Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Seguiment de les evolucions d'un gos ensinistrat per part d'un drone per a l'optimització d'operacions de rescat

Document: 1. Memòria

Alumne: Robert Gifreu Pons

Tutor: Xavier Cufi Soler

Departament: Arquitectura i Tecnologia de Coputadors

Àrea: Arquitectura i Tecnologia de Coputadors

Convocatòria (mes/any) setembre / 2016

Índex

1. Introducció 2
1.1 Antecedents 2
1.2 Objecte 2
1.3 Especificacions i abast 2
2. Descripció del dispositiu 3
2.1 Components del drone 4
3. Linux Ubuntu 14.04LTS 10
4. Robot Operating System 11
4.1 Funcionament 11
4.2 Instalació ROS 13
4.3 Espai de treball 14
5. Onboard SDK 16
6. Guidance SDK 18
7. Open CV 19
7.1 Instal·lació 20
8. Resolució 22
9. Resum del pressupost 28
10. Conclusions 29
11. Relació de documents 30
12. Bibliografia 31
13. Glossari 32
A Codi Dispositius 33
A.1 Programa Automatització 33
A.2 Programa Tractament de la Imatge 39

1. Introducció

Amb l'objectiu d'augmentar l'eficiència en les operacions de rescat de persones que es porten a terme per part d'equips formats per agents dels cossos de seguretat i gossos ensinistrats, es pretén utilitzar robots mòbils de suport.

1.1 Antecedents

Es pretén que aquest robots, equipats amb recursos suficients, puguin proporcionar dades addicionals d'interès per a la missió, i puguin captar de forma eficient les evolucions del gos ensinistrat mentre aquest realitza el seu treball. El que es pretén aconseguir és que el robot mòbil (drone) esdevingui un company habitual del gos ensinistrat sense afectar negativament la seva feina. D'aquesta manera s'ha d'aconseguir que els agents de seguretat puguin supervisar les operacions del gos i proporcionar-li ordres senzilles sense que sigui necessària la seva presència física constant.

1.2 Objecte

Amb el desenvolupament d'aquest TFG s'ha d'escollir un element tipus drone amb les característiques adients i l'equipament necessari per a poder detectar el gos ensinistrat de forma autònoma, s'ubiqui sobre la seva vertical a una alçada determinada, i realitzar-ne el seguiment en temps real. Aquesta part del TFG s'haurà de portar a terme en col·laboració estreta amb els responsables de la Brigada Canina dels cossos de seguretat. Un cop escollits tots els components necessaris s'haurà de poder operar el drone des de terra de forma eficient per una banda, i s'haurà d'implementar el funcionament descrit en mode de seguiment automàtic del gos ensinistrat per l'altra.

1.3 Especificacions i abast

La detecció i el seguiment del gos ensinistrat sobre la seva vertical per part del drone s'haurà de realitzar utilitzant tècniques de detecció d'objectes mitjançant visió per computador.

2. Descripció del dispositiu

Un drone és un vehicle aeri no tripulat dissenyat per volar de manera autònoma o bé controlat a distància per control remot.
Aquesta tecnologia primerament es va desenvolupar amb finalitat militar, per espionatge i llançament d'explosius, l'aeronau era de grans dimensions de manera que permetia carregar una gran quantitat d'explosius. Amb el pas dels anys s'ha pogut anar reduint la mida i fins i tot han aparegut nous dissenys. Avui en dia hi ha dos tipus de drones amb ales fixes i multi-rotor.

Aquests primers poden funcionar tant amb motor de combustió com amb motor elèctric, depenent de les seves característiques. La principal desavantatge d'aquests drones és que no es poden enlairar verticalment de manera que necessiten una pista d'enlairament i aterratge o bé una llançadora. D'altra banda, tenen una estructura més senzilla tenint d'aquesta manera un manteniment més econòmiques. També són més aerodinàmics i gràcies a que planegen, poden realitzar vols més llargs, tant en temps com en distància, apart de poder volar a més velocitat i portar més càrrega.

Per altre vanda els drones multi-rotor funciona amb motors elèctrics (rotors). Poden tenir des de 1 sol rotor fins a 16, essent els més usuals els de 3, 4, 6 i 8 rotors. Els rotors estan distribuïts de manera equilibrada per tal de mantenir estable el drone. En augmentar el nombre de rotors, també s'augmenta la mida del drone i la seva càrrega útil. Pot enlairar-se verticalment, de manera que no necessita la generació de cap corrent d'aire per enlairar-se i conseqüentment tampoc una pista d'enlairament i una llançadora. Això permet l'aterratge i l'enlairament a qualsevol punt. La seva estructura és complexa de manera que tenen un alt cost de manteniment però li aporta agilitat i maniobrabilitat.

El principi es va optar per realitzar el procés mitjançant el model de drone Dji Phantom 4 , pensant que per les característiques que presentava es podria realitzar les tasques, i poder facilitar el desenvolupament el disposar d'aplicacions pre-dissenyades que servirien per al projecte. No obstant es va observar que no disposava de una plataforma d' open software, el qual poder realitzar el disseny del projecte ni permetia al ser encapsulat l'addició de hardware extern, per aquest motiu finalment es va optar per el model DJI Matrice100.

2.1 Components del drone

Les característiques principals del Matrice100, és la capacitat d'autonomia de fins a 40 minuts, suportar un pes màxim d'enlairament de 3.6kg, comptant amb tots els complements carregats, i un rang de temperatures a les que pot operar, tenint com a element limitant les bateries, de entre -10ºC a 40ºC. Els elements principals del drone es descriuen a continuació.

El Xassís és l'estructura de l'aeronau sobre la qual s'instal·len tota la resta de components. Està fet de fibra de carboni per aconseguir que sigui lleuger i resistent. Aquesta estructura està dissenyada amb ranures i forats per fer l'aparell més modulable, i podent d'aquesta manera poder-hi incorporar-hi "hardware" addicional. A part Incorpora diversos ports per tal de poder fer les connexions necessàries, A la Figura 1 es pot observar el xassís sense braços del DJI Matrice 100.

Figura 1. Xassís

El DJI Matrice 100 el tractar-se d'un quadcopter, el xassís consta de 4 braços dels quals l'angle és totalment regulable per tal d'augmentar el parell. A més a més cada un d'aquests braços consta d'un sistema d'absorció de vibracions degudes als motors i també de coixinets per a protegir l'aeronau en l'aterratge.

Els motors que incorpora són el són el DJI 3510 350KV son uns petits motors elèctrics sense escombretes, es poden observar a la Figura2. Aquests motors tenen un pes de 106g i 35mm de diàmetre. Incorporen uns imants corbats que eliminen els espais dins el motor, evitant els fluxos d'aire i així incrementant-ne l'eficiència.

Figura 2. Motor DJI

L' ESC (Electronic Speed Control), tal i com el seu nom diu és una controladora de velocitat i reparteix la potència per executar les maniobres correctament. En cal una per a cada motor, ja que regula la velocitat i direcció que ha prendre cada un segons la trajectòria que hagi de seguir. El model utilitzat és el que es pot veure a la Figura 3, el DJI E SERIES 620D. Aquest model permet reduir el consum d'energia i augmentar l'eficiència i precisió dels moviments gràcies a una arquitectura de generació d'ona sinusoïdal. De la mateixa manera permet conèixer l'estat del sistema de propulsió a temps real, també actualitzar de forma independent el firmware gràcies a que incorpora un port de comunicació dúplex. El circuit d'aquesta ESC està fet a prova d'espurnes, de manera que protegeix la pròpia ESC i els seus connectors d'aquest fenomen.

Figura 3. ESC

El model d'hèlix utilitzat pel drone DJI Matrice 100, és el DJI 1345s que es veu a la Figura1 . Són hèlix de fibra de carboni reforçada que estan proveïdes d'un sistema que evita que s'afluixin durant el vol, per tal d'evitar desestabilitzar el drone i provocar-ne una caiguda

Les hèlix són aquell element que el motor fa rotar per tal d'aconseguir l'elevació de l'aparell, transformen energia mecànica en empenta. La configuració del muntatge i rotació de les hèlix es pot veure a la Figura 4. S'han de configurar els motors per tal de que girin en el sentit adequat. També és necessari equilibrar bé les hèlix per tal de tenir estabilitat.

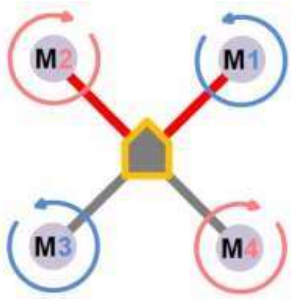

Figura 4. Rotació de les hèlix

Per tal d'evitar possibles danys a les hèlix del DJI Matrice 100 s'han afegit uns protectors amb uns materials lleugers i resistents , dissenyats per dur a terme la seva funció conjuntament amb l'ús de les hèlix, aquestes inclouen una corda de seguretat per tal de evitar que puguin entrar objectes entre les hèlix.

