Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi:Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Detecció i seguiment d'un objecte amb el dron DJI Matrice 100

Documents: 1. Memòria

Alumne: Jonah Fernández González

Tutor: Dra. Bianca Mariela Innocenti Badano

Departament: Enginyeria Elèctrica, Electrònica i Automàtica

Àrea: Enginyeria de Sistemes i Automàtica

Convocatòria (mes/any) setembre/2019

Índex

1. INTRODUCCIÓ

Amb aquest projecte es pretén desenvolupar una algorisme de detecció i seguiment d'objectes pel dron DJI Matrice 100, mitjançant l'ordinador d'abord Manifold connectat a la càmara a color Zenmuse X3. Un portàtil executarà un escriptori remot, on es podran visualitzar totes les variables del dron i les imatges de la càmera després de passar per l'algorisme de detecció i seguiment.

1.1. Antecedents

El dron Matrice 100 s'utilitzarà per a realitzar el seguiment d'objectes en moviment que, posteriorment, s'aplicarà en missions de rescat amb gossos. Uns anys enrere, es va voler implementar aquest seguiment, però no es disposava de l'ordinador d'abord Manifold i, com a conseqüència, no es va poder implementar amb la càmera a color Zenmuse X3 que porta el dron i es va haver d'optar a utilitzar únicament les càmeres en blanc i negre del sistema Guidance. Al llarg de la meva estada en entorn laboral a l'UdG, havia d'implementar aquest seguiment amb la càmera a color, però degut a uns problemes que venien d'estades anteriors, es va haver de prioritzar la reparació del dron. Això va impedir que es pogués assolir l'objectiu inicial i no es va implementar el seguiment d'objectes.

1.2. Objecte

Amb aquest TFG es pretén reprendre la implementació del seguiment d'objectes amb la càmera a color Zenmuse X3 i l'ordinador d'abord Manifold. Es desenvoluparà un algorisme de tractament de la imatge capaç de distingir un objecte en moviment i realitzar el seu seguiment de manera automàtica. D'aquesta manera, el dron s'anirà posicionant a sobre de l'objecte en qüestió i l'operari podrà visualitzar en tot moment el seguiment.

1.3. Especificacions i abast

El seguiment d'objectes amb el dron Matrice 100 es durà a terme mitjançant tècniques de visió artificial, en concret, algorismes de detecció i seguiment d'objectes.

2. CONFIGURACIÓ DEL DRON

Per a la realització del projecte es disposarà diversos elements, tal i com mostra la Figura 1, que es comuniquen entre si per a transmetre informació, paràmetres i ordres.

Figura 1. Elements utilitzats en l'elaboració del projecte: M100, portàtil i RC

El dron DJI model Matrice 100 incorpora el següents components: un ordinador d'abord DJI Manifold, una càmera a color Zenmuse X3 i un sistema Guidance. L'ordinador Manifold disposa del sistema operatiu Linux Ubuntu 14.04 LTS i dels softwares ROS Indigo, OpenCV i Python 2.7. A part del software, es disposa d'un entorn de treball *catkin* amb l'Onboard-SDK-ROS-3.2, que inclou el programari bàsic de DJI, el driver ROS per la càmera a color i els codis realitzats en aquest projecte. El dron es comunicarà amb el controlador remot model C1. Aquest es connectarà a través d'un cable USB amb un mòbil amb l'aplicació DJI GO instal·lada. L'ordinador d'abord Manifold es comunicarà a través del protocol SSH amb un portàtil amb sistema operatiu Linux Ubuntu 14.04 LTS. A través del portàtil es podran visualitzar paràmetres del dron com la velocitat o l'estat de les bateries. Per altra banda, es podrà accedir a la imatge de la càmera Zenmuse i activar l'algorisme de detecció i el codi de seguiment. És molt important aclarir que en cap cas s'utilitzarà l'ordinador portàtil per pilotar el dron. Servirà per passar trajectòries i visualitzar els valors dels diferents sensors del dron i la càmera. El programa que hi ha en el Manifold i que es comunica amb l'interfície del portàtil és l'encarregat de pilotar el dron, en mode automàtic. Tot i això, el RC ha d'estar present i en funcionament en tot moment, per a que, en cas d'emergència o perquè el pilot ho vulgui, pugui tenir el control del dron.

2.1. DJI Matrice 100

El DJI Matrice 100 és un dron multi-rotor, concretament un quadricòpter, dissenyat especialment per a desenvolupadors, ja que disposa d'una àmplia gamma d'accesoris i és totalment programable. La seva capacitat d'autonomia és de 40 minuts amb dues bateries TB48D i 23 minuts si se li monta la càmara a color Zenmuse X3 i amb una sola bateria. El seu pes és de 2,431 kg i pot suportar un pes de 3,6 kg en l'enlairament. Les bateries limiten el seu rang de temperatura entre -10 i 40ºC.

Al ser un quadricòpter, es pot enlairar verticalment des de qualsevol superfície plana i pot aterrar en qualsevol punt al llarg del vol, gràcies als quatre motors elèctrics DJI 3510/350 KV, tal i com es mostra en la Figura 2. Aquests motors sense escombretes disposen d'imants que eviten l'aparició de fluxos d'aire en l'interior d'aquests.

Figura 2. Motor elèctric DJI 3510

Cada motor es col·loca en un dels 4 braços del xassís, els quals estan protegits en cas de vibracions. Els braços és poden reajustar 3 graus per tal d'obtenir un parell més gran, tal i com es pot veure en la Figura 3.

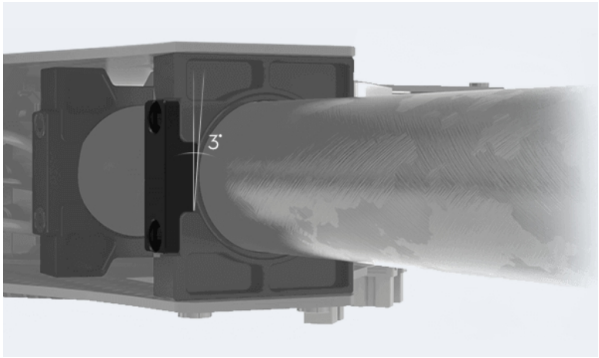


Figura 3. Braços reajustables

El dron DJI Matrice 100 utilitza el model d'hèlix de fibra de carboni 1345s, tal i com es mostra en la Figura 4. Disposen d'un sistema d'anclatge que evita que es despreguin en mig del vol.

Figura 4. Hèlix DJI 1345s

Per el correcte funcionament de l'aparell, s'ha de configurar cada parell d'hèlix per tal de que girin en el mateix sentit. La configuració de la rotació dels motors del DJI Matrice 100 es pot observar en la Figura 5.

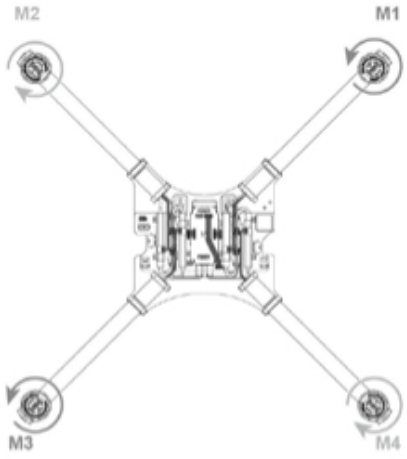


Figura 5. Sentit de rotació de les hèlix dels motors

Els encarregats de controlar el moviment del vehicle són els ESC (Electronic Speed Control), model DJI E SERIES 620D, com es veu en la Figura 6. Es requereix un per a cada motor. Amb els ESC es pot variar la velocitat relativa de cada rotor per canviar la direcció de vol i modificar la trajectòria del dron.

Figura 6. DJI ESC

Els components i accesoris del DJI Matrice 100 es munten en un xassís lleuger de fibra de carboni, com el de la Figura 7. Disposa de ranures per ampliar el hardware i de ports per realitzar les connexions dels components.

Figura 7. Xassís amb els motors i el GPS

El DJI Guidance és un sistema de navegació visual. Disposa d'un potent processador central i de cinc mòduls de sensors. Cada mòdul està format per dues càmares monocolor i sensors d'ultrasons, com es pot observar en la Figura 8. Permet evitar obstacles en 360º, amb la limitació que només pot detectar un obstacle alhora. El seu rang d'imatges és molt ampli, anant des de 15 fins a 20000 lux.

Figura 8. DJI Guidance

El mòdul GPS PRO PLUS, Figura 9, és l'utilitzat en el dron M100. Es pot orientar amb diferents inclinacions per facilitar el seguiment de la posició del dron en temps real. Es situa a la part superior del xassís per a evitar interferències i aconseguir un millor senyal.

Figura 9. Mòdul GPS

La càmera Zenmuse X3, Figura 10, s'utiliza per a capturar la imatge i posteriorment a través d'un algorisme, realitzar el seguiment d'objectes en temps real. És tracta d'una càmara a color, a diferència de les càmares del Guidance que són en blanc i negre. Va muntada en un gimbal que uneix la càmara amb el xassís del dron i li proporciona estabilitat per a l'obtenció òptima d'imatges al llarg del vol. Té una resolució de 12 Mpíxels i permet gravar vídeos en 4k o 1080p. Disposa d'una ranura per a targetes SD on es poden emmagatzemar tant vídeos com captures.

Figura 10. Càmera Zenmuse X3

Les bateries compatibles amb el Matrice 100 són les TB47D i TB48D. Es tracta de bateries intel·ligents de tipus Li-Po. El present projecte utilitza el model TB48D, Figura 11. En concret, és una Li-Po 6S i consta de 6 cel·les de voltatge connectades entre si en sèrie. La seva capacitat és de 5700 mAh i té un pes de 676 g. La seva temperatura de funcionament ca des de -10 fins a 40ºC i condiciona el rang de funcionament del dron M100.

Proporciona una autonomia de vol màxima de 28 minuts, amb una bateria i sense cap altre element connectat, i de 23 minuts amb una bateria i la càmera Zenmuse X3. Al tractar-se de bateries intel·ligents, se'ls ha d'anar actualitzant el software periòdicament.

Figura 11. Bateria intel·ligent TB48D

El control remot model C1, Figura 12, permet maniobrar el dron a distància mitjançant la radiofreqüència. Permet la connexió d'un mòbil o tablet per USB per poder visualitzar la càmara i altres dades, com el recorregut i les interferències, mitjançant l'aplicació DJI GO. Les freqüències més utilizades són 2,4GHz i 5,85GHz. La màxima distància de transmissió és de 3,5 km, sense interferències.

Figura 12. Control remot C1

2.2. Ordinador d'abord DJI Manifold

El DJI Manifold, tal i com s'observa en la Figura 13, és un ordinador d'abord amb sistema embedded Linux. És un equip integrat d'alt rendiment dissenyat per l'Onboard SDK de DJI.

Permet als desenvolupadors programar els drons per a la realització de tasques complexes així com el processament d'imatges en temps real. Es basa en un processador NVIDIA Tegra K1, el qual conté una CPU i una GPU.

Figura 13. Ordinador Manifold

Disposa de tres ports USB, un port HDMI i un port d'Ethernet. D'aquesta manera, permet la connexió amb un teclat, un ratolí i un monitor funcionant com un PC portàtil.

2.2.1. Linux Ubuntu 14.04 LTS

La distribució de Linux Ubuntu disposa de dues versions, LTS i no LTS. El Manifold disposa de la versió LTS, ja que permet que aquest s'actualitzi durant més temps. La versió 14.04 facilita la compatibilitat amb la resta de softwares del dron.

2.2.2. ROS

El ROS (Robot Operating System) és un entorn de treball que permet el desenvolupament de software per robots. Està basat en una arquitectura de grafs, on els nodes reben, envien i multiplexen la informació i les dades provinents de sensors i actuadors entre d'altres. Està pensat per utilitzar-se en un sistema UNIX com Linux.

A més de la part de sistema operatiu *ros*, també disposa de *ros-pkg*, un conjunt de paquets organitzats en piles o *stacks* que aporten funcionalitats com la percepció, simulació o localització. ROS és un software lliure.

L'arquitectura de grafs de computació consisteix en una xarxa peer-to-peer que consta de diversos elements: nodes, master, services, topics, messages i bags.

Els nodes són programes executables amb una funció concreta (planificació de rutes, mapeig d'entorn, etc.). Els nodes es compilen i són executats individualment pel ROS Master o node principal. Els nodes es comuniquen entre si per mitjà de missatges i topics.

