EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Eng. Tècn. Informàtica de Sistemes. Pla 2001

Títol: Disseny d'un comportament en ROS per a mantenir la posició d'un vehicle autònom submarí a partir de referències visuals

Document: Memòria

Alumne: ÓSCAR SIMÓN CASTILLO

Director/Tutor: MARC CARRERAS PÉREZ / NARCÍS PALOMERAS ROVIRA
Departament: Arquitectura i Tecnologia de Computadors
Àrea: A.T.C

Convocatòria (mes/any): setembre / 2011

Índex

Índex de Figures

CAPÍTOL 1

1. Introducció

1.1. Antecedents

En el Centre d'Investigació en Robòtica Submarina (CIRS) de la Universitat de Girona es disposa de diferents robots submarins els quals utilitzen una arquitectura software anomenada Component Oriented Layered-based Architecture for Autonomy (COLA2), la qual ha estat desenvolupada per estudiants i professors del mateix centre. Per tal de fer aquesta arquitectura més accessible per a professors i estudiant d'altres centres la COLA2 s'està adaptant al Robot Operative System (ROS [**1**]) que és un framework genèric per al desenvolupament d'aplicacions amb robots.

Aquest projecte pretén dissenyar un comportament per al robot Girona500 que estigui desenvolupat dins la versió ROS de l'arquitectura COLA2. Tot seguit es farà un repàs de les motivacions per a aquest projecte, els propòsits i els objectius a assolir.

1.2. Motivacions i propòsits

Com a estudiant d'ETIS, al 3er curs les assignatures que més interès van inspirar al autor d'aquest projecte van ser Robòtica i Visió per Computador. Per sort a la Universitat de Girona té un dels millors Laboratoris de Robòtica Submarina d'Europa.

L'autor va demanar si hi havia propostes de projectes i hi havia la proposta de treballar amb la càmera de l'últim robot del laboratori, el Girona500, un projecte en el que estaven implicades més universitats i que s'havia de treballar amb el framework ROS per tal de que el que es desenvolupes fos el més universal possible.

Així el repte de treballar amb la càmera tant a nivell de hardware com amb la part de software va cridar l'atenció a l'autor d'aquest projecte.

A més, la proposta de dissenyar i implementar un mòdul per mantenir una posició per a un robot va ser molt interesant ja que, per a un robot submarí, el fet de mantenir-se quiet a una posició es molt important per a quan estan realitzant intervencions a sota l'aigua. El fet de que es faci tant amb referencies

visuals com amb dades de navegació farà que el sistema sigui molt robust i una proposta de treball molt interesant.

1.3. Objectius

L'objectiu del projecta consisteix en implementar un comportament per al robot Girona500 desenvolupat a la versió ROS de l'arquitectura COLA2 que ens permeti realitzar un control en posició del robot utilitzant informació visual d'una càmera o procedents del mòdul de navegació. Per aconseguir això s'haurà de:

- Desenvolupar un driver per la càmera del robot.
- Implementar el software que processi les imatges de la càmera per ubicar el robot en posició i determinar les correccions que s'hagin de transmetre als sistemes de navegació i control dels motors del robot per a realitzar la seva tasca.
- Desenvolupar un comportament que faci moure el robot en funció de l'error en posició detectat per el software de tractament d'imatges o dades de navegació.

1.4. Abast

L'abast del projecte consistirà en entendre com funcionen l'arquitectura COLA2 per tal de preparar un driver per la càmera en ROS i després implementar un sistema que per mitja de les imatges obtingudes de la càmera i dades de navegació es pugui ubicar el robot en una posició determinada i mantenir aquella posició a partir de referències visuals captades per la càmera del mateix robot o amb informació obtinguda del mòdul de navegació.

Per fer-ho s'aplicaran diverses tècniques de visió per computador com la de Template Matching o SURF d'imatges i s'estudiarà el seu rendiment i quina serà més adequada segons cada situació.

1.5. Estructura del document

L'estructura d'aquest document segueix la Guia dels projectes/treballs de final de carrera de les enginyeries informàtiques. A continuació es presenta una breu descripció de cada Capítol d'aquesta memòria.

- CAPÍTOL 2, Estudi de viabilitat. En aquest Capítol s'explica perquè l'autor d'aquest projecte troba viable el desenvolupament del projecte.

- CAPÍTOL 3, Metodologia. El aquest Capítol s'explica d'una manera resumida la metodologia emprada per al desenvolupament del projecte.

- CAPÍTOL 4, Planificació: En aquest Capítol s'exposa la temporització de les diverses fases del projecte

- CAPÍTOL 5, Marc de treball i Conceptes previs. En aquest Capítol s'explica l'entorn de treball i el vehicle amb el que es treballa.

- CAPÍTOL 6, Requisits del sistema. En aquest Capítol s'exposa el requisits funcionals i no funcionals que haurà de complir el sistema.

- CAPÍTOL 7, Estudis i decisions. En aquest Capítol s'exposen les eines de treball utilitzades i s'expliquen els conceptes bàsics de l'arquitectura COLA2 i ROS.

- CAPÍTOL 8, Anàlisis i Disseny del sistema. En aquest Capítol es detalla la anàlisis i el disseny per passos del mòduls implementats.

- CAPÍTOL 9, Implementació i proves. En aquest Capítol es remarquen les parts més importants de la implementació del sistema i s'exposen proves del seu correcte funcionament.

- CAPÍTOL 10, Resultats. En aquest Capítol es fa un estudi del rendiment i de la qualitat que ofereix el sistema amb jocs de proves adequats a diferents situacions en les que es pugi fer servir el sistema.

- CAPÍTOL 11, Conclusions. En aquest Capítol es fa una recapitulació de les conclusions que es poden extreure després d'acabar el projecte.

- CAPÍTOL 12, Treball futur. En aquest Capítol es tanca la memòria fen propostes dels possibles treballs futurs que poden realitzar-se un cop finalitzat aquest projecte.

CAPÍTOL 2

2. Estudi de viabilitat

2.1. Coneixements Previs

Com ja s'ha dit anteriorment, l'estudiant va trobar les assignatures de Robòtica i de Visió per Computador les més interesants, per tant ja hi ha tant coneixements previs com un interès personal per el tema. També es pot fer una idea de les dificultats que suposa treballar en aquest àmbit.

També s'ha tingut en compte de que en departament en el que es treballa hi ha gent amb amplis coneixement sobre la matèria als quals es pot recorre per a resoldre els temes que per a l'estudiant suposin un problema fora del seu abast.

2.2. Elecció de llibreria

La llibreria escollida per fer la majoria de la feina va ser l'OpenCV [2]. Es va escollir aquesta llibreria bàsicament per dues raons:

- Al departament ja es feia servir anteriorment i satisfeia les necessitats dels projectes desenvolupats i les del a desenvolupar.
- ROS és un projecte desenvolupat per l'equip Willow Garage. Aquest equip també l'encarregat de mantenir les OpenCV i per tant estan perfectament integrades a ROS i ja hi ha moltes eines de treball que estan implementades dintre de ROS amb OpenCV, cosa que facilitarà la integració.

Per tant és una bona elecció perquè esta espatllada tant per l'equip de treball del departament com pel framework de l'arquitectura en la que es treballarà.

CAPÍTOL 3

3. Metodologia

3.1. Metodologia incremental

En aquest projecte l'autor ha seguit una metodologia incremental [3].

La metodologia incremental consisteix en fixar un objectiu, passar per les fases d'anàlisis, disseny, implementació i test i un cop s'han assolit totes, es torna a començar incrementant en un pas l'objectiu fixat (Figura 1).

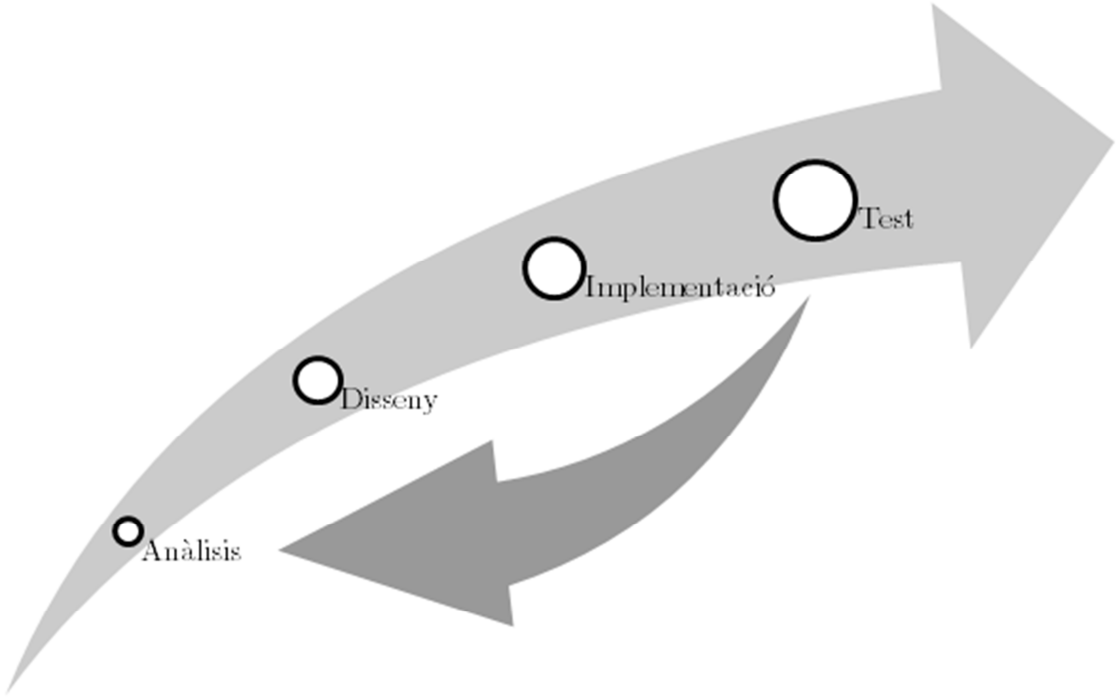


Figura 1: Metodologia incremental

Per anar fent els increments s'han fet revisions amb els tutors cada dues setmanes si no ha fet falta reunir-se abans per a poder avançar més ràpidament o resoldre algun dubte.

CAPÍTOL 4

4. Planificació

4.1. Organització de les setmanes

En aquesta Secció s'explicarà la planificació de la fenia dividida per setmanes.

4.1.1. 24/01 – 30/01 : Plantejament del Projecte

Descripció: Es va fer un plantejament inicial del projecte.

4.1.2. 31/01 - 06/02 : Estudi del COLA2

Descripció: Llegir la documentació i comprendre el disseny i funcionament de l'arquitectura COLA2.

4.1.3. 07/02 – 13/02 : Estudi del ROS

Descripció: Llegir la documentació de ROS disponible a la web.

4.1.4. 14/02 – 20/02 : Instal · lació de l'Entorn COLA2 i ROS

Descripció: Preparació de tot el software necessari per a treballar amb el l'arquitectura COLA2 i el framework ROS.

4.1.5. 21/02 – 13/03 : Disseny i implementació del Node de control de correccions de posició

Descripció: Dissenyar i implementar un node que amb enviï les correccions adients al sistema de navegació per mantenir-se en una posició determinada i desitjada.