El DJI Matrice 100 utilitza bateries tipus Li-Po, més concretament el model TB47D que es pot observar a la Figura 5. Aquest model és un Li-Po 6S que indica que està format per 6 cel·les de 3.7V cada una col·locades en sèrie. Proporciona un voltatge de 22.2V i té una capacitat de 4500mAh. Aquest element aporta la potència necessària per accionar i fer funcionar qualsevol dispositiu electrònic que disposa el drone. Aquest drone pot utilitzar un altre model de bateria que disposa de més capacitat, el TB48D, amb 22.8V i 5700mAh. També és una Li-Po 6S, de manera que consta de 6 cel·les de 3.8V connectades en sèrie.

Figura 5. Bateria Li-Po

En l'emmagatzematge d'ambdós models, les temperatures permeses que mantindran les propietats i capacitats de les bateries variaran segons el temps que romanguin sense us. A la Taula 1 es pot veure el rang de temperatures que suporten segons el temps d'emmagatzematge.

Temps	Temp. Mínima (ºC)	Temp. Màxima (ºC)
Menys de 3 mesos	-25	45
Més de 3 mesos	-22	28

Taula 1. Rang de temperatures

Aquest drone permet afegir un compartiment extra per a una altra bateria juntament amb una badia d'expansió que augmenta el temps de vol. S'ha de tenir en compte que posar una bateria addicional augmenta també el pes de l'aeronau, això implica que la relació de temps no sigui proporcional al nombre de bateries que carrega. El temps màxim de vol aconseguit és de 40 minuts, amb els models TB48D i sense cap altre element que consumeixi energia.

El model utilitzat per carregar la bateria del drone és el A14-100P1A. El balancejador necessari per a la càrrega equilibrada de les bateries l'incorporen aquestes mateixes, de manera el carregador consisteix simplement en una font d'alimentació de 100W i una tensió de 26.3V.

El model de la controladora de vol és la N1, aquest element recull i processa totes les dades generades pels diferents sensors que duu l'aparell, inclosa la imatge real en alta definició. També interpreta i efectua suaument les ordres generades pel pilot tot mantenint estable l'aeronau.

El controlador remot *C1* utilitzat pel drone *DJI Matrice 100* que es pot observar a la Figura 6, té un rang de transmissió de 2km. El rang de transmissió és aquella distància a la qual pot estar el controlador remot del drone, si es sobrepassa es perdrà el control de l'aeronau. Aquest l'element te la finalitat que el pilot doni ordres al drone per control remot, mitjançant la radiofreqüència. Aquesta la rep el receptor i la transmet a la placa controladora que la interpreta i la fa efectiva. Els dos elements han d'estar en la mateixa freqüència. La més utilitzada és la de 2,4GHz, tot i que també pot operar a 5.85GHz.

Figura 6. Controlador remot

El mòdul GPS utilitzat per a l'aeronau és el GPS PRO PLUS que s'observa a la Figura 7. Aquest conté un suport plegable que ajuda realitzar el seguiment a temps real de l'aparell, podent orientar-lo en la direcció desitjada. Tot i ser compatible amb el "A2 GPS PRO PLUS", no es recomana fer-lo servir ja que el "GPS PRO PLUS" millora molt el sistema anti-interferència, la velocitat d'adquisició de dades i comunicació amb el satèl·lit i l'exactitud amb què ho fa.

Figura 7. Mòdul GPS

El Manifold, Figura 8, és un ordinador amb sistema encastat amb Linux. És un element molt útil per a desenvolupadors per la seva gran flexibilitat i capacitat d'ampliació. Té molt bon funcionament i és molt útil per treballar amb *software* basat en DJI SDK i en el desenvolupament d'aplicacions per a aeronaus. Conté una "NVIDIA Tegra K1", un processador multimèdia que incorpora una CPU (Central Processing Unit), una GPU (Graphic Processor Unit) i tecnologia ISP (InSystem Programming). Aquest té un alt rendiment de manera que permet treballar amb molt bons gràfics i processar a molta alta velocitat sense gaires costos d'autonomia.

Figura 8. Manifold

Es va comprar el gimbal i la càmera Zenmuse X3 , es poden observar en la figura 9, aquest primer serveix com a element per unir el drone amb la càmera i dona estabilitat a aquesta, per tal de poder aconseguir una bona imatge siguin com siguin les condicions de vol, aquest model en particular conste de 3 eixos per poder fer el control de estabilitat de la càmera. Les propietats principals de la càmera són que te una resolució de 12Mpixels amb una longitud focal de 20mm, un angle de visió de 94º i una distorsió de 0.9. Aquest model pot gravar vídeos a 4k a 30FPS (frames per second) o bé a 1080p a 60FPS.

Figura 9. Càmera i Gimbal Zenmuse X3

El sistema guidance, Figura10, és un kit de navegació basat en un sistema de detecció visual. Conté un potent processador, diferents càmeres (en grups de dos càmeres mono-color de resolució 640x480) que gairebé garanteixen la visió a 360º i varis sensors d'ultrasons. Els elements anteriors juntament amb avançats algoritmes de visió permeten al drone evitar obstacles durant vols a un màxim de 7m/s amb un rang de reacció de 0.5m a 2m. Tot i així no pot evitar dos o més obstacles alhora, de manera que s'ha d'intentar evitar aquestes situacions ja que poden ser perilloses per l'aparell. Els algoritmes estèreo d'alta precisió proporcionen informació del posicionament de l'aparell sobre gairebé qualsevol terreny obert i fins a 20m d'altitud. La foscor no és un greu problema ja que les càmeres poden obtenir imatge des amb un rang de 15 lux a 20000 lux.

En el nucli de processament s'inclou una FPGA SoC de baix cost (Altera Cyclone V) sensors de inèrcia (MPU 6050) i la capacitat de poder muntar 5 mòduls de sensors (grup càmera més ultrasons) .

Figura 10. Guidance

3. Linux Ubuntu 14.04LTS

Linux va ser creat l'any 1991 per Linus Torvald amb la idea de fer un sistema operatiu UNIX de codi lliure. El sistema operatiu Unix va ser creat per l'empresa americana AT&T per facturar totes les trucades dels seus clients l'any 1969. Linus Torvald va agafar Minix, un sistema operatiu creat per ensenyar a alumnes d'informàtica les parts d'un sistema operatiu, per crear el kernel Linux. El kernel és un programa que gestiona el hardware de l'ordinador, decideix quin programa fa ús de quin recurs o dispositius i durant quant de temps.

Linux també és conegut com GNU/Linux per que quan Linus Torvald va llençar Linux, el projecte GNU ja portava molt de temps en marxa i oferia eines bàsiques com: un command-line, biblioteca C i un compilador, però aquest projecte tenia un nucli (un kernel anomenat Hurd) que tenia molts errors. Per tal d'omplir aquest forat que tenia el sistema operatiu de GNU es va fer servir el nucli de Linux i per això es coneix com a GNU/Linux. Així el nucli del sistema operatiu és Linux i la part fonamental d'interacció entre hardware i usuari són les eines del projecte GNU. Així Ubuntu utilitza el kernel de Linux. El seu patrocinador es Canonical, una empresa britànica que ofereix el sistema de manera gratuïta i es finança mitjançant serveis vinculats al sistema operatiu. Ubuntu va néixer del codi base de Debian amb l'objectiu de fer una distribució fàcil d'utilitzar i entendre per usuaris finals.

La distribució Ubuntu deriva de la distribució Debian i és una de les més famoses per la seva senzillesa i per estar enfocat a un usuari final i no a un servidor. Ubuntu està recolzat per el kernel de Linux i el seu entorn gràfic pot ser escollit però per defecte ve amb Unity des de la versió 10.10 que va reemplaçar la versió GNOME desktop environment.

Ubuntu disposa de 2 tipus de versions estables. Una versió LTS (Long Term Support) i una no LTS. La diferència entre les dues es que Canonical, l'empresa desenvolupadora d'aquesta popular distribució de Linux, ofereix ajuda tècnica i actualitzacions de seguretat durant més o menys temps. Canonical llença cada 6 mesos una versió d'Ubuntu per escriptori. D'aquestes versions, cada 2 anys, una d'elles, és una versió LTS.

Sabent això, s'escollirà una versió LTS per obtenir un sistema operatiu que estarà actualitzat durant més temps, entre elles s'ha escollit la versió 14.04 per a la compatibilitat amb la resta de programari que necessitem.