Els nodes s'escriuen amb la llibreria client de ROS (*roscpp* o *rospy*), la qual és una colecció de codi que permet crear els nodes, publicadors i subscriptors dels missatges i topics en els nodes, a més dels serveis i el paràmentres. Es compatible amb Python i C++.

Els topics són busos amb un nom determinat mitjançant els quals els nodes s'intercanvien missatges. Tenen una semàntica de publicació/subscripció anònima. Un node publica un missatge en una determinada direcció i varis nodes es subscriuen a aquest missatge per rebre la informació.

Els nodes no són conscients de amb qui es comuniquen, sinó que es subscriuen al topic que té dades relevants per ells. Els topics estan pensats per dur a terme comunicacions unidireccionals i de transmissió seqüencial.

El model de publicació/subscripció és flexible però per ser un tranport unidireccional no és apropiat per les interaccions de sol·licitud/resposta, necessàries en un sistema distribuit. Aquesta interacció es pot fer a través d'un service, un conjunt de missatges: un per la sol·licitud i un altre per la resposta. Un node ROS subministra el servei amb un nom en *string* i un node client el crida enviant-li una sol·licitud i esperant una resposta.

Un client pot realitzant una connexió continua a un service, tenint millor rendiment però menys rigidesa en els canvis del proveïdor.

Per definir un servei, s'utiliza un arxiu SRV, que es compila en el codi font per una biblioteca client com *roscpp* o *rospy* de ROS.

Els nodes es comuniquen entre ells a través de la publicació de missatges en un topic. Un missatge pot ser enter, *float*, booleà, etc. També poden intercanviar-se missatges de

sol·licitud i resposta, definits en arxius srv.

Un bag és un arxiu ROS que serveix per emmagatzemar els missatges. La majoria de bags es crean a través d'una eina anomenada rosbag, la qual es subscriu als topics de ROS. Les dades del missatge es van emmagatzemant en un arxiu que es pot reproduir en ROS pels topics subscrits o es poden reasignar a diferents topics.

El ROS Master té la funció de proporcionar els serveis de registre i denominació als diferents nodes, a més de seuir els editors i subscriptors de serveis i topics. El Master permet que els nodes es localitzin entre ells i es puguin comunicar d'igual a igual. Per executar el Master, s'utiliza la comanda *roscore*.

Per entendre el funcionament del Master, suposem que tenim una càmara i volem visualitzar la imatges amb ROS. Els nodes corresponents serien el *Camera node* i el I*mage_viewer node*.

Primer, la *Camera* notifica al Master que vol publicar les imatges en el topic *images*, tal i com mostra la Figura 14.

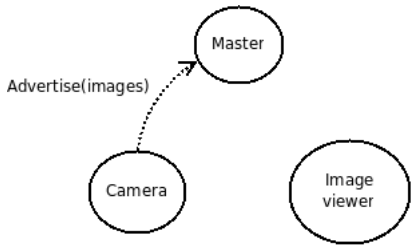


Figura 14. Camera notifica al MASTER

A continuació, la *Camera* comença a publicar les seves imatges en el topic *images*, però no s'envien dades ja que cap node està subscrit.

Tot seguit, *Image_viewer* sol·licita al Master subscriure's al topic *images* per buscar les imatges, com en la Figura 15.

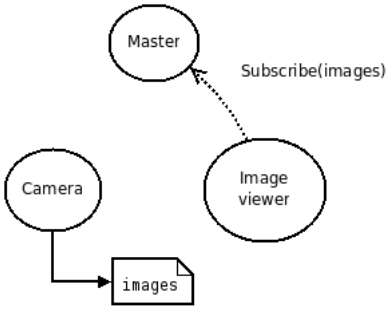


Figura 15. Image_viewer notifica al Master

Finalment, el topic *images* té un editor, *Camera*, i un subscriptor, *Image_viewer*. El Master notifica als dos nodes les seves existències perquè es puguin localitzar i es transfereixin les imatges entre si, com es mostra en la Figura 16.

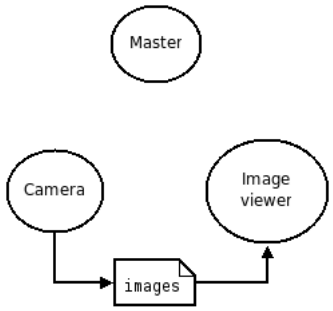


Figura 16. Camera i Image_viewer es transmeten les imatges

Una altra funció del Master és proporcionar el *Parameter Server*, una llibreria de variables accesible a través de les API de la xarxa. Els nodes emmagatzemen i recuperen els paràmetres en el servidors mentre s'executen.

2.2.3. OpenCV

OpenCV és una biblioteca lliure de visió artificial, que inclou funcions de programació. Permet realitzar projectes on és necessari fer un seguiment en temps real.

ROS no es compatible amb la versió normal d'OpenCV, sinó que s'utiliza un stack anomenat vision opencv, que permet la interacció entre ROS i la llibreria d'OpenCV. Disposa de dos paquets, cv_bridge i image_geometry.

El paquet cv_bridge s'encarrega de comunicar OpenCV amb ROS, és a dir que fa de pont entre els seus missatges. El seu funcionament és el següent. cv_bridge defineix una imatge d'OpenCV. Aquesta imatge conté la imatge codificada amb una capçalera que permet la lectura per part de ROS. Aquest procés és bidireccional, ja que els dos formats contenen la mateixa informació, tal i com es mostra en la Figura 17.

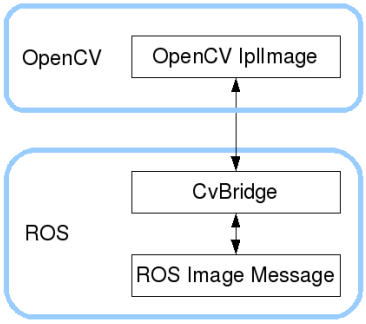


Figura 17. Comunicació entre OpenCV i ROS

El paquet image_geometry conté un seguit de llibreries per interpretar les imatges amb el paràmetres de sensor_msgs/CameraInfo.

2.3. ONBOARD SDK

L'Onboard SDK (OSDK) de DJI és una biblioteca de programari de codi obert que permet la comunicació entre drons i Pcs. El SDK permet al desenvolupador connectar el seu propi dispositiu informàtic (PC, Manifold, etc.) a bord del dron i controlar-lo durant el vol, gràcies a les funcions de telemetria i control que ofereix. Les dades són enviades en temps real.

Amb l'OSDK es pot controlar la trajectòria del dron i capturar imatges amb la càmera. Aquestes captures es poden enviar al mòbil connectat al controlador remot amb l'app DJI GO.

Hi ha dos opcions per utilitzar l'Onboard SDK: amb un PC o amb l'ordinador d'abord Manifold. El present projecte es basa en l'opció del Manifold, ja que aquest està pensat per agilitzar el procés de transmissió de dades del dron. El Manifold s'ha de connectar al Matrice 100 a través del port UART TTL del dron.

L'OSDK consisteix en un conjunt de paquets compatibles amb el Matrice 100 i el controlador remot A3. Les funcions de l'OSDK s'agrupen en una llibreria anomenada dji_drone.h. Aquest

header es pot incloure en qualsevol dels programes que dissenyi el desenvolupador.

Per gestionar l'OSDK és necessari el programa DJI Assistant, compatible només amb un PC amb Windows, a diferència de la resta de programari del projecte. Amb el DJI Assistant i connectant el M100 amb el PC a través del seu port microUSB, es pot actualitzar el firmware del dron, realitzar simulacions de vol i gestionar diversos paràmetres, com la localització GPS.

Per poder utilitzar aquestes funcions, primer s'ha d'activar l'Onboard SDK, connectant el PC, dron i comandament com indica la Figura 18 i seguint els posteriors pasos.

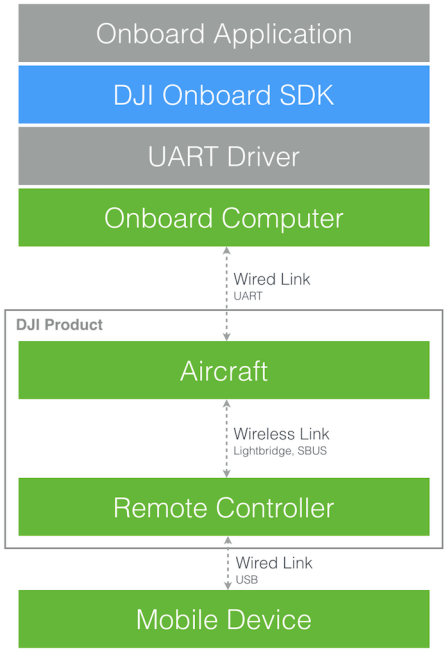


Figura 18. Connexions del sistema

Abans de començar, l'usuari ha de crear una compta DJI com a desenvolupador. S'obtindrà una clau d'accés i una contrasenya per poder desenvolupar projectes amb el M100.

En el programa DJI Assistant 2 s'ha d'habilitar Enable API control per que el Matrice 100 es pugui comunicar amb el Manifold i establir el baud rate a 230400.

A continuació, s'han de copiar tots els fitxers del repositori DJI-Onboard-SDK-ROS, prèviament descarregat i posar-los a la carpeta *src* del nostra entorn de treball *catkin_ws*. El

següent pas consisteix en modificar el fitxer *sdk_manifold.launch* situat a la carpeta *dji_sdk/launch/* modificant els seus paràmetres amb les següents dades de la Figura 19.

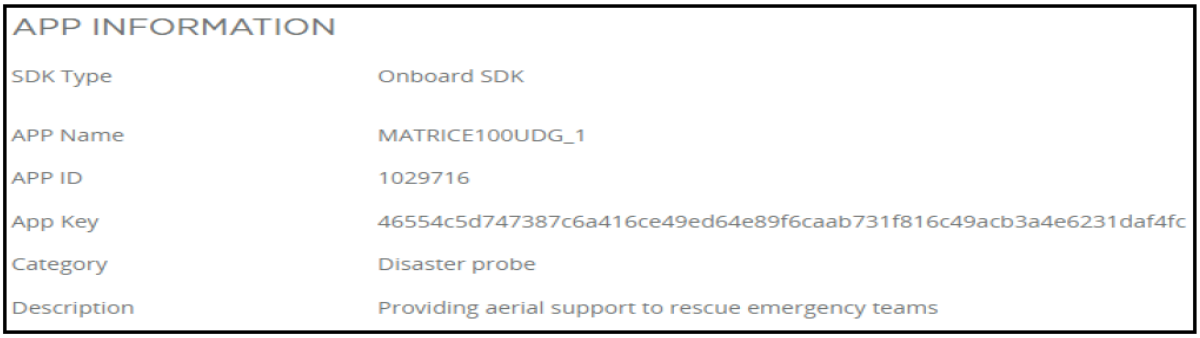


Figura 19. Paràmetres del fitxer *sdk_manifold.launch*

Com que s'utiliza el Manifold, el serial_name del fitxer .launch és per defecte **/dev/ttyTHS1** i el baud_rate **230400**. El baud rate ha de coincidir sempre amb el valor establert al DJI Assistant 2.

Finalment, es compila el fitxer .launch per construir les dependències necessàries.

D'entre tots els paquets que disposa l'OSDK ROS, cal destacar el ROS core. Aquest paquet publica en diversos ROS topics totes les dades que rep del Matrice 100 i envia totes les comandes de control a la controladora de vol del dron.

2.4. GUIDANCE SDK

El dispositiu Guidance disposa d'un programari de desenvolupament similar a l'Onboard SDK, anomenat Guidance SDK, que permet ontenir dades dels diferents sensors (càmeres monocolors i sensors d'ultrasons).

Els paràmetres del kit de sensors es poden gestionar amb el programa DJI-Guidance Assistant, similar en funcionament al DJI Assistant 2. S'ha de connectar l'entrada micro-USB del Guidance Core a l'entrada USB de l'ordinador amb Windows i obrir el programa.

DJI-Guidance Assistant disposa de diferents pestanyes: *View*, *Standard*, *DIY*, *Calibration* i *Upgrade*.

