, una de les coses que el nostre programa haurà de fer serà transformar els colors de RGB a HSV.

El mètode per a transformar els colors de RGB a HSV ve determinat per les expressions matemàtiques de la Fig. 2; on *MAX* representa el valor màxim de les components R, G i B i *MIN* és el valor mínim d'aquestes mateixes components.

Val a dir que la transformació de RGB a HSV és una funció definida per la llibreria OpenCV.

$$H = \begin{cases} \text{no definido}, & \text{si } MAX = MIN \\ 60° \times \frac{G-B}{MAX-MIN} + 0°, & \text{si } MAX = R \\ & \text{y } G \geq B \\ 60° \times \frac{G-B}{MAX-MIN} + 360°, & \text{si } MAX = R \\ & \text{y } G < B \\ 60° \times \frac{B-R}{MAX-MIN} + 120°, & \text{si } MAX = G \\ 60° \times \frac{R-G}{MAX-MIN} + 240°, & \text{si } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{si } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{en otro caso} \end{cases}$$

$$V = MAX$$

Figura 2 – Formules transformació RGB a HSV

4. Descripció del drone

En primer lloc és necessària una explicació de què és un drone. Un drone és una aeronau que vola sense tripulació sovintment utilitzat en l'àmbit militar. L'exemple més antic de drone va ser desenvolupat durant la primera guerra mundial i utilitzat en la segona guerra mundial per entrenar els operaris de canons. No va ser fins a finals del segle XX quan es varen començar a operar a través de ràdio control.

En aquests darrers anys, l'evolució de la tecnologia, ha fet que es puguin popularitzar un tipus d'aeronaus conegudes com a multirotors. El nom multirotor ve del fet de tenir varis motors que giren en diversos sentits. Aquestes aeronaus són capaces de donar a l'usuari una molt bona estabilitat de vol gràcies a l'electrònica que incorporen, i fins i tot, volar de forma totalment autònoma a partir de sistemes per gps. Aquests multirotors s'han convertit en els drones que avui en dia podem trobar pel carrer o en botigues d'aparells electrònics. Per tant, la tecnologia drone ha passat de ser d'ús exclusiu militar, a gairebé un be de consum que, fins i tot, ha obert noves oportunitats de negoci.

En aquest apartat s'explicaran les funcions de cadascun dels components que formen el drone utilitzat en aquest projecte i el perquè de la seva elecció. També s'aprofitarà per explicar alguns conceptes que ajudaran a la comprensió de la solució adoptada. En la Figura 3 podem veure l'esquema de connexió més comú d'un drone de 4 motors:

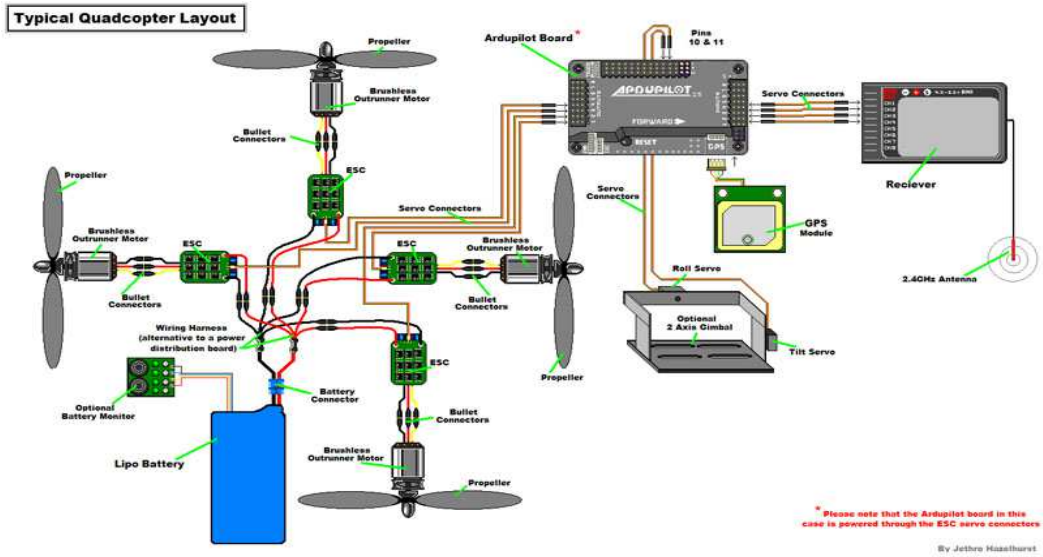

Figura 3 - Esquema de connexió dels elements d'un drone

4.1.Graus de llibertat del drone

Per entendre una mica millor el funcionament d'una aeronau d'aquest tipus és important entendre quines són les variables que deteminen els seus desplaçaments i elevacions: els seus graus de llibertat. Un drone té 4 graus de llibertat a diferència d'un avió que en té 3. Aquests graus de llibertat, representats en la Fig. 4 a excepció del throtle, són els explicats a continuació:

- **Throttle:** és el gas que donem a l'aeronau i serà el grau de llibertat que ens permet controlar el desplaçament vertical. Així doncs el throttle és el que ens determina l'elevació del drone.
- **Yaw:** el yaw és el gir que pot tenir el drone sobre el seu propi eix vertical. Aquest no genera cap tipus d'inclinació ni desplaçament, però ens permet encarar l'aeronau a la direcció desitjada.
- **Pitch:** el pitch és la inclinació respecte el pla horitzontal del drone que es transforma en avançament endavant o endarrere.
- **Roll:** el roll vindria a ser el mateix que el pitch a nivell conceptual però en aquest cas les inclinacions es tradueixen en un desplaçament lateral sobre el pla horitzontal.

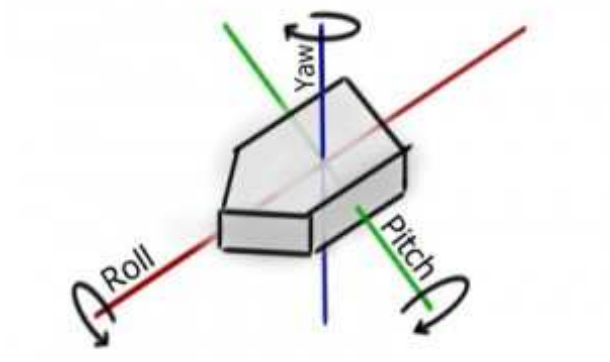

Figura 4 - Graus de llibertat d'un drone a excepció del throttle

4.2.El xassís

L'elecció del xassís del drone és una de les coses més importants a considerar per a que pugui tenir la funcionalitat desitjada. El primer que s'ha tingut en compte en l'elecció del xassís d'aquest drone és el nombre de motors. Considerant que la seva funció ha de ser en missions de rescat, l'autonomia de l'aeronau és un factor essencial a tenir en compte. A més motors, més consum i per tant menys autonomia. Així doncs,

en aquest projecte s'ha optat per un xassís de 4 motors. Això fa que ens puguem referir a la nau com a quadcopter. Un altre factor que influeix en l'autonomia és el pes. És per això que el xassís ha de ser lleuger a més a més de ser resistent als cops.

Una característica que determina la funcionalitat del drone és la distància entre motors. Els quadcopters destinats a competició solen ser molt petits i amb molt poca distància entre motors per a aconseguir una màxima agilitat i resposta a qualsevol petita variació que faci el pilot. Un exemple d'aquests tipus de xassís el podem trobar a la Figura 5.

Figura 5 - Xassís drone tipus "race" 250

En el cas que ens ocupa, és necessària la màxima estabilitat possible. Que no hi hagi canvis sobtats en el comportament del drone quan es produeixin les acceleracions i desacceleracions per a que el seguiment del gos sigui el més precís possible.

El xassís escollit per a aquest drone és un Iron Man 650 de la marca Tarot (Fig.6). L'Iron Man 650 és un xassís completament de fibra de carboni de quatre eixos, amb un diàmetre de 650 mm i un pes de 476g.

Figura 6 - Xassís Tarot Iron Man 650

4.3.Descripció de la placa de control

La placa de control del drone és el cervell de l'aeronau. Aquest tipus d'aeronaus basen el seu funcionament en bucles de control. La placa de control és l'encarregada de dur a terme lectures d'inclinacions en temps real mitjançant acceleròmetres i corregir aquestes inclinacions donant ordres als diversos motors a través dels diversos PID de navegació. Algunes d'aquestes plaques disposen de baròmetre per a calcular l'altura a la que es troba el drone i així poder-la mantenir en tot moment si l'usuari ho desitja.

Per a la realització d'aquest projecte és necessària una placa de control capaç de donar una màxima estabilitat al drone i que sigui de codi obert per a poder generar una comunicació entre el drone, un altre ordinador i una estació terrestre. A part d'això, també és necessari que aquesta placa pugui tenir un mòdul GPS per a millorar l'eficàcia del seguiment del gos i per a poder tenir la posició d'aquest en temps real.

La placa escollida és l'anomenada ArduPilot Mega 2.6 (APM 2.6) de 3DRobotics. Aquesta placa basada en arduino incorpora acceleròmetres, baròmetre i se l'hi pot acoblar un mòdul GPS. APM 2.6 forma part del projecte Open Source Arducopter.

Figura 7 - Kit APM 2.6 amb el GPS uBlox

Com es pot veure en la figura 7, tant la placa com el mòdul GPS tenen una orientació marcada per una fletxa i la inscripció "front". Segons com orientem aquests dos elements en el xassís, obtindrem un drone de tipus "+" o de tipus "x". En aquest cas s'ha optat pel tipus "x" degut a que és el més comú entre els quadcopters i a que la placa s'adaptava millor al xassís Tarot en aquesta disposició.

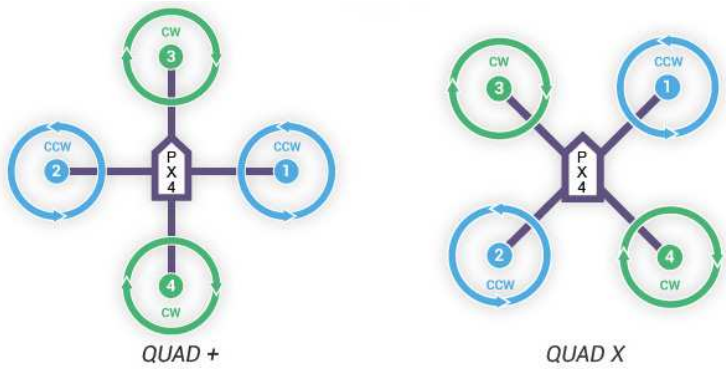

Figura 8- Tipus de quadcopter depenent de l'orientació de la placa

Arducopter consta d'un programa anomenat Mission Planner en el qual es duen a terme totes les configuracions inicials dels acceleròmetres, el GPS i la ràdio. També es poden dur a terme configuracions de nivell avançat com és el "tunning" dels diversos PIDs de navegació. A part de poder dur-hi a terme totes les configuracions, també ens ofereix una pantalla en la qual podem tenir informació de posició en temps real si tenim un mòdul de telemetria connectat a la placa o programar punts de geolocalització als quals automàticament el drone s'hi dirigeix sense necessitat de fer servir la ràdio.

4.4.Sistema motriu

El sistema motriu d'aquesta aeronau està compost per 4 motors, 4 ESC i 4 hèlices. A continuació es durà a terme una descripció de la funció de cadascun d'aquests elements.

4.4.1. Motors

Els motors escollits són de la marca eMax. Són motors trifàsics amb escombretes de 870 Kv. En aquest cas, quan parlem de Kv no s'ha de confondre en quilovolts. Kv és una velocitat constant, en voltes per minut, a la qual girarà el motor quan apliquem 1V. Això vol dir que si apliquem 1V a aquest motor tindrem una velocitat de 870 voltes per minut. Per norma general, quan es fan servir xassís grans en els quals es busca estabilitat, els motors han d'estar per sota dels 1000Kv. La raó d'això és senzilla: a menys voltes per volt i minut, menys bruscos seran els canvis de direcció i velocitat

que es produeixin al donar més o menys voltatge, traduint-se en un comportament estable i suau.

Dos dels motors giraran en sentit horari i els altres dos giren en sentit antihorari. El sentit de gir dels motors el podem veure en la figura 8. Per a modificar el sentit de gir del motor l'únic que hem de fer és intercanviar la connexió de dues de les fases com en qualsevol motor trifàsic.

4.4.2. Controlador de Velocitat Electrònic - ESC

ESC són les sigles de l'anglès Electronic Speed Controller. Tal i com diu el seu nom, l'ESC és un circuit electrònic que té com a propòsit controlar la velocitat i la posició del motor trifàsic. Els ESC escollits són de la marca Afro de 20 A.

4.4.3. Hèlices

La característica a destacar de les hèlices és la seva longitud i amplada. A més longitud i més amplada major serà la força que farà per a generar sustentació. Aquesta longitud però, va lligada al tipus de motors que haguem escollit. En aquest cas, el fabricant recomana col·locar pales de 10 o 11 polzades. Les que s'han muntat en aquest cas són unes pales de 10" de longitud i 4.7" d'amplada màxima.