4. Robot Operating System

ROS (Robot Operating System) va ser un projecte que va començar sota el nom de Switchyard el 2007 al departament d'Intel·ligència Artificial de Stanford per tal de donar suport al projecte STAIR2, (Stanford AI Robot 2). Aquest projecte des del 2008, es desenvolupa a Willow Garage, un institut d'investigació robòtica amb més de vint institucions col·laborant en un model de desenvolupament federat.

ROS és un framework, o plataforma de treball, pel desenvolupament de softwares en robòtica. Aquest framework està pensat per proporcionar les funcionalitats d'un sistema operatiu de clúster heterogeni, el que significa que ens aporta les característiques d'un clúster habitual, permeten l'agrupació de varis ordinadors dins d'un mateix sistema operatiu per distribuir la computació, aconseguint així repartir la carrega. Al mateix temps, al ser heterogeni, permet la diversificació de hardware i sistema operatiu, aconseguint unir ordinadors amb característiques diferents.

Una de les principals raons que està convertint el ROS en el framework més utilitzat en el món de la robòtica és per ser de programari lliure o open source, permeten així una adquisició lliure del producte, tant al món laboral com en la investigació. Fet que ha generat una gran divulgació i intercanvi de programes, tant per aficionats com per professionals d'aquest sector. Per facilitar aquest intercanvi, els desenvolupadors de ROS han realitzat una WIKI on s'incorporen uns tutorials, per usuaris novells com avançats, molt complets i ben estructurats pel fàcil enteniment del seu funcionament. També, ROS ha incorporat un apartat de respostes, ROS Answers, on els aficionats i desenvolupadors del sistema, responen als diferents dubtes que puguin sortir amb el programa.

Aproximadament, cada 8 mesos apareix una nova versió de ROS. Les versions intenten estar sincronitzades amb les versions d'Ubuntu LTS, donant suport d'una versió de ROS per les següents 3 versions posteriors d'Ubuntu. Per a la compatibilitat amb els paquets de *DJI* s'ha optat per ROS indigo.

4.1 Funcionament

ROS ofereix les biblioteques i eines per ajudar als programadors a crear aplicacions robòtiques. Proporciona abstracció del maquinari, dels controladors, biblioteques, visualitzadors, enviament de missatges, gestió de paquets, i molt més.

L'estructura bàsica del funcionament de ROS és l'enomenat graf de computació, que tracta d'una xarxa peer-to-peer dels processos de ROS, que són els que processen dades conjuntament. Aquesta xarxa consta d'uns elements bàsics, aquests són nodes, master, messages, services, tòpics, i bags.

El ROS Master proporciona registre de noms i de consulta per a la resta del graf de computació. Sense el Master, els nodes no serien capaços de trobar cada un dels missatges, intercanviar, o invocar els serveis.

Els nodes són processos que porten a terme els càlculs. ROS està dissenyat per ser modular; l'arquitectura d'un robot normalment està compresa per molts nodes. Els nodes es comuniquen entre si enviant-se messages.

Un message és simplement una estructura de dades, que comprèn els camps escrits. Tipus primitius estàndards (sencers, flotants, booleans, etc) com també tuples de tipus primitius. Els missatges també poden incloure estructures aniuades i taules (igual que les estructures de C).

Els missatges s'enruten a través d'un sistema de transport de publicació/subscripció. Un node envia un missatge mitjançant la seva publicació a un tòpic determinat. El tòpic és un nom que s'utilitza per identificar el contingut del missatge. Un node que està interessat en un determinat tipus de dades es subscriurà amb el tòpic apropiat. És possible que hi hagi múltiples publicadors i subscriptors concurrents a un mateix tòpic, i un sol node pot publicar i/o subscriure's a múltiples tòpics. En general, els publicadors i subscriptors que no són conscients de l'existència dels altres. La idea és separar la producció d'informació del seu consum. Lògicament, es pot pensar en un tema com bus de missatges de tipus inflexible. Cada bus té un nom, i qualsevol persona pot connectar-se al bus per enviar o rebre missatges, sempre que siguin del tipus correcte.

La publicació/subscripció model és un paradigma de la comunicació molt flexible, però els seus molts-a-molts, d'un mitjà de transport no és adequat per a la interacció de petició/resposta, que sovint es requereixen en un sistema distribuït. La petició/resposta es realitza a través dels serveis, que es defineixen per un parell d'estructures dels missatges: un per a la sol·licitud i un altre per la resposta. Un node ofereix un servei amb un nom i un client utilitza el servei enviant el missatge de sol·licitud i en espera de la resposta. Les biblioteques ROS client en general presenten aquesta interacció per al programador, com si es tractés d'una crida a un procediment remot.

Els bags són un format per a guardar i reproduir les dades de ROS. Són un mecanisme important per a emmagatzemar dades, com dades dels sensors, que poden ser difícils d'obtenir, però són necessàries per desenvolupar i provar algorismes.

El ROS Master actua com un servei de noms en el graf de computació. Emmagatzema els Tòpics i serveis d'informació de registre per als nodes ROS. Els nodes es comuniquen amb el Master per a que informi de la seva informació al registre. A mesura que aquests nodes es comuniquen amb el Master, aquests poden rebre informació sobre altres nodes registrats i realitzar les connexions, segons correspongui. El Master també farà devolucions de crides a aquests nodes, quan aquesta informació canviï de registre, de manera que permet que els nodes pugin crear dinàmicament les connexions dels nous nodes s'executen.
Els nodes es connecten a altres nodes directament, el Mestre només proporciona informació de cerca, igual que un servidor DNS. Els nodes que es subscriuen a un tòpic demanaran connexions des dels nodes de publicació aquest tòpic, i establiran aquesta connexió a través d'un protocol de connexió. El protocol més comú usat en un ROS es diu TCPROS, que utilitza l'estàndard TCP / IP sockets.
Els noms tenen un paper molt important en ROS: els nodes, els tòpics, els serveis, i tots els paràmetres tenen nom. Cada biblioteca de client de ROS suporta línies d'ordres de reassignació de noms, el que significa que un programa compilat pot ser reconfigurat en temps d'execució per operar en una tipologia de computació de graf diferent.

Per a l'ús que s'ha utilitzat per el projecte s'ha hagut de modificar els arxius XML, que es troben a l'arrel dels paquets per tal de incloure les dependències necessàries per al nostre disseny. També s'ha modificat els arxius CmakeLists.txt, per tal de incloure en el projecte les llibreries extres utilitzades.

Per instalar Ros i crear un entorn de treball pel qual poder desenvolupar el projecte és necessari seguir un seguit de passos.

4.2 Instalació ROS

Primer de tot s'ha de configurar ubuntu per tal de que accepti els software de ROS, en el nostre cas ROS Indigo, i configurar les claus d'accés, es pot realitzar escrivint les següents comandes en un terminal.

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

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
```

Ara que ja s'ha configurat Ubuntu , confirmem que tenim tots els paquets de Debian actualitzat, amb la comanda `sudo apt-get update .`

I ja podem passar a instal·lar ros, per això simplement utilitzarem la comanda `sudo apt-get install ros-indigo-desktop-full .`

Un cop realitzat tot el procés d'instal·lació és necessari inicialitzar rosdep, el qual permet instal·lar dependències del sistema fàcilment i és necessari per executar algun component central de ROS, per això només es te que executar les comandes en el termina.

```
sudo rosdep init
rosdep update
```

A continuació es mostra com fer que les variables d'entorn s'afegeixin automàticament a bash a cada nou terminal que s'executi, simplement executem les dos següents comandes.

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

Com a complement s'ha instal·lat el rosinstall, que és una eina de comanda, per tal de facilitar la descarrega dels arxius i les arrels de diferents paquets, simplement s'ha de executar `sudo apt-get install python-rosinstall` .
Per últim s'ha de configurar l'entorn de ROS instal·lat aplicant la comanda *source /opt/ros/indigo/setup.bash* , Aquesta comanda serà necessària per cada terminal shell que utilitzis, per tal de configurar-ho automàticament es pot modificar l'arxiu .bashrc aplicant aquesta mateixa comanda el final de l'script.

4.3 Espai de treball

Un cop instal·lat ROS i configurat els paràmetres per el funcionament, fa falta crear un entorn de treball, s'ha optat per el format catkin, on crear i compilar els paquets i scripts que volem fer servir. Per això primerament crearem la carpeta que utilitzarem com el nostre espai de treball amb la seva corresponent subcarpeta *src* on aniran els arxius que volem utilitzar. Per fer tot això farem servir les següents comandes.

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

Seguidament tot i encara no tenir cap arxiu en l'espai de treball compilarem per primer cop tot l'espai, mitjançant la següent comanda.