View ens indica quins dels cinc sensors d'orientació VBUS1-5 estan en funcionament i ens permet realitzar una *Preview* de la imatge. On posa *Work Type*, s'ha de seleccionar *Standard*, ja que s'utilitza el Matrice 100. Si en la columna de *Working* posa *Yes*, s'habilita el botó *Preview*, que ens permet comprovar la imatge que capta cada càmara, com en la Figura 20.

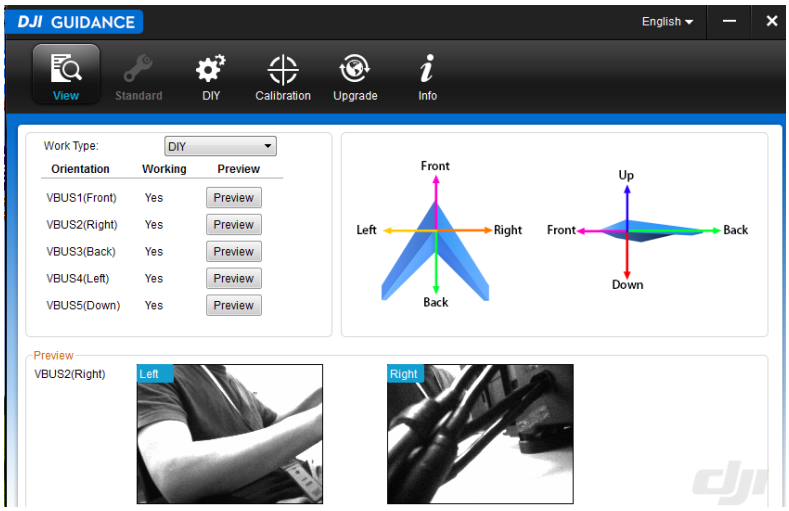


Figura 20. *Preview* dels sensors del Guidance

La segona pestanya és la de *Standard Settings*. On posa *Aircraft Type*, s'ha de seleccionar DJI MATRICE 100. Si s'utilitza un muntatge amb les posicions dels sensors estàndard, no s'ha de modificar res. També es pot modificar la distància de seguretat del *Sensing Mode* desplaçant el selector d'esquerra a dreta. El muntatge del dron M100 no és l'estàndard, ja que al disposar de la càmera Zenmuse X3, s'ha hagut de desplaçar el sensor frontal de la part inferior a la part superior del xassís, tal i com mostra la Figura 21.

Figura 21. Muntatge dels sensors del sistema Guidance

Al tenir un muntatge dels sensors diferent a l'estàndar, es realitzarà una configuració DIY de la forma següent, tal i com indica la Figura 22.

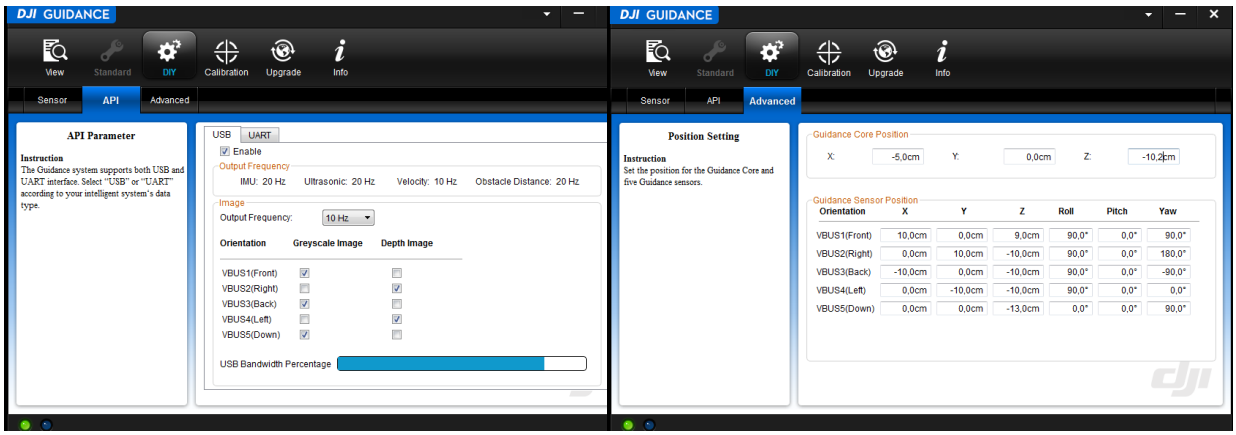


Figura 22. Configuració DIY del Matrice 100

La tercera pestanya és la de *DIY Settings*. Aquesta permet configurar els paràmetres DIY en cas que s'utilitzi el Guidance SDK. Disposa de tres pestanyes: Guidance Sensor Parameter, API Parameter, i Advanced Parameter. El Guidance Sensor Parameter, al igual que la pestanya *View*, permet fer un *Preview* de cada sensor per comprovar el seu funcionament. En l'API Parameter s'ha d'escollir un dels dos ports del Guidance: USB o UART. Per establir la freqüència de sortida de les imatges i el contingut d'imatges s'ha de seleccionar el port USB. Per últim, els Advanced Parameters s'utilitzen per indicar les posicions del Guidance Core i dels Guidance Sensors.

La quarta pestanya és *Calibration*, Figura 23. Aquest pas és necessari abans de volar el dron.

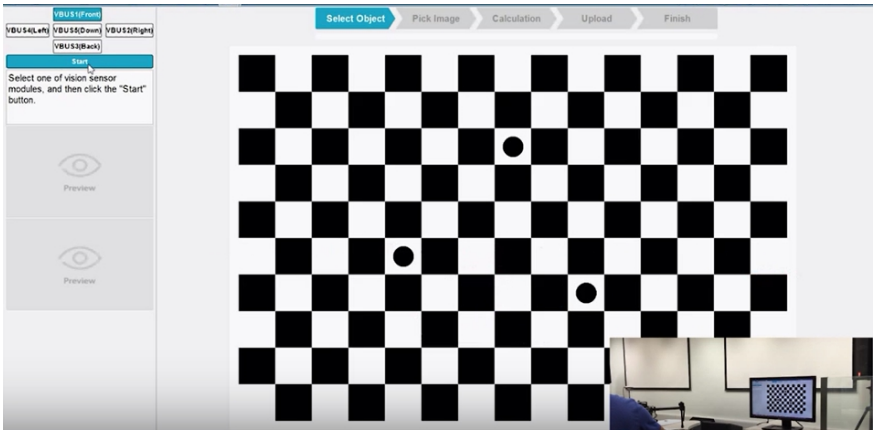


Figura 23. Calibració dels sensors

La calibració s'ha de realitzar per cada un dels cinc sensors.

El Guidance-SDK disposa d'un paquet de ROS que gestiona i publica les dades de la velocitat, distancia de l'obstacle, les senyals dels ultrasons i la imatge de les càmeres monocolor. Per activar la comunicació amb els paquets de ROS, s'utiliza la següent comanda.

```
sudo sh -c 'echo "SUBSYSTEM == \"usb\", ATTR{idVendor} == \"fff0\" , ATTR{idProduct} == \"d009\", MODE = \"0666\"" > /etc/udev/rules.d/51-guidance.rules'
```

3. DISSENY DE L'ALGORISME DE SEGUIMENT

La metodologia i el procés de desenvolupament del projecte es centra en el disseny d'un algorisme de tractament de la imatge pel dron DJI Matrice 100 capaç de realitzar el seguiment d'un objecte en temps real mitjançant la càmara a color Zenmuse X3 i l'ordinador d'abord Manifold. El programa s'ha realitzat amb l'entorn catkin de ROS i les llibreries d'OpenCV.

Existeixen diversos tipus d'algorismes de detecció, entre els quals destaquen els detectors preentrenats i els detectors de taques o *blobs*.

Un model o detector preentrenat és un model que ha estat entrenat amb una gran base de dades per detectar una imatge similar a la que volem detectar. Hi ha diversos tipus de detectors preentrenats, com el R-CNN o el YOLO. Per dur a terme la detecció, primer divideixen la imatge en una quadrícula SxS. Per cada objecte present en la imatge, una cel·la de la quadrícula, on es situa el centre de l'objecte, és responsable de la predicció. Per cada cel·la es prediu N possibles *bounding boxes* o quadres delimitadors i es calcula el nivell de probabilitat de cada una d'elles, és a dir, es calculen SxSxN caixes diferents, la majoria amb un nivell de certesa molt baix. A continuació, s'eliminen les caixes que es trobin per sota d'un límit i a les caixes restants se'ls aplica un procediment que consisteix en eliminar objectes detectats per duplicat i deixar únicament el més exacte d'entre ells. Finalment, la imatge de la caixa es compara amb la base de dades per classificar l'objecte detectat. El procés de detecció i classificació es pot observar en la Figura 24.

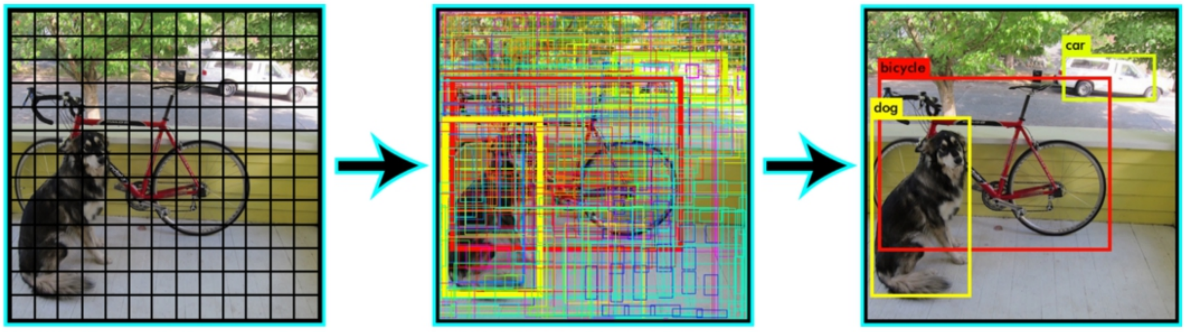


Figura 24. Detecció i classificació d'objectes amb YOLO

Un detector de *blobs*, en canvi, detecta una taca d'una mida i color determinat. Primer, s'aplica un filtre Gaussià a la imatge per suavitzar-la i difuminar-la en un *blur* i centra-se en els elements o píxels amb més pes sobre la imatge, tal i com es pot observar en la Figura 25.

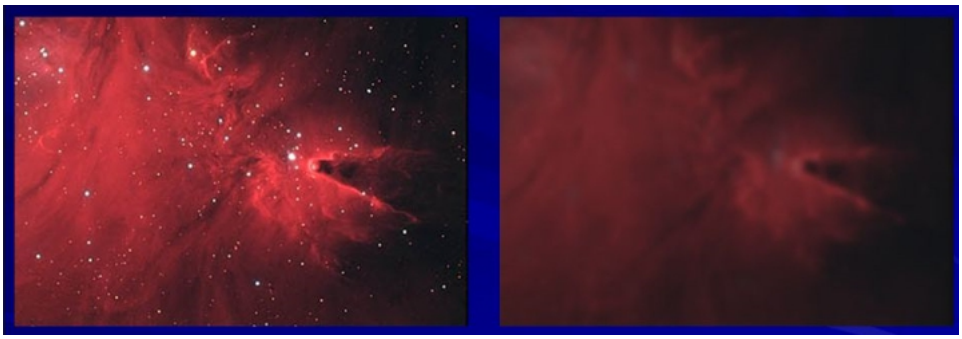


Figura 25. Imatge original (esquerra) i filtrada (dreta)

A continuació converteix la imatge a l'espai de color HSV, mostrat en la Figura 26. En l'espai HSV, el conjunt de colors es representa per un con, on la regió circular representa el matís del color i la regió triangular s'utilitza per representar la saturació i el valor del color. Per tant, l'eix horitzontal representa la saturació i el vertical, el valor del color. Així, per escollir un color, primer es selecciona el matís en la regió circular i, a continuació, es tria la saturació i el valor del color en la regió triangular.

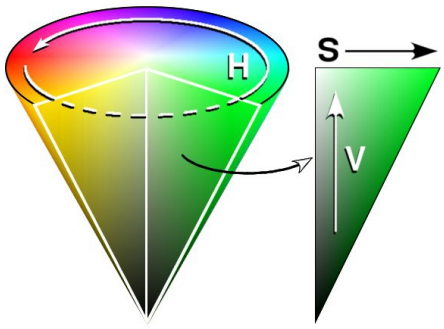


Figura 26. Espai de color HSV

El següent pas consisteix en crear una màscara de color amb els rang superiors i inferiors ajustats per detectar un color determinat. Tot seguit, el detector filtra els elements més petits i busca contorns definits. Al trobar una taca, calcula el seu centre (x, y). Finalment, filtra el contorn més gran d'entre tots els que ha detectat l'algorisme i dibuixa el cercle i el centroide de l'objectiu.