Hem de tenir en compte que dues de les hèlices han de ser de sentit horari i les altres dues de sentit antihorari.

4.5.Bateria

Per alimentar tot el sistema és necessària una bateria. Les més comunes per a drones o aeronaus de ràdio control solen ser les Lithium-Polymer o més conegudes com a Li-Po. Les bateries de tipus Li-Po estan constituïdes per un càtode de liti aliat amb algun altre metall com pot ser el cobalt, un ànode de grafit i un polymer que fa d'electrolit. Aquestes bateries solen estar formades per varies celes que donen una tensió entorn als 3,6V. En el nostre cas, la bateria és de la marca Turnigy de 3 celes connectades en sèrie o 3S. Té una capacitat de 4000 mAh i ofereix 11,1 V a mitja càrrega.

4.6.Ràdio

La ràdio és el comandament utilitzat per controlar el drone. Com s'ha comentat anteriorment, un drone consta de quatre graus de llibertat que hem de ser capaços de controlar. Això significa que són necessaris quatre canals de ràdio per a controlar-los.

També hem de disposar d'un canal addicional per a poder canviar de mode de vol quan ho desitgem. A part de necessitar 5 canals destinats a nivell operacional, va bé poder tenir algun canal més lliure a la ràdio per a poder controlar servomotors o utilitzar altres funcions del Arducopter com l'AutoTune per a realitzar la configuració dels diversos PIDs de control del drone (aquesta funció serà explicada amb detall en el Capítol 6.7 d'aquesta memòria).

Per a aquestes raons s'ha escollit una ràdio de 9 canals de la marca Turnigy de 2,4GHz de freqüència que incorpora també el receptor que va col·locat al drone.

5. Projecte ArduPilot

ArduPilot és un projecte de codi obert que ens ofereix, el hardware, el firmware i el software per a que siguem capaços de crear el nostre propi vehicle terrestre o aeri. Aquest projecte neix l'any 2007 de la mà de la comunitat d'internet DIY Drones degut a la creació de la placa ArduPilotMega (APM) basada en la plataforma open-source Arduino.

Actualment el projecte ArduPilot ens ofereix la possibilitat de construir 3 tipus diferents de vehicles ràdio control que es divideixen en l'ArduCopter, ArduPlane i ArduRover. Per tant amb la placa APM podrem construir o bé un vehicle terrestre gràcies al firmware d'ArduRover o bé un vehicle aeri gràcies a ArduCopter o ArduPlane. En tots ells podrem incorporar unitats GPS per a poder programar missions autònomes en les quals el nostre vehicle es desplaçarà als punts que li indiquem per mitjà d'un ordinador o una tableta. Si disposem d'un mòdul de telemetria, també podrem saber quina és la posició del vehicle en temps real.

5.1. Modes de vol

Amb el firmware d'ArduCopter, APM 2.6 ens permet configurar fins a 6 modes de vol que podrem seleccionar gràcies al control remot. Podem triar entre una extensa varietat de modes de vol que van des dels més simples, sense cap tipus d'ajuda en els quals hem de ser capaços de controlar tots els graus de llibertat del drone, o podem seleccionar un mode de vol automàtic en el qual programem uns punts de pas en coordenades GPS i simplement hem de vigilar que el drone no es descontroli o xoqui contra algun obstacle.

Un dels modes de vol més interessants és l'anomenat ALT HOLD. Amb aquest mode de vol fixem l'altura i gràcies a les lectures en temps real del baròmetre, el drone manté l'altura en tot moment amb molt poques pèrdues de sustentació. Això fa que el comportament del drone sigui molt estable i molt més senzill de maniobrar ja que no ens hem de preocupar de les rectificacions d'elevació per a corregir les pèrdues de sustentació que es generen quan el drone es desplaça o gira. Val a dir que les lectures del baròmetre no són del tot constants i van variant en funció de les condicions climatològiques.

A part de les facilitats que ens brinda aquest mode de vol a nivell operacional, també ens pot facilitar el reconeixement visual ja que d'aquesta manera no hi haurà canvis sobtats en la dimensió de l'objectiu en les captures de vídeo i ens oblidem d'una altre variable a controlar.

6. Configuracions inicials de l'aeronau

Per a que el drone funcioni de forma correcta i amb la màxima estabilitat possible, són necessàries una sèrie de configuracions. Hem de tenir en compte que els drones basats en plaques de control APM no són del tipus "plug'n'play" és a dir, no és simplement endollar els cables allà on toca i el drone ja està preparat per a funcionar.

Així doncs tenim una sèrie de configuracions que són de caràcter obligatori: selecció del tipus de xassís, calibrat d'acceleròmetres, calibrat de la ràdio i la configuració del mòdul extern de GPS i magnetòmetre.

Per altre banda tenim una sèrie de configuracions que no són necessàries per a fer volar l'aeronau però que ens poden ajudar a millorar-ne el comportament i la funcionalitat d'aquesta com l'AutoTune.

Totes aquestes configuracions es duen a terme a la GCS Mission Planner a excepció de la configuració dels ESC i de l'AutoTune.

6.1. Tipus de xassís

El primer que hem de fer és dir-li a la placa de control quin tipus de xassís hem escollit i quina serà la disposició dels motors. ArduCopter ens permet múltiples configuracions d'aeronau en les quals els bucles que control que utilitzarà la placa per a donar les instruccions als motors no seran iguals.

Així doncs, aquesta primera configuració serà la base de l'aeronau i serà l'encarregada d'instal·lar el pertinent software a la placa APM 2.6.

6.2. Accelerometres

La placa APM 2.6 està dotada de diversos acceleròmetres. En aquesta configuració el que es fa és ensenyar a la placa quin és el seu davant, darrera, esquerra i dreta. Seguint els passos que ens marca el Mission Planner, anem realitzant les inclinacions que ens demana al drone i anem capturant les lectures que fan els acceleròmetres quan es troben en aquella posició.

6.3.Ràdio

Gràcies a aquesta configuració, la placa APM aprendrà quins són els límits màxims i mínims de PWM que la ràdio pot enviar. També aprendrà quins son els valors de PWM corresponent a les posicions neutrals dels joysticks. Aquest procés es duu a terme simplement movent tots els joysticks a les seves posicions més extrems movent-los amunt, avall, esquerra i dreta.

6.4. GPS i compàs

Per a aquesta configuració hem de moure el mòdul GPS a tantes posicions diferents com puguem fent-lo rotar sobre sí mateix durant 60s. Durant aquet temps, la placa APM enregistra prop de 1000 punts.

6.5.ESCs

Igual que en el cas del calibrat de la ràdio, els controladors de velocitat també necessiten saber quina és la posició màxima a la que pot arribar l'emissora de ràdio.

6.6.Modes de vol

Tal i com s'ha comentat anteriorment en l'apartat 5.2, APM disposa de la capacitat de configurar fins a 6 modes de vol diferents. En aquest cas els dos modes de vol que s'han prioritzat són:

- **Stable:** és el mode de vol més senzill en quant al paper que juga l'electrònica. Amb aquest mode de vol serà necessari controlar tots els graus de llibertat de la nau i fer les compensacions d'elevació a través del throttle necessàries per aconseguir una navegació on es mantingui l'altura constant.
- **Alt hold:** amb aquest mode de vol ens podem oblidar de les compensacions d'altura ja que el drone ho fa automàticament mitjançant lectures de pressió de l'aire a través d'un baròmetre. Pel que fa al Pitch, Roll i Yaw els podem controlar de la mateixa manera que en el mode de vol Stable.

6.7.AutoTune

ArduPilot Mega consta de 5 controladors de tipus proporcional P, un de tipus proporcional derivatiu PD i 5 de proporcionals integrals i derivatius PID tal i com es pot veure en la Figura 9. Tots ells venen amb unes constants predeterminades per a que es

pugui dur a terme un vol de qualitat sense necessitat d'haver de canviar cadascuna de les constants [4].

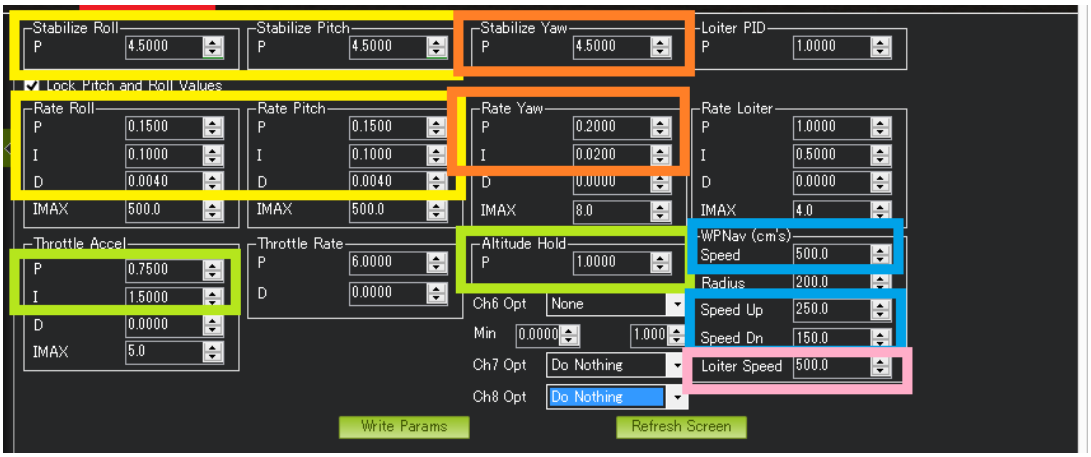

Figura 9 - Apartat de configuració de controladors que ens ofereix la GCS Mission Planner

En el cas que ens ocupa però, la resposta dels controladors que mantenen l'altura i donen les acceleracions pot condicionar molt el comportament que tingui el drone a l'hora de seguir l'objectiu, per tant és necessari afinar el màxim el comportament de cadascun dels controladors. Això s'aconsegueix amb el procediment anomenat Tuning. Arducopter té una opció per a optimitzar les constants dels controladors de forma automàtica anomenat AutoTune.

L'AutoTune intenta establir els valors òptims de la P Stabilize i la P i D dels Rate Pitch i Rate Roll per proveir la màxima resposta sense canvis bruscos. Això ho fa inclinant el drone en els eixos del roll i el pitch i canviant els valors anteriorment nombrats buscant obtenir una resposta determinada. Un cop s'obté la resposta desitjada aquests valors queden guardats.

7. Navegació autònoma

Com ja s'ha comentat anteriorment en aquesta memòria, el principal objectiu d'aquest projecte és aconseguir una navegació de forma totalment autònoma de l'aeronau per mitjà de tècniques de visió per computador. Per assolir aquest objectiu són necessaris una sèrie d'instruments i programari.

Degut a que la placa de control del propi drone no és capaç d'admetre la connexió d'una càmera i dur a terme el processament de les imatges ens és necessari un ordinador de dimensions reduïdes que sigui capaç de dur a terme aquest tractament, una càmera i finalment una font d'alimentació.

Per tant tindrem dues plaques a bord de l'aeronau que hauran de mantenir una comunicació estable entre elles. Per a dur a terme aquesta comunicació entre les dues ens serà necessari un programa que actuï de nexe entre les dues i que no consumeixi masses recursos per a que no hi hagi risc de que una de les dues plaques quedi saturada fent perdre el control de l'aeronau.