```
cd ~/catkin_ws/
catkin_make
```

Per últim posarem les dependències del nou espai de treball executant la *comanda source devel/setup.bash .* Finalment per comprovar que totes les dependències estan ven posades podem comprovar-ho mitjançant la comanda `echo ROS_PACKAGE_PATH` i ens tindria que sortir en el mateix terminal :

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

5. Onboard SDK

Onboard SDK habilita una interacció entre un ordinador/OES i el controlador de vol de DJI. Utilitzant l'Onboard SDK es pot utilitzar els sensors connectats per controlar la trajectòria del vehicle, emmagatzemar dades de telemetria en temps real o controlar la càmera i el gimbal per poder gestionar com i on capturar les imatges o inclús enviar dades en el dispositiu mòbil. Per això s'ha de connectar el ordinador/OES a través del un port UART TTL del drone. En general consisteix en un conjunt de llibreries i paquets per tal de tenir comunicació amb el Matrice100 i el controlador de vol A3. El tipus de controlador de vol es pot definir en l'arxiu launch del paquet central en qualsevol moment. Les funcions de l'Onboard SDK estan agrupades dins la llibreria anomenada dji_drone.h el qual es poden incloure directament en els programes.

Per gestionar el desenvolupament es disposa de un programa anomenat DJI Assistant. Aquest programa un cop es comunica per el port microUSB del drone amb el nostre ordinador ens permet actualitzar el firmware del drone, gestionar la comunicació de les funcions de desenvolupament i inclús poder crear una simulació de vol, el qual el no disposar del manifold i no poder realitzar tasques autònomes amb el drone ens ha permès poder utilitzar els experiments i tests utilitzats per la realització d'aquest programa.
DJI ofereix diferents modalitats per poder desenvolupar amb el Matrice100, utilitzant comandes de línux, QT, STM32 o ROS. Per aquest projecte s'ha escollit utilitzar ROS, pel fet d' oferint-n'hos una gran flexibilitat de programació, ens permet poder publicar les dades del drone utilitzant formats estàndards de missatges simplifica el problema i dona com a resultat una aplicació més robusta, i disposa d'una excel·lent comunitat de suport .
Avans de començar a explorar les funcions del DJI Onboard SDK a traves dels exemple de ROS, és necessari anar a través del procés d'activació, per això s'ha de connectar tot el sistema (PC, drone i comandament) com indica en la figura 11 i seguir els següents passos.

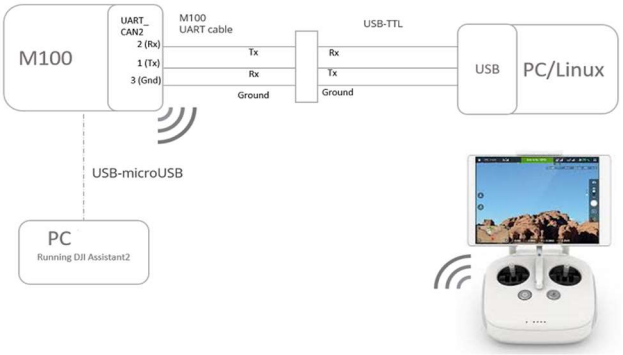

Figura 11. Communicació del sistema

Primer s'ha de crear una conta d'usuari a dji i activar la funció de desenvolupador, el qual serà necessari posar les dades personals de l'administració que en vol fer ús i a continuació es te que crear un projecte dins de la pàgina de DJI especificant el l'aplicació que es vol realitzar, per tal de obtenir una clau d'accés i contrasenya el qual seran necessaris per poder desenvolupar amb el drone.

A continuació s'ha d' habilitar, dins del programa DJI Assistant 2, la casella d' Enable API control i especificar la comunicació amb un baud rate de 230400, per poder tenir accés a la comunicació amb el drone a través del port UART.

Un cop s'han complert els primers preparatius per poder desenvolupar, el següent pas és copiar els fitxers del stack (conjunt de package) que disposa el repositori de DJI-Onboard-SDK-ROS i posar-los dins la carpeta src que hem creat dins el nostre espai de treball. Seguidament es te que modificar el fitxer central anomenat sdk_manifold.launch situat a la carpeta dji_sdk/launch/ de la següent manera.

A l' apartat anomenat drone_versions ens indica quin dispositiu estem utilitzant, en el nostre cas escrivim "M100", per referent-nos que volem utilitzar tot el dispositiu del matrice100, si hagues volgut comunicar-nos exclusivament amb el controlador de vol tindríem que indicar escrivint A3 o en el cas de tenir la versió de DJI matrice 600 ens tindríem que referir a ells amb M600.

En les comandes app_id i enc_key, tenim que introduir la id i contrasenya obtinguda en el procés de registre i creació de la aplicació que volem realitzar.

Finalment en l'apartat serial_name s'introdueix el nom del port per el que es connecta amb el drone, en el nostre cas ha sigut mitjançant el port /ttyUSB0 , i s'ha de comprovar que el baud rate estigui amb el valor 230400.

Un cop s'ha configurat amb els valors pertinents es pot passar a compilar, per tal de construir totes els dependències necessàries pel desenvolupament.

Entre els diferents paquets que disposa ONBOARD SDK ROS, hi ha el ROS core el qual publica totes les dades del drone en ROS tòpics i envia totes les comandes de control a la controladora de vol. Executant aquest amb les configuracions establertes, es desencriptarà la controladora.

6. Guidance SDK

Per gestionar el desenvolupament del dispositiu de visió Guidance, DJI ofereix uns arxius de desenvolupaments anomenats Guidance SDK, aquests permeten l'obtenció dels diferents sensors, i el tipus de transmissió d'aquest.
Guidance SDK suporta 2 protocols de comunicació, per port USB i per port UART.
El tenir la comunicació UART un ample de banda molt limitat per a l'ús de imatge, s'ha optat per l'ús de comunicació per USB.

Per gestionar els paràmetres dels sensors, DJI ofereix un programa anomenat DJI-Guidance, per habilitar l'ús de comunicació pel port USB, es te que polsar el quadre *Enable* de dins el submenú *API* de dins de la pestanya *DIY*. En aquesta finestra a part es pot seleccionar altres paràmetres de control com la velocitat de transferència de la informació i els sensors que es vol configurar, ja que tot i usar el bus USB, l'ample de banda no permet poder utilitzar-los tots simultàniament.

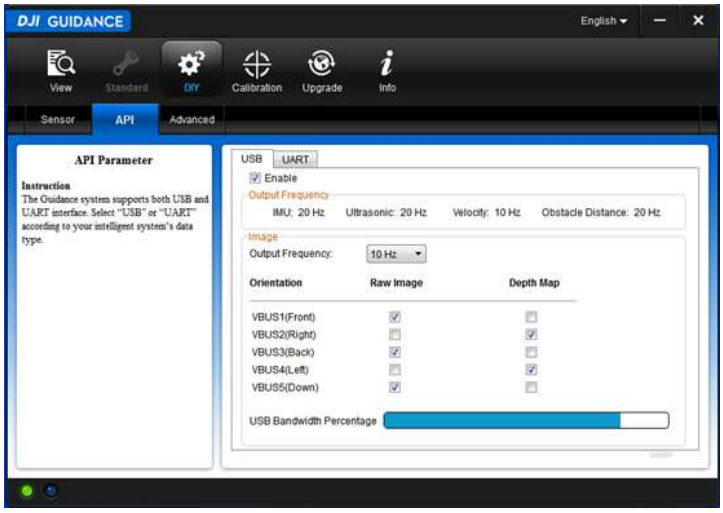

Figura 12. Programa DJI-Guidance

Guidance-SDK disposa d'un paquet de ROS que gestiona i publica les dades de la velocitat, distancia d'obstacle, inèrcia, les senyals dels ultrasons i la imatge de les càmeres. No obstant per aconseguir la comunicació en els paquets de ROS primerament s'ha d'aconseguir el privilegis *root* per a la interfície USB. Això s'aconsegueix aplicant una regle de dispositiuw, aplicant la següent comanda en un terminal.

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

7. Open CV

OpenCV és una llibreria de funcions de programació, principalment enfocats en la visió per computadora en temps real, va ser desenvolupat originalment per un centre de recercar de Intel a Nizhny Novgorod, més tard gestionada per Willow Garage i ara mantingut per Itseez, la qual ha sigut adquirida per Intel. La llibreria està construïda per múltiples plataformes i és de lliure accés sota la llicència de codi obert BSD.

Les noves versions de ROS no són compatibles directament amb OpenCV, per això es disposa de un stack de ROS, anomenat *vision opencv*.

Vision opencv ofereix la interacció de la llibreria OpenCV amb ROS. Aquest està separat en dos paquets, el *cv_bridge* que s'encarrega de fer de pont entre els missatges de ROS i l'OpenCV i el paquet image_geometry el qual és un conjunt de mètodes per tractar amb imatges i les geometries dels píxels.

El cv_bridge és el paquet, com s'ha comentat anteriorment, que s'encarrega de la comunicació entre ROS i openCV degut que aquests no son compatibles directament. cv_bridge defineix una imatge d'OpenCV que conté la imatge codificada conjuntament amb una capçalera per la lectura amb ROS. Aquesta imatge de tipus OpenCV conté exactament la informació de la mateixa manera que ho fa sensor_msgs/Image, per la qual cosa es pot convertir de un format a l'altre com es mostra la figura 13.

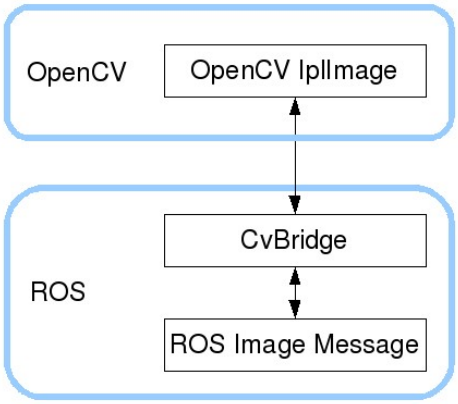

Figura. 13 Configuració OpenCV-ROS

Per altre banda el paquet *Image geometry*, que conte llibreries que simplifiquen d'interpretació de la geometria de les imatges utilitzant els paràmetres de sensor_msgs/CameraInfo. I les classes utilitzades a *image geometry* són escrites per utilitzar-se en missatges de retorn de *image/ CameraInfo*, similar a com ho fa el *cv_bridge*, per aquest motiu, per mantenir invariància en l'informació, per un segut de registres només es te accés a la lectura, podent realitzar canvis només quan s'apliquen les funcions específiques.

7.1 Instal·lació

És necessari intalar moltes dependències, per tal de poder utilitzar la lectura i escripture d'imatges, mostrar els resultats a la pantalla o utilitzar varies eines de visió, per dur-ho a terme s'ha hagut d'escriure i executar la següent comanda en un terminal.