Per escollir quin detector era més adequat per el present projecte, es van avaluar els seus avantatges i inconvenients.

Una de les opcions que es van considerar era aplicar el sistema de detecció en temps real YOLO Darknet. Aquest programa està dissenyat en C++ i utiliza un model preentrenat que pot distingir, per exemple, un cotxe d'un gos ja que ha estat prèviament entrenat amb una gran base de dades enfocada a la classificació d'imatges i, per tant, té l'avantatge no només de detectar sinó de classificar la imatge captada. No obstant això, el YOLO només funcionava amb un portàtil amb Ubuntu 14.04 i no amb el Manifold, tot i tenir el mateix sistema operatiu. Això es devia a que el Manifold no té un processador de 32 o 64 bits convencional, sinó que és un model de 32 bits anomenat Armv7l i, com a conseqüència, no es compatible amb YOLO. Es van buscar altres detectors basats en models preentrenats però tenien l'inconvenient que ocupaven massa espai en la memòria del Manifold, la qual es molt limitada i, per tant, es van descartar completament.

En canvi, un algorisme de detecció de taques té el gran avantatge de que, al no dependre d'una gran base de dades, és més ràpid i només ocupa la memòria que necessiti el codi. Per aquest motiu i perquè s'adequava als requeriments del projecte es va optar per dissenyar un algorisme de detecció de *blobs*.

Un cop escollit el tipus de detector, es va procedir a confeccionar l'algorisme en llenguatge Python. L'avantatge de Python és que té una sintaxis que fa el codi sigui fàcilment llegible. A més, disposa de moltes llibreries i funcions dissenyades expressament per la visió artificial. L'API de OpenCV per Python és OpenCV-Python, que combina les característiques del API C++ de Open CV amb la programació amb Python, dissenyada per resoldre problemes de visió per computador. OpenCV-Python disposa d'una llibreria optimitzada per càlculs numèrics similar a Matlab anomenada Numpy. Amb aquesta llibreria, tots els arrays pasen a ser Numpy arrays, el que permet la integració amb llibreries que utilitzen Numpy, com SciPy o Matplotlib. El software ROS Indigo instal·lat prèviament en el Manifold disposa de Python 2.7 i, per tant, no és necessari descarregar-se'l paral·lelament.

L'algorisme de detecció de *blobs* s'incorporarà en el codi de detecció d'objectes, *object_tracker.py.*

4. RESOLUCIÓ DEL PROJECTE

El desenvolupament del present projecte es centra en el disseny d'un programa de detecció i seguiment d'objectes amb el dron DJI Matrice 100, mitjançant la càmera a color Zenmuse X3 i l'ordinador d'abord Manifold. Per a la realització d'aquest projecte es va instal·lar prèviament el software ROS Indigo, OpenCV i Python 2.7.

El funcionament del disseny es basa en l'execució del paquet OnboardSDK, encarregat de comunicar-se amb la controladora de vol del dron i del driver ROS de la càmera Zenmuse X3. Un cop executats, s'inicialitza el disseny desenvolupat en aquest projecte, el qual consta de dos programes, escrits en Python i C++, que es comuniquen entre si a través de tòpics.

El programa en Python és el detector i s'encarrega de tractar la imatge captada amb la càmera. El segon programa, en llenguatge C++, té la funció de rebre la posició de l'objecte detectat i realitzar el seu seguiment. L'estructura del disseny es pot observar en la Figura 27.

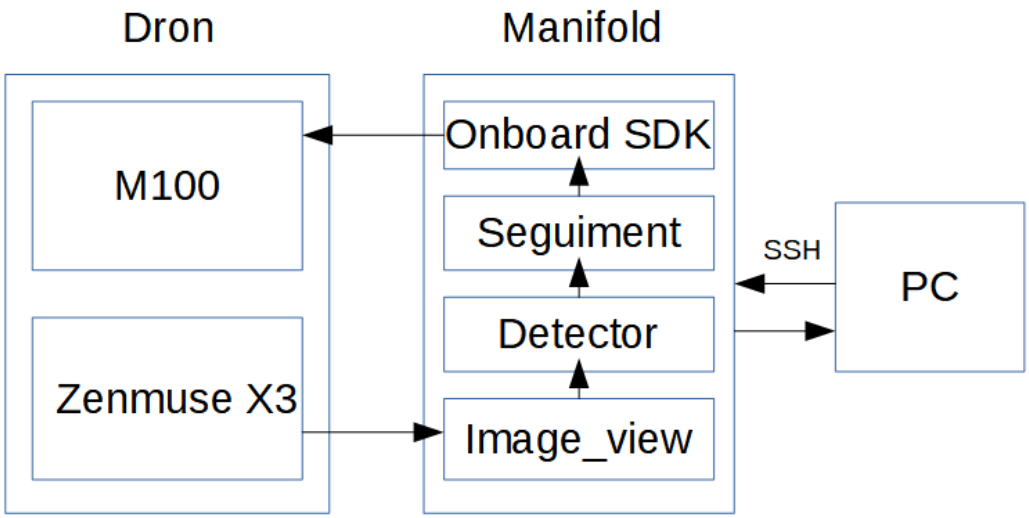


Figura 27. Estructura del disseny

El primer pas va ser crear un espai de treball *catkin* de ROS en el Manifold. En aquest *workspace* s'inclouen els SDK, codis i scripts necessaris pel disseny de l'aplicació. Per a crear el catkin_ws, s'han de seguir els passos següents.

Primer s'activa l'entorn.

```
source /opt/ros/indigo/setup.bash
```

A continuació, es construeix l'espai de treball i es compila.

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/
```

```
catkin_make
```

Finalment, es comprova que l'entorn de treball està en el directori actual.

```
source devel/setup.bash
```

```
echo $ROS_PACKAGE_PATH
```

Un cop s'ha configurat l'entorn de treball, es passa a descarregar i instal·lar l'Onboard-SDK-ROS, tal i com s'ha explicat prèviament en l'apartat 2.3. El SDK és el conjunt d'eines necessari pel desenvolupament de software amb el dron i inclou el driver ROS de la càmera i llibreries per compatibilitat amb C++ i Python, entre d'altres.

Un cop dintre de l'espai de treball, s'ha d'activar el master amb l'arxiu *sdk_manifold.launch* modificat tal i com s'ha explicat en l'apartat 2.3, a través de la comanda següent.

```
roslaunch dji_sdk sdk_manifold.launch
```

El següent pas és activar la càmera Zenmuse a través de l'*image_view*.

```
rosrun image_view image_view image:=/dji_sdk/image_raw
```

Amb l'*image_view,* es transmet la imatge de la càmera a color sense tractar. Aquesta imatge es publica en un topic en el qual es subscriu el detector *object_tracker.py*.

El programa de tractament de la imatge *object_tracker.py* té l'objectiu d'agafar una imatge amb la càmera a color Zenmuse X3, detectar l'element desitjat i seguir-lo en temps real. La finalitat del projecte és poder utilitzar-se en missions de rescat amb gossos. Com que normalment aquestes missions tenen lloc en muntanyes, boscos o zones amb molta vegetació, el gos ha de portar un element de color vistós que es pugui filtrar amb facilitat en zones on abunden els colors terra, com en la Figura 28.

Figura 28. Gos amb armilla de color vistós

El rang de colors escollit té el límit inferior a (20, 100, 100) i el límit superior a (30, 255, 255). Per tant, comprén els tons groguencs. El programa de tractament d'imatges *object_tracker* segueix l'algorisme mostrat en la Figura 29.

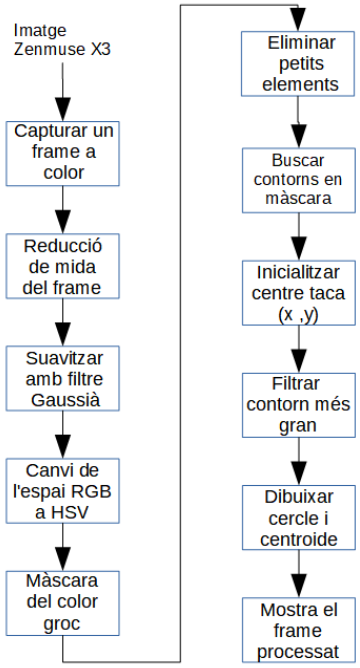


Figura 29. Algorisme tractament de la imatge a color

Primer de tot, captura un frame que rep a través de la càmera Zenmuse X3. Un cop té el primer frame, canvia la mida de la imatge a una amplada de 600 px. El fet de reduir la mida del frame permet processar la imatge més ràpidament, augmentant el FPS, al tenir menys imatge per processar. Tot seguit, se li aplica un filtre Gaussià que converteix la imatge a un *blur*, suavitzant-la i eliminant el soroll d'aquesta.

A continuació, es passa el color de la imatge filtrada de l'espai RGB, amb el qual treballa la càmera, a l'espai HSV. Després, es crea una màscara pel color groc delimitat entre els límits inferior i superior establerts anteriorment i, seguidament, es procedeix a erosionar i dilatar la imatge per eliminar petites taques que es puguin trobar en la màscara. Tot seguit, es busquen contorns en la màscara i s'estableix l'actual centre de la taca (x, y).

El programa només seguirà si ha trobat com a mínim un contorn. Si n'ha trobat més d'un, es queda amb el més gran i l'utilitza per computar el cercle mínim de tancament i el centroide. Seguidament, si el radi del cercle és superior al radi mínim, dibuixa el cercle i el centroide en el frame actual, es mostra el frame en pantalla i es torna a inicialitzar l'algorisme en bucle. Per últim, el programa publica les coordenades x i y de l'objecte detectat en el tòpic *position* en forma d'una cadena *string* amb les coordenades separades per una coma.

A continuació, el programa de seguiment listener.cpp es subscriu al tòpic *position* per rebre les coordenades del centre de l'objecte. Aquest programa s'encarrega de modificar la trajectòria de vol del dron per tal que l'element detectat estigui sempre al centre del *frame*.

El programa disposa d'un menú, Figura 30, amb el qual es pot seleccionar l'acció que realitzarà el dron. Inclou funcions bàsiques com l'aterratge o la calibració del gimbal i la detecció i seguiment d'objectes.