7.1.Estructura del sistema

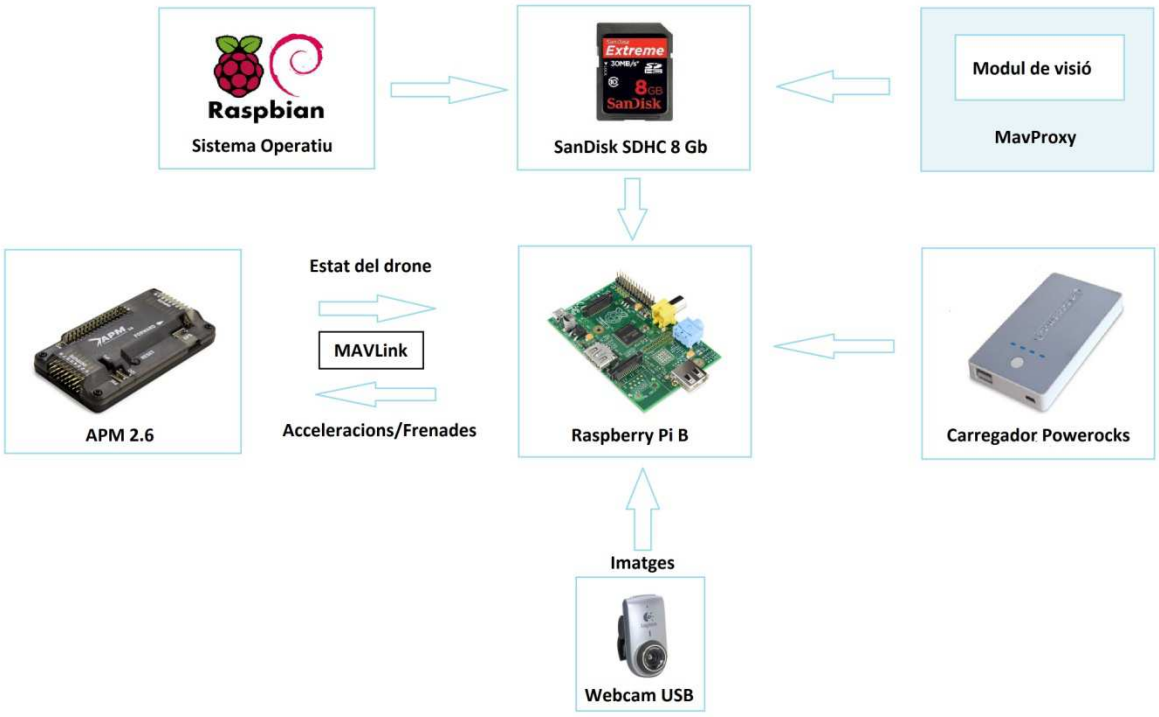

Figura 10 - Estructura del sistema de visió

La Figura 10 il·lustra el sistema que s'ha utilitzat en aquest projecte. Aquest sistema consta d'un ordinador de placa de dimensions reduïdes anomenat Rasberry Pi, una webcam amb connexió USB, un carregador portable per a smartphones com a sistema d'alimentació, i dos cables USB/microUSB. Pel que fa al programari emprat, s'ha instal·lat la GCS MAVProxy a la Raspberry Pi per a poder executar un programa de visió per computador que enviï comandes al drone. Tot el sistema ha estat pensat per a que tingui un consum baix i que no suposi un increment molt gran en el pes de l'aeronau per a que l'autonomia d'aquesta no se'n vegi afectada. Val a dir que tant la Raspberry Pi com la GCS MAVProxy seran necessàries també per a dur a terme la comunicació entre el drone i algun dels altres elements del projecte MATE com poden ser el robot BIGBOT o un ordinador que faci d'intermediari. Això ho podem veure explicat amb més detall al Capítol 10 d'aquesta memòria.

7.1.1. MAVProxy

MAVProxy és una GCS (de l'anglès "Ground Control Station") completament funcional i minimalista pensada per a que no consumeixi excessius recursos de l'ordinador i d'aquesta manera tenir la capacitat de ser portable. MAVProxy utilitza el protocol MAVLink per a comunicar-se amb la placa APM 2.6.[5]

Aquesta GCS disposa de diversos mòduls que un cop són importats ens ofereixen la possibilitat de rebre informació del drone o bé enviar informació a aquest. Aquests mòduls, programats en el llenguatge Python, són aplicacions que ens permeten augmentar la funcionalitat de MAVProxy, per exemple podem importar un mòdul que carrega un mapa de google maps i en el qual es veu la posició en temps real del nostre drone. A més a més tenim la possibilitat de programar el nostre propi mòdul ja que tots ells són de codi obert com el mateix programa MAVProxy.

7.1.2. MAVLink

MAVLink o Micro Air Vehicle Link, és un protocol de comunicació per a vehicles sense pilot. S'utilitza per a transferir la posició GPS, la velocitat , etc d'un vehicle sense pilot a una GCS instal·lada en un ordinador.

Un missatge de MAVLink és bàsicament una transmissió de bytes que han sigut codificats per la GCS i són enviats a la placa APM via USB o antenes de telemetria.

Cadascun dels paquets de MAVLink té una dimensió de 17 bytes dels quals 6 són per l'encapçalament del missatge, 9 per a la informació que conté el missatge i 2 de checksum (determina sí es tracta d'un missatge de MAVLink vàlid).

7.1.3. Mòdul de visió

El mòdul de visió és un programa en Python creat per Collin Doolittle i David Wurtz, dos estudiants de la Portland State University durant un projecte d'investigació de sis mesos. El que fa aquest mòdul és capturar imatges a partir d'una càmera USB i tractar-les mitjançant tècniques de visió per computador. Llavors un cop detectada la regió d'interès es fa una predicció del desplaçament que ha tingut l'objecte que s'està seguint i amb aquesta predicció de posició es donen una sèrie de comandes al drone.

Degut a que el mòdul de visió va ser programat pels voltants del 2013 i no ha estat actualitzat d'ençà, hi ha una incompatibilitat entre l'última versió oficial de MAVProxy i el mòdul de visió. Així doncs s'utilitzarà una versió més antiga. Aquesta la podem descarregar en un arxiu .zip en el següent enllaç: ***www.github.com/dwrtz/MAVProxy***.

7.1.4. Raspberry Pi

La Raspberry Pi és un ordinador de placa de dimensions reduïdes i de baix cost desenvolupada al Regne Unit per la Fundació Raspberry Pi. Existeixen diversos models de Raspberry Pi amb diverses característiques, en aquest projecte s'ha utilitzat la model B de 512 MB de memòria RAM. Disposa d'una sortida d'imatge per HDMI, dos ports USB, un port ethernet, una ranura per a targes SD i una presa de corrent per mitjà d'un connector microUSB. La Raspberry Pi no té disc dur; normalment s'utilitza com a disc dur una tarja SD en la qual s'instal·la el sistema operatiu.

El projecte Raspberry Pi disposa d'un sistema operatiu propi anomenat Raspbian que està basat en Linux, a part d'oferir-nos l'oportunitat d'instal·lar qualsevol altre sistema operatiu basat en Linux. En aquest projecte s'ha optat per la instal·lació del sistema operatiu Raspbian en una SDHC de 8 GB Classe 10.

El roll d'aquest ordinador és fonamental en aquest projecte. La Raspberry Pi portarà instal·lada la GCS MAVProxy amb el mòdul de visió i una webcam connectada a

aquesta per mitjà d'un dels ports USB. Així doncs, la Raspberry Pi realitza l'adquisició d'imatges, el corresponent tractament i envia ordres de desplaçament al drone.

7.1.5. La càmera

El mòdul de visió està pensat per funcionar amb una càmera que estigui connectada a l'ordinador per mitja d'un port USB o bé que estigui integrada al propi ordinador com són el cas de les webcams dels ordinadors portàtils o en el cas de la Raspberry Pi seria la Pi cam.

La càmera utilitzada en aquest projecte és una webcam capaç de capturar imatges a una velocitat màxima de 30 fotogrames per segon, té una resolució màxima de 1,3 megapixels a 640x480 pixels. A part d'aquestes característiques, aquesta webcam té unes dimensions i un pes bastant reduïts.

En l'elecció de la càmera el que és important, a part de que sigui compatible amb la Raspberry Pi, és la qualitat del sensor. Si el sensor és de mala qualitat no obtindrem una imatge prou nítida i el sistema reaccionarà malament als possibles canvis sobtats de les condicions de llum.

7.1.6. Sistema d'alimentació

La Raspberry Pi ha de ser alimentada a 5V i 1A, i llavors aquesta transferirà l'electricitat necessaria a la webcam per mitjà de l'USB. Per a realitzar aquesta alimentació s'ha utilitzat un carregador portàtil per a smartphones d'una capacitat de 1500mAh. D'aquesta manera podem connectar un cable USB/microUSB directament del carregador a la Raspberry Pi.

8. Entrenament visual

Abans de ser capaços d'utilitzar el mòdul de visió computacional "mavproxy_vision.py", és necessari executar quatre programes previs que generen un fitxer .txt anomenat "classifier_and_feature_scaler.txt" el qual conté informació encriptada que ens servirà per a millorar l'eficàcia del nostre sistema de seguiment.

Val a dir que els paràmetres CAPTURE_WIDTH, CAPTURE_HEIGHT, HSV_UPPER_LIMIT, i HSV_LOWER_LIMIT en tots quatre programes serà necessari que siguin iguals respectivament i també serà necessari l'ús de la mateixa càmera. Els dos primers paràmetres seran els que marcaran la resolució a la que capturarem les imatges. Aquesta resolució ha de ser compatible amb la càmera utilitzada, suficientment gran com per distingir la taca a detectar i, per altre banda, no pot ser excessivament gran ja que el sistema es torna molt més lent. Els altres dos paràmetres són els que marquen els valors d'HSV límit entre els quals es troba el color objectiu. La determinació dels valors d'HSV serà explicada a continuació.[6]

8.1.Elecció paràmetres color HSV

L'elecció dels paràmetres HSV seran els que ens condicionaran en gran part l'eficàcia del nostre sistema de seguiment per computació visual. Així doncs hauran de ser suficientment acurats, sense que siguin massa restrictius per a que si varien les condicions de llum continuï duent a terme una detecció de qualitat però també sense ser massa tolerants per a que no sorgeixin falsos objectius o pertorbacions durant el seguiment.

Per a dur a terme l'elecció d'aquests paràmetres s'ha utilitzat el script "hsvtool.py" el qual consta d'un panell amb una barra lliscant (Figura 11) per a cadascun dels paràmetres HSV del límit superior i del límit inferior. Gràcies a aquestes barres podem anar jugant amb els diversos valors i visualitzar en una petita pantalla en temps real com es veu el nostre objectiu.

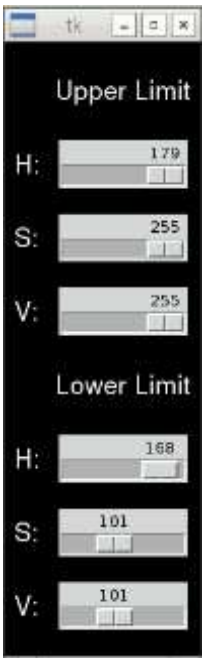

Figura 11 – Sliders "hsvtool.py"

Un cop obtingut el resultat desitjat s'anoten els valors obtinguts i aquests seran els que utilitzarem durant tot el procés que es detallarà en els següents apartats i durant el procés de seguiment que realitzarà el drone. Val a dir que és necessari utilitzar la mateixa càmera en totes les fases del procés degut a que cada càmera utilitza uns filtres de color diferents i això afectaria al bon funcionament del sistema.

Figura 12 – Resultat de l'eina "hsvtool.py"

8.2.Entrenament positiu i negatiu

L'entrenament positiu i negatiu es duen a terme amb els scripts anomenats "pytrainingpositive.py" i "pytrainingnegative.py". Bàsicament els dos scripts contenen el mateix codi, la única diferència entre els dos és el procediment que durem a terme

nosaltres mentre aquests s'estiguin executant i com s'interpretaran llavors els fitxers .txt que en derivin de cadascun.

En l'entrenament positiu el que ens interessa és que el nostre objectiu estigui dins de la imatge en tot moment. Així doncs mentre s'executa el script "pytrainingpositive.py" el que s'ha de fer és variar l'enquadrament i la distància de la webcam però sempre vigilant que l'objectiu aparegui a la imatge en tot moment.

Per altre banda, durant l'execució de l'entrenament negatiu, durem a terme el mateix procediment però en aquest cas l'objectiu no ha de sortir en cap moment en pantalla. D'aquesta manera el programa recull dades de com serà la imatge quan apareix l'objectiu i de com serà quan no apareix en la imatge.

Com ja s'ha dit, la funció d'aquests dos scripts serà la de generar els fitxers "training_positive.txt" i "training_negative.txt". Aquests dos fitxers contindran una sèrie de mitjanes, variàncies i màxims (x_coord, y_coord, xmean, ymean, xpeakvalue, ypeakvalue, etc.) sobre les imatges que s'han capturat i tractat durant l'entrenament positiu i negatiu.

El procediment per a generar aquests valors comença amb la captura d'un fotograma provinent de la webcam. Llavors aquesta imatge es transforma a HSV i es compara amb els límits superiors i inferiors HSV_UPPER_LIMIT i HSV_LOWER_LIMIT. Tot el que no es troba dins d'aquests límits passa a ser negre i el que es troba dintre passa a ser de color blanc.

Abans de poder prosseguir és important explicar una de les funcions més importants d'aquest codi. MatchTemplate [7] és una instrucció de la llibreria OpenCV en la qual tenim una imatge font i una plantilla. La plantilla es va desplaçant d'esquerra a dreta i de dalt a baix a través de la imatge font. Per a cadascuna de les posicions en les que es troba la plantilla respecte la imatge font se n'extreu un valor que mesura el grau de coincidència entre la imatge font i la plantilla. Amb tots aquests valors es genera una nova matriu de resultats.

A continuació en el nostre codi, es declaren dues matrius d'uns amb la funció "ones"; una matriu fila i una matriu columna de les mateixes longituds que l'amplada i l'alçada

de la imatge respectivament. Aquestes dues matrius ens serviran de plantilles per a poder trobar els histogrames respecte els eixos x i y de la imatge a través de la comanda matchTemplate. S'obtenen dues noves matrius x_ccor i y_ccor amb els valors de l'histograma. D'aquestes dues en podrem extreure tota la informació necessària per a generar el fitxer .txt.