Test: Provar el correcte funcionament de node.

4.1.6. 14/03 – 27/03 : Disseny i implementació del Pont COLA2-ROS

Descripció: Disseny i Implementació d'un node en ROS per enviar missatges TCP cap a un component de la versió anterior de COLA2 que no funcionava utilitzant el framework de ROS i viceversa.

Test: Enviar i rebre dades de navegació a les dues bandes. Experiments Conjunts amb la universitat Jaume I (UJI) i la universitat de les Illes Balears (UIB).

4.1.7. 28/03 – 10/04 : Disseny i implementació del driver per la càmera

Descripció: Estudiar les possibles solucions per a implementar el driver de la càmera i escollir-ne la més viable. Dissenyar i implementar el driver per la càmera.

Test: Comprovació de que es capturaven imatges de la càmera del robot.

4.1.8. 11/04 – 1/05 – { del 17/04 al 25/04 Setmana Santa } : Disseny i implementació Node per calibrar la càmera

Descripció: Dissenyar i implementar un Node en ROS per a calibrar la càmera del robot.

Test: provar el correcte funcionament de l'algorisme.

4.1.9. 2/05 – 22/05 : Disseny i implementació del Node de tractament d'imatges implementant Template Matching

Descripció: Estudiar funcionament del Template Matching i dissenyar i implementar el Node que tractes les imatges amb aquesta tècnica.

Test: Fer jocs de proves amb la càmera per comprovar el correcte funcionament de l'algorisme.

4.1.10. 23/05 – 12/06 : Disseny i implementació del Node de tractament d'imatges implementant SURF d'imatges

Descripció: Estudiar el funcionament del SURF d'imatges i dissenyar i implementar el Node que tractes les imatges amb la tècnica.

Test: Fer jocs de proves amb la càmera per comprovar el correcte funcionament de l'algorisme.

4.1.11. 13/06 – 03/07 : Disseny i implementació del Node de processat d'imatges

Descripció: Unificar els nodes de Template Matching i SURF per tal de tenir un únic node per al tractament d'imatges.

Test: Provar que el node fes la mateixa feina que els dos nodes per separat però en un de sol.

4.1.12. 04/07 – 24/07 : Proves i repàs dels codis

Descripció: Fer proves per corroborar el correcte funcionament de tot el sistema i repassar tota la feina elaborada.

4.1.13. 25/08 – 07/09 : Memòria i retocs finals

Descripció: Escriptura de la memòria i donar les ultimes pinzellades als petits detalls del projecte.

4.2. Calendari amb Gantt Project (Figura 2)

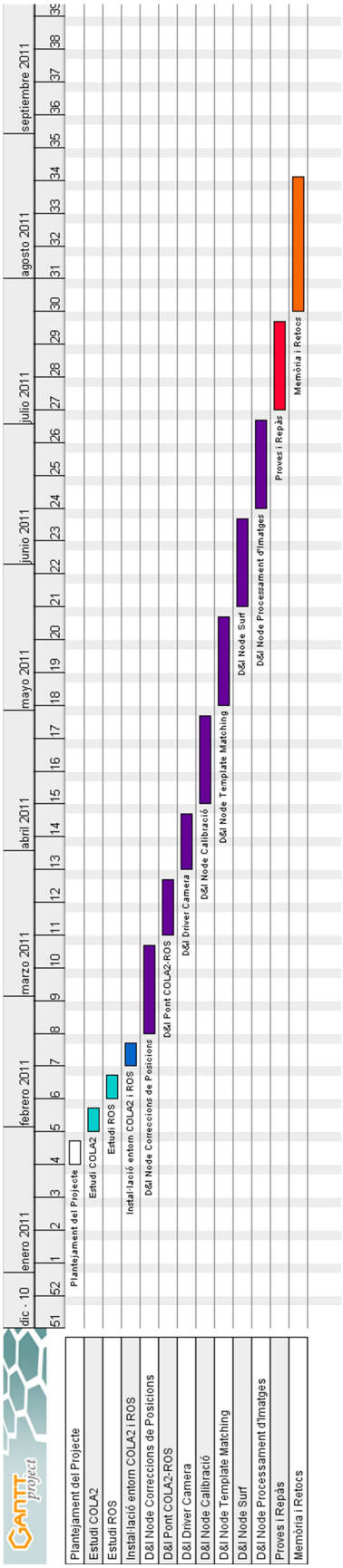


Figura 2: Calendari amb Gantt Project

CAPÍTOL 5

5. Marc de treball i Conceptes previs

5.1. Entorn de treball

L'entorn de treball serà el CIRS [4].

Figura 3: Centre CIRS

El CIRS (Centre de Recerca en Robòtica Submarina, Figura 3) és l'edifici que alberga el Laboratori de Robòtica Submarina de Visió per Computador i grup de recerca de Robòtica de la Universitat de Girona. Aquest edifici està ubicat al Parc Científic i Tecnològic de la Universitat. El complex està compost per dos edificis principals. Un (a l'esquerra, Figura 3) conté laboratoris, oficines i tallers, i l'altre (a la dreta, Figura 3) conté un tanc d'aigua, i una sala de control de i supervisió amb vista directa al tanc d'aigua.

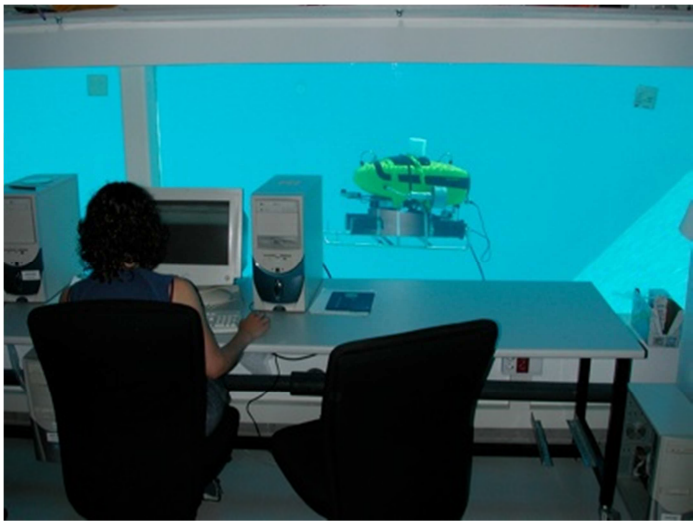


Figura 4: Sala Control Piscina

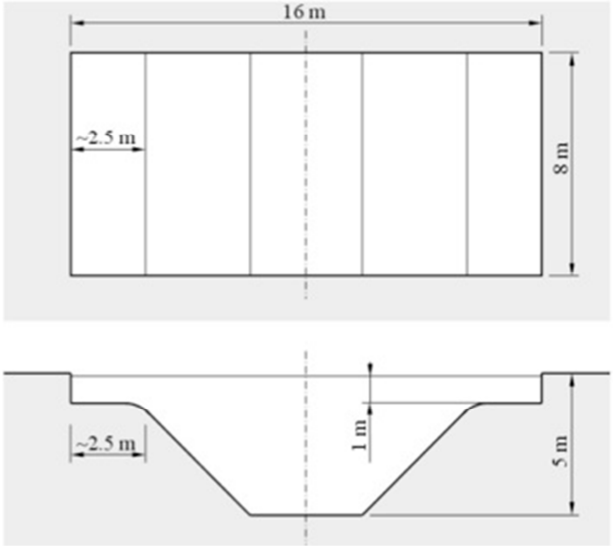


Figura 5: Mesures Piscina

El fet de disposar d'una sala de control sota l'aigua amb vista directa al tanc d'aigua (Figura 4) fa més senzill realitzar experiments i observar els seus resultats. Les dimensions dels tancs d'aigua són 16x8 m2 amb una profunditat que varia d'1 a 5 m (Figura 5).

Figura 6: Oficina Piscina

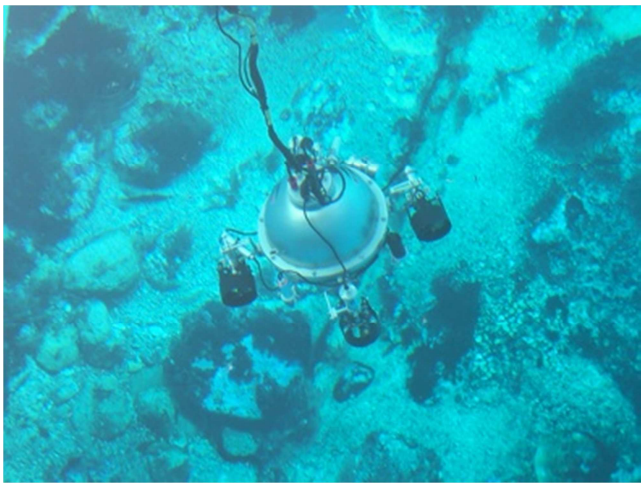


Figura 7: URIS UUV

La imatge de esquerra mostra una de les oficines ubicades a l'edifici contigu al tanc d'aigua (Figura 6). La imatge dreta mostra URIS UUV (Figura 7) en navegació sobre un pòster posat a la part inferior del tanc d'aigua. Aquest pòster ens permeten dur a terme experiments de control basat en la visió (mosaics d'imatges, el seguiment de cable, ...) en condicions de laboratori.

5.2. Vehicle de treball

El vehicle amb el que es treballarà serà un AUV, en concret el Girona500.

5.2.1. Autonomous Underwater vehicle (AUV)

Un vehicle submarí autònom (AUV) és un robot que viatja sota l'aigua sense necessitat d'un operador. Els AUVs formen part d'un grup més ampli de sistemes submarins no tripulats coneguts com *Unmanned Underwater Vehicles*, una classificació que inclou els no autònoms *Remotely Operated Underwater Vehicles (ROV)* - controlats i alimentat des de la superfície per un operador / pilot a través d'un cordó umbilical o l'ús de control remot. En aplicacions militars els AUVs són més sovint coneguts simplement com *Unmaded Undersea vehicles (UUVs).*

5.2.2. Girona500

Figura 8: Girona500

El Girona500 (Figura 8) és un AUV (Autonomous Underwater Vehicle), dissenyat per a navegar a una profunditat màxima de fins a 500 metres. El vehicle es compon d'un marc d'alumini que suporta tres cascos en forma de torpede de 0,3 m de diàmetre i 1,5 m de longitud, així com altres elements com els propulsors. Aquest disseny ofereix un bon comportament hidrodinàmic i un gran espai per a l'allotjament d'equips, mantenint una mida compacte que permet operar el vehicle de petites embarcacions. Les dimensions totals del

vehicle són d'1 m d'alçada, 1 m d'ample, 1,5 m de longitud i un pes de menys de 200 kg. Els dos cascos superiors contenen l'escuma de flotació i la carcassa de l'electrònica i són de flotabilitat positiva, mentre que l'inferior conté els elements més pesats com les bateries i la càrrega útil. Aquesta particular disposició dels components separa el centre de gravetat del centre de flotabilitat per uns 11 cm, que és significativament major que la trobada en un disseny de forma de torpede típic (Figura 9). Això proporciona al vehicle una estabilitat passiva de capcineig i balanceig, el que és adequat per a les tasques que es beneficiaran d'una plataforma estable, com les intervencions o estudis d'imatges.