```
+-------------------------- < Main menu > ------------------------+
| [1]  SDK Version Query        | [6]  Go Home                     |
| [2]  Request Control          | [7]  Gimbal Control Sample       |
| [3]  Release Control          | [8]  Attitude Control Sample     |
| [4]  Takeoff                  | [9]  Take a Picture              |
| [5]  Landing                  | [10] Object Detection and Tracker|
+-----------------------------------------------------------------+
input 1/2/3 etc..then press enter key
use `rostopic echo` to query drone status
--------------------------------------
Enter Input Val:
```

Figura 30. Menú del programa

El primer pas és calcular les distàncies x i y en píxels entre el centre del *frame* (300, 225) i el centre de l'objecte, mitjançant la següent equació:

$$\text{Distància}_x = x - 300; \text{Distància}_y = 225 - y \quad \text{(Eq. 1)}$$

On:

Distància: distància x o y del centre del *frame* al centre de l'objecte en px

x: coordenada x de l'objecte

y: coordenada y de l'objecte

Aquestes equacions tenen en compte el sistema de coordenades de la càmera, Figura 31. Per aquest motiu, la coordenada x és el minuend per a calcular la distància x i, en canvi, la y és el subtrahend per a calcular la distància y. D'aquesta manera, es passen les distàncies x i y a uns eixos de coodenades estàndard.

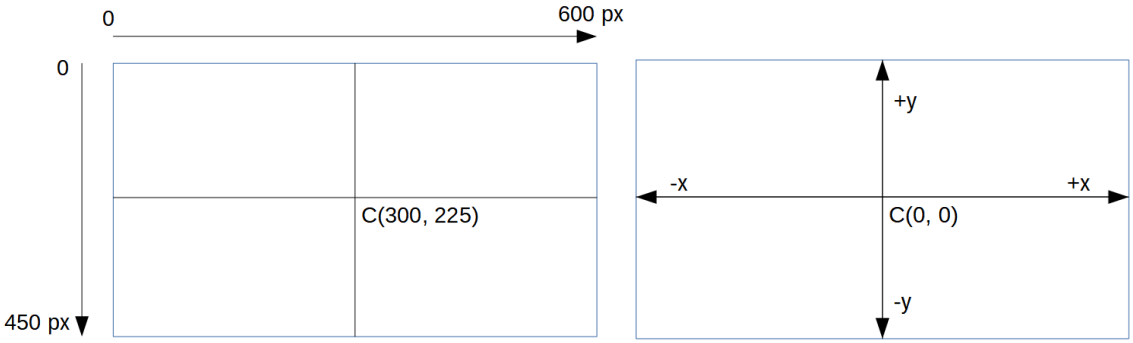


Figura 31. Coordenades càmera (esquerra) i eixos de coordenades estàndard (dreta)

Tot seguit, s'ha de convertir la distància en píxels a metres, ja que aquesta relació varia en funció de l'alçada del dron. Per fer-ho, s'utilitza la *Ground Sampling Distance* (GSD). La GSD és la distància entre dos centres de píxels consecutius mesurats a terra. Per exemple, un GSD de 5 cm significa que un píxel a la imatge representa linealment 5 cm al terra (5 * 5 = 25 centímetres quadrats). Quant més gran sigui el valor de la imatge GSD, menor serà la resolució espacial de la imatge i menys detalls seran visibles. El GSD està relacionat amb l'altura del vol: com més gran sigui l'altitud del vol, més gran serà el valor GSD i, per tant, un píxel representarà més espai, tal i com mostra la Figura 32.

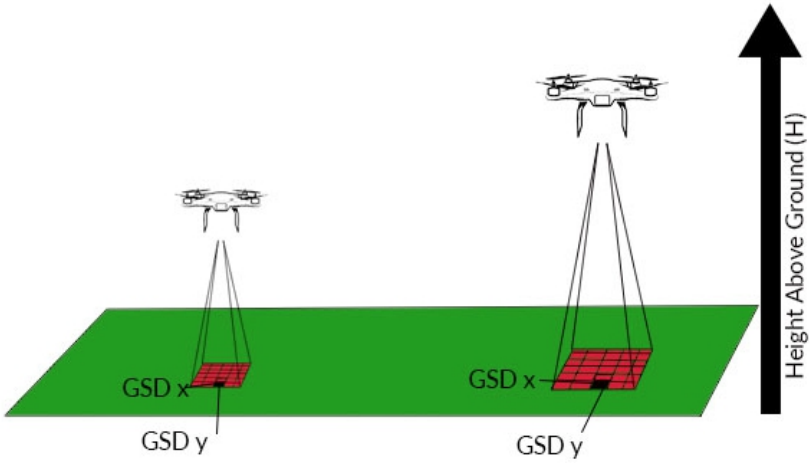


Figura 32. Ground Sampling Distance

El GSD depén de les mides tant del sensor com de la imatge, així com la longitud focal i l'alçada de vol, tal i com es pot veure en la Figura 33:

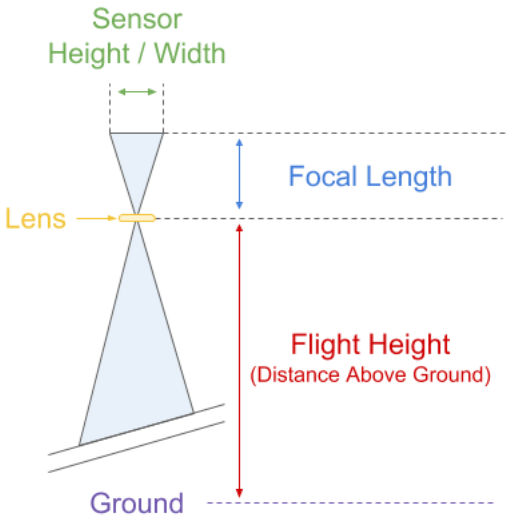


Figura 33. Paràmetres del GSD

Per a calcular el GSD, s'utilitza l'equació següent:

$$GSDw = \frac{Alçada * Amplada\,sensor}{Longitud\,focal * Amplada\,imatge}; GSDh = \frac{Alçada * Alçada\,sensor}{Longitud\,focal * Alçada\,imatge} \quad \text{(Eq. 2)}$$

On:

GSDw: Ground Sampling Distance width (eix x) en cm/px

GSDh: Ground Sampling Distance height (eix y) en cm/px

Les mides del sensor són 6.17mm x 4.55mm, la longitud focal és de 3.6mm i le mides de la imatge són 600px x 450px. Multiplicant les distàncies x i y calculades anteriorment amb GSDw i GSDh, respectivament, i dividint-les per 100 obtenim les distàncies en metres.

El següent pas és delimitar una zona central en la imatge, tal I com mostra la Figura 34. Si l'objecte es situa dintre d'aquesta zona, el dron no es mourà ja que, independentment de l'alçada de vol, estarà sempre a una distància de com a màxim 1 metre de l'objecte, en aquest cas.

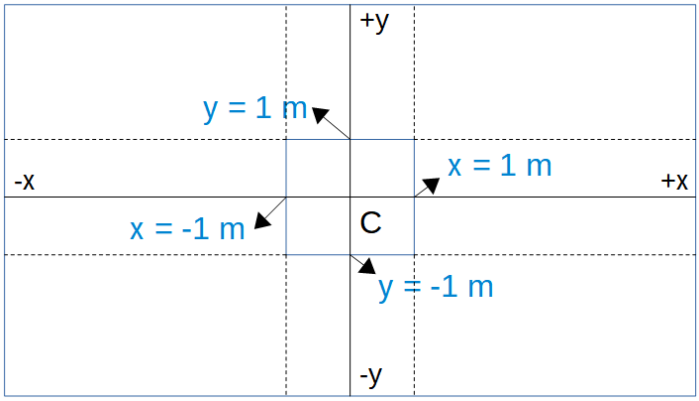


Figura 34. Delimitació de la zona central de la imatge

Com es pot observar, el *frame* queda dividit en 9 zones: la zona central més 8 zones exteriors. En funció de en quina zona es trobi l'element detectat, el dron realitzarà un moviment diferent per tal de seguir l'objectiu i mantenir-lo en la zona central del *frame*. D'aquesta manera, si per exemple el centre de l'objecte es situa en x > 1m i -1m < y < 1m, es mourà cap a l'esquerra i si es situa en x > 1m i y > 1m, es mourà en direcció sud-oest (esquerra i endarrere).

El moviment d'un dron com el Matrice 100 es basa en petites rotacions en tres eixos, com mostra la Figura 35.

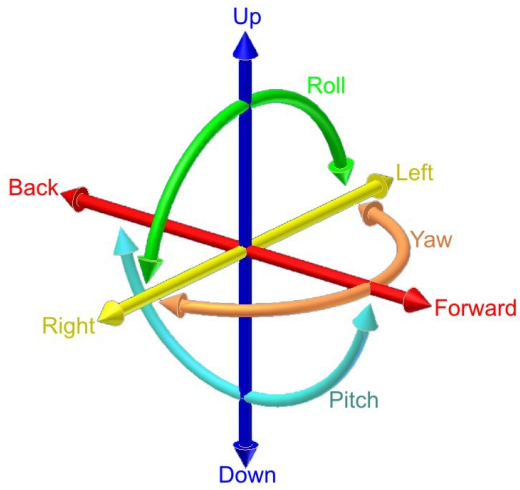


Figura 35. Eixos de moviment d'un dron

L'eix de pitch s'encarrega de controlar el moviment d'avançament o retrocés del dron. L'eix de roll controla el desplaçament lateral del dron. Per últim, l'eix de yaw té la funció de modificar l'orientació del dron.

Per controlar el moviment del dron s'utilitza la funció *drone->attitude_control(0x40, y, x, h, yaw)*, la qual depén dels desplaçaments x (moviment lateral) i y (moviment d'avançament i retrocés) en metres, l'altura h en metres i l'angle de yaw. D'aquesta manera, depenent de la zona de detecció de l'objecte, el dron realitzarà els moviment descrits en la Taula 1.

Zona de detecció de l'objecte (m)		Desplaçament		Moviment del dron
x	y	y	x	
-1 < x < 1	-1 < y < 1	0	0	Repòs, sense moviment
x > 1	-1 < y < 1	0	+	Moviment cap a la dreta
x < -1	-1 < y < 1	0	-	Moviment cap a la l'esquerra
-1 < x < 1	y > 1	+	0	Moviment cap endavant
-1 < x < 1	y < -1	-	0	Moviment cap endarrere
x > 1	y > 1	+	+	Moviment cap a la diagonal superior dreta
x < -1	y > 1	+	-	Moviment cap a la diagonal superior esquerra
x > 1	y < -1	-	+	Moviment cap a la diagonal inferior dreta
x < -1	y < -1	-	-	Moviment cap a la diagonal inferior esquerra

Taula 1. Moviment del dron depenent de la posició de l'element detectat

El programa de seguiment seguirà executant-se en bucle fins que l'usuari l'aturi i sol·liciti el control manual del dron amb el RC.

En la Figura 36 es mostra la interconnexió de tots el nodes ROS en funcionament.

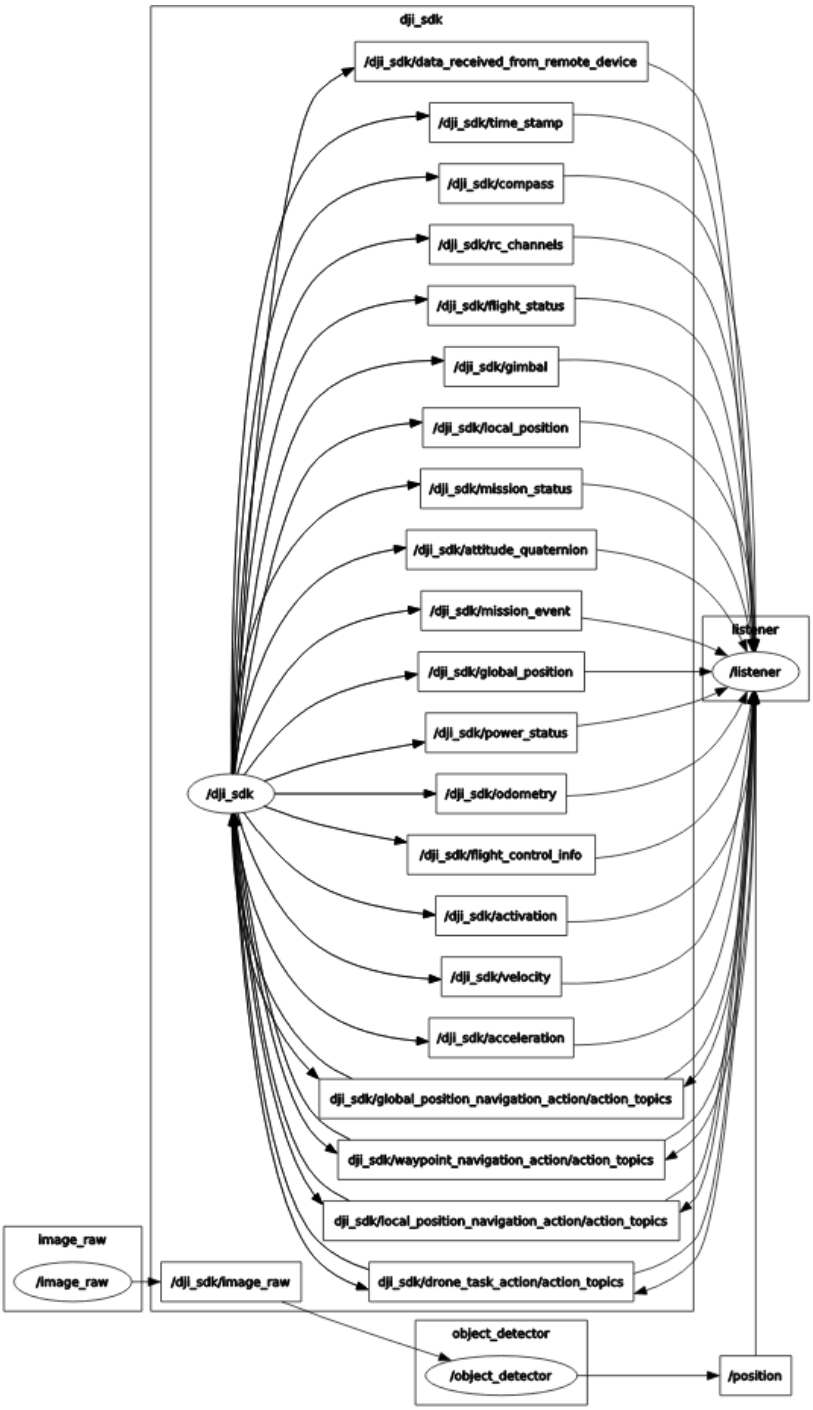
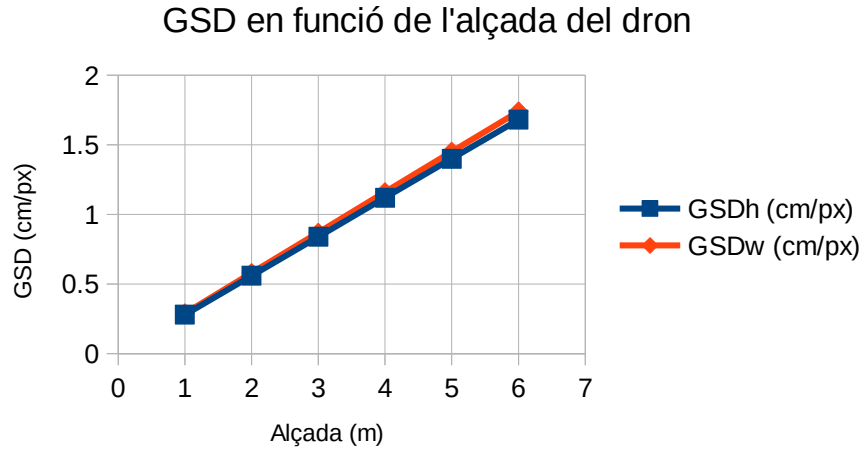


Figura 36. Interconnexió dels nodes ROS

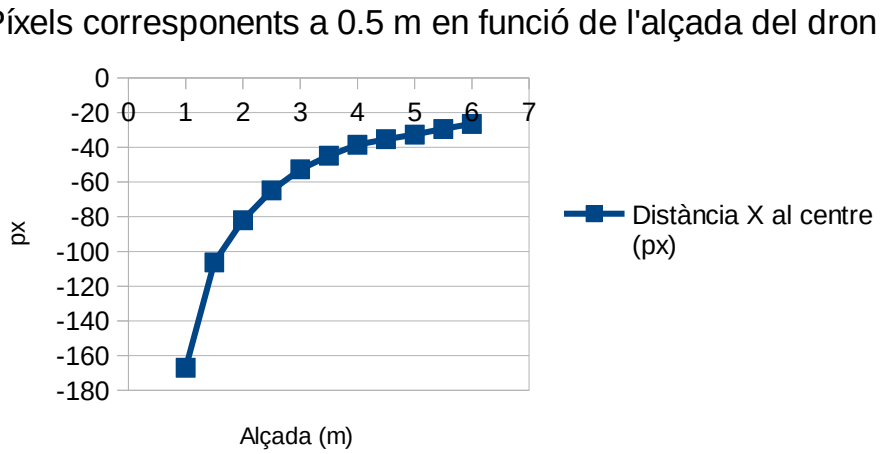
5. RESULTATS

Primer es va realitzar un vol de prova amb el dron. L'objectiu era comprovar que la relació del GSD, utilitzant el valors del càmera Zenmuse X3, era correcte. Per realitzar la comprovació, es va fer volar el dron verticalment a diverses alçades i a una distància x de 0.5 metres del centre de l'objecte a detectar.