La comanda minMaxLoc ens ajuda a determinar quins són els mínims i els màxims d'una matriu i determinar-ne la seva posició. Per a establir quina és la posició del nostre objectiu s'utilitza la comanda minMaxLoc per a determinar quina és la posició dels valors màxims de les matrius x_ccor i y_ccor. Aquests dins del codi reben el nom de x_coord i y_coord.

Un cop ja tenim localitzat l'objectiu, per recol·lectar la resta d'informació necessària per a generar els fitxers .txt es fa una mitjana de tots els valors que contenen x_ccor i y_ccor, trobant els valors xmean i ymean respectivament. També obtenim la variància xvariance i yvariance de cadascuna de les matrius. Xpeakval i ypeakval seran els valors màxims que hem obtingut abans amb la comanda minMaxLoc dels quals hem dit que la seva posició en la matriu ens determinen la posició de l'objectiu. Per acabar es busca quin és el 90% de l'amplitud del pic de les matrius que hem trobat mitjançant la funció "peak_width"; que està definida en les primeres línies del codi. Tota aquesta informació s'encripta i es guarda en el fitxer .txt. El programa finalitza quan s'aconsegueixen 1000 línies d'informació.

Mentre anem executant el programa, podem anar veient en pantalla la imatge en blanc i negre resultant de la computació i en vermell un punt senyalant el punt marcat per les coordenades x_coord i y_coord.

En aquesta memòria no s'han adjuntat cap d'aquests dos codis en els annexes perquè aquest codi que s'ha explicat també es troba de forma implícita en el mòdul de visió que es pot trobar en l'annex A d'aquesta memòria.

8.3.Generació del fitxer classificador i escalador

Arribats a aquest punt ens trobem amb dos fitxers .txt de característiques similars que contenen els mateixos tipus de paràmetres. La correcta interpretació de la informació que contenen aquests recau sobre el programa "trainclassifier.py". Amb l'execució d'aquest script generarem el fitxer "classifier_and_feature_scaler.txt".

Aquest script està basat en la llibreria d'anàlisis de dades i aprenentatge per a màquines Sikit-learn. En el nostre cas el script "trainclassifier.py" analitza les dades dels fitxers "positive_training.txt" i "negative_training.txt" generant un sol fitxer en el qual les dades que provenen de l'entrenament positiu seran interpretades com a una resposta vertadera i les dades provinents de l'entrenament negatiu seran interpretades com a falses.

8.4.Verificació de l'entrenament

Un cop generat el fitxer "classifier_and_feature_scaler.txt" podem provar el seu funcionament executant el script "vision.py" en el qual veurem en pantalla la imatge provinent de la càmera i la imatge un cop tractada amb un punt en vermell indicant quina és la posició que el programa considera que té l'objectiu tal i com podem veure ne la Figura 13.

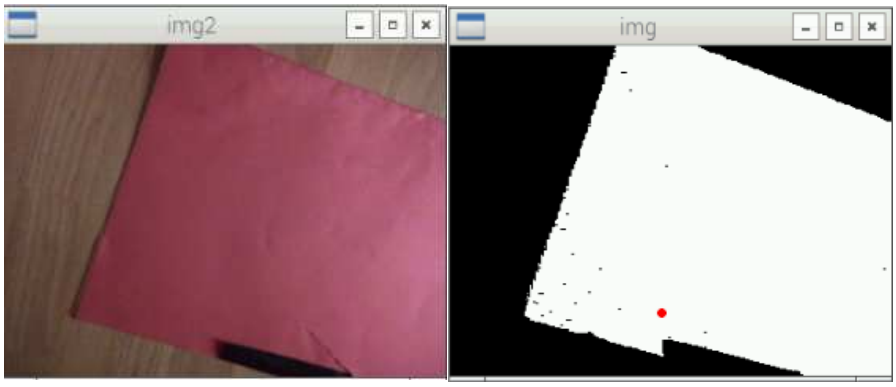

Figura 13 – Imatge raw i imatge processada amb la posició marcada amb un punt vermell

El programa "vision.py" conté el mateix codi per ala realització del reconeixement visual utilitzat en el mòdul de visió "mavproxy_vision.py". Mentre el script estigui funcionant, imprimirà en pantalla la paraula TRUE si detecta l'objectiu i FALSE si no el detecta com es pot veure en la Figura 14.

Un cop obtingut el resultat desitjat és necessari copiar el fitxer "classifier_and_feature_scaler.txt" en el mateix directori de la Raspberry Pi on es troba instal·lat el MAVProxy.

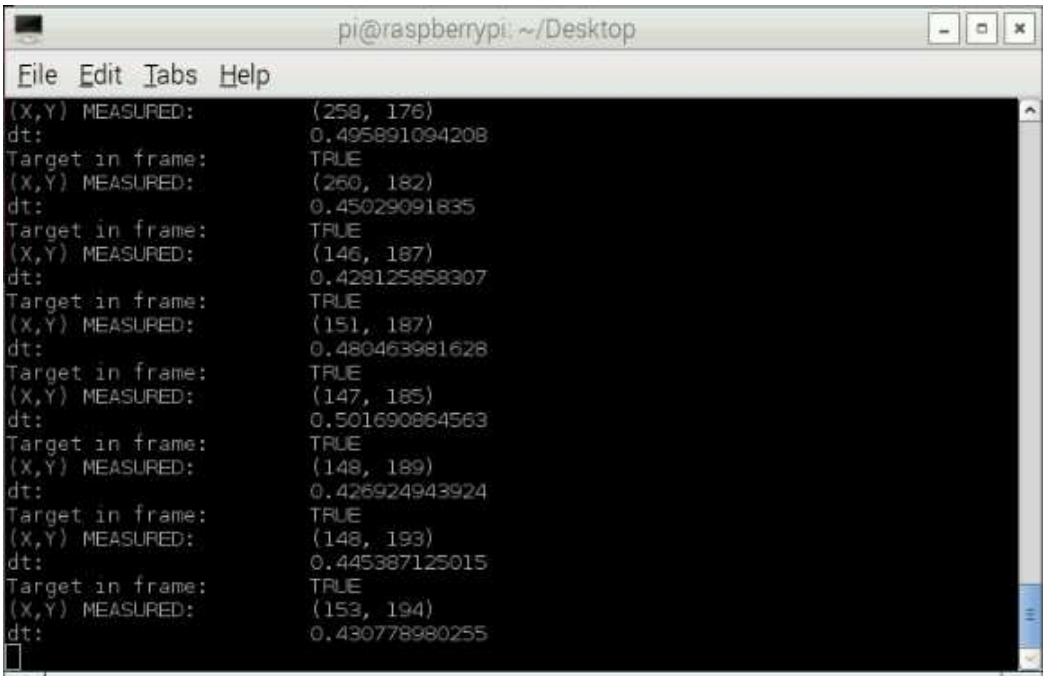

Figura 14 – Resultats de la detecció

9. Modul de visió "mavproxy_vision.py"

El mòdul de visió és el cor d'aquest projecte. Gràcies a ell aconseguim una navegació automàtica del drone a partir del reconeixement visual. Aquest mòdul el podem dividir principalment en dues parts les quals estan estretament unides. En primer lloc tenim la part en la qual es duu a terme el reconeixement visual a partir de les imatges captades per la càmera i en segon lloc, un cop detectat l'objectiu i posicionat, tenim una part del codi en el qual s'envien una sèrie de comandes al drone per a posicionar-lo sobre l'objectiu. A continuació es durà a terme una explicació del funcionament de cadascuna d'aquestes parts i en què consisteix un mòdul de MAVProxy.

9.1. Estructura d'un mòdul de MAVProxy

L'estructura d'un mòdul de MAVProxy és bàsicament la mateixa que la de qualsevol script programat en Python amb algunes petites peculiaritats. Aquests scripts bàsicament consten d'unes variables que s'han de declarar en primer lloc com a classes amb la instrucció "self" al davant de cadascuna d'aquestes.

Llavors es defineixen una sèrie de funcions que seran els processos que es duran a terme mentre el mòdul estigui en funcionament. Algunes d'aquestes funcions són obligatòries en qualsevol dels mòduls de MAVProxy, com són la funció que dona inici al mòdul un cop és invocat en la consola de MAVProxy o la funció que s'utilitza per enviar paquets de MAVLink, d'altres són les que donaran la funció i funcionalitat del mòdul.

9.2. Detecció de l'objectiu

En primer lloc el mòdul realitza una comprovació en la qual es dona inici al "loop" de visió. Aquesta comprovació consisteix en determinar si el nostre drone es troba en el mode de vol "ALT HOLD". Aquest mòdul està dissenyat per donar autonomia al drone un cop es determina una altura fixa, per tant, per molt que el mòdul de visió estigui carregat a MAVProxy i el drone estigui volant, si no es troba a una altura fixada pel mode de vol "ALT HOLD" aquest no funcionarà.

Un cop el mòdul ha comprovat que es troba en el mode de vol correcte s'inicia la detecció de l'objectiu. Aquesta utilitza el mateix procés que s'ha explicat en l'apartat 9.2 "Entrenament positiu i negatiu". En primer lloc tindrem una imatge font provinent

de la càmera que es compararà amb dues matrius d'uns; una matriu columna i una matriu fila amb la funció d'OpenCV matchTemplate. Llavors com ja s'ha vist, s'extreu una sèrie d'informació com són la posició de l'objectiu i una sèrie de mitjanes i variàncies. Mentre es duien a terme els entrenaments positiu i negatiu, la posició de l'objecte al programa li era una mica indiferent, el que buscava recol·lectar principalment eren les altres dades. En aquest cas les mitjanes, variàncies i valors màxims en els eixos x i y ens serveixen per a comprovar si l'objectiu es troba en la imatge, de no ser així, no es transmet cap posició al "loop" de seguiment. Això ho fa comparant les dades que s'han extret de l'últim fotograma captat per la càmera, amb la informació encriptada que conté el fitxer "classifier_and_feature_scaler.txt". Aquesta comprovació evita que el drone es torni boig buscant l'objectiu on no hi és. Llavors si la resposta és vertadera i per tant l'objectiu es troba en la imatge, es dona l'ordre d'iniciar el seguiment.

9.3.El seguiment

Un cop s'han complert les dues condicions que s'imposen per a que es pugui dur a terme el seguiment; que el drone estigui en "ALT HOLD" i l'objectiu es trobi en imatge, es pot iniciar el seguiment.

Després de realitzar la detecció de l'objectiu el programa ja en coneix la seva posició, o si menys no en té una estimació. Hem de tenir en compte que el programa sap en quin pixel de la imatge es troba el que ell considera la posició de l'objectiu, però aquesta posició s'ha de traslladar a una sèrie de comandes que donin acceleracions i desacceleracions al drone per a que s'hi pugui situar just a sobre. La forma de fer-ho és utilitzant dos controladors PID diferents. Un serà per l'eix de les x i l'altre per l'eix de les y. El resultat de cadascun es traduirà en un impuls en l'eix del "roll" i l'eix del "pitch" respectivament si aquests són necessaris.

Sí el drone estigués sobre l'objectiu, el punt que s'ha trobat en la detecció de l'objecte coincidiria amb el centre de la imatge. Si es dona aquest cas el drone roman en la seva posició, de no ser així es calcula l'error relatiu en els eixos x i y de la posició que ocupa l'objectiu en la imatge respecte el centre de la pròpia imatge. Així doncs, d'aquí en podem extreure dues informacions molt importants a l'hora d'entendre el

funcionament del sistema i la modelització del PID; la primera és que el valor de consigna serà un valor constant que es referirà al centre de la imatge, la segona és que aquest error relatiu que s'ha trobat serà l'alimentació del PID de control.