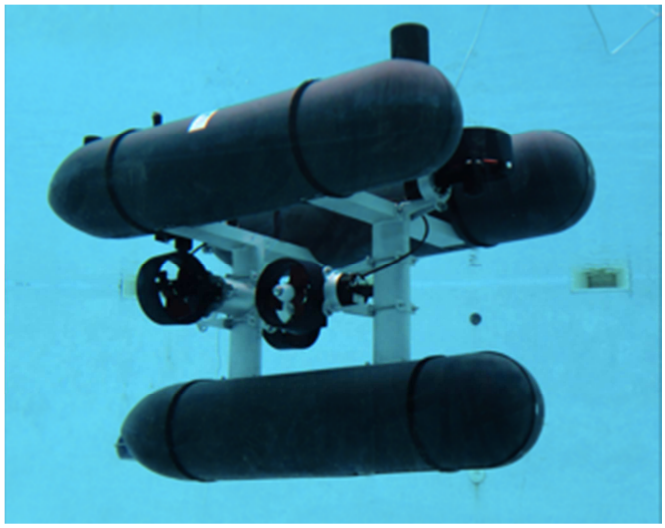


Figura 9: Girona500 sota l'aigua

La característica més notable del Girona500 és la capacitat poder-lo configurar per a diferents tasques. En la seva configuració estàndard, el vehicle està equipat amb els típics sensors de navegació (DVL, AHRS, manòmetre i USBL) i l'equip bàsic de supervivència (sonar, sides can sonar, videocàmera un sensor de velocitat). A més d'aquests sensors, gairebé la meitat del volum del casc inferior està reservada per a equips de càrrega que poden ser configurat d'acord als requeriments d'una missió particular (Figura 10). La mateixa filosofia s'ha aplicat al sistema de propulsió, que també és reconfigurable. El disseny bàsic té 4 propulsors, dos verticals per accionar l'oscil · lació vertical i capcineig i dos horitzontals per a l'orientació i l'onatge. No obstant això, és possible tornar a configurar el vehicle per funcionar amb només tres propulsors (un vertical i dos horitzontals) i amb un màxim de vuit propulsors per controlar tots els graus de llibertat.

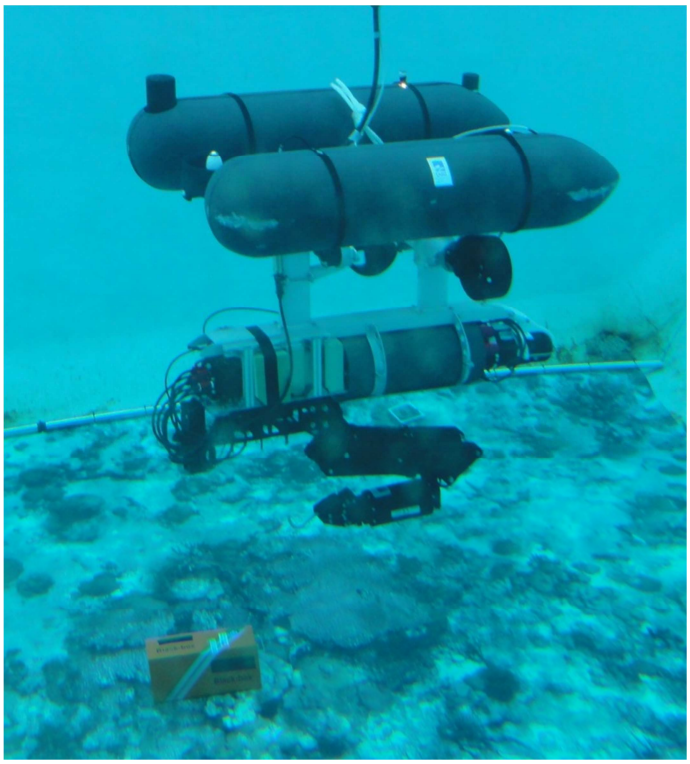


Figura 10: Girona500 amb braç robòtic

CAPÍTOL 6

6. Requisits del sistema

En aquest apartat s'explicaran els requisits funcionals i els requisits no funcionals del sistema.

6.1. Requisits funcionals del sistema

- Desenvolupar un driver que obtingui imatges de la càmera del robot.
- Desenvolupar una aplicació que calibri la càmera.
- Proporcionar una interfície visual intuïtiva que permeti seleccionar l'àrea de la imatge que es vulgui agafar com a referència visual.
- Donar la possibilitat d'agafar la referencia visual d'un fitxer.
- Per mitja de dos mètodes (Template Matching i SURF) obtenir les característiques de la referencia visual.
- Amb el mètode escollit per captar les característiques de la referencia visual, fer un seguiment d'aquesta referencia en les següents imatges tractades per determinar les variacions en posició d'aquesta.
- Depenent de si es vol mantenir en posició la referencia o es vol centrar aquesta, calcular les correccions a aplicar en la posició del robot.
- Desenvolupar un comportament per transmetre les correccions al sistema de navegació del robot.

6.2. Requisits no funcionals del sistema

El driver de la càmera ha de permetre obtenir imatges a una freqüència adient per obtenir una fluïdesa de visió suficient per a controlar el vehicle sense saturar el sistema.

L'aplicació que calibri la càmera ha de permetre visualitzar correctament les imatges corregint les alteracions que es puguin produir degudes a l'entorn.

El sistema ha de permetre carregar una imatge des de fitxer o escollir un retall de les imatges capturades per a fer un seguiment d'aquest segons la configuració escollida i anar calculant en temps real les correccions que s'haurien d'aplicar al robot per a mantenir aquella escena captada per les imatges.

CAPÍTOL 7

7. Estudis i decisions

7.1. Eines de treball

En aquesta Secció es farà una introducció ràpida de les eines/llibreries que s'han empleat per a desenvolupar el projecte.

7.1.1. Llenguatge C++

El projecte s'ha desenvolupat en llenguatge C++ [5] ja que l'anterior versió de l'arquitectura estava implementada en aquest llenguatge i perquè dels dos llenguatges que permet la versió basada en ROS que són el C++ i el Python i l'autor d'aquest projecte domina millor el C++ que el Python.

7.1.2. Llibreria OpenCV

OpenCV [2] està alliberada sota llicència BSD, que és gratuïta per a ús acadèmic i comercial. Pot corre sota C++, C, Python i aviat en Java i està disponible per Windows, Linux, Android i Mac. La llibreria compta amb més de 2500 algorismes optimitzats. Els seus usos van des de l'art interactiu, a la inspecció de mines, construcció de mapes en web i també aplicats a la robòtica avançada.

7.1.3. Eclipse (IED)

Eclipse [6] és un entorn integrat de desenvolupament de codi obert programat principalment en Java (per tant, multiplataforma), per a desenvolupar projectes en C, C++, COBOL, Python, Perl, PHP, i molts altres, sempre i quan s'instal · li l'entorn i les llibreries corresponents per a cada llenguatge de programació.

S'ha escollit a aquest entorn ja que els paquets ros utilitzen el CMake i estan preparats per generar el projecte en format eclipse.

7.1.4. Git

Git [7] és un programari lliure i obert, un sistema distribuït per control de versions dissenyat per a utilitzar des dels més petits fins als més grans projectes amb gran amb velocitat i eficiència.

Cada *clone* de Git és un repositori amb tot l'historial complet i totes les possibilitats de seguiment de control de revisions sense la dependència d'accés a la xarxa o un servidor central. Pot branquejar i barrejar els projectes i ho fa ràpid i fàcil de fer.

7.1.5. GanttProject

Gantt Project [8] és una eina d'escriptori multi plataforma per a la programació i gestió de projectes. S'executa en Windows, Linux i MacOSX i el seu codi és opensource. Pot fer diagrames de Gantt, crear una estructures de desglossament de treball, dependències dibuixar, definir fites, assignar recursos humans per treballar en les tasques, veure la seva assignació a la taula de càrrega de recursos, generar diagrames PERT de diagrames de Gantt, guardar gràfics com imatges PNG, generar informes PDF i HTML, la importació de projectes i l'exportació als formats de Microsoft Project. Exportació a fulls de càlcul amb CSV i compartir projectes amb els seus col · legues utilitzant WebDAV.

7.1.6. Dia

Dia [9] és una aplicació per a la creació de diagrames tècnics. Les característiques més destacables de Dia són les de tenir diverses pàgines d'impressió, possibilitat d'exportar a molts formats (EPS, SVG, CGM i PNG), i la capacitat de poder usar objectes personalitzats creats per l'usuari en simples descripcions en XML. Dia és útil per a dibuixar diagrames UML, mapes de la xarxa, i diagrames de flux.

7.2. Arquitectura COLA2

Component Oriented Layer-Based Architecture for Autonomy(COLA2), segueix un model basat en capes, que organitza els components en tres capes: missió, execució i reactiva. Es pot veure un exemple de arquitectura organitzada en 3 capes (Figura 11).

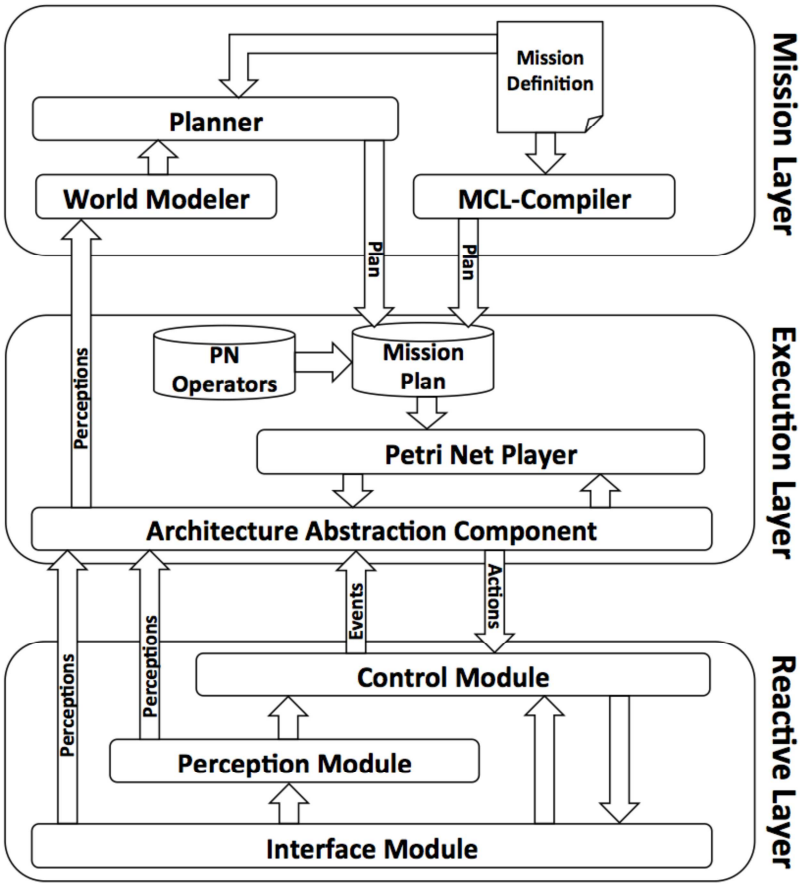


Figura 11: Arquitectura 3 capes

7.2.1. Capa Reactiva

És una capa molt depenen en els sensor i actuadors utilitzats, no obstant és divideix en tres mòduls per reduir la dependència:

- Vehicle Interface: Aquest mòdul conté els components, anomenats drivers, que interactuen directament amb el hardware, llegint dades dels sensor o enviant comandes els actuadors. Els drivers converteixen totes les dades en unitats coherents i com a referència fixed body frame.