```
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk libtbb-dev libeigen3-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev 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
```

Seguidament s'ha de de descarregar la llibreria OpenCV, en el nostre cas s'ha optat per la versió 2.4.9, que és la que recomana DJI

```
cd ~
wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/opencv-2.4.9.zip
unzip opencv-2.4.9.zip
cd opencv-2.4.9
```

El següent pas és compilar els arxius obtinguts, primer s'ha creat la carpeta *build* que s'usarà per dipositar els arxius resultants de la compilació, restringint els nodes que volem emprar utilitzant una serie de flags, i acavant la comanda amb dos punts per denominar que els arxius a compilar estan a la carpeta pare de la que acabem de crear.

```
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 ..
```

Seguidament es passa a la instal·lació de l'OpenCV 2.4.9.

```
make
sudo make install
```

Un cop finalitzat la instal·lació, s'ha de configurar l' OpenCV, primerament s'ha obert l'arxiu opencv.conf amb la comanda *sudo gedit /etc/ld.so.conf.d/opencv.conf* i s'introdueix la següent línia */usr/local/lib* a la part posterior i guardem l'arxiu.

Es configurar la llibreria mitjançant la comanda *sudo ldconfig i per* configurar les dependències s'obre el següent arxiu amb *sudo gedit /etc/bash.bashrc* i s'afegeix les següents dos línies el final de tot l'arxiu:

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

Per últim tanquem el terminal i reiniciem el ordinador.

8. Resolució

El desenvolupament del projecte s'ha centrat en la creació d'una aplicació de visió per computadora encarregada del reconeixement d'un objecte per tal de fer el seguiment d'aquest mitjançant el drone utilitzat. El programa com s'ha comentat prèviament s'ha desenvolupat mitjançant l'entorn de treball de ROS i amb l'ajuda de les llibreries d' OpenCV.

A l'inici el projecte estava centrat amb l'objectiu d'utilitzar la càmera central del drone, la Zenmuse X3, no obstant com s'ha comentat anteriorment, a causa d'un problema de trencament d'estoc, no ha sigut possible l'obtenció de la central de processament Manifold, encarregada tant del processament autònom, com de la descodificació i tractament del vídeo de la càmera. Per aquest motiu s'ha optat per a la realització del projecte utilitzant les càmeres dels sensors que incorpora el modul Guidance. I amb aquest nou sistema gestionar tota una interfície encarregada de realitzar l'aplicació desitjada, i facilitar d'aquesta manera un model útil per utilitzar-lo quan es disposi del dispositiu. Un cop s'ha tingut el drone amb tot el programari instal·lat i els productes pel desenvolupament activat, com es comenta en l'apartat d' Onboard SDK, s'ha gestionat un seguit de tests per a la realització d'un programa encarregat de desenvolupar l'aprenentatge, comprovació d'un correcte funcionament i la realització de l'objectiu final per la creació d'un sistema de seguiment.

Pel funcionament del disseny triat, primer s'han d'executar els dos paquets centrals, encarregats en els seus respectius mòduls, el de OnboardSDK que s'encarrega de la comunicació amb la controladora de vol i el de GuidanceSDK de la captació dels sensors del modul Guidance, amb aquest dos nuclis executats es passa a inicialitzar el nostre disseny el qual s'ha optat per la realització en dos programes independents comunicats entre ells mitjançant tòpics, un encarregat del tractament de la imatge i un segon dedicat a la automatització del drone, el qual formant una estructura com es mostra en la figura 14.

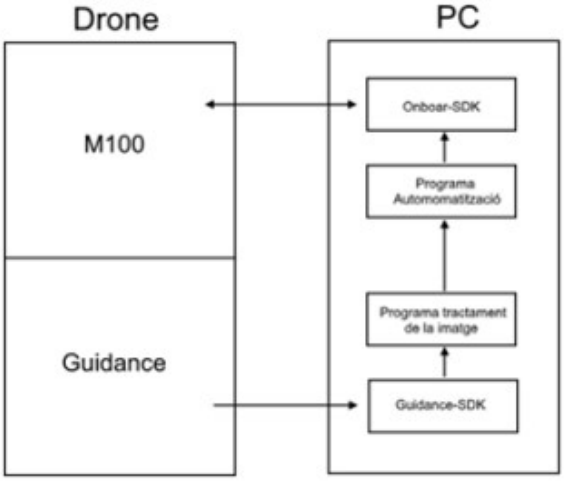

Figura 14. Estructura del projecte

Per a la inicialització del modul core de l'Onboard SDK, un cop tenim tot el programari configurat, és necessari executar-lo amb la comanda següent.

```
roslaunch dji_sdk sdk_manifold.launch
```

Amb aquesta comanda s'aconsegueix la comunicació bilateral entre el drone Matrice100 i l'ordinador, a partir del connexió UART-USB que s'ha descrit anteriorment, i s'encarrega de publicar totes les accions disponibles pel seu ús, permetent que altres programes independents simplement subscrivint-se/publicant-se en el tòpic pertinent puguin gestionar tot l'automatisme pertinent. A part ens mostra l'estat de cada acció que es produeix i de quina forma s'ha configurat la comunicació, es pot observar en la Figura 15, el resultat de la execució de la comanda dins del terminal.

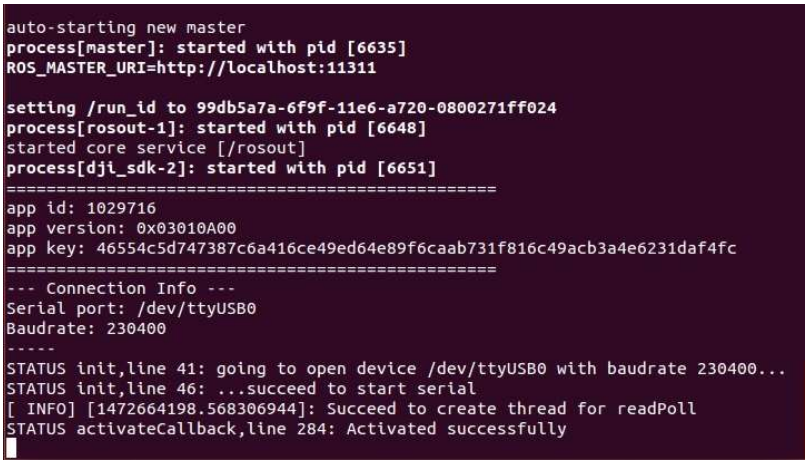

Figura 15. Onboard SDK core

I per a la inicialització del modul core del Guidance SDK es tindrà que executar la següent comanda.