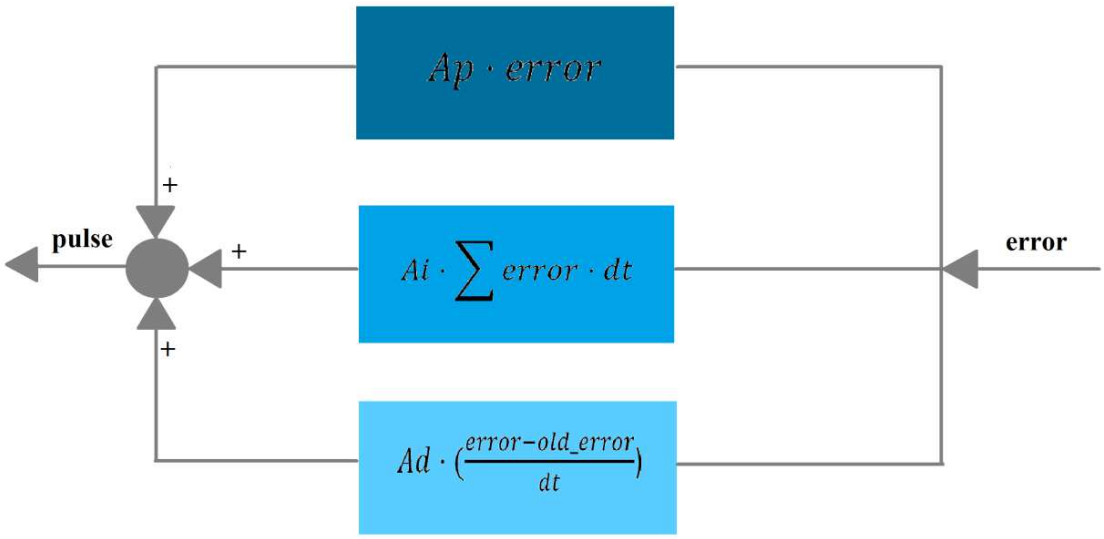

Figura 15 – PID de control del mòdul de visió

Com es pot veure en la figura 15, l'error entra al PID i després de ser tractat obtenim la variable "pulse". Al tenir un PID per a cadascun dels eixos que controlen els desplaçaments del drone obitndrem un pulse per al "roll" i un per al "pitch". Aquesta sortida del PID però, no és el valor definitiu per a enviar les comandes al drone.

Com ja s'ha dit anteriorment en aquesta memòria, les ordres es donen en PWM (Polsos Modulats en Amplada). Durant la configuració del mòdul en la que posarem els rangs de HSV i la resolució de la imatge que captura la nostre càmera, també haurem de donar els valors en els que es troben el PWM del "pitch" i el "roll" quan es troben en la seva posició neutra, és a dir, quan no estan donant cap tipus d'acceleració. Sobre aquests valors seran els que es sumaran o restaran (depenent del sentit del desplaçament i de l'eix en el que es realitzin) el "pulse" multiplicat per un factor de 50. D'aquesta manera s'obtenen els valors de PWM que s'envien de la Raspberry PI a l'APM 2.6 i per tant es transformen en acceleracions en el cas de que els valors siguin diferents a els valors neutres de PWM. Val a dir que aquests valors neutres de PWM tenen un petit punt mort el qual si no es sobrepassa el drone quedarà immòbil encara que el "pulse" obtingut del PID no sigui 0.

El comportament d'un drone és similar al d'un cotxe, per molt que deixem de donar el gas, el cotxe continua en moviment fins que no premem el fre o ha patit suficientment fricció com per aturar-lo. En aquest cas, després de donar l'ordre de desplaçament, per molt que aquesta ordre ja estigui executada, el drone continuarà el moviment fins que no se li doni una ordre de frenada. El que fa el mòdul per a contrarestar aquest desplaçament involuntari és el següent; deixa que continuï amb el desplaçament provocat per la nova comanda de PWM durant tres segons i llavors envia una ordre nova a la placa APM per a que es restableixin els valors de PWM neutrals per aconseguir una frenada del drone.

10. La comunicació amb el robot BIGBOT

Un cop explicada la solució adoptada per a l'objectiu principal d'aquest projecte, es discutirà una solució per al segon objectiu plantejat. En aquest capítol es discutirà la solució per a dur a terme la transferència d'informació del drone al robot terrestre BIGBOT. Aquesta solució s'ha acordat entre l'autor d'aquest projecte i l'altre projectista encarregat d'adaptar el robot BIGBOT per a que sigui capaç de rebre la informació del drone.

Abans però, és important entendre el perquè d'aquesta comunicació i quan es realitzarà; ja que el drone i el sistema de visió que s'han plantejat fins ara en aquesta memòria tenen la finalitat de ser un suport més per al robot terrestre BIGBOT. Recordem que el robot BIGBOT segueix al gos de rescat, a partir de tècniques de visió per computador, per a donar-li suport i ordres. El robot BIGBOT no sempre és capaç de veure en imatge el gos de rescat i quan això succeeix, no sap quina és la següent posició cap a la que ha d'avançar. En la Figura 16 podem veure una de les situacions en les quals el robot podria perdre de vista el gos de rescat en un terreny irregular.

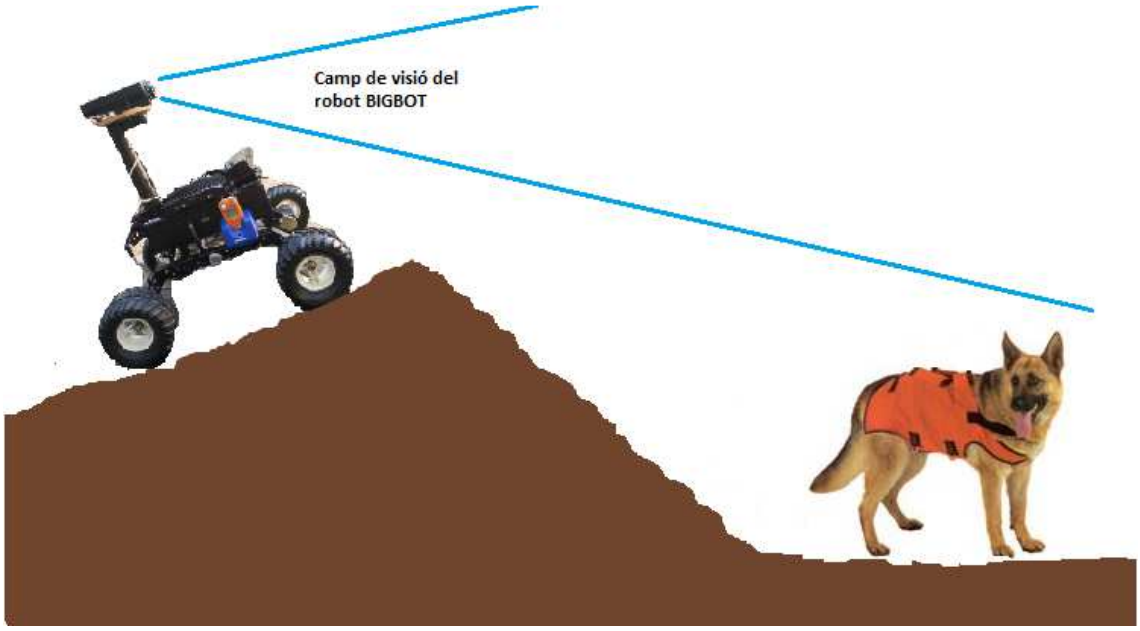

Figura 16 – Exemple robot BIGBOT sense visió del gos.

Amb la implementació del drone es poden corregir aquestes mancances del sistema de seguiment del robot BIGBOT, tal i com es pot veure en la Figura 17. Per tant, el drone entra en joc, dins d'aquest sistema, en el moment en el qual el robot no sap cap on anar. És llavors quan aquest demana la posició GPS al drone. Suposadament (dependrà de l'eficàcia del sistema de seguiment del drone) el drone es trobarà en tot moment sobre el gos de rescat i per tant aquesta posició de GPS podem dir que serà la del gos.

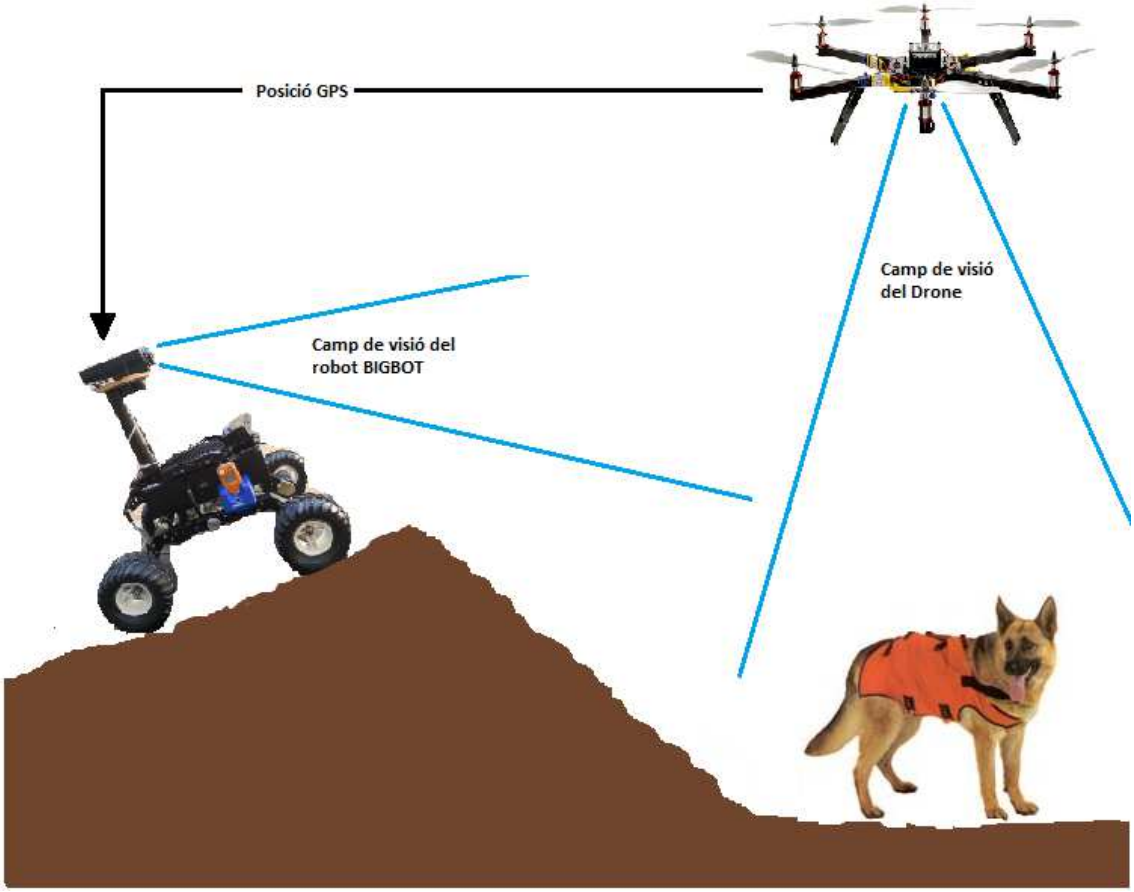

Figura 17 – Implementació del Drone en el sistema de seguiment del projecte MATE

Des del punt de vista aeri del qual disposa el drone, el sistema de seguiment de visió per computadora amb el que compta pot abastar més terreny que el robot BIGBOT, sempre i quant la resolució de la càmera sigui bona.

En la Figura 18 podem veure els elements que conformen el sistema escollit per a realitzar aquesta comunicació entre el drone i el robot BIGBOT o un ordinador que faci d'intermediari entre els dos.

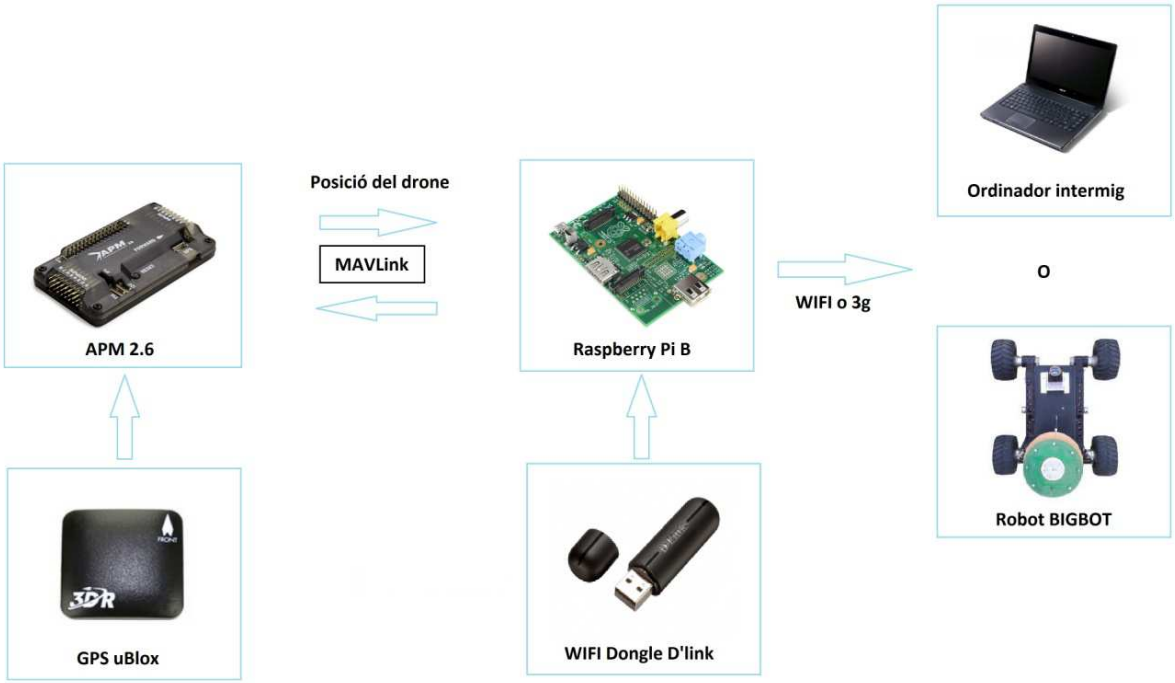

Figura 18 – Esquema comunicació

10.1. Robot Operating System – ROS

ROS és una plataforma basada en Ubuntu, que consta d'eines, llibreries i expressions, amb la qual podem programa software per a comportaments complexes d'un robot d'una forma simplificada [9]. Aquest sistema operatiu és el que porta instal·lat el robot BIGBOT per a poder tenir un comportament operacional de qualitat.