- Percepció: Aquest mòdul rep dades reunides per el mòdul Vehicle Interface. Està format per diversos components anomenats processing units. El navegador, detector d'obstacles i detector d'objectius.

- Guia i Control: Aquest mòdul inclou un conjunt de comportaments, el coordinador i el controlador de velocitat. Els comportaments, son funcionalitats bàsiques del robot que poden anar des d'un procés per controlar al nivell de la bateria fins a un que segueixi trajectòries. En general, tots els comportaments busquen complir un objectiu. Els comportaments reben dades tant del Vehicle Interface com de la percepció, d'aquesta manera és independent del sensor físics i actuadors. El coordinador combina les respostes generades per els comportaments habilitats i el control de velocitat converteix aquestes dades a un vector de força per cada motor.

Encara que el COLA2 és una arquitectura basada en capes, la capa reactive pot ser vista com una arquitectura basada en comportament, en la qual hi ha un conjunt de comportaments habilitats que són coordinats per arribar a un objectiu. No obstant, qui decideix quin comportament està habilitat per la capa executiva seguint les instruccions de la capa de missió.

7.2.2. Capa executiva

La capa executiva actua com una interfície entre la capa reactiva i la de missió. Aquesta capa tradueix els plans de alt nivell a comandes de baix nivell habilitant i deshabilitant comportaments de la capa reactiva. Està formada per dos components, Architecture Abstraction Component i Petri Net Player. El AAC ens ofereix una interfície a la capa reactiva a través de tres senyals:

- Accions: Permeten activar i desactivar accions.
- Esdeveniments: Enviats per la capa reactiva i permeten notificar canvis.
- Percepcions: Són valors de sensors específics o processing units que s'extreuen directament de la capa reactiva a la de missió per extreure informació rellevant sobre l'estat actual del mon que s'està utilitzant un planificador.

El PNP executa la missió planejada, donava per la capa de missió. La missió és definida per mitjanes de Xarxes de Petri que descriu quina acció ha de ser executada depèn dels esdeveniments rebuts. Bàsicament, actua com un Discrete Event System (DES) connectant plans discrets amb comportaments continus.

7.2.3. Capa Missió

Plans de missions predefinides són les utilitzades actualment en missions d'AUV. No obstant els plans off-line poden fallar durant l'execució. Per altre banda, el us de plans generats on-line poden arribar a un comportament impredictibles en els comportaments del vehicle. Per tant, val la pena trobar un compromís entre els plans off-line i els on-line. COLA2 introdueix un llenguatge d'alt nivell, anomenat Mission Control Language(MCL), per descriure les missions off-line que són compilats a una xarxa de Petri. Aquest llenguatge pot incloure un operador de planificació. D'aquest manera es pot planificar una missió capaç de seleccionar quin és l'objectiu més apropiat per completar la missió.

7.3. Antic framework amb el qual estava desenvolupada l'arquitectura COLA2

En aquesta Secció farem un resum de l'arquitectura COLA2 així com de l'antic framework utilitzat. Tota la informació ha sigut extreta del Projecte de Final de Carrera de l'Enric Galceran, anomenat Disseny i Implementació d'una Arquitectura de Control per a Robots Autònoms.

L'antic framework es pot dividir en 5 mòduls (Figura 12):

- Networking : permet a les estacions comunicar-se entre elles, i al sistema amb altres sistemes externs a traves de la xarxa.
- DataManipulator : el format en que les dades s'intercanvien dins el sistemes es XML. Així doncs, cal poder convertir aquestes dades en format XML a dades de tipus que puguin operar-se aritmèticament o lògica: enters, reals, booleans, vectors, matrius... així com fer l'operació inversa. Aquest mòdul proporciona les eines per fer-ho.
- Threading : proporciona les classes per crear nous fils d'execució i poder executar tasques simultàniament.
- I/O : proporciona un accés uniforme i fàcil als dispositius d'entrada/sortida.
- Core : S'encarrega de les següents funcions:
    - Gestió de Components.
    - Comunicació entre components(compartida amb el mòdul Networking).
    - Comunicació d'esdeveniments.
    - Gestió de configuracions. Enregistrament de logs.
    - Distribució dels components.
    - Centralització de la informació persistent en un sol node.

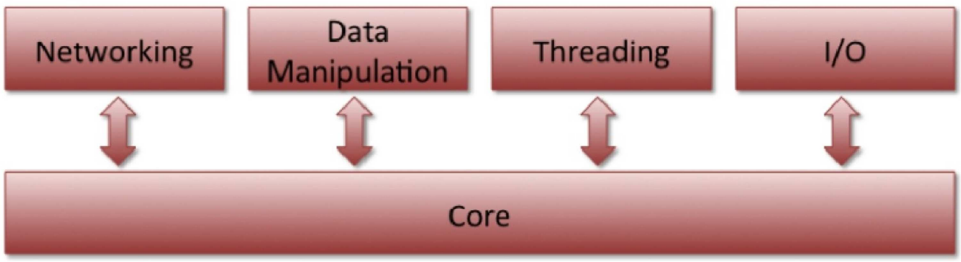


Figura 12: Mòduls antic framework

S'entrarà en detall en la Gestió dels components, la Comunicació entre components, la Gestió de configuració i El mòdul Data Manipulator per tal de poder entendre com funciona aquest sistema (Figura 13).

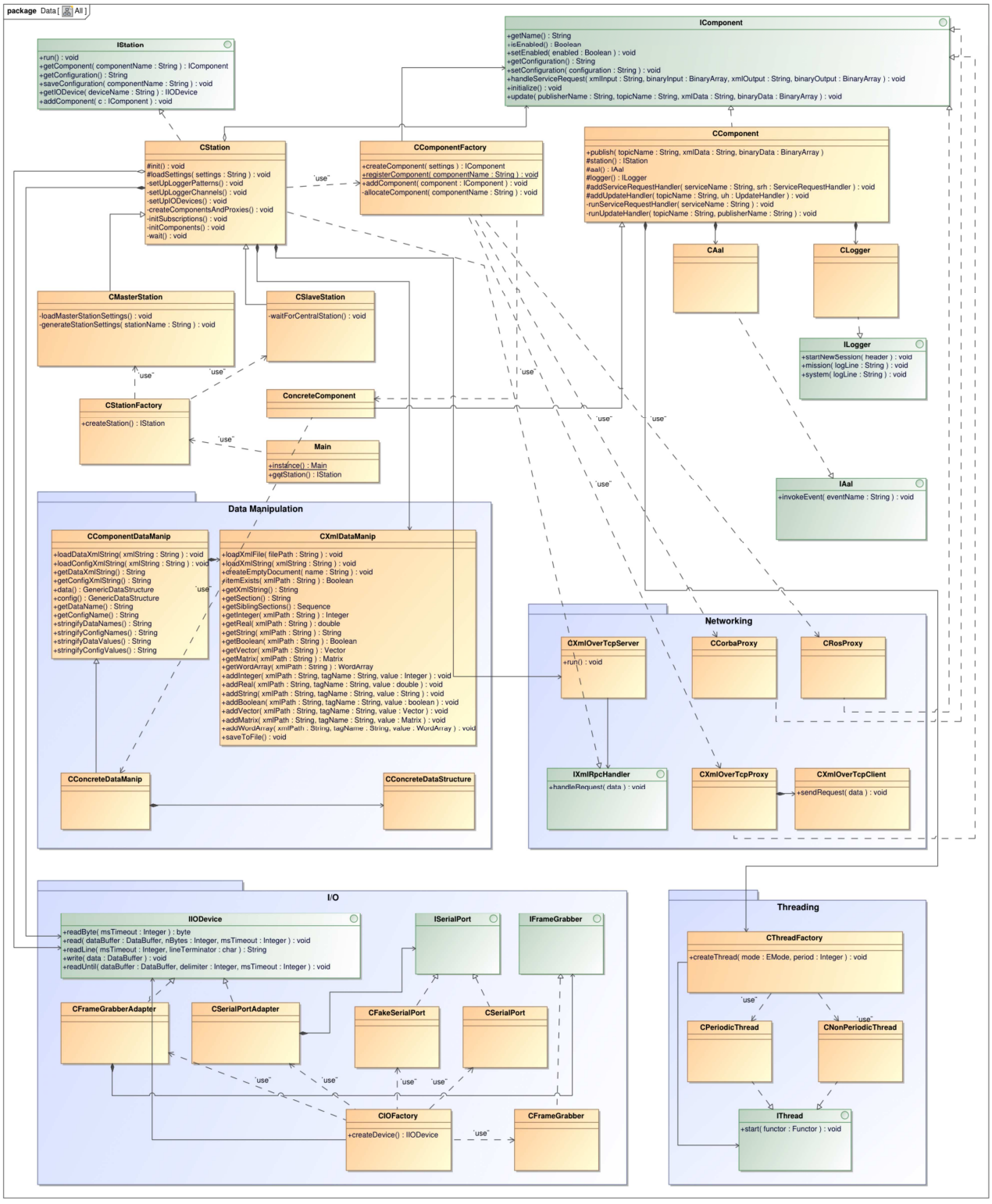


Figura 13: Arquitectura COLA2

7.3.1. Gestió dels components

La interfície IComponent representa un component del sistema robòtic (p.e. el driver d'un motor o un segmentador d'imatges). Les seves operacions son:

- getName: permet obtenir el nom del component.
- isEnabled: permet comprovar si el component es troba habilitat. setEnabled: permet habilitar o deshabilitar el component.
- initialize: operació que es cridada nomes pel sistema i nomes una sola vegada per realitzar accions d'inicialització. Cada component programat per l'usuari decideix quines accions executa aquesta operació. T picament el que s'hi farà es demanar la configuració del propi component al sistema portant a terme accions d'acord amb els seus paràmetres i crear un o mes threads per permetre al component executar tasques asíncronament.
- getConfiguration: permet obtenir la configuració del component.
- setConfiguration: permet aplicar una configuració al component.
- handleServiceRequest: permet fer una petició de servei al component i obtenir-ne el resultat.
- update: permet publicar dades que el component rep com a subscriptor.

Per entendre millor els mètodes de update i handleServiceRequest els s'explicaran en la següent Secció, que són els que permeten la comunicació entre components.

7.3.2. Comunicació entre components

La comunicació entre components ve descrita per la interfície IComponent. Per maximitzar la flexibilitat del sistema, els components poden fer servir dos estils diferents de comunicació: publicació/subscripció i orientada a servei.