```
rosrun guidance guidanceNode
```

Aquest executa la comunicació amb el Guidance a través del port USB, obtenint i publicant les dades de tots els sensors que estan presents, quan s'executa tot el programa en el terminal, es canvia la imatge de la càmera que es vol utilitzar, per a la prova que es realitza en el projecte s'ha utilitzat la càmera esquerra del grup de sensor3 connectat en el Guidance, aquest s'adquireix polsant la tecla 'D' dins del terminal, s'ha escollit aquest ja que amb tota la connexió del drone ens permetia fer els experiments de manera més òptima, es pot observar l'execució de la comanda en la Figura 16.

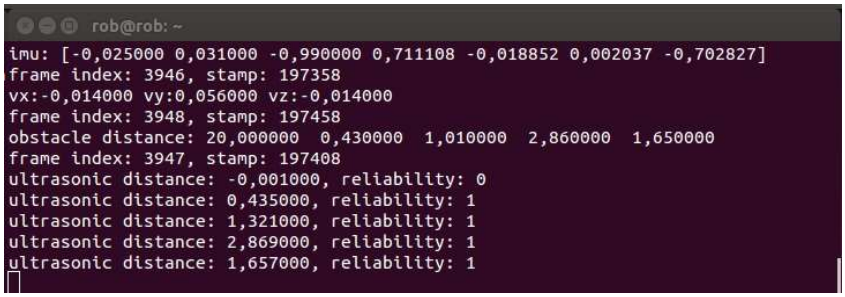

Figura 16. Guidance SDK core

El programa encarregat del tractament de la imatge, te com a objectiu l'obtenció de la imatge rebuda dels tòpics publicats per el programa core del GuidanceSDK, per tal de mitjançant un tractament de la imatge, obtenir les coordenades del centre de l'objecte que volem seguir. Un cop es te l'objecte identificat es publiquen aquestes dades mitjançant un tòpic per transmetre'ls al segon programa, l' automatització del vol. Pel tractament de la imatge i la identificació de l'objecte a seguir, per a realitzar aquestes tasques s'han utilitzat funcions de la llibreria OpenCV.

Pel programa de tractament de la imatge s'ha seguit un algoritme concret, descrit en la figura 17, que com es pot observar el primer que s'ha fet un cop obtinguda la imatge del sensors de visió i tenint el cas particular de comptar amb la imatge en escala de grisos es passa a aplicar un treshold de la intensitat d'aquesta, per tal de discretitzar la imatge en dos únics nivells. La imatge binaritzada resultant se li ha aplicat un filtratge, aquest s'ha realitzat en tres passos, el primer ha sigut un filtratge gaussià per suavitzar la imatge, aquest come el nom indica es tracta d'un filtre que la resposta en un impuls te la forma de una funció gaussiana, permetent reduir el soroll de la imatge, seguidament s'ha eliminat els objectes petits que pot contindre la imatge i per últim s'ha aplicat un filtre per emplenar els possibles forats que pot haver en el fons de la imatge.Un cop s'ha tractat la imatge es passa a la detecció de l'objecte a seguir, això s'ha fet amb la funció SimpleBlobDetector, el qual extreu els components connectats dins la imatge binaritzada mitjançant la localització dels seus contorns i en determina el seu centre, després uneix tots els grups trobats que estan en una distància propera i n'extreu el centre del grup. El programa seguidament n'extreu les coordenades de l'objecte i les publica en tòpics, per tal de poder ser utilitzades en el programa d'automatització. Els grups creats es filtren per tal d'habitar la identificació de formes del fons de la imatge que no pertanyen a l'objecte que volem identificar, per això es seleccionen només els que estan compresos dins d'una mida determinada.

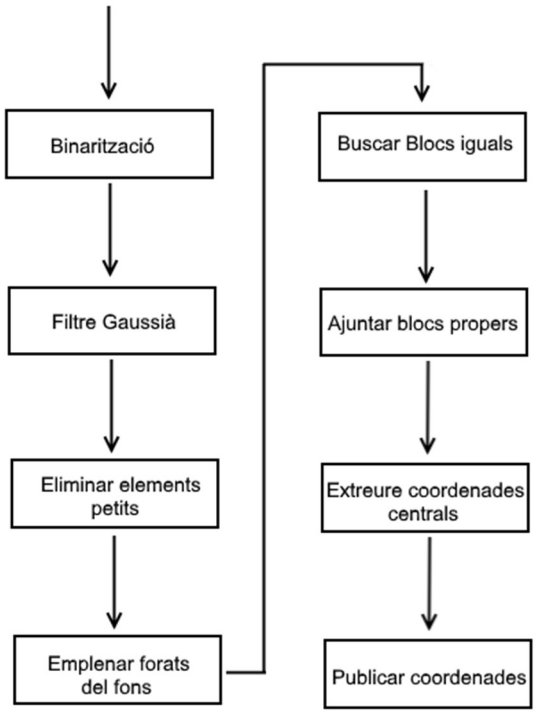

Figura.17 Algoritme tractament de la imatge

El segon programa, s'encarrega de l'automatització del drone, primer es subscriu en els tòpic que envien les coordenades del centre de l'objecte trobat, enviats per el programa de tractament de la imatge i a part també s'encarrega de publicar i rebre missatges dels tòpics que gestionen el control del drone, aquests proporcionats per el paquet core de l'OnboardSDK.

Aquest programa inclou varies funcions, les quals es poden seleccionar polsant el botó que ens indica una finestra de text, poden canviar d'aquesta manera el mode de funcionament entre el mode de comprovació i el mode de funcionament.

Les funcions encarregades del testeig o comprovació del funcionament del drone, s'encarrega de realitzar les tasques centrals del vol, aquest està compost per un seguit de accions seqüencials. L'algoritme de testeig comença demanant els permisos a la controladora de vol per gestionar les accions del drone, seguidament realitzar la tasca d'enlairament, un desplaçament horitzontal i finalment s'efectua la maniobra d'aterratge.

Per altre banda també disposa del programa central del projecte, el qual és la realització de l'automatització que acciona el control de l'aeronau a partir de les dades obtingudes del procés de visió per computadora del primer programa. L'algoritme principal del programa, com es pot observar en la figura 18 consta de l'esquema central igual que el programa de testeig, realitza les tasques d'obtenció del control del drone i l'acció d'enlairament, no obstant quan el drone està enlairat el seu desplaçament varia en funció de d'ubicació de l'objecte a detectar, permeten d'aquesta manera el seguiment d'aquest.

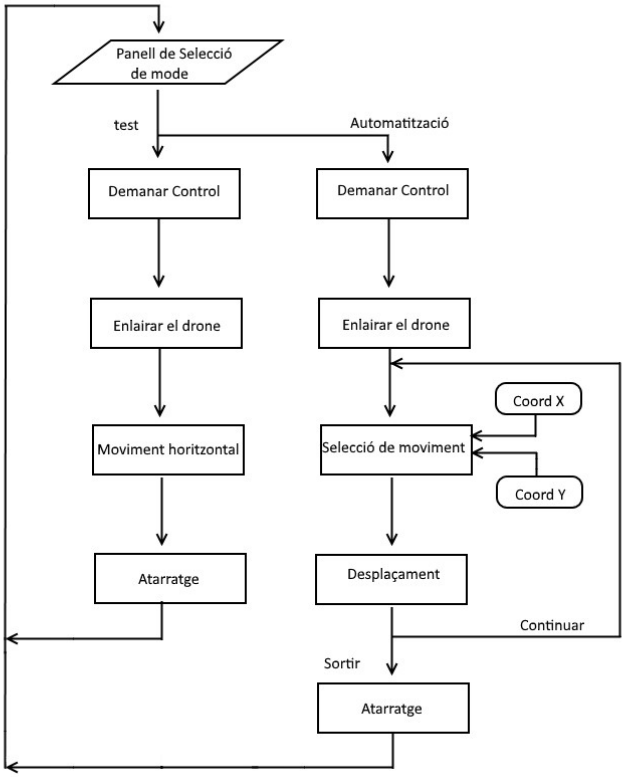

Figura18. Algoritme Automatització

La direcció del moviment s'ha realitzat discretitzant a on està disposat l'objecte, es a dir si el centre de l'objecte trobat per la càmera es troba per sobre d'un límit el drone enviarà la comanda de desplaçar-se en aquella direcció. A més a més s'ha creat una zona de histèresis central per reduir les oscil·lacions de vol.

A causa de falta del dispositiu Manifold com s'ha comentat, no s'han pogut realitzar comprovacions de vols reals, per la qual cosa tots els tests s'han tingut que efectuar a partir del simulador de vol que incorpora el programa d'assistència de vol DJIAssistant2.

S'ha comprovat el correcte funcionament tant de la comunicació entre els dispositius com de l'execució del programa realitzat, donant a terme interconnexió entre els diversos aplicacions com es mostra en la Figura 19, que fent proves en un format més òptim, tenint en compte en les característiques que s'han presentat en el projecte, s'ha pogut obtenir un model per tal de realitzar la tasca de seguiment d'un objecte, i permetent d'aquesta manera un fàcil trasllat per a poder realitzar, en projectes posteriors, una aplicació semblant utilitzant la càmera d'alta resolució i el control de vol autònom.

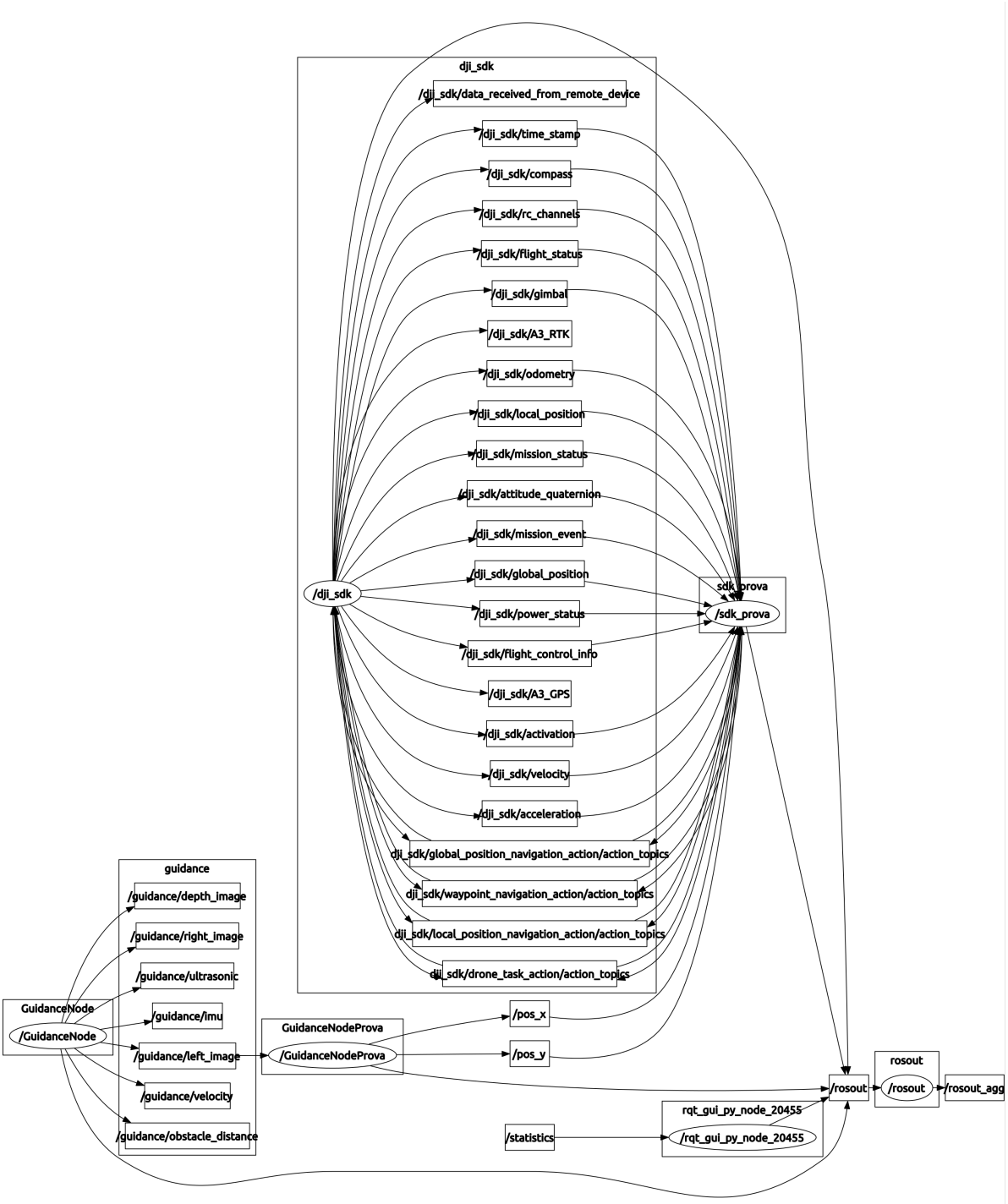

Figura.19 Interconexió nodes ROS

9. Resum del pressupost

Segons es detalla en el Document nº 4: Pressupost, es pot determinar que el cost del present projecte, tenint en compte el cost material necessari per a la construcció el cost derivat dels serveis humans per tal de portar-lo a terme, ascendeix a nou mil cent catorze euros amb trenta-tres cèntims (9.114,33 €), sense IVA .

10. Conclusions

En els apartats d'aquesta memòria s'ha exposat el desenvolupament d'aquest projecte, el qual ha complert amb l'objectiu principal de realitzar en temps real el seguiment d'un objecte mitjançant un drone, el qual complís amb les característiques que es proposaven.

Per fer-ho s'ha dissenyat un sistema, compres per varies aplicacions interconnectades entre elles, utilitzant l'eina de desenvolupament ROS. El núcli principal de les aplicacions han sigut un programa encarregat del tractament i indentificació de l'objecte que es vol fer el seguiment i encarregat d'envia les coordenades d'aquests i un segon programa que discretitza la zona de l'objecte i mou el dron per intentar sempre posicionar-se sobre de l'objecte.

Tot i haver creat una aplicació que complia amb els objectius del seguiment de un objecte, mitjançant el drone que s'ha triat, de cares a futurs projectes s'hauria d'adaptar-lo per la funcionalitat dins del dispositiu Manifold, per tal d'automatitzar el projecte i poder fer ús de la càmera estàndard Zenmuse X3, el qual permetria poder treballar amb resolucions més grans i disposar de la imatge en color el que milloraria la discretització de l'objecta a seguir.

Robert Gifreu Pons
Graduat en Enginyeria Electrònica Industrial i Automàtica

Girona, 30 agost de 2016

11. Relació de documents

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

12. Bibliografia

Clearpathrobotics, ROS Tutorials, (https://www.clearpathrobotics.com/assets/guides/ros/ , 13 de juliol de 2016)

DJI Onboard, DJI(https://developer.dji.com/onboard-sdk/documentation/quick-start/index.html , 8 de juny de 2016)

DJI Guidance, DJI (https://developer.dji.com/guidance-sdk/documentation/quick-start/ index.html, 12 de juliol de 2016)

MARTINEZ, AARON i FERNANDEZ, ENRIQUE. Learning ROS for Robotics Programming. Packt publishing, 1a Edició. Birmingham ,2013.

OpenCV, OpenCV Org, (http://www.opencv.org/, 15 d'agost de 2016)

Opencv-srf , OpenCV Lessons(http://opencv-srf.blogspot.com.es/p/opencv-c-tutorials.html, 25 de juny de 2016)

ROS, Ros, (http://www.ros.org/wiki/, 5 de juliol de 2016)

13. Glossari

ESC: Electronic Speed Control.

GPS: Global Positioning System

LTS: Long Term Support

ROS: Robotic Operational System.

USB: Universal Serial Bus.

A Codi Dispositius

En aquest Annex s'incorporaran els dos codis de programa que s'han fet creat per a la realització d'aquest projecte.

A.1 Programa Automatització