ROS permet l'ús de diversos llenguatges a l'hora de programar. Els llenguatges oficials que pot soportar són el Python, C++ i Lisp. A més, hi ha molts altres llenguatges que es troben en fase experimental com és el cas de java.

10.2. Connexió de MAVProxy en remot

Una de les opcions que ens ofereix MAVProxy és a possibilitat d'establir una connexió en remot per a poder enviar paquets de MAVLink a un altre ordinador, ja sigui a través d'una connexió per mitjà d'un USB o bé via WIFI. En aquest cas ens interessa realitzar aquesta connexió via WIFI davant de la impossibilitat de tenir un cable que connecti el

drone amb el robot BIGBOT. La Raspberry Pi no compta amb una tarja de xarxa per a connexions via WIFI. Així doncs, s'ha utilitzat un USB HUB per a poder tenir una connexió WIFI a la Raspberry Pi.

Per a realitzar la connexió en remot de la Raspberry a un altre element, simplement hem d'especificar quina és la direcció IP del dispositiu al qual volem realitzar aquesta connexió en el moment en el qual iniciem el programa MAVProxy.

10.3. La transmissió de dades

MAVProxy utilitza el protocol de comunicació MAVLink per a enviar i rebre paquets de dades provinents de la placa de control APM. En aquest cas les dades que ens interessa rebre de la placa APM són les provinents del mòdul de GPS uBlox connectat a aquesta. És en aquest moment en el que entra en joc el programa facilitat per l'altre projectista: el "roscopter_node.py" (Annex C).

El programa "roscopter_node.py" és el programa que completa la connexió remota del MAVProxy. "roscopter_node.py" està instal·lat en el robot BIGBOT i el pc intermediari, aquest s'encarrega de buscar una connexió que tingui com a protocol de comunicació MAVLink, que en aquest cas serà la connexió en remot que hem generat a partir de MAVProxy.

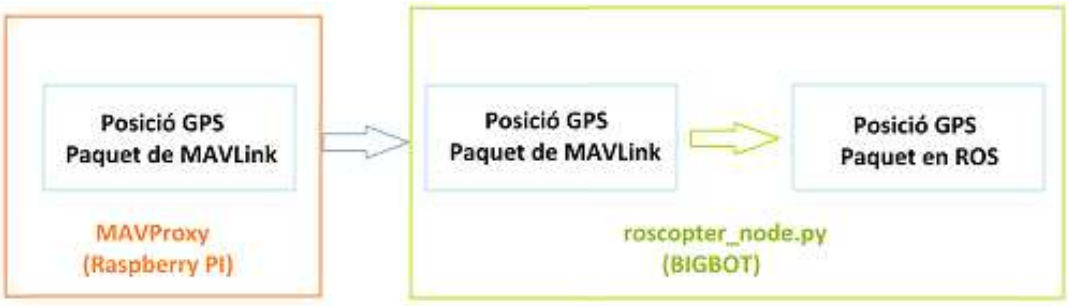

Figura 19 – Esquema traspàs de paquets

Un cop establerta aquesta connexió entre MAVProxy i "roscopter_node.py", aquest últim li demana la posició del drone a MAVProxy i aquest li envia en un paquet de MAVLink. En aquest punt, com es pot veure en la Fig. 19, les dades de posició ja estan al robot BIGBOT (en cas de que "roscopter_node.py" s'estigui executant des del robot BIGBOT), però com que estan en forma de paquets de MAVLink, el robot no les pot

interpretar. Com s'ha dit en l'apartat 10.3, el robot BIGBOT funciona amb ROS i per tant és necessari transformar aquests paquets de MAVLink en paquets de ROS. Aquesta funció també la duu a terme el propi programa "roscopter_node.py".

11. Prova del sistema de comunicació

En aquets capítol es plantejarà el procediment a seguir per a realitzar una prova del sistema de comunicació. Tal i com en el plantejament del sistema de comunicació, aquesta prova ha estat acordada entre l'autor d'aquesta memòria i el projectista encarregat d'adaptar els sistemes del robot BIGBOT.

La prova es realitzarà amb els següents elements del sistema: el drone amb la Raspberry Pi, un ordinador intermediari i un router. El router té la funció d'establir una xarxa local per a poder tenir un mitjà de comunicació entre el drone i el pc.

Connectarem l'ordinador a través d'un cable ethernet al router. També connectarem la Raspberry Pi al router a través del WIFI. La direcció IP de l'ordinador és fixe, per altre banda, la de la Raspberry Pi serà una IP dinàmica. Com que la direcció IP de l'ordinador ja és coneguda, només haurem de buscar la de la Raspberry PI.

A continuació executarem MAVProxy des de la Raspberry Pi, connectada a la placa APM 2.6. En aquest cas afegirem un "out" a la comanda que dona inici a MAVProxy. Cal destacar dues coses d'aquest "out": haurem d'especificar quin protocol de comunicació utilitzarem (en aquest cas serà el protocol UDP) i la IP de l'ordinador intermediari.

Des de l'ordinador intermediari s'estarà executant el programa "roscopter_node.py". En aquest haurem d'especificar quina serà l'entrada per on rebrà els paquets de MAVLink. Això ho farem indicant el protocol de comunicació (que serà UDP), la direcció IP de la Raspberry Pi i el port de la comunicació.

Considerarem que s'ha obtingut un resultat satisfactori d'aquesta prova en el cas de reclamar la posició GPS del drone a través del programa "roscopter_node.py" i que aquesta aparegui en la pantalla de l'ordinador intermediari.

12. Resultats

Totes les proves del sistema de visió s'han realitzat amb condicions climatològiques favorables o en espais tancats. També s'han realitzat proves de vol, amb el drone controlat des de la ràdio, en condicions de vent sense el sistema de visió.

Per a la realització de les proves del drone amb el sistema de visió en funcionament s'ha utilitzat la següent metodologia: en primer lloc armem el drone, llavors l'enlairem en el mode de vol STABILIZE fins a una altura de 1 – 2 m, llavors fixem l'altura passant al mode de vol ALT HOLD i finalment situem el drone sobre de l'objectiu amb l'ajuda del comandament de ràdio.

El drone amb el sistema de visió incorporat és capaç de seguir l'objectiu ja estigui aquest parat o en moviment. El millor dels casos és quan l'objectiu es mou a una velocitat baixa per a que aquest continuï dins de la imatge en tot moment i el sistema de visió continuï sent capaç de realitzar el seguiment. Aquest cas és més favorable que en el cas on l'objectiu es troba estàtic degut al fet comentat anteriorment en l'apartat 9.3 sobre les acceleracions i frenades que duu a terme el drone. Si l'objectiu continua el moviment en l'última direcció en la qual s'ha desplaçat el drone, l'aeronau poques vegades haurà de realitzar una rectificació per acabar-se de posicionar sobre l'objectiu, simplement continuarà cap a la següent posició on ha detectat l'objectiu.

En el cas de que l'objectiu es trobi parat, el drone té un comportament oscil·latori entorn a la posició de l'objectiu. Suposem que l'objectiu es troba a la dreta del drone i per tant li és necessari un desplaçament cap a la dreta per poder posicionar-se sobre aquest. Després de fer aquest desplaçament, degut al cicle d'acceleracions i frenades, ara l'objectiu es troba una mica a l'esquerra de la cartolina i per tant aquest cop fa un desplaçament cap a l'esquerra. Tot i que cada vegada els desplaçaments són més petits i s'hi situa millor sempre tenim aquest comportament oscil·latori del drone quan l'objectiu es troba parat. En la Figura 20 podem veure algunes de les iteracions del sistema de visió. i les pertinents acceleracions que s'han transmès al drone.

Els valors més significatius d'aquestes iteracions són el "roll_pwm" i el "pitch_pwm". Un valor superior de 1460, en el cas del "roll" es tradueix en un desplaçament a la

dreta; en el cas del "pitch" és un desplaçament endavant. Per altre banda, valors inferiors a 1460 generen un desplaçament a l'esquerra o endarrere respectivament.

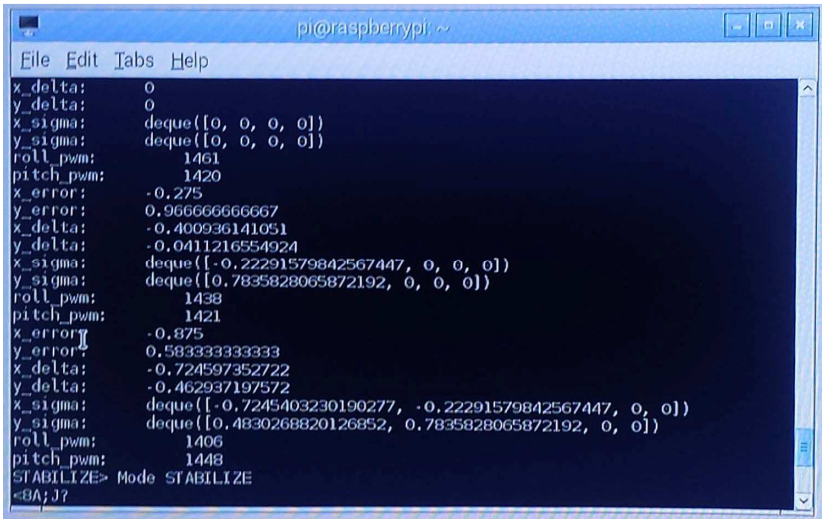

Figura 20 - Resultats de les iteracions sistema de visió

Degut a la poca resolució que s'ha pogut utilitzar de la webcam per la falta de capacitat de la Raspberry Pi, tots els vols s'han hagut de realitzar a baixa altura, d'entre 1m – 1,5m. Això causa dos problemes; per una banda, a aquesta altura es genera turbulència degut a la proximitat amb el terra i això ha fet que algunes vegades la nau s'hagi desestabilitzat, per altre banda, les lectures del baròmetre són lectures que oscil·len dintre d'uns marges i aquestes oscil·lacions no afecten igual a l'aeronau quan aquesta vola a 25m d'altura que quant vola a 1m d'altura. En el nostre cas aquesta imprecisió del baròmetre ha perjudicat el comportament de l'aeronau fent que algun cop hagi perdut el control i s'hagi estavellat. En la Figura 21 podem veure les oscil·lacions de les lectures del baròmetre en la línia blava, la línia verda és l'altura desitjada i la vermella és l'altura real.

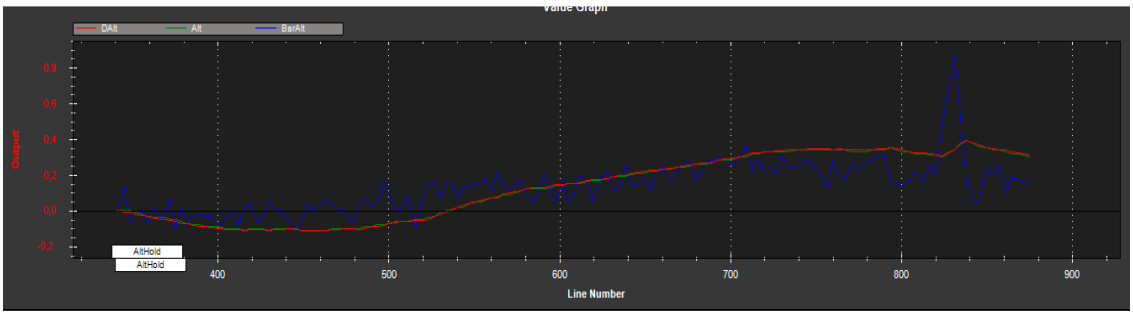

Figura 21 – Gràfica telemetria mode de vol Altitude Hold

Pel que fa l'autonomia del drone, la incorporació del sistema de visió no la fa variar. Estem parlant d'una autonomia entre els 15 i 20 min que dependrà del tipus de vol que realitzem i a l'altura a la que ens trobem. Pel que fa a l'autonomia del sistema de visió està prop dels 15 min. L'estabilitat del drone tampoc s'ha vist afectada degut a que en col·locar els diversos instruments del sistema de visió s'ha intentat equilibrar el pes de l'aeronau per a que no se'n vegi afectada.

13. Resum pressupost

Puja el Pressupost d'Execució per Contracte la quatitat de SET-CENTS CINQUANTA-QUATRE EUROS AMB SETANTA-SET CÈNTIMS (754,77€).