La comunicació via publicació/subscripció es realitza aplicant el patró arquitectònic Publish-Subscribe. Es tracta d'un sistema de comunicació que va només en una direcció: un publicador envia dades cap a un subscriptor. D'entrada, aquest patró només permet especificar qui envia i qui rep, no permet discriminar el flux d'informació segons el tipus de dades. Per això s'ha incorporat el concepte de tòpic. Un tòpic representa una categoria de dades. Els publicadors publiquen diferents tòpics i cada tòpic es distingeix per un nom. Així, els subscriptors poden subscriure's només als tòpics que els interessin de cada publicador. A més, com que cada tòpic s'identifica amb un nom, els subscriptors poden saber quin tipus de dades han rebut abans de processar-les. Com es veurem més endavant, el mòdul Data Manipulation permet distingir les dades per tòpics. La informació sobre quin component es subscriu a quins tòpics de quin altre component s'especifica al fitxer de configuració de sistema.

La comunicació orientada a servei segueix els principis de disseny Service-Oriented Architecture. És una comunicació bidireccional: un component demana un servei a un altre, aquest l'executa i en torna el resultat al primer. En aquest cas, el component que demana el servei ja coneix per endavant a quin component ho demana, i també com s'estructuren tant la petició de servei com la resposta que rebrà.

L'operació update s'encarrega de la comunicació via publicació/subscripció. Quan un component vol publicar dades, fa una crida a la seva operació publish. El que fa l'operació publish és cridar l'operació update de tots els components que s'han registrat com a subscriptors del que publica.

L'operació update crida el handler que té associat al valor del paràmetre topicName (mitjançant l'operació runUpdateHandler). Perquè això sigui possible, el component subscriptor haurà d'haver associat cada nom de tòpic que és capaç de manipular amb un handler que serà cridat quan arribin dades d'aquell tòpic. Aquesta associació la fa el component subscriptor mitjançant l'operació addUpdateHandler.

Similarment, l'operació handleServiceRequest s'encarrega de la comunicació orientada a servei. El component que fa la petició farà una crida a l'operació handleServiceRequest del component al qual va dirigida, i aquest executarà una crida a la seva operació runServiceRequestHandler per executar el handler associat al nom del servei demanat (paràmetre serviceName). L'Associació entre el nom de servei i el handler l'haurà fet prèviament el component que rep la petició, cridant addServiceRequestHandler.

7.3.3. El Mòdul Data Manipulator

Aquest mòdul s'encarrega d'una tasca aparentment senzilla, però clau en un sistema on totes les comunicacions es basen en XML. El que fa és convertir estructures de dades (i.e. tuples) a documents XML i viceversa.

Totes les classes d'aquest mòdul operen basant-se en documents XML que tenen una estructura determinada.

La classe CXmlDataManip és l'encarregada de parsejar i carregar a memòria els documents XML i de comprovar que tenen l'estructura correcta. Les seves operacions permeten fer referencia als elements dels documents XML mitjançant la sintaxi arrel/fill/fill del fill..., a l'estil d'XPath.

La classe CGenericDataManip fa servir CXmlDataManip per convertir el text XML en valors de tipus enter, real, cadena de caràcters, booleà, vector, matriu o vector de paraules o per fer l'operació inversa.

Un cop definida la tupla que es vol convertir, cal crear una subclasse de CGenericDataManip per poder-la traduir a XML. Cada subclasse representa un tòpic: una categoria de dades. Per exemple, pot haver-hi el tòpic Navegació o el tòpic Estat del Sistema. Les dades contingudes en un tòpic són una tupla amb elements que poden ser dels tipus que CXmlDataManip pot gestionar. Tots els tòpics s'identifiquen amb un nom.

7.4. ROS

ROS [1] 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.

ROS té tres nivells: el sistema de fitxers, el graf de computació, i el nivell de la comunitat. Aquests nivells i conceptes es resumeixen a continuació en les següents Seccions.

7.4.1. Sistema de fitxers

Els nivells del sistema de fitxers són els recursos de ROS que es troben en el disc, com són:

- Packages: Els Packages són la unitat principal per a l'organització de programari en ROS. Un Package ROS contenir els processos en temps d'execució (nodes), les biblioteques dependents de ROS, bases de dades, arxius de configuració, o qualsevol altra cosa que sigui útil organitzar conjuntament.

- Manifests: Els Manifests (manifest.xml) proporcionen les metadades sobre un paquet, incloent la seva informació de llicència i dependències, així com informació específica de l'idioma o els flags del compilador.

- Stacks: Els Stacks són un conjunt de paquets que proporcionen una funcionalitat afegida, com un "Stack de navegació".

- Stack Manifests: Els Stack Manifests (stack.xml) proporcionen dades sobre una pila, incloent la informació sobre la llicència i les seves dependències en altres Stacks.

- Message (msg) types: Les descripcions de missatges emmagatzemats a my_package/msg/MyMessageType.msg, defineixen les estructures de dades dels messages enviats per ROS.

- Services (SRV) types: les descripcions de serveis, emmagatzemats a my_package/srv/MyServiceType.srv, defineixen les estructures de dades de les peticions i respostes per dels serveis de ROS.

7.4.2. Graf de computació

El graf de computació és la xarxa peer-to-peer dels processos de ROS que són els que processen dades conjuntament. Els elements bàsics del graf de computació de ROS són els nodes, master, el parameter server, messages, services, tòpics, i bags. Tots proporcionen dades al graf de diferents maneres (Figura 14).

- Nodes: 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. Per exemple, un node de control d'un telèmetre làser, un node de control dels motors, un node que realitza la localització, un node que realitza la planificació de ruta, un node proporciona una vista gràfica del sistema, etc. Un node de ROS està escrit amb l'ús d'una biblioteca de ROS, com roscpp o rospy.

- Master: 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.

- Parameter Server: El Parameter Server permet que les dades s'emmagatzemin per clau en un lloc central. Actualment forma part del Master.

- Messages: 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).

- Tòpics: 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.

- Services: 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.

- Bags: Els bags són un format per a guardar i reproduir les dades de ROS. Els bags 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.

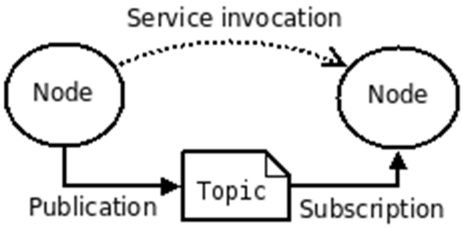


Figura 14: Esquema Teòric ROS

7.4.3. Nivell de Comunitat

Els conceptes de la Comunitat ROS són els recursos que permetran a les comunitats l'intercanvi de coneixements. Aquests recursos inclouen:

- Distributions: ROS Distributions són col · leccions de stacks que et pots instal · lar. Les Distributions juguen un paper similar al de les distribucions de Linux: que sigui més fàcil per instal · lar un conjunt de programari, i també mantenen versions consistents en un conjunt de programari.

- Repositories: ROS es basa en una xarxa federada de repositoris de codi, on diferents institucions poden desenvolupar i publicar els seus propis components.

- La ROS Wiki: La ROS Wiki de la comunitat és el principal fòrum per documentar la informació sobre ROS. Qualsevol persona pot aportar pel seu compte i contribuir amb la seva documentació, facilitar les correccions o actualitzacions, escriure tutorials, i molt més.

- Llistes de correu: La llista de correu-ROS d'usuaris és el principal canal de comunicació sobre noves actualitzacions de ROS, així com un fòrum per fer preguntes sobre el programari ROS.

- ROS Answers: Un Q&A per ajudar amb preguntes relacionades amb ROS.

- Blog: El bloc de Willow Garage ofereix actualitzacions periòdiques, incloent fotos i vídeos.

CAPÍTOL 8

8. Anàlisis i Disseny del sistema

8.1. Anàlisis i Disseny del Node Keep Pose

Aquest node és el que s'encarregarà d'enviar les consignes al sistema de navegació.

Te dos modes de treball:

- Dades de navegació: amb aquest mode, el node rep dades del sistema de navegació i segons la posició desitjada, va calculant les correccions per conservar la posició desitjada.
- Dades del node d'imatges: amb quest mode, el node rep els desplaçaments en píxels, calcula les consignes adequades i les transmet al sistema de navegació i control dels motors.

El segon mode de treball el primer cop que es va provar va ser fent experiments amb els companys de la UJI. Ells disposaven d'un node que utilitzant visió per computador aplicava tècniques de Template Matching. Aquest node enviava les dades a traves del pont COLA2-ROS i el node de Keep Pose era l'encarregat de processar les dades que arribaven. Després d'aquests experiments és quan l'equip de treball va decidir desenvolupar una versió pròpia d'aquell node i s'experimentés amb Template Matching i també amb SURF.

Amb el mode que només tracta dades de navegació, el que es fa és:

- Llegir la posició absoluta que s'ha de conservar.
- Computar el desplaçament necessari per a conservar la posició desitjada (graus de llibertat x, y, z i yaw).
- Calcular les consignes que ha de transmetre al sistema de navegació amb l'error computat utilitzant un PID (controlador de tipus Propocional Integratiu i Derivatiu).
- Publicar les dades pel tòpic behavior_response.

Per computar el desplaçament, s'haurà de calcular una transformació homogènia per tal de mapejar el punt desitjat respecte el robot (Figura 15).

Per fer-ho és necessària la següent formula:

$$T=\begin{pmatrix} R & p \\ 000 & 1 \end{pmatrix} \Rightarrow T^{-1}=\begin{pmatrix} R^T & -R^T * p \\ 000 & 1 \end{pmatrix}$$

On:

- R és la matriu de rotació. En aquest cas la Rotació homogènia sobre z que és: $\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- p és el punt on és troba el robot
- R^T és la matriu transposada de R: $\begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$

I per obtenir la posició desitjada respecte el robot es multiplicarà la matriu T^{-1} per la posició desitjada.

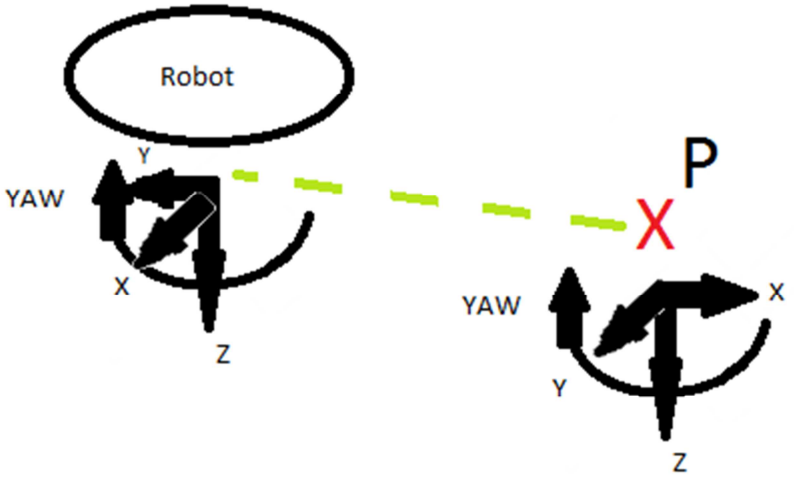


Figura 15: Desplaçament de Posició Absoluta

Amb el node subscrit al tòpic píxel_correction, el que es fa és:

- Si és el primer cop es guardarà la posició en z i l'orientació en yaw del tòpic de navigator_data (encara que el robot variï en els altres graus de llibertat, aquest fent una simple resta podrà corregir-los).
- Si no és el primer cop, calcular les consignes que ha de transmetre al sistema de navegació amb les correccions del tòpic píxel_correction(graus de llibertat x i y), restant les dades actuals de z i yaw obtingudes del sistema de navegació amb les guardades com a referencia (com que són

absolutes no s'ha de replantejar el sistema de coordenades) i utilitzant PID (controlador de tipus Propocional Integratiu i Derivatiu).