```
#include <ros/ros.h>
#include <stdio.h>
#include <dji_sdk/dji_drone.h>
#include <cstdlib>
#include <actionlib/client/simple_action_client.h>
#include <actionlib/client/terminal_state.h>
#include <std_msgs/Float32.h>
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

ros::Subscriber subx;
ros::Subscriber suby;

float x;
float y;

using namespace DJI::onboardSDK;
using namespace cv;
using namespace std;

static void Display_Main_Menu(void)
{
    printf("\r\n");
    printf("+---------------------- <Programa de Prova > ----------------------+\n");
    printf("| [r] Test1                  | [s] Test2                |\n");
    printf("| [t] Test3                  | [0] Mission Hotpoint Download    |\n");

    printf("+-------------------------------------------------------------------+ \n");
    printf("input r/s/t etc..then press enter key\r\n");
    printf("use `rostopic echo` to query drone status\r\n");
    printf("----------------------------------------\r\n");
    printf("input: ");
}
```

```
void chatterCallback(const std_msgs::Float32::ConstPtr& subx)
{
       x = subx->data;
}

void chatterCallback2(const std_msgs::Float32::ConstPtr& suby)
{
       y = suby->data;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "sdk_prova");
    ROS_INFO("sdk_service_client_test");
    ros::NodeHandle nh;

   subx = nh.subscribe("pos_x", 1000, chatterCallback);
   suby = nh.subscribe("pos_y", 1000, chatterCallback2);

    ros::Rate loop_rate(10);
    int main_operate_code = 0;
    int temp32;

    bool valid_flag = false;
    bool err_flag = false;