Puja el preu final de l'estudi i la redacció del projecte del drone de suport per a tasques de rescat la quantitat de SIS MIL SET-CENTS CINQUANTA-QUATRE EUROS AMB SETANTA-SET CÈNTIMS (6754,77€).

14. Conclusions

En aquest projecte es proposava com a objectiu central la creació i modificació d'un drone per tal de que pogués dur a terme el seguiment d'un objectiu d'un color determinat a través de tècniques de visió per computador. Aquest objectiu principal es pot donar per complert de forma satisfactòria tenint en compte que ens trobem encara en una fase beta del que haurà d'acabar essent aquest dispositiu.

Pel que fa a l'estudi de la comunicació entre el drone i el robot terrestre BIGBOT, en aquest projecte s'ha proposat un sistema de comunicació juntament amb el projectista encarregat de dur a terme les modificacions del robot BIGBOT. Degut a que l'altre projecte ha avançat de forma més lenta que aquest, no s'han pogut dur a terme les proves necessàries per tal de poder veure si realment el sistema plantejat en aquest projecte funciona. És per això que s'ha dut a terme l'explicació del procediment de la prova que es volia realitzar per a comprovar el funcionament del sistema de comunicació en el Capítol 11.

En general, el comportament i estabilitat del drone són molt òptims tant en l'interior d'edificis com en exteriors en condicions meteorològiques favorables, per tant podem dir que és una molt bona base per a continuar amb les millores que puguin venir en un futur. L'únic inconvenient del drone és l'autonomia de la bateria que és d'uns 20 min. amb tots els dispositius carregats al drone i duent a terme vols a baixa altura (entre 2 i 4m). Per tant s'hauria d'estudiar la possible implementació d'una carcassa per a tot el xassís en la qual hi haguessin plaques solars per a poder millorar l'autonomia de l'aeronau si fos possible.

La incorporació d'una carcassa, possiblement de plàstic, seria un punt molt positiu a tenir en compte degut a que podria contenir uns allotjaments o bé suports per a que els elements del sistema de visió quedessin ben incorporats en el drone. A més a més, d'aquesta manera es podrien prevenir possibles averies degudes a canvis climatològics que poguessin fer que l'electrònica del drone es mullés. Aquesta carcassa es podria dissenyar a mida i imprimir gràcies a una impressora 3D.

Pel que fa al sistema de visió, com s'ha pogut veure, és un sistema de visió molt "low cost" i no dona els millors resultats possibles. La Raspberry Pi mode B no té la suficient

capacitat com per a suportar bucles de programa més complexes que els utilitzats en aquest programa. La resolució que s'ha utilitzat durant la realització de tot el projecte ha estat la resolució mínima permesa ja que si no es saturava tot el sistema. També s'ha pogut apreciar un "delay" de gairebé mig segon entre la imatge que entra de la webcam i la imatge que es processa. Fins i tot, algunes vegades s'ha detectat que el mòdul de visió ha deixat de funcionar sense motiu aparent. Això pot ser degut a les poques capacitats que té la Raspberry Pi que, en un futur, hauria de ser substituïda per un altre ordinador de placa de dimensions reduïdes amb millors prestacions.

Un altre problema detectat d'aquest aspecte "low cost" del sistema de visió és la qualitat de la webcam. La mala qualitat del sensor d'aquesta fa que no s'adapti de forma ràpida al canvis sobtats de les condicions de llum. També s'ha vist que la imatge provinent de la webcam conté molt de soroll i està molt adulterada amb filtre gamma fent que tota la imatge tendeixi a tenir un color més càlid vermellós.

Per altre banda una de les sorpreses d'aquest sistema "low cost" és el sistema d'alimentació utilitzat. El carregador per a smartphones emprat ha estat capç d'aguantar gairebé 30 min alimentant la Raspberry Pi, tenint en compte que és un dels carregadors portàtils per a smartphones més bàsics del mercat amb una capacitat de 1500mAh. Gràcies a ella s'ha estalviat molt de pes i espai al no haver d'utilitzar una altre bateria de tipus Lipo per a alimentar aquest sistema i a més a més no ha estat necessari fer cap modificació en els connectors per a que s'adaptin al microUSB de l'alimentació de la Raspberry PI.

Com s'ha vist en l'apartat de 11. Resultats, alguns cops el drone no manté de forma prou precisa l'altura només utilitzant les lectures del baròmetre. Això altera de forma dràstica l'estabilitat d'aquest i el bon comportament del seguiment de l'objectiu. Una possible solució a aquest problema seria la incorporació d'un "Optical Flow Sensor".

Per acabar, un dels problemes que s'haurà de tenir en compte en un futur serà la incorporació de sistemes de detecció d'obstacles per tal de poder-los esquivar i que el drone no impacti contra ells.

FIRMA: JOAN BATALLER QUIÑONES

GIRONA, SETEMBRE 2015

15. Relació de documents

- **Document 1:** Memòria i Annexes
    1. Memòria
    2. Annexes

- **Document 2:** Estat d'amidaments

- **Document 3:** Pressupost

16. Bibliografia

[1] DR. JOSEP LLUIS DE LA ROSA. MATE: Robot-Dog interaction for efficient urban search and rescue operations. p. 4 – 6. 2015.

[2] L. ENRIQUE SUCAR I GIOVANI GÓMEZ. Visión Computacional. p. 1 – 3. 2015.

[3] L. ENRIQUE SUCAR I GIOVANI GÓMEZ. Visión Computacional. p. 59. 2015.

[4] Tuning dels controladors Arducopter (http://copter.ardupilot.com/wiki/tuning/?lang=en , 7 de Febrer de 2015)

[5] Pagina suport MAVProxy (http://tridge.github.io/MAVProxy/ , 13 d'Agost de 2015)

[6] Readme mòdul de visió (https://github.com/dwrtz/MAVProxy/blob/master/README.vision.md , 15 de Març de 2015)

[7] MatchTemplate Python (http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html , 15 de juny de 2015)

[8] What you need for an Arducopter (http://copter.ardupilot.com/wiki/introduction/what-you-need/ , 22 d'Octubre de 2014)

[9] About ROS (http://www.ros.org/about-ros/, 23 d'Agost de 2015)

17. Glossari

Abreviatures

- **GCS:** Ground Control Station
- **APM:** ArduPilot Mega
- **HSV:** Hue Saturation Value
- **RGB:** Red Green Blue
- **PWM:** Pulse Width Modulation
- **ROS:** Robot Operating System
- **UDP:** User Datagram Protocol

Definicions

- **Drone:** Aeronau no identificada de petit tamany i tipus ràdio control.
- **Quadcopter:** Drone de quatre motors.
- **BIGBOT:** Robot terrestre del projecte MATE.
- **Low cost:** Baix cost
- **Plug'n'play:** Capacitat d'un sistema informàtic per a configurar els dispositius només endollant-los.
- **Joystick:** Palanca de control**.**

II-ANNEXOS

A. Imatges del drone

Figura 22 – Imate superior del drone amb el sistema de visió

Figura 23 – Imatge frontal del drone amb la Raspberry Pi al davant i la webcam enfocada