Abans de la prova de vol, es van calcular els GSD per alçades compreses entre 1 i 6 metres. L'increment d'alçada suposa un increment en el valor del GSD linealment proporcional. Per tant, a 1 metre, un píxel representa una àrea de 0.28 cm x 0.29 cm i a 2 metres l'àrea és de 0.56 cm x 0.58 cm.



Els valors de la distància x i y en px es van enregistrar en un fitxer amb el qual es van obtenir el següents resultats:



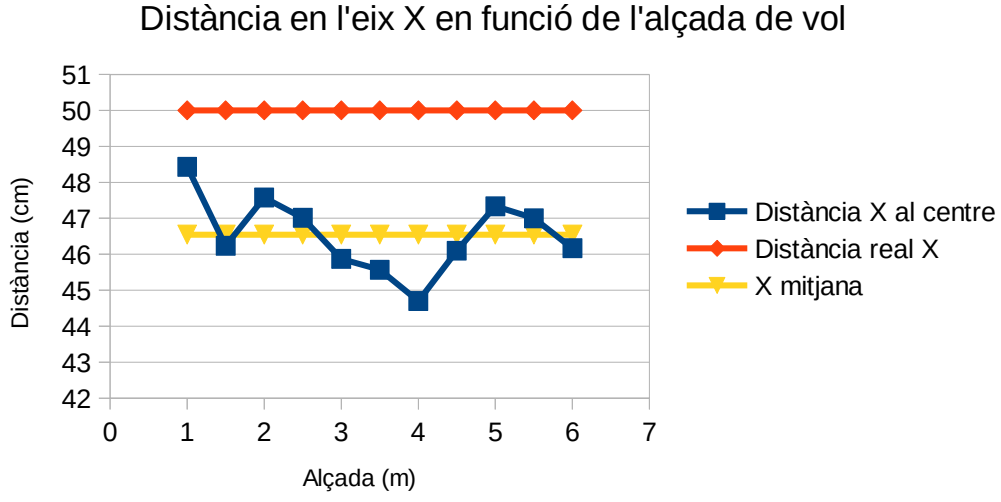
El centre de l'objecte estava situat a l'esquerra del centre del *frame*, per tant, el valor de x és negatiu. Com es pot observar, quant més gran és l'alçada de vol del dron, menor és el valor de la distància en píxels, ja que, com s'ha pogut comprovar anteriorment, la distància que representa un píxel s'incrementa amb l'alçada.

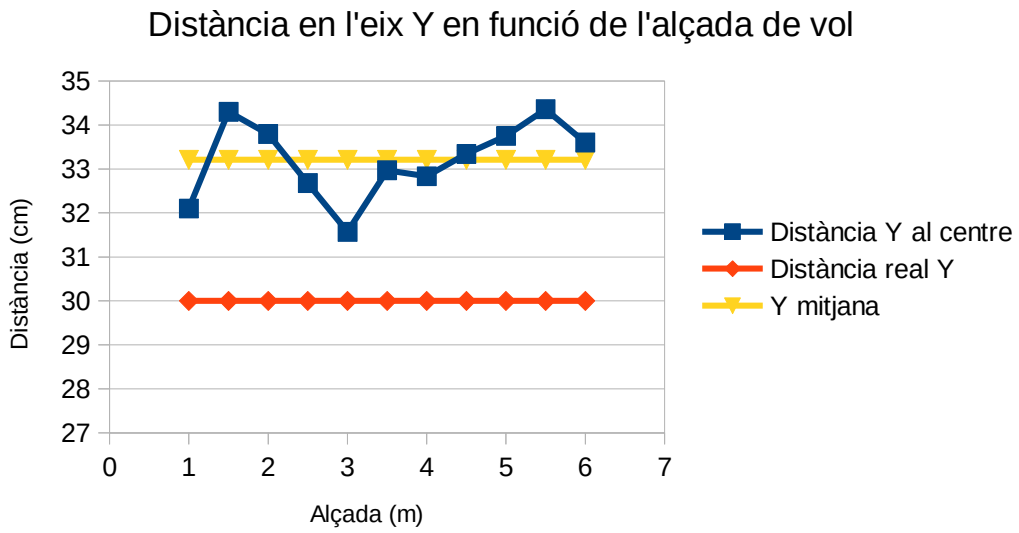
A continuació, es van passar els valors obtinguts de px a cm utilitzant el GSD corresponent a cada alçada i es van comparar amb les distàncies x i y reals, 0.5 m i 0.3 m respectivament, per trobar l'error en la mesura, tal i com mostra la Taula 2.

Alçada (m)	Distància X (px)	Distància Y (px)	Distància X (cm)	Distància Y (cm)	Error abs. X (cm)	Error abs. Y (cm)
1.0	-167.0100	-114.6443	48.4329	32.1004	1.5671	2.1004
1.5	-106.2862	-81.6636	46.2345	34.2987	3.7655	4.2987
2.0	-82.0307	-60.3443	47.5778	33.7928	2.4222	3.7928
2.5	-64.8447	-46.6826	47.0124	32.6778	2.9876	2.6778
3.0	-52.7252	-37.5846	45.8709	31.5711	4.1291	1.5771
3.5	-44.8958	-33.6402	45.4592	32.9674	4.4308	2.9674
4.0	-38.5334	-29.3161	44.6987	32.8340	5.3013	2.8340
4.5	-35.3240	-26.4602	46.0978	33.3398	3.9022	3.3398
5.0	-32.6445	-24.1066	47.3345	33.7492	2.6655	3.7492
5.5	-29.4660	-22.3095	46.9982	34.3567	3.0018	4.3567
6.0	-26.5332	-19.9995	46.1678	33.5991	3.8322	3.5991

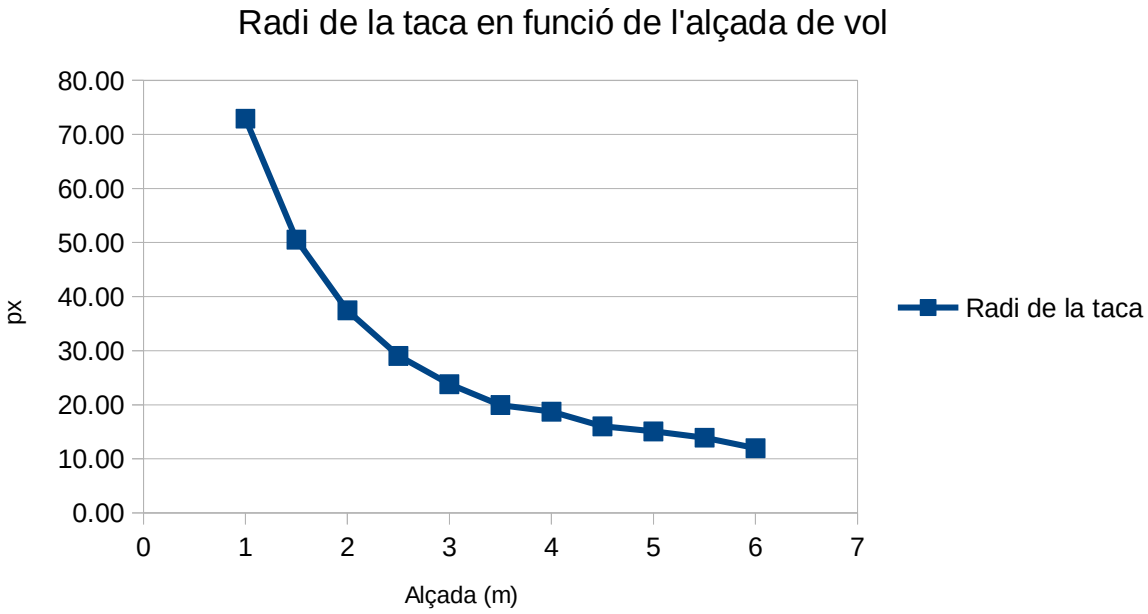
Taula 2. Distàncies X i Y i errors de mesura

S'observa que l'error en l'eix x oscil·la entre 1.5 i 4.5 cm i l'error en l'eix y, entre 1.5 i 4.3 cm. La mitjana de les mesures en l'eix x té un error d'aproximadament 3.5 cm i, en l'eix y, de 3.2 cm.





Per altra banda, es van agafar les dades del radi de la taca detectada a diverses alçades I es van obtenir els resultats següents:



Es pot observar que, al igual que la distància al centre analitzat prèviament, el valor en px del radi es redueix en funció de l'alçada.

Els tests del programa de detecció i seguiment es realitzar amb el simulador de vol que incorpora el software DJI Assistant 2, tal i com mostra la Figura 37, per comprovar que les trajectòries de vol en funció de la posició de l'objecte són correctes.

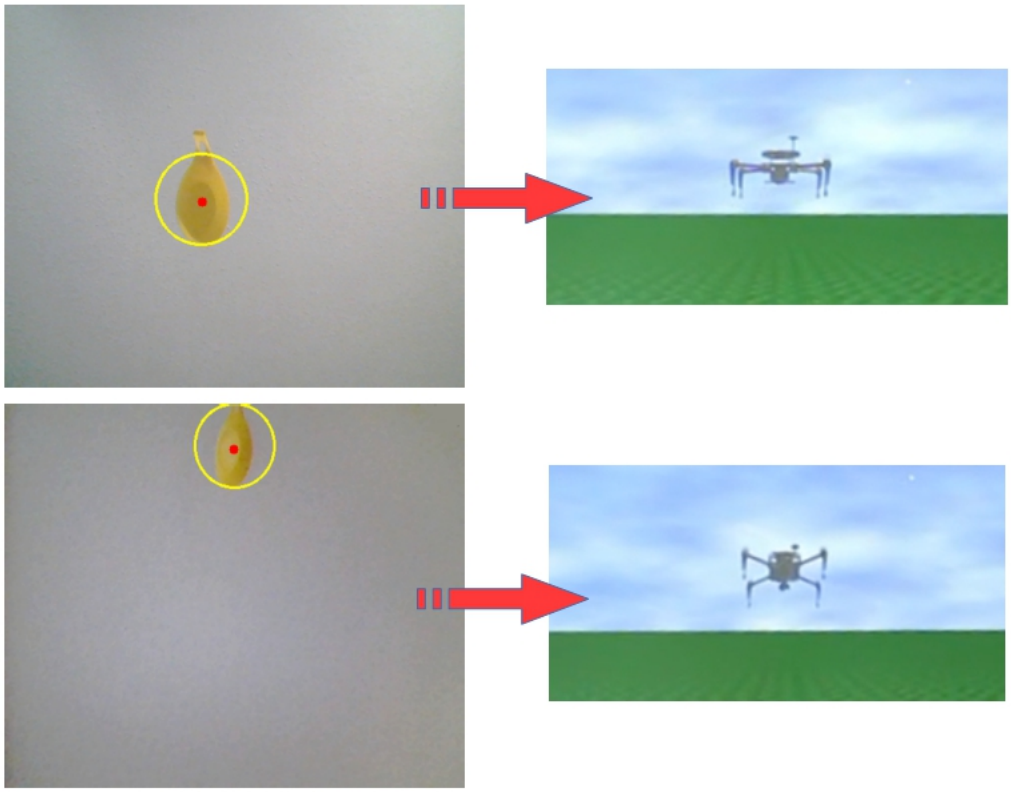
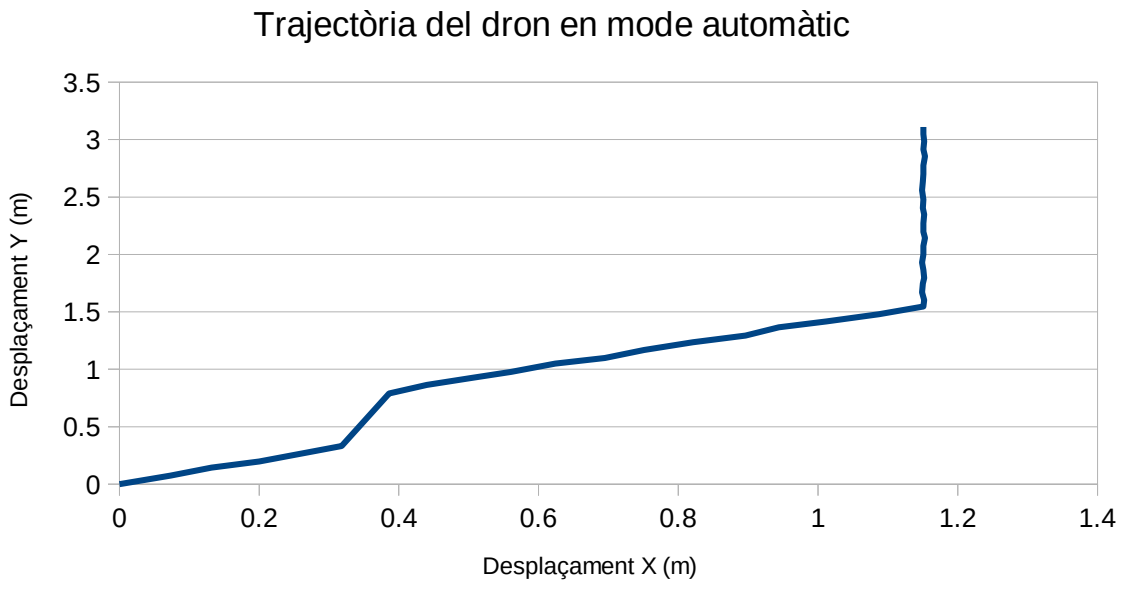


Figura 37. Programa de detecció (esquerra) i simulació del seguiment (dreta)

Finalment, es va realitzar un vol en mode automàtic, en el qual l'objecte estava situat a una distància d'aproximadament 4 metres endavant del dron i 2 metres a la dreta en paral·lel a l'eix y del dron, i es van obtenir els resultats següents:



Tal i com es pot observar, el dron es va moure en diagonal fins a haver-se desplaçat 3 metres en l'eix x i estar, per tant, a menys d'un metre de l'objectiu. A continuació, es va desplaçar cap endavant fins a estar a menys d'un metre en l'eix y de l'objecte detectat. Finalment, com que estava a -1 < x < 1m i -1 < y < 1m, va romandre en repòs.

6. RESUM DEL PRESSUPOST

El projecte de programació d'un dron per a detecció i seguiment d'objectes ascendeix a un cost de catorze mil sis-cents vint euros amb vuitanta-sis cèntims, sense IVA.

7. CONCLUSIONS

En aquesta memòria s'ha detallat el procés de desenvolupament del present projecte i la seva resolució. S'han complert els objectius proposats, complint amb les requesits prèviament establerts, per realitzar el seguiment d'un objecte mitjançant el dron Matrice 100 amb la càmera a color Zenmuse X3 i l'ordinador d'abord Manifold.

La resolució del projecte ha consistit en el disseny d'un sistema basat en el ROS format per dos programes i diversos nodes que es comuniquen entre si. El primer programa, escrit en Python, s'encarrega de tractar la imatge rebuda directament de la càmera, detectar la posició de l'element a seguir i transmetre les coordenades de l'objectiu. El segon programa, en llenguatge C++, converteix les coordenades rebudes a distàncies reals en metres i modifica la trajectòria de vol del dron per tal que l'objecte es mantingui al centre de la imatge i, per tant, que el dron es posicioni sempre a sobre d'aquest.

Jonah Fernández González

Graduat en Enginyeria Electrònica Industrial i Automàtica

Girona, 30 de juliol de 2019

8. RELACIÓ DE DOCUMENTS

El present projecte consta dels següents documents: memòria, plec de condicions, estat d'amidaments i pressupost.

9. BIBLIOGRAFIA

ACADEMIA. Ros. (http://www.academia.edu/36437179/Ros_robotics_by_example_Indigo_, 20 de novembre de 2018).

GITHUB. Manifold Cam. (https://github.com/dji-sdk/Manifold-Cam, 24 d'octubre de 2018).

PJREDDIE. Darknet YOLO Real-Time Object Detection. (https://pjreddie.com/darknet/yolo/, 10 de gener de 2019).

PYIMAGESEARCH. Real-Time Detection. (https://www.pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/, 22 d'abril de 2019).

STANFORD. Deep Drone. (https://web.stanford.edu/class/cs231a/prev_projects_2016/deep-drone-object__2_.pdf, 14 de desembre de 2018).

10. GLOSSARI

API: Application Programming Interface

CNN: Convolutional Neural Network

CPU: Central Processing Unit

GPU: Graphics Processing Unit

GSD: Ground Sampling Distance

HSV: Hue, Saturation, Value

LTS: Long Term Support

PC: Personal Computer

RC: Remote Controller

RGB: Red, Green, Blue

ROS: Robot Operating System

SDK: Software Development Kit

SSH: Secure Shell

USB: Universal Serial Bus

YOLO: You Only Look Once

A. CODIS

En aquest annex es mostraran els codis realitzats per la implementar la detecció i el seguiment d'objectes amb el dron.

A.1. Codi Detecció d'objectes