    DJIDrone* drone = new DJIDrone(nh);
    ROS_INFO("I heard: [%f]", x);

   Display_Main_Menu();

    while(1)
    {

       ros::spinOnce();
        temp32 = getchar();
        if(temp32 != 10)
        {
            if(valid_flag == false)
            {
                main_operate_code = temp32;
                valid_flag = true;
            }
            else
            {
                err_flag = true;
```

```
            }
            continue;
        }
        else
        {
            if(err_flag == true)
            {
                printf("input: ERROR\n");
                Display_Main_Menu();
                err_flag = valid_flag = false;
                continue;
            }
        }
//Comença el codi!!

        switch(main_operate_code)

        {
             case 'r':

                    /* request control ability*/
                     drone->request_sdk_permission_control();
                    sleep(1);
                    /* take off */
                    drone->takeoff();
                    sleep(8);
                    /* Moures */
                    for(int i = 0; i < 100; i ++)
                    {
                           if(i < 90)
                                  drone->attitude_control(0x40, 8, 0, 0, 0);
                           else
                                  drone->attitude_control(0x40, 0, 0, 0, 0);
                                  usleep(20000);
                    }
                    sleep(1);
                    /* landing*/
                    drone->landing();
                    sleep(8);
                    break;

             case 's':

                    /* request control ability*/
                    drone->request_sdk_permission_control();
                    sleep(1);
```

```
                    /* take off */
                    drone->takeoff();
                    sleep(8);
                    /* Moures */
                    for(int i = 0; i < 100; i ++)
                    {
                           if(i < 90)
                                  drone->attitude_control(0x40, -8, 0, 0, 0);
                           else
                                  drone->attitude_control(0x40, 0, 0, 0, 0);
                                  usleep(20000);
                    }
                    sleep(1);
                    /* landing*/
                    drone->landing();
                    sleep(8);
                    break;

             case 't':

                    /* request control ability*/
                    drone->request_sdk_permission_control();
                    sleep(1);
                    /* take off */
                    drone->takeoff();
                    sleep(8);
                    /* Moures */

                    //while(cv::waitKey(1) != 113){ // 27 = ascii value of ESC
                    //for(int z = 0; z < 999999999999999999; z ++){
                    for(;;)
                    {   //line 34: Port closedERROR sendData
                           ros::spinOnce(); //Will call all the callbacks waiting to
be  called at that point in time.
                           if (x> 140)
                           {
                           //if( waitKey(1) == 113 ) break;
                           if (y> 140)
                           {
                                  for(int i = 0; i < 100; i ++)
                                  {
                                  if(i < 90)
                                         drone->attitude_control(0x40, 1, 1, 0, 0);
                                  else
                                         drone->attitude_control(0x40, 0, 0, 0, 0);
                                         usleep(20000);
```

```
        }
    }
    else if (y<120)
    {
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
            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 < 100; i ++)
        {
        if(i < 90)
            drone->attitude_control(0x40, 1, 0, 0, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
            usleep(20000);
        }
    }
    }
    else if (x< 120)
    {
    //if( waitKey(1) == 113 ) break;
    if (y> 140)
    {
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
            drone->attitude_control(0x40, -1, 1, 0, 0);
        else
            drone->attitude_control(0x40, 0, 0, 0, 0);
            usleep(20000);
        }
    }
    else if (y<120)
    {
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
            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 < 100; i ++)
        {
        if(i < 90)
                drone->attitude_control(0x40, -1, 0, 0, 0);
        else
                drone->attitude_control(0x40, 0, 0, 0, 0);
                usleep(20000);
         }
}
}
else
{
//if( waitKey(1) == 113 ) break;
if (y> 140)
{
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
                drone->attitude_control(0x40, 0, 1, 0, 0);
        else
                drone->attitude_control(0x40, 0, 0, 0, 0);
                usleep(20000);
        }
}
else if (y<120)
{
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
                drone->attitude_control(0x40, 0, -1, 0, 0);
        else
                drone->attitude_control(0x40, 0, 0, 0, 0);
                usleep(20000);
         }
}
else
{
        for(int i = 0; i < 100; i ++)
        {
        if(i < 90)
```

```
                                        drone->attitude_control(0x40, 0, 0, 0, 0);
                                  else
                                        drone->attitude_control(0x40, 0, 0, 0, 0);
                                        usleep(20000);
                                  }
                            }
                             //usleep(20000);
                            }
                            //if( waitKey(1) == 113 ) break;
                             sleep(1);
                            //} while (fi != '.');
                      }
                      printf("automatització finalitzada");
                      sleep(8);
                      /* landing*/
                      drone->landing();
                      sleep(8);
                      break;

               case '0':
                      /* SDK version query*/
                      drone->check_version();

                      default:
                            break;
        }
        main_operate_code = -1;
        err_flag = valid_flag = false;
        Display_Main_Menu();
       loop_rate.sleep();
}
return 0;
// while (ros::ok());
}
```

A.2 Programa Tractament de la Imatge

```
#include <std_msgs/Float32.h>
#include <stdio.h>
#include <string.h>
#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/opencv.hpp>
```

```
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

ros::Subscriber left_image_sub;
ros::Publisher posx_pub;
ros::Publisher posy_pub;

using namespace cv;
#define WIDTH 320
#define HEIGHT 240

std_msgs::Float32 posx ;
std_msgs::Float32 posy ;

void like(const sensor_msgs::ImageConstPtr& left_img)
{

float antx = 120;
float anty = 120;
cv_bridge::CvImagePtr cv_ptr;
try {
       cv_ptr = cv_bridge::toCvCopy(left_img, sensor_msgs::image_encodings::MONO8);
    }
catch (cv_bridge::Exception& e)
       {
        ROS_ERROR("cv_bridge exception: %s", e.what());
        return;
       }

threshold(cv_ptr->image,cv_ptr->image,50,255,CV_THRESH_BINARY_INV);

erode(cv_ptr->image,cv_ptr->image,getStructuringElement(MORPH_ELLIPSE,Size(5,5)));
dilate(cv_ptr->image,cv_ptr->image,getStructuringElement(MORPH_ELLIPSE,Size(5,5)));

dilate(cv_ptr->image,cv_ptr->image,getStructuringElement(MORPH_ELLIPSE,Size(5,5)));
erode(cv_ptr->image,cv_ptr->image,getStructuringElement(MORPH_ELLIPSE,Size(5,5)));

cv::GaussianBlur(cv_ptr->image, cv_ptr->image, Size(3, 3), 0, 0);

cv::SimpleBlobDetector::Params params;
params.minDistBetweenBlobs = 10.0;  // minimum 10 pixels between blobs
params.filterByArea = true;         // filter my blobs by area of blob
params.minArea = 500.0;              // min 20 pixels squared
params.maxArea = 8000.0;             // max 500 pixels squared
params.filterByColor = true;
```

```
params.blobColor = 255;

SimpleBlobDetector Blob(params);
std::vector<cv::KeyPoint> myBlobs;
Blob.detect(cv_ptr->image, myBlobs);

cv::Mat blobImg;
cv::drawKeypoints(cv_ptr->image, myBlobs, blobImg);
cv::imshow("image", cv_ptr->image);
cv::imshow("image", blobImg);

for(std::vector<cv::KeyPoint>::iterator blobIterator = myBlobs.begin(); blobIterator
!= myBlobs.end(); blobIterator++)
{
       std::cout << "size of blob is: " << blobIterator->size << std::endl;
       std::cout << "point is at: " << blobIterator->pt.x << " " << blobIterator-
       >pt.y << std::endl;

//Prova de modul de discretització de diferents objectes

       if (blobIterator->pt.x <= (posx.data+2) && blobIterator->pt.x >= (posx.data-
2)  ){posx.data = blobIterator->pt.x;}
       if (blobIterator->pt.y <= (posy.data+2) && blobIterator->pt.y >= (posy.data-
2)  ){posy.data = blobIterator->pt.y;}

}
posx_pub.publish(posy);
posy_pub.publish(posx);
cv::waitKey(1);
}

int main(int argc, char** argv)
{
     ros::init(argc, argv, "GuidanceNodeProva");
     ros::NodeHandle nh;
    left_image_sub  = nh.subscribe("/guidance/left_image",  10, like);
    posx_pub = nh.advertise<std_msgs::Float32>("pos_x", 1000);
    posy_pub = nh.advertise<std_msgs::Float32>("pos_y", 1000);
              ros::Rate loop_rate(10);
     ROS_INFO("%f", posx.data);
     ROS_INFO("%f", posy.data);
        ros::spin();
 while (ros::ok())
   return 0;
}
```