- Publicar les dades pel tòpic behavior_response.

Es pot observar el funcionament del node en el següent diagrama (Figura 16).

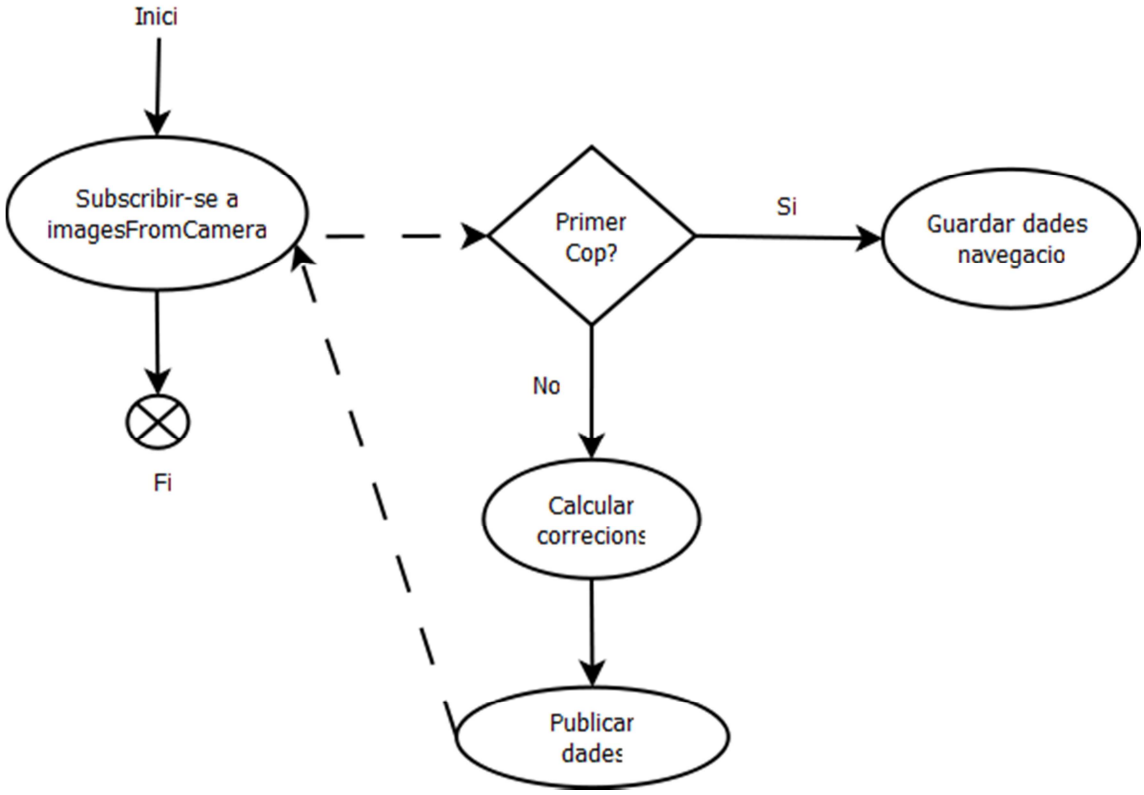


Figura 16: Diagrama Node Keep Pose

8.2. Anàlisis i Disseny del Bridge COLA2-ROS

Aquesta Secció ha sigut desenvolupada de forma conjunta entre l'autor d'aquest projecte i l'Arnau Carrera Viñas.

En un primer moment es va prendre la decisió d'implementar el ProxyROS que s'havia dissenya en l'anterior framework. Aquest framework però va ser dissenyat sense conèixer en profunditat el ROS.

Així doncs després de tenir el primer contacte amb el ROS i observar les dificultat d'enviar o rebre missatges simulant que un objecte del framework COLA2 era un node ROS resultava complexa. Es va prendre la decisió d'enviar les dades a traves de missatges del COLA2 a un socket TCP/IP en format XML i a dins del ROS convertir aquest tipus de missatges al Message de ROS apropiat.

8.2.1. Anàlisis i disseny dins del COLA2

Al ser una decisió provisional i tenir una urgència important, es va decidir que només seria necessari crear un component. Aquest enviarà les dades de Navegació i es volen poder realitzar les següents accions:

- La primera acció es afegir un component de navegació diferent al que utilitza el COLA2 per un d'extern, aquest component calcularà la navegació utilitzant visual-odometry. Aquesta tècnica consisteix en calcular la posició a partir d'imatges.

- La segona acció és mantenir una posició del vehicle a partir del feedback enviat per el mòdul extern. Per mantenir aquesta posició s'utilitzarà el node Keep Position. Es pot observar el diagrama de classes del disseny d'aquest objecte (Figura 17).

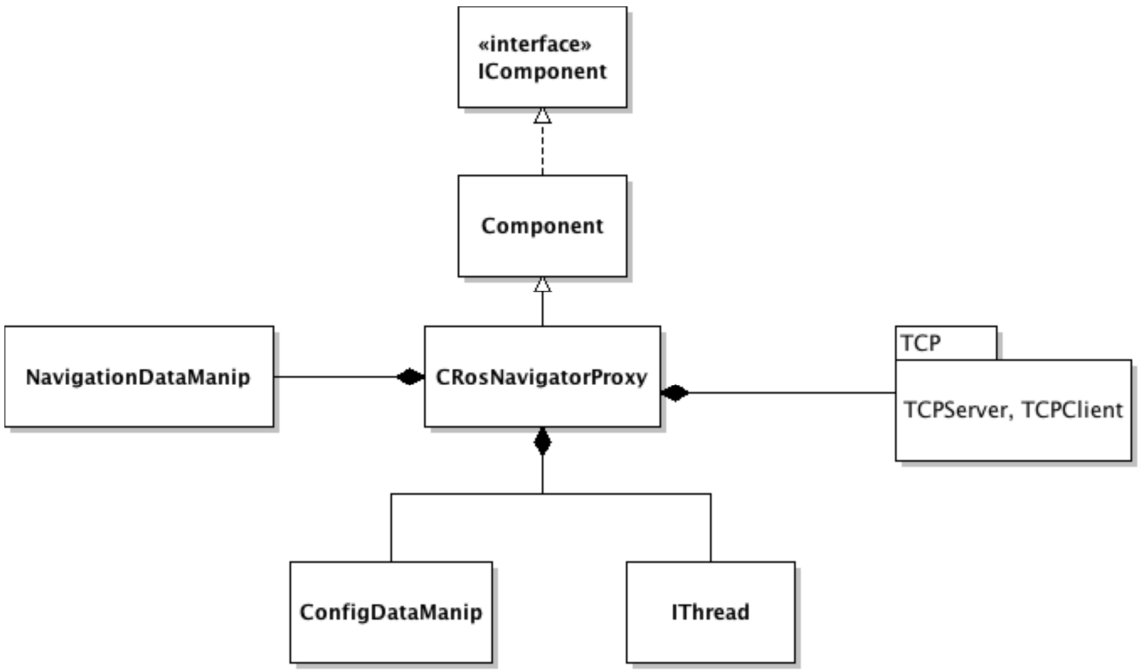


Figura 17: Diagrama de Classes de Disseny del ROSNavigatorProxy

Per altra banda, per comprendre millor el funcionament de tots els objectes es pot observar un diagrama on es veu representat els diferents components i com són les relacions del publish/subscrive, entre ells (Figura 18).

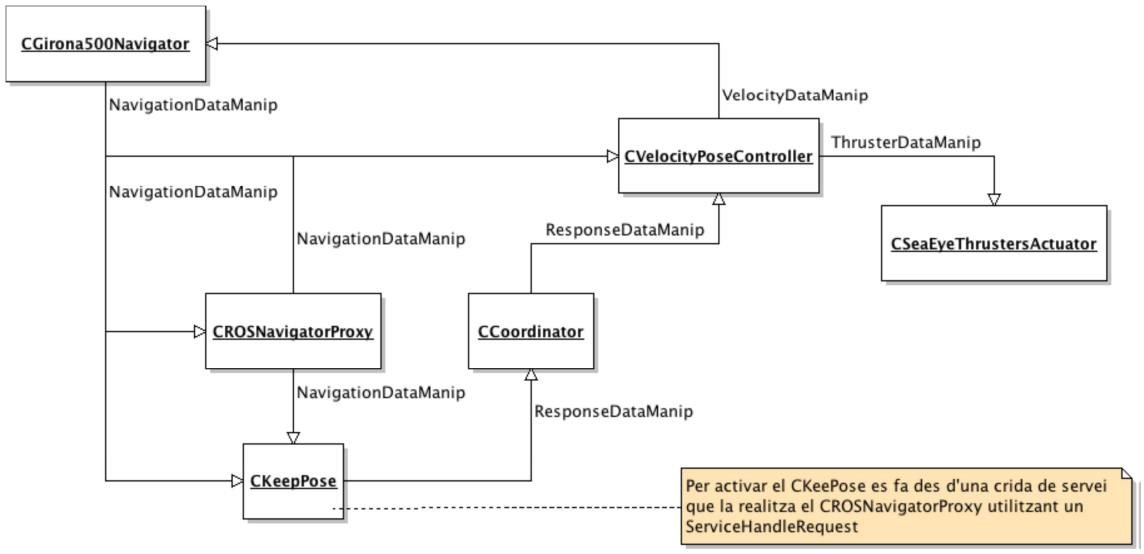


Figura 18: Diagrama de Publish/Susbcribe del ROSNavigatorProxy

8.2.2. Anàlisis i Disseny dins del ROS

Per tal de desenvolupar aquesta part es va decidir de crear un sol package on crearem totes les llibreries necessàries per tal de utilitzar un servidor/client TCP com les classes necessàries del DataManipulators per rebre i enviar els missatges corresponents.

En aquest cas necessitarem dos processos diferents. Això es degut a que no tots els sistemes necessiten aquesta dada i per tant volem tenir la opció de no rebre-la.

Un procés mainClient serà un client del servidor del COLA2, aquest estarà subscrit als missatges que envien els sistemes externs de ROS. Convertirà de missatges Odometry (tipus de Message del framework ROS que permet representar un vector de posició linear i angular) a NavigationDataManip (tipus de missatge del anterior framework que permet representar un vector de posició linear i angular), per fer això s'ha de convertir el sistema d'orientació de Quaternion a Euler. Això es degut a que el ROS utilitza el Quaternion i per altre banda el COLA2 utilitza el sistema Euler que és més intuïtiu.