```
#!/usr/bin/env python
from __future__ import print_function

# importa els paquets necessaris
from collections import deque
import argparse
import imutils
import roslib
import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image, BatteryState, NavSatFix, Imu, Joy
from geometry_msgs.msg import QuaternionStamped, Vector3Stamped,
PointStamped, Pose, PoseStamped
from rosgraph_msgs.msg import Log
from cv_bridge import CvBridge, CvBridgeError


class blob_tracker:

  def __init__(self):
    #self.image_pub =
rospy.Publisher("image_topic",Image,queue_size=10)

    self.pub = rospy.Publisher('position', String, queue_size=10)

    self.bridge = CvBridge()
    self.image_sub =
rospy.Subscriber("/dji_sdk/image_raw",Image,self.callback)


  def callback(self,data):
    try:
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
      print(e)


    # defineix el límits superior i inferior de la taca groga en
    # l'espai de color HSV
    yellowLower = (20, 100, 100)
    yellowUpper = (30, 255,255)
```

```
    (rows,cols,channels) = cv_image.shape
    if cols > 60 and rows > 60 :

      # canvia la mida del frame, ho pasa a blurs i ho converteix a
      # l'espai de color HSV
      cv_image = imutils.resize(cv_image, width=600)
      blurred = cv2.GaussianBlur(cv_image, (11, 11), 0)
      hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

      # construeix una màscara pel color groc i, a continuació,
      # realitza una sèrie de dilatacions i erosions per eliminar
      # petites taques en la màscara
      mask = cv2.inRange(hsv, yellowLower, yellowUpper)
      mask = cv2.erode(mask, None, iterations=2)
      mask = cv2.dilate(mask, None, iterations=2)

      # busca contorns en la màscara i inicialitza l'actual centre de
      # l'objecte (x, y)
      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)
      cnts = imutils.grab_contours(cnts)
      center = None

      # només continua si s'ha trobat com a mínim un contorn
      if len(cnts) > 0:
            # troba el contorn més gran de la màscara i l'utiliza per
            # computar el cercle mínim de tancament i el centroide
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center= (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

            # només continua si el radi té un tamany mínim de 10 px
            if radius > 10:
                  # dibuixa el cercle i el centroide en el frame
                  cv2.circle(cv_image, (int(x), int(y)), int(radius),
                        (0, 255, 255), 2)
                  cv2.circle(cv_image, center, 5, (0, 0, 255), -1)


    cv2.imshow("Image window", cv_image)
    cv2.waitKey(3)

    try:

      position_str = '{:0.2f}'.format(x)+","+'{:0.2f}'.format(y)
+","+'{:0.2f}'.format(radius)
      rospy.loginfo(position_str)
      self.pub.publish(position_str)


    except CvBridgeError as e:
      print(e)

def main(args):
  ic = blob_tracker()
  rospy.init_node('blob_tracker', anonymous=True)
```

```
  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

A.2. Codi Seguiment

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <string>
#include <iostream>
#include <stdio.h>
#include <dji_sdk/dji_drone.h>
#include <cstdlib>
#include <stdlib.h>
#include <actionlib/client/simple_action_client.h>
#include <actionlib/client/terminal_state.h>
#include <fstream>

using namespace DJI::onboardSDK;

class Escolta
{
  public:
        float x=10.0;
        float y=10.0;
        float r=10.0;
        float GSDh=0.28;
        float GSDw=0.29;
        float x_meters=0.00;
        float y_meters=0.00;

    void chatterCallback(const std_msgs::String::ConstPtr& msg);
};
```

```
static void Display_Main_Menu(void)
{
    printf("\r\n");
    printf("+-------------------- < Main menu > -----------------+\n");
    printf("| [1] SDK Version Query| [6] Go Home                 |\n");
    printf("| [2] Request Control  | [7] Gimbal Control Sample   |\n");
    printf("| [3] Release Control  | [8] Attitude Control Sample |\n");
    printf("| [4] Takeoff          | [9] Take a Picture          |\n");
    printf("| [5] Landing          | [10] Object Detection and Tracker
|\n");
    printf("+-----------------------------------------------------+\n");
    printf("input 1/2/3 etc..then press enter key\r\n");
    printf("use `rostopic echo` to query drone status\r\n");
    printf("-------------------------------------\r\n");
}

void Escolta::chatterCallback(const std_msgs::String::ConstPtr& msg)
{
     std::sscanf(msg->data.c_str(), "%f,%f,%f", &x, &y, &r);