B. Codi "module_vision.py"

```
'''
mavproxy_vision.py
LAST UPDATED 2013/06/21 2:00 PM
written by Colin Doolittle and David Wurtz

Portland State University 2013 Capstone Project

By importing this module in mavproxy.py, the user can engage/disengage
a
color-based computer vision system on channel 5, and use channel 6 to
select
which color to track.

Released under the GNU GPL licence
'''

from cv2 import *
from numpy import *
from numpy.linalg import inv, det
from time import time, sleep
from sklearn import preprocessing, svm
from collections import deque
import pickle
import threading
import RPi.GPIO as GPIO

# only for Raspberry Pi LED indicator
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11,GPIO.OUT)

mpstate = None

class vs(object):
    def __init__(self):

        # camera parameters
        self.cam = None
        self.most_recent = None
        self.CAPTURE_WIDTH = 160.0
        self.CAPTURE_HEIGHT = 120.0
        self.HALF_CAPTURE_WIDTH = self.CAPTURE_WIDTH/2
        self.HALF_CAPTURE_HEIGHT = self.CAPTURE_HEIGHT/2

        #HSV thresholds (hue: 0-179, saturation: 0-255, value: 0-255)
        self.color_3_upper_limit = (179, 255, 255)   # red
        self.color_3_lower_limit = (170, 100, 100)   # red
        self.color_5_upper_limit = (80, 255, 255)   # green
        self.color_5_lower_limit = (60,  100, 75)  # green
        self.color_7_upper_limit = (179, 255, 255)  # pink
        self.color_7_lower_limit = (120, 10, 100)  # pink

        # hue currently selected for tracking, init to pink
        self.HSV_UPPER_LIMIT = self.color_7_upper_limit
        self.HSV_LOWER_LIMIT = self.color_7_lower_limit

        #create cross-correlation templates
        self.col_temp = ones((self.CAPTURE_HEIGHT,1), uint8) #ROWS by
1 array
```

```
        self.row_temp = ones((1,self.CAPTURE_WIDTH), uint8) #1 by COLS
array

        # for keeping track of vision loop speed
        self.t = 0

        # open text file and store as python opject, then recover
classifier and feature scaler
        self.data_file = open('classifier_and_feature_scaler.txt',
'r')
        self.classifier_and_feature_scaler =
pickle.load(self.data_file)# features with target present
        self.data_file.close()
        self.classifier = self.classifier_and_feature_scaler[0]
        self.feature_scaler = self.classifier_and_feature_scaler[1]

        # classifier output
        self.target_in_frame = False

        # RC radio data, must reflect your unique settings!
        self.ch1_trim = 1460
        self.ch2_trim = 1460

        # hue parameters
        self.tracking_hue = 'Color 7'
        self.hue = 7

        # Initialize PID coefficients
        self.x_Ap = 0.7
        self.x_Ai = 0.05
        self.x_Ad = 0.6
        self.y_Ap = 0.8
        self.y_Ai = 0.05
        self.y_Ad = 0.6

        # initialize errors
        self.x_error = 0
        self.y_error = 0
        self.old_x_error = 0
        self.old_y_error = 0

        # initialize error accumulators
        self.x_sigma = deque([0,0,0,0])
        self.y_sigma = deque([0,0,0,0])

        # initialize deltas
        self.x_delta = 0
        self.y_delta = 0

        # initialize timer
        self.t = time()
        self.dt = 0.5

        # flag for pid first output behavior
        self.delta_flag = True
        self.sigma_flag = True


# A local copy of send_rc_override() in MAVProxy
def send_motor_override():
    if mpstate.sitl_output:
```

```
        buf = struct.pack('HHHHHHHH',*mpstate.status.override)
        mpstate.sitl_output.write(buf)
    else:

mpstate.master().mav.rc_channels_override_send(mpstate.status.target_s
ystem,

mpstate.status.target_component,

*mpstate.status.override)

# threaded in __init__
def vision_loop():
    while 1:
        if mpstate.status.flightmode == 'ALT_HOLD':
            track()

# threaded in __init__
def camera_loop():
    while 1:
        s, mpstate.vs.most_recent = mpstate.vs.cam.read()
        sleep(.05)

# returns n% peak width
def peak_width(curve, peak_index, percent):
    x = peak_index

    upper_index = 0
    lower_index = 0
    while (x < size(curve)):
        if (curve[x] > percent*curve[peak_index]):
            x += 1
        else:
            upper_index = x
            break

    x = peak_index
    while (x > 0):
        if (curve[x] > percent*curve[peak_index]):
            x -= 1
        else:
            lower_index = x
            break

    return (upper_index - lower_index)

def name():
    '''return module name'''
    return "vision"

def description():
    '''return module description'''
    return "This module tracks an object based on its HSV specs"

def init(_mpstate):
    '''initialize module'''
    global mpstate
    mpstate = _mpstate
    mpstate.vs = vs()

    # thread the vision loop
```

```
    vision_thread = threading.Thread(target=vision_loop)
    vision_thread.daemon = True
    vision_thread.start()

    '''initialize camera'''
    try:
        mpstate.vs.cam = VideoCapture(0)
    except:
        print("Could not initialize camera")
    mpstate.vs.cam.set(3, mpstate.vs.CAPTURE_WIDTH)
    mpstate.vs.cam.set(4, mpstate.vs.CAPTURE_HEIGHT)
    print("vision initialized")

    # thread the camera loop
    camera_thread = threading.Thread(target=camera_loop)
    camera_thread.daemon = True
    camera_thread.start()

def update_hue(hue_int):
    '''update tracking hue based on RC channel 6'''
    if hue_int != mpstate.vs.hue:
        mpstate.vs.hue = hue_int
        if hue_int == 3:
            mpstate.vs.HSV_UPPER_LIMIT =
mpstate.vs.color_3_upper_limit
            mpstate.vs.HSV_LOWER_LIMIT =
mpstate.vs.color_3_lower_limit
            mpstate.vs.tracking_hue = 'Color 3'
        elif hue_int == 5:
            mpstate.vs.HSV_UPPER_LIMIT =
mpstate.vs.color_5_upper_limit
            mpstate.vs.HSV_LOWER_LIMIT =
mpstate.vs.color_5_lower_limit
            mpstate.vs.tracking_hue = 'Color 5'
        elif hue_int == 7:
            mpstate.vs.HSV_UPPER_LIMIT =
mpstate.vs.color_7_upper_limit
            mpstate.vs.HSV_LOWER_LIMIT =
mpstate.vs.color_7_lower_limit
            mpstate.vs.tracking_hue = 'Color 7'
        else:
            pass

def track():
    '''isolate target hue, classify target detection, send rc
override'''

    # grab frame and convert to hsv
    img = cvtColor(mpstate.vs.most_recent, cv.CV_BGR2HSV)

    # apply hsv threshold
    upperb = array(mpstate.vs.HSV_UPPER_LIMIT, uint8)
    lowerb = array(mpstate.vs.HSV_LOWER_LIMIT, uint8)
    img = inRange(img, lowerb, upperb)

    # cross-correlate image with templates
    x_ccor = matchTemplate(img, mpstate.vs.col_temp, cv.CV_TM_CCORR)
#1 by COLS array
    y_ccor = matchTemplate(img, mpstate.vs.row_temp, cv.CV_TM_CCORR)
#ROWS by 1 array
```

```
    # get x and y cordinates of maximum
    x_minVal, x_maxVal, x_minLoc, x_maxLoc = minMaxLoc(x_ccor)
    y_minVal, y_maxVal, y_minLoc, y_maxLoc = minMaxLoc(y_ccor)
    x_coord = x_maxLoc[0]
    y_coord = y_maxLoc[1]

    # Classifier
    xmean = mean(x_ccor)
    xvariance = var(x_ccor)
    xpeakval = x_maxVal
    x90peakwidth = peak_width(x_ccor[0], x_maxLoc[0], .90)
    ymean = mean(y_ccor)
    yvariance = var(y_ccor)
    ypeakval = y_maxVal
    y90peakwidth = peak_width(y_ccor, y_maxLoc[1], .90)


    features =[ xmean,
                xvariance,
                xpeakval,
                x90peakwidth,
                ymean,
                yvariance,
                ypeakval,
                y90peakwidth
                ]

    # determine if target is in frame or not
    if
mpstate.vs.classifier.predict(mpstate.vs.feature_scaler.transform([fea
tures])) == 1:
        mpstate.vs.target_in_frame = True
    else:
        mpstate.vs.target_in_frame = False

    if mpstate.vs.target_in_frame == True and
mpstate.status.flightmode == 'ALT_HOLD':

        ''' PID Controller for Navigation '''
        # normalize x_error and y_error
        mpstate.vs.x_error = (x_coord -
mpstate.vs.HALF_CAPTURE_WIDTH)/mpstate.vs.HALF_CAPTURE_WIDTH
        mpstate.vs.y_error = (mpstate.vs.HALF_CAPTURE_HEIGHT -
y_coord)/mpstate.vs.HALF_CAPTURE_HEIGHT
        print 'x_error:     ', mpstate.vs.x_error
        print 'y_error:     ', mpstate.vs.y_error

        # compute deltas
        mpstate.vs.x_delta = (mpstate.vs.x_error -
mpstate.vs.old_x_error)/mpstate.vs.dt
        mpstate.vs.y_delta = (mpstate.vs.y_error -
mpstate.vs.old_y_error)/mpstate.vs.dt
        if mpstate.vs.delta_flag == True:    # initial deltas are
undefined because dt is undefined
            mpstate.vs.x_delta = 0
            mpstate.vs.y_delta = 0
            mpstate.vs.delta_flag = False    # subsequent deltas are
defined

        print 'x_delta:     ', mpstate.vs.x_delta
        print 'y_delta:     ', mpstate.vs.y_delta
```

```
        # update accumulators
        mpstate.vs.x_sigma.pop()

mpstate.vs.x_sigma.appendleft(mpstate.vs.x_error*mpstate.vs.dt)
        mpstate.vs.y_sigma.pop()

mpstate.vs.y_sigma.appendleft(mpstate.vs.y_error*mpstate.vs.dt)
        if mpstate.vs.sigma_flag == True:    # initial sigmas are
undefined because dt is undefined
            mpstate.vs.x_sigma = deque([0,0,0,0])
            mpstate.vs.y_sigma = deque([0,0,0,0])
            mpstate.vs.sigma_flag = False    # subsequent sigmas are
defined
        print 'x_sigma:     ', mpstate.vs.x_sigma
        print 'y_sigma:     ', mpstate.vs.y_sigma

        # compute pulse
        mpstate.vs.x_pulse = mpstate.vs.x_Ap * mpstate.vs.x_error +
mpstate.vs.x_Ai * sum(mpstate.vs.x_sigma) + mpstate.vs.x_Ad *
mpstate.vs.x_delta
        mpstate.vs.y_pulse = mpstate.vs.y_Ap * mpstate.vs.y_error +
mpstate.vs.y_Ai * sum(mpstate.vs.y_sigma) + mpstate.vs.y_Ad *
mpstate.vs.y_delta

        # motor overrides
        roll_pwm = mpstate.vs.ch1_trim + int(mpstate.vs.x_pulse*50)
        pitch_pwm = mpstate.vs.ch2_trim - int(mpstate.vs.y_pulse*50)
        print 'roll_pwm:        ', roll_pwm
        print 'pitch_pwm:       ', pitch_pwm

        # send a sequence of commands that accelerates and brakes the
quadcopter
        GPIO.output(11,1)
        mpstate.status.override = [roll_pwm,pitch_pwm,0,0,0,0,0,0]
        send_motor_override()  # start movement
        sleep(0.3)             # wait

        mpstate.status.override = [mpstate.vs.ch1_trim,
                                   mpstate.vs.ch2_trim,0,0,0,0,0,0]
        send_motor_override()  # stop brake
        sleep(0.3)             # wait
    else:
        # release to RC radio
        GPIO.output(11,0)
        mpstate.status.override = [0,0,0,0,0,0,0,0]
        send_motor_override()
        # reset pid
        mpstate.vs.x_sigma = deque([0,0,0,0])
        mpstate.vs.y_sigma = deque([0,0,0,0])
        mpstate.vs.old_x_error = 0
        mpstate.vs.old_y_error = 0
        mpstate.vs.dt = 0.5
        mpstate.vs.delta_flag = True
        mpstate.vs.sigma_flag = True

    mpstate.vs.old_x_error = mpstate.vs.x_error
    mpstate.vs.old_y_error = mpstate.vs.y_error
    mpstate.vs.dt = time() - mpstate.vs.t
    mpstate.vs.t = time()
```

```
def mavlink_packet(m):
    '''update color filter hue'''
    if m.get_type() == 'RC_CHANNELS_RAW':
        rc_temp = m.get_payload()
        the_hue = rc_temp[15]  # grabs the int on ch6
        update_hue(the_hue)
```

C. Codi "roscopter_node.py"

```
#!/usr/bin/env python
import os
import sys
import struct
import time
import socket

import roslib; roslib.load_manifest('roscopter')
import rospy
from std_msgs.msg import String, Header
from std_srvs.srv import *
from sensor_msgs.msg import NavSatFix
from sensor_msgs.msg import NavSatStatus

import roscopter.msg


mavlink_dir = os.path.realpath(os.path.join(
    os.path.dirname(os.path.realpath(__file__)),
    '..', 'mavlink'))
sys.path.insert(0, mavlink_dir)

pymavlink_dir = os.path.join(mavlink_dir, 'pymavlink')
sys.path.insert(0, pymavlink_dir)


from optparse import OptionParser
parser = OptionParser("roscopter.py [options]")

parser.add_option("--baudrate", dest="baudrate", type='int',
                  help="master port baud rate", default=57600)
parser.add_option("--device", dest="device", default="/dev/ttyUSB0",
help="serial device")
parser.add_option("--rate", dest="rate", default=10, type='int',
help="requested stream rate")
parser.add_option("--source-system", dest='SOURCE_SYSTEM', type='int',
                  default=255, help='MAVLink source system for this
GCS')
parser.add_option("--enable-control",dest="enable_control",
default=False, help="Enable listning to control messages")

(opts, args) = parser.parse_args()


import mavutil

# create a mavlink serial instance
master = mavutil.mavlink_connection(opts.device, baud=opts.baudrate)

if opts.device is None:
    print("You must specify a serial device")
    sys.exit(1)

def wait_heartbeat(m):
    '''wait for a heartbeat so we know the target system IDs'''
    print("Waiting for APM heartbeat")
    m.wait_heartbeat()
```

```
    print("Heartbeat from APM (system %u component %u)" %
(m.target_system, m.target_system))


def send_rc(data):
    master.mav.rc_channels_override_send(
        master.target_system,
        master.target_component,
        data.channel[0],
        data.channel[1],
        data.channel[2],
        data.channel[3],
        data.channel[4],
        data.channel[5],
        data.channel[6],
        data.channel[7])
    print "sending rc: %s" % data


def set_arm(req):
    master.arducopter_arm()
    return []

def set_disarm(req):
    master.arducopter_disarm()
    return []

pub_gps = rospy.Publisher('gps_drone', NavSatFix)
pub_rc = rospy.Publisher('rc', roscopter.msg.RC)
pub_state = rospy.Publisher('state', roscopter.msg.State)
pub_vfr_hud = rospy.Publisher('vfr_hud', roscopter.msg.VFR_HUD)
pub_attitude = rospy.Publisher('attitude', roscopter.msg.Attitude)
pub_raw_imu =  rospy.Publisher('raw_imu',
roscopter.msg.Mavlink_RAW_IMU)
if opts.enable_control:
    rospy.Subscriber("send_rc", roscopter.msg.RC , send_rc)

#define service callbacks
arm_service = rospy.Service('arm', Empty, set_arm)
disarm_service = rospy.Service('disarm', Empty, set_disarm)


#state
gps_msg = NavSatFix()



def mainloop():
    rospy.init_node('roscopter')
    while not rospy.is_shutdown():
        rospy.sleep(0.001)
        msg = master.recv_match(blocking=False)
        if not msg:
            continue
        #print msg.get_type()
        if msg.get_type() == "BAD_DATA":
            if mavutil.all_printable(msg.data):
                sys.stdout.write(msg.data)
                sys.stdout.flush()
        else:
            msg_type = msg.get_type()
```

```
            if msg_type == "RC_CHANNELS_RAW" :
                pub_rc.publish([msg.chan1_raw, msg.chan2_raw,
msg.chan3_raw, msg.chan4_raw, msg.chan5_raw, msg.chan6_raw,
msg.chan7_raw, msg.chan8_raw])
            if msg_type == "HEARTBEAT":
                pub_state.publish(msg.base_mode &
mavutil.mavlink.MAV_MODE_FLAG_SAFETY_ARMED,
                                  msg.base_mode &
mavutil.mavlink.MAV_MODE_FLAG_GUIDED_ENABLED,
                                  mavutil.mode_string_v10(msg))
            if msg_type == "VFR_HUD":
                pub_vfr_hud.publish(msg.airspeed, msg.groundspeed,
msg.heading, msg.throttle, msg.alt, msg.climb)

            if msg_type == "GPS_RAW_INT":
                fix = NavSatStatus.STATUS_NO_FIX
                if msg.fix_type >=3:
                    fix=NavSatStatus.STATUS_FIX
                pub_gps.publish(NavSatFix(latitude = msg.lat/1e07,
                                          longitude = msg.lon/1e07,
                                          altitude = msg.alt/1e03,
                                          status =
NavSatStatus(status=fix, service = NavSatStatus.SERVICE_GPS)
                                          ))
            #pub.publish(String("MSG: %s"%msg))
            if msg_type == "ATTITUDE" :
                pub_attitude.publish(msg.roll, msg.pitch, msg.yaw,
msg.rollspeed, msg.pitchspeed, msg.yawspeed)


            if msg_type == "LOCAL_POSITION_NED" :
                print "Local Pos: (%f %f %f) , (%f %f %f)" %(msg.x,
msg.y, msg.z, msg.vx, msg.vy, msg.vz)

            if msg_type == "RAW_IMU" :
                pub_raw_imu.publish (Header(), msg.time_usec,
                                     msg.xacc, msg.yacc, msg.zacc,
                                     msg.xgyro, msg.ygyro, msg.zgyro,
                                     msg.xmag, msg.ymag, msg.zmag)


wait_heartbeat(master)

print("Sending all stream request for rate %u" % opts.rate)
master.mav.request_data_stream_send(
    master.target_system,
    master.target_component,
    mavutil.mavlink.MAV_DATA_STREAM_ALL,
    opts.rate,
    1)

if __name__ == '__main__':
    try:
        mainloop()

    except rospy.ROSInterruptException: pass
```