L'altre procés mainServer serà un servidor que es connectarà amb el client del COLA2. El client de COLA2 envia les dades de navegació cada cop que s'actualitzen i el nostre node ROS convertirà aquestes el missatge de NavigationDataManip a Odometry.

8.2.3. Comunicació entre els dos sistemes

Per acabar l'anàlisi i el disseny d'aquesta Secció en la Figura 19 es mostrarà la comunicació entre els diferents components que formaran tot el sistema i com actuen entre ells (Figura 19).

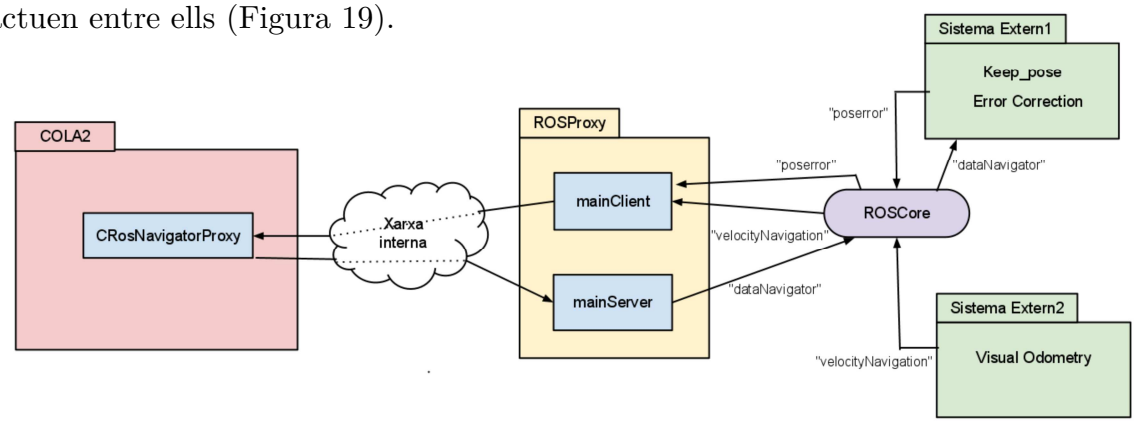


Figura 19: Diagrama esquemàtic de tot el sistema COLA2 ROS

8.3. Anàlisis i Disseny del driver per la càmera i el Node de cal · libració

Aquesta Secció es dividirà en dues parts, en una s'explicarà el driver de la càmera i en l'altre s'explicarà un node extra que es va implementar per al seu calibratge.

El node de calibratge és va decidir de fer després d'estar fent proves amb la càmera perquè es va detectar que quan la càmera canviava d'entorn es detectaven distorsions considerables.

Així dons el driver per la càmera fet és va dissenyar perquè pogués utilitzar-se en qualsevols dels robots del CIRS i inclús per una webcam convencional.

8.3.1. Anàlisis i Disseny del driver per la càmera

Al fer-se servir les llibreries d'OpenCV el camí a seguir era ample. Aquestes llibreries ajuden molt en aquestes tasques. Les accions del node son:

- Seleccionar de quin dispositiu es volen llegir dades.
- Capturar un frame de la càmera.
- Segons el tipus de dispositiu (webcam o framegrabber), pot ser que sigui necessari un retall en la imatge per eliminar píxels "brossa". En el cas del robot, el qual porta instal · lada una framegrabber, quan es capturaven imatges apareixien uns marcs negres (Figura 20) que no pertanyin a les dades que es volien obtenir, i per eliminar-los es fa un crop a les imatges si és el cas. Això es degut a que les OpenCV estan pensades per treballar amb webcams i a l'hora d'adquirir dades en determinades framegrabber fallen, al menys en Linux.

Figura 20: Imatge de framegrabber sense retallar

- Si es vol calibrar les imatges, fer-ho.
- Mostrar la imatge per pantalla.
- Empaquetar la imatge dins d'un paquet de ROS, en aquest cas un sensor_msgs::Image, per a poder enviar-lo per el tòpic imagesFromCamera. Per empaquetar la imatge dintre d'un paquet ros es fa servir una classe de ROS que agafa una matriu d'OpenCV i la fica dintre d'una estructura de dades emplenant automàticament tota la informació necessària de la imatge com és el tipus de codificació (el sistema treballarà amb imatges amb format bgr8), amplada, altura, canals i un vector amb la matriu de la imatge. D'aquesta manera el altres nodes que es subscrivin a aquest tòpic podran obtenir les imatges capturades per la càmera amb tota aquesta informació.
- Publicar les dades.

I així, si es van capturant imatges una darrera l'altra, obtenim video de la càmera.

Es pot observar el funcionament de la càmera en el següent diagrama (Figura 21).

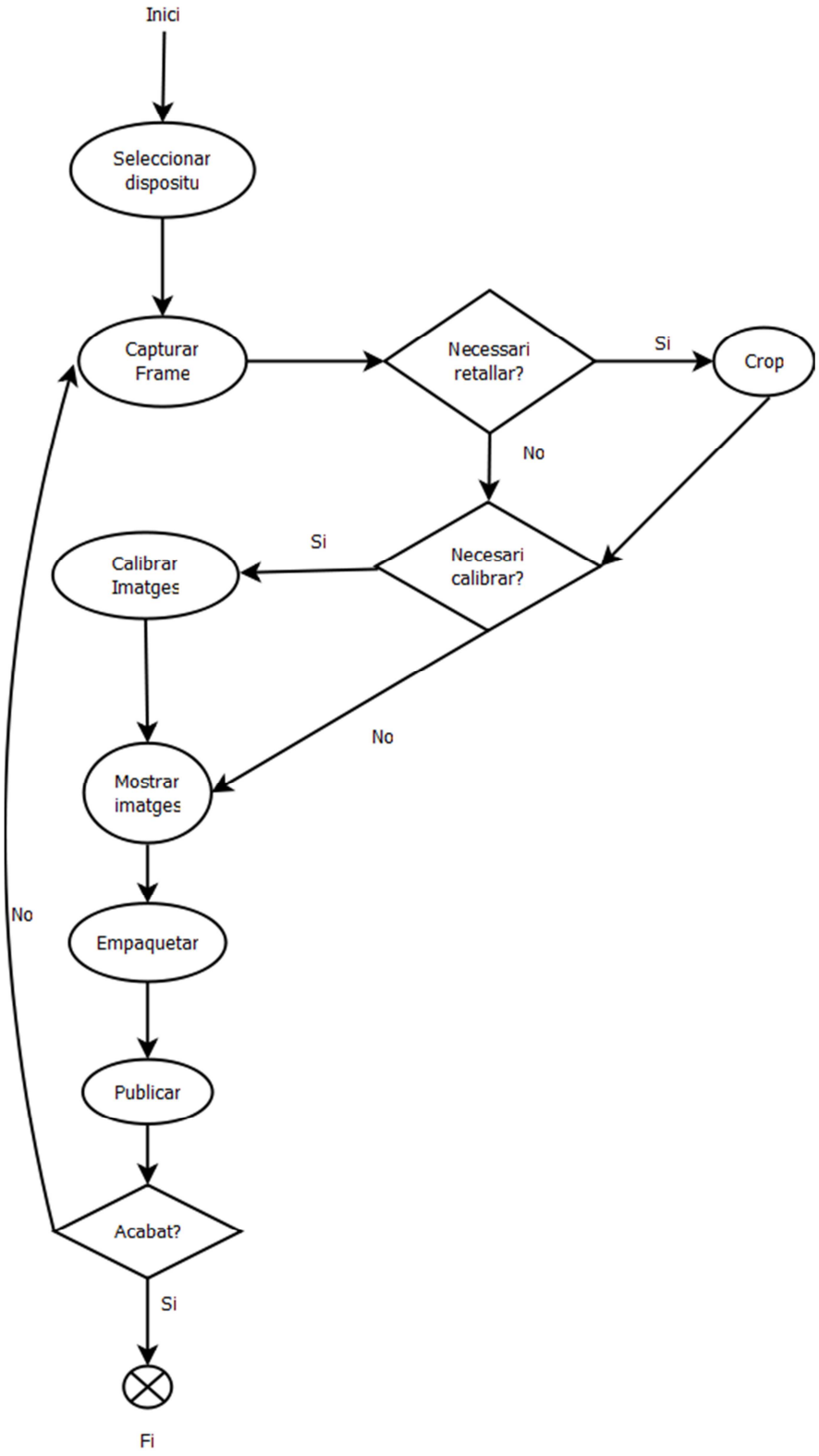


Figura 21: Diagrama Driver Càmera

8.3.2. Anàlisis i Disseny del Node de Cal · libració

Després d'investigar i fer recerca per fòrums i la documentació oficial d'OpenCV és va fer una idea de com es podia fer.

Per començar aquest node ROS, en el seu main no farà pràcticament res, simplement estarà escoltant al tòpic per on s'envien les imatges per a, en el moment que n'hi hagin, fer un Call Back a una funció que serà el que faci tota la fenia.

Funció CallBack

Tractar Imatge Per Calibrar

FFunció

Main

Subscribir-se al Tòpic d'Imatges

FMain

En aquesta funció es van agafant imatges que estiguin gravant un tauler d'escacs de mesures conegudes, llavors es capturen les imatges en les que es detectin totes les cantonades dels quadres del tauler. D'aquesta manera, quan es fa un seguit de vegades, podrà calcular la relació entre les mesures reals i les obtingudes per la càmera. D'aquesta manera es sabrà la matriu intrínseca i els coeficients de distorsió que te la nostra càmera i podrem rectificar les imatges. Les passes que seguirà són:

- Subscriure el main al tòpic imagesFromCamera.

Després d'això, esperar a que ens arribin dades pel tòpic per a que el Call Back sigui invocat. Aquest el que farà és:

- Si el número d'elements necessaris a les taules per calibrar la càmera no és suficient, trobar els corners del tauler d'escacs. Si troba tots els corners del tauler, afegirà les dades d'on es troba cada corner a les taules per a fer els càlculs de cal · libració.

- Quan ja es tenen dades suficients per calibrar la càmera, calcular la matriu intrínseca i la de coeficients de distorsió.

- Salvar aquestes dues matrius en dos fitxer .xml per a poder calibrar les imatges obtingudes per aquesta càmera. Aquest procés només s'ha de realitzar una vegada per a cada càmera si les condicions no varien.

Es pot observar el funcionament del node de cal · libració en el següent diagrama (Figura 22).