     GSDw = 2.9; // Ground Sampling Distance width en cm/px
     GSDh = 2.8; // Ground Sampling Distance height en cm/px

     x_meters=(x-300)*GSDw/100; // Distància x al centre del frame en m
     y_meters=(225-y)*GSDh/100; // Distància y al centre del frame en m

}

int main(int argc, char **argv)
{
   int main_operate_code = 0;
   int temp32;
   bool valid_flag = false;
   bool err_flag = false;
   ros::init(argc, argv, "listener");
   ros::NodeHandle nh;
   DJIDrone* drone = new DJIDrone(nh);
```

```
   Escolta myDron;

   ros::Subscriber sub = nh.subscribe<std_msgs::String>("position",
1000, &Escolta::chatterCallback, &myDron);

   FILE *f;
   f = fopen("data.txt", "a");

   Display_Main_Menu();
   while(1)
    {
        ros::spinOnce();
        std::cout << "Enter Input Val: ";
        while(!(std::cin >> temp32))
         {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        std::cout << "Invalid input.  Try again: ";
         }

          if(temp32>0 && temp32<38)
        {
            main_operate_code = temp32;
        }
        else
        {
            printf("ERROR - Out of range Input \n");
            Display_Main_Menu();
            continue;
        }
        switch(main_operate_code)
        {
          case 1:
            /* SDK version query*/
            drone->check_version();
            break;
             case 2:
```

```
        /* request control ability*/
        drone->request_sdk_permission_control();
        break;
    case 3:
        /* release control ability*/
        drone->release_sdk_permission_control();
        break;
    case 4:
        /* take off */
        drone->takeoff();
        sleep(7);
        drone->attitude_control(0x40, 0, 0, 9, 0);
        sleep(2);
        drone->gimbal_angle_control(0, -1000, 0, 20);
        break;
    case 5:
        /* landing*/
        drone->landing();
        sleep(2);
        drone->gimbal_angle_control(0, 0, 0, 20);
        break;
    case 6:
        /* go home*/
        drone->gohome();
        break;
    case 7:
        /*gimbal test*/

        drone->gimbal_angle_control(0, 0, 1800, 20);
        sleep(2);
        drone->gimbal_angle_control(0, 0, -1800, 20);
        sleep(2);
        drone->gimbal_angle_control(300, 0, 0, 20);
        sleep(2);
        drone->gimbal_angle_control(-300, 0, 0, 20);
        sleep(2);
        drone->gimbal_angle_control(0, 300, 0, 20);
        sleep(2);
```

```
        drone->gimbal_angle_control(0, -300, 0, 0);
        sleep(2);
        drone->gimbal_speed_control(100, 0, 0);
        sleep(2);
        drone->gimbal_speed_control(-100, 0, 0);
        sleep(2);
        drone->gimbal_speed_control(0, 0, 200);
        sleep(2);
        drone->gimbal_speed_control(0, 0, -200);
        sleep(2);
        drone->gimbal_speed_control(0, 200, 0);
        sleep(2);
        drone->gimbal_speed_control(0, -200, 0);
        sleep(2);
        drone->gimbal_angle_control(0, 0, 0, 20);
        break;

    case 8:
        /* attitude control sample*/
        drone->takeoff();
        sleep(8);


        for(int i = 0; i < 100; i ++)
        {
            if(i < 90)
                drone->attitude_control(0x40, 0, 2, 0, 0);
            else
                drone->attitude_control(0x40, 0, 0, 0, 0);
            usleep(20000);
        }
        sleep(1);

        for(int i = 0; i < 200; i ++)
        {
            if(i < 180)
                drone->attitude_control(0x40, 2, 0, 0, 0);
            else
```

```
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0x40, -2, 0, 0, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0x40, 0, 2, 0, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0x40, 0, -2, 0, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
```

```
            drone->attitude_control(0x40, 0, 0, 0.5, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0x40, 0, 0, -0.5, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0xA, 0, 0, 0, 90);
        else
            drone->attitude_control(0xA, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    for(int i = 0; i < 200; i ++)
    {
        if(i < 180)
            drone->attitude_control(0xA, 0, 0, 0, -90);
        else
            drone->attitude_control(0xA, 0, 0, 0, 0);
        usleep(20000);
    }
    sleep(1);

    drone->landing();
```

```
        break;

    case 9:
        /*take a picture*/
        drone->take_picture();
        break;

case 10:
        /*object detection and tracker*/

    for(;;)
    {
        ros::spinOnce();

         if (myDron.x_meters < 1)
         {
             if (myDron.y_meters < 1)
             {
               for(int i = 0; i < 200; i ++)
               {
                  if(i < 180)
                    drone->attitude_control(0x40, 1, 1, 0, 0);
                  else
                     drone->attitude_control(0x40, 0, 0, 0, 0);
                     //usleep(20000);
               }
             }
              else if (myDron.y_meters < -1)
              {
                for(int i = 0; i < 200; i ++)
                {
                   if(i < 180)
                      drone->attitude_control(0x40, -1, 1, 0, 0);
                   else
                      drone->attitude_control(0x40, 0, 0, 0, 0);
                      //usleep(20000);
                }
```

```
        }
        else
        {
          for(int i = 0; i < 200; i ++)
          {
             if(i < 180)
                drone->attitude_control(0x40, 0, 1, 0, 0);
             else
                drone->attitude_control(0x40, 0, 0, 0, 0);
                //usleep(20000);
          }
        }
    }

    else if (myDron.x_meters < -1)
    {
        if (myDron.y_meters < 1)
        {
          for(int i = 0; i < 200; i ++)
          {
             if(i < 180)
                drone->attitude_control(0x40, 1, -1, 0, 0);
             else
                drone->attitude_control(0x40, 0, 0, 0, 0);
                //usleep(20000);
          }
        }
        else if (myDron.y_meters < -1)
        {
          for(int i = 0; i < 200; i ++)
          {
             if(i < 180)
               drone->attitude_control(0x40, -1, -1, 0, 0);
             else
               drone->attitude_control(0x40, 0, 0, 0, 0);
               //usleep(20000);
          }
        }
```

```
    else
    {
      for(int i = 0; i < 200; i ++)
      {
         if(i < 180)
           drone->attitude_control(0x40, 0, -1, 0, 0);
         else
            drone->attitude_control(0x40, 0, 0, 0, 0);
            //usleep(20000);
      }
    }
}

else
 {
    if (myDron.y_meters < 1)
    {
      for(int i = 0; i < 200; i ++)
      {
         if(i < 180)
           drone->attitude_control(0x40, 1, 0, 0, 0);
         else
            drone->attitude_control(0x40, 0, 0, 0, 0);
            //usleep(20000);
      }
    }
    else if (myDron.y_meters < -1)
    {
      for(int i = 0; i < 200; i ++)
      {
         if(i < 180)
           drone->attitude_control(0x40, -1, 0, 0, 0);
         else
            drone->attitude_control(0x40, 0, 0, 0, 0);
            //usleep(20000);
      }
    }
```

```
                    else
                    {
                      for(int i = 0; i < 200; i ++)
                      {
                         if(i < 180)
                           drone->attitude_control(0x40, 0, 0, 0, 0);
                         else
                            drone->attitude_control(0x40, 0, 0, 0, 0);
                            //usleep(20000);
                      }
                     }
                  }

            } //END del FOR
                 break;
        }
        main_operate_code = -1;
        Display_Main_Menu();
    }


   while (ros::ok())
   {
      ros::spinOnce();
      printf("X=[%0.2f]\n",myDron.x);
      printf("Y=[%0.2f]\n",myDron.y);
      printf("R=[%0.2f]\n",myDron.r);

      fprintf(f, "X=[%0.2f], Y=[%0.2f], R=[%0.2f]\n", myDron.x,
myDron.y, myDron.r);

      sleep(1);
   }

  fclose(f);

  return 0;
}
```

B. INSTAL·LACIÓ DE ROS INDIGO

El primer pas consisteix en configurar els respositoris d'Ubuntu. D'aquesta manera es podrà descarregar software provinent de package.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

A continuació, configurem les claus d'accés.

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Tot seguit, comprovem que els paquets Debian estan actualizats.

```
sudo apt-get update && sudo apt-get install dpkg
```

El següent pas és instal·lar ROS. De les diverses opcions disponibles, farem una instal·lació completa.

```
sudo apt-get install ros-indigo-desktop-full
```

Si es volen descarregar paquets opcionals, utilizarem la següent comanda.

```
apt-cache search ros-indigo
```

Un cop instal·lat ROS, s'ha d'inicialitzar rosdep. Facilita la instal·lació de dependències alhora de compilar i es necessari per executar components clau de ROS.

```
sudo rosdep init
```

```
rosdep update
```

Per què les variables d'entorn s'afegeixin automàticament al bash cada cop que s'obri un

shell nou o terminal, s'utilizaran les comandes següents.

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

Per finalitzar la instal·lació de ROS, utilizarem la següent comanda .

```
sudo apt-get install python-rosinstall
```

rosisntall és una eina que permet descarregar arrels i arxius de diferents paquets de ROS amb una sola comanda.

S'ha de tenir en compte que cada cop que s'obri un nou terminal, s'ha de configurar l'entorn ROS prèviament instal·lat amb la comanda següent.

```
source /opt/ros/indigo/setup.bash
```

El següent pas es crear un entorn de treball *catkin*. En aquest *workspace* es crearan i compilaran tots els scripts i paquets utilitzats en el present projecte.

Es crearà una carpeta anomenada *catkin_ws* com a entorn de treball i una subcarpeta *src* que inclourà els codis i arxius del projecte.

```
mkdir -p ~/catkin_ws/src
```

Tot seguit, compilarem l'entorn de treball amb les següents comandes, la qual s'ahurà d'utilizar cada vegada que afegim un arxiu a *src*.

```
cd ~/catkin_ws/
```

```
catkin_make
```

Com que es el primer cop que es compila l'entorn de treball, es crearà un arxiu anomenat CmakeLists.txt a la carpeta *src*. En el directori principal *catkin_ws* es crearan dos noves subcarpetes, *build* i *devel*. En la subcarpeta *devel* s'inclouen els arxius setup. *sh. Per continuar, apliquem la comanda següent.

```
source devel/setup.bash
```

Finalment, comprovem que l'entorn de treball està correctament enllaçat amb l'arxiu setup. Per fer-ho, ens hem d'assegurar que el *path* de l'entorn inclou el directori actual.

```
echo $ROS_PACKAGE_PATH
```

```
/home/youruser/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

D'aquesta manera, ROS s'ha instal·lat completament i l'entorn de treball està operatiu.

C. INSTAL·LACIÓ D'OPENCV

Abans d'instal·lar OpenCV, s'han de descarregar varies dependències per poder visualitzar i escriure imatges a la pantalla i altres eines de visió artificial.

```
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk libtbbdev libeigen3-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwbdev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev default-jdk ant libvtk5-qt4-dev
```

A continuació es descarregarà la versió d'OpenCV compatible amb els productes DJI, 2.4.9.

```
cd ~

wget http://sourceforge.net/projects/opencvlibrary/files/opencvunix/

2.4.9/opencv-2.4.9.zip

unzip opencv-2.4.9.zip

cd opencv-2.4.9
```

Tot seguit, es crearà una carpeta *build* on es guardaran els arxius compilats amb la comanda *cmake*.

```
mkdir build

cd build

cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON
-D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_
```

```
EXAMPLES =ON -D WITH_QT=ON -D WITH_OPENGL=ON -D WITH_VTK=ON ..
```

Per instal·lar OpenCV 2.4.9, utilitzem les següents comandes.

```
make
```

```
sudo make install
```

Finalment, s'han de modificar dos arxius. Amb la comanda *sudo gedit /etc/ld.so.conf.d/opencv.conf* obrim l'arxiu corresponent i afegim /usr/local/lib al final i guardem. Seguidament, configurem la llibreria amb *sudo ldconfig* i amb la comanda sudo gedit /etc/bash.bashrc i afegim dos línies el final de l'arxiu corresponent:

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
```

```
export PKG_CONFIG_PATH
```

Per què aquests canvis tinguin efecte, s'ha de reiniciar el Manifold.