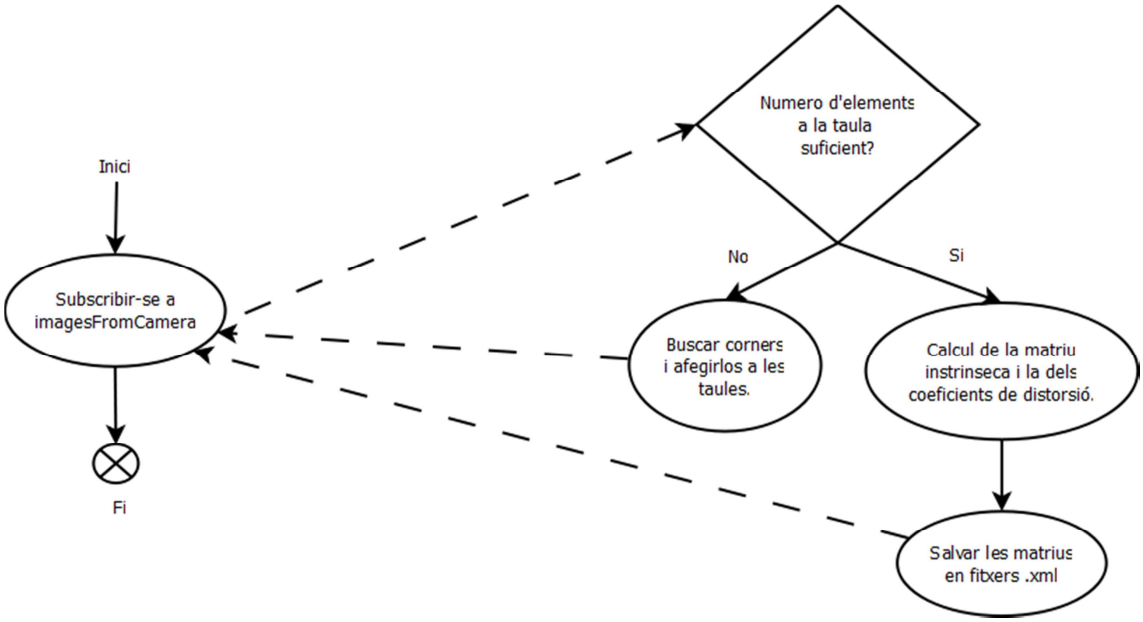


Figura 22: Diagrama Node Cal · libració

8.4. Anàlisis i Disseny del Node Template Matching

Una de les tècniques més comunes per buscar un fragment d'una imatge dins d'una altre és el Template Matching. Consisteix en localitzar un model dins d'una imatge. Aquesta tècnica es poc costosa computacionalment si prèviament a la imatge si li fan pretractaments com, per exemple, una binarització. Per utilitzar aquesta tècnica és necessari disposar del template a buscar. Aquest node pot treballar en dos modes, o carregant el model des de un fitxer o en temps d'execució marcant una regió de les imatges capturades. Si es fa servir en el primer cas es pot pretractar les imatges capturades per millorar la localització del model.

Aquest node tindrà un main que es subscriurà al tòpic imagesFromCamera. Aquesta funció tindrà un Call Back que s'encarregarà de fer tota la feina. I el Call Back també tindrà un altre Call Back dintre, que aquest detectarà les interrupcions de mouse per el cas en que l'usuari esculli el template en temps d'execució.

Funció CallBackMouse

Capturar Clicks

FFunció

Funció CallBackImatges

Habilitar Interrupcions Mouse

Processar Imatges

FFunció

Main

Subscribir-se Tòpic Imatges

FMain

El Call Back del mouse el que farà és:

- Si es fa per primer cop un click esquerre, s'agafarà aquest punt com una cantonada del Template. Quan es fa un click esquerre per segon cop, serà la cantonada contraria del Template. En aquest moment la funció Call Back de la imatge és quan començarà a fer realment feina.

- Si es fa click dret, en qualsevol moment, es desseleccionarà el template i la funció Call Back de la imatge deixarà de fer feina fins que es torni a agafar un altre template.

La funció Call Back d'imatges farà les següents tasques:

- El primer cop es guardarà el Template.

- Determinar una "resolució" per a buscar el template en aquesta regió de la imatge i no en tota ella.

- Buscar el punt de la imatge o trobi més semblances amb el template. Segons la fiabilitat del valor, s'augmentarà o reduirà la resolució de cerca. El Template Matching només troba correspondències en el cas que es produeixi una translació. No pot detectar quan es produeixen rotacions o escalats. Aquests dos graus de llibertat que no pot detectar vindran controlats a traves de la navegació del vehicle, ja que l'angle i la profunditat són valors absoluts que venen donats per la brúixola i el sensor de pressió (Figura 23).

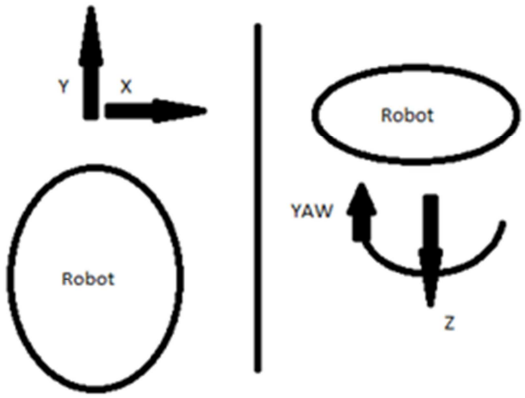


Figura 23: Graus en que Template Matching pot detectar desplaçaments (esquerra) i no pot (dreta)

- Si la fiabilitat del match es suficientment alta, es renovarà el template ja que, a mesura que es vagi movent el robot, pot anar variant la il · luminació i la perspectiva de les imatges. D'aquesta manera sempre es tindrà un template actualitzat.

- Un cop ubicat el template i el resultat sigui prou fiable, es calcularà el desplaçament que s'hauria d'aplicar al template per tal de que estigues allà on es vulgui.

- Publicar les dades pel tòpic pixel_correction.

Durant tota l'estona s'aniran mostrant les imatges, la ubicació de template i la resolució on s'està buscant-lo de manera de fer la interacció amb l'usuari més còmoda e intuïtiva. Es pot observar el funcionament del node en el següent diagrama (Figura 24).

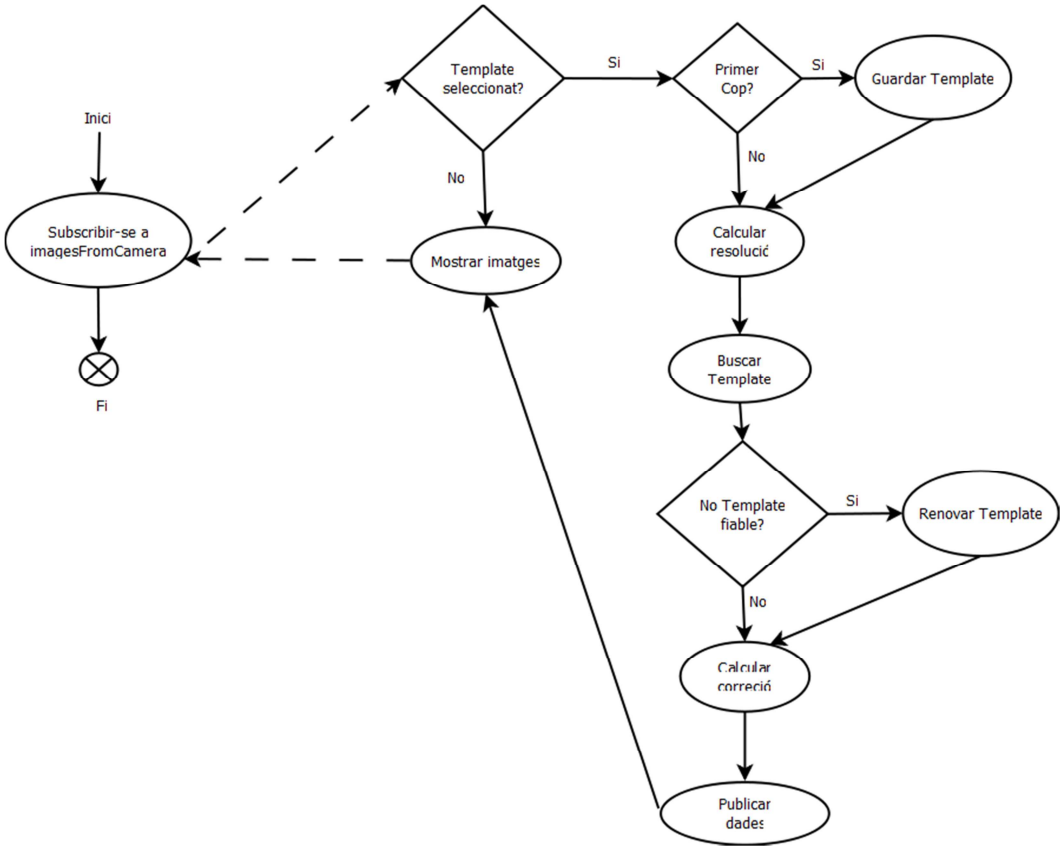


Figura 24: Diagrama Node Template Matching

8.5. Anàlisis i Disseny del Node SURF

Una tècnica més complexa per al tracking d'imatges és la de "SURFejar" una imatge. SURF [10] (Speeded Up Robust Feature) és una tècnica robusta de detecció d'imatges i descriptors. Es va presentar per primera vegada per Herbert Bay el 2006, utilitzada en tasques de visió per computador, com en reconeixement d'objectes o de reconstruccions en 3D. És en part inspirat pel descriptor SIFT. La versió estàndard de l'SURF és varies vegades més ràpida que el SIFT i proclamada pels seus autors per ser més robusta davant les transformacions d'imatge en comparació a la SIFT. SURF es basa en sumes aproximades de respostes 2D *Haar waveled* i fa un ús eficient d'imatges integrades.

Aquest node té un flux semblant a l'anterior però canviant el mètode amb el que s'identifica el Template.

Aquest node tindrà un main que es subscriurà al tòpic imagesFromCamera. Aquesta funció tindrà un Call Back que s'encarregarà de fer tota la feina. I el Call Back també tindrà un altre Call Back dintre, que aquest detectarà les interrupcions de mouse per el cas en que l'usuari esculli el template en temps d'execució.

Funció CallBackMouse

    Capturar Clicks

FFunció

Funció CallBackImatges

    Habilitar Interrupcions Mouse

    Processar Imatges

FFunció

Main

    Subscribir-se Tòpic Imatges

FMain

El Call Back del mouse el que farà és:

- Si es fa per primer cop un click esquerre, s'agafarà aquest punt com una cantonada del Template. Quan es fa un click esquerre per segon cop, serà la cantonada contraria del Template. En aquest moment la funció Call Back de la imatge és quan començarà a fer realment feina.

- Si es fa click dret, en qualsevol moment, es desseleccionarà el template i la funció Call Back de la imatge deixarà de fer feina fins que es torni a agafar un altre template.

La funció Call Back d'imatges farà les següents tasques:

- El primer cop es guardarà el Template com a vector amb els descriptors dels punts d'interès que aquesta contingui.

- Si no és el primer cop, agafar la imatge actual i buscar els punts d'interès de la imatge i guardar-los a un vector.

- Buscar les relacions entre el vector de referencia i el de la imatge actual.

- Si al menys s'han pogut trobar 4 relacions podrà ubicar a on esta el template escollit.

- Un cop ubicat el template, si el resultat és prou fiable, calcular el desplaçament que s'haurien d'aplicar al template per tal de que estigues allà on pertoca.

- Publicar les dades utilitzant el tòpic pixel_correction.

Durant tota l'estona s'aniran mostrant les imatges i la ubicació de template de manera de fer la interacció amb l'usuari més còmoda e intuïtiva.

Es pot observar el funcionament del node en el següent diagrama (Figura 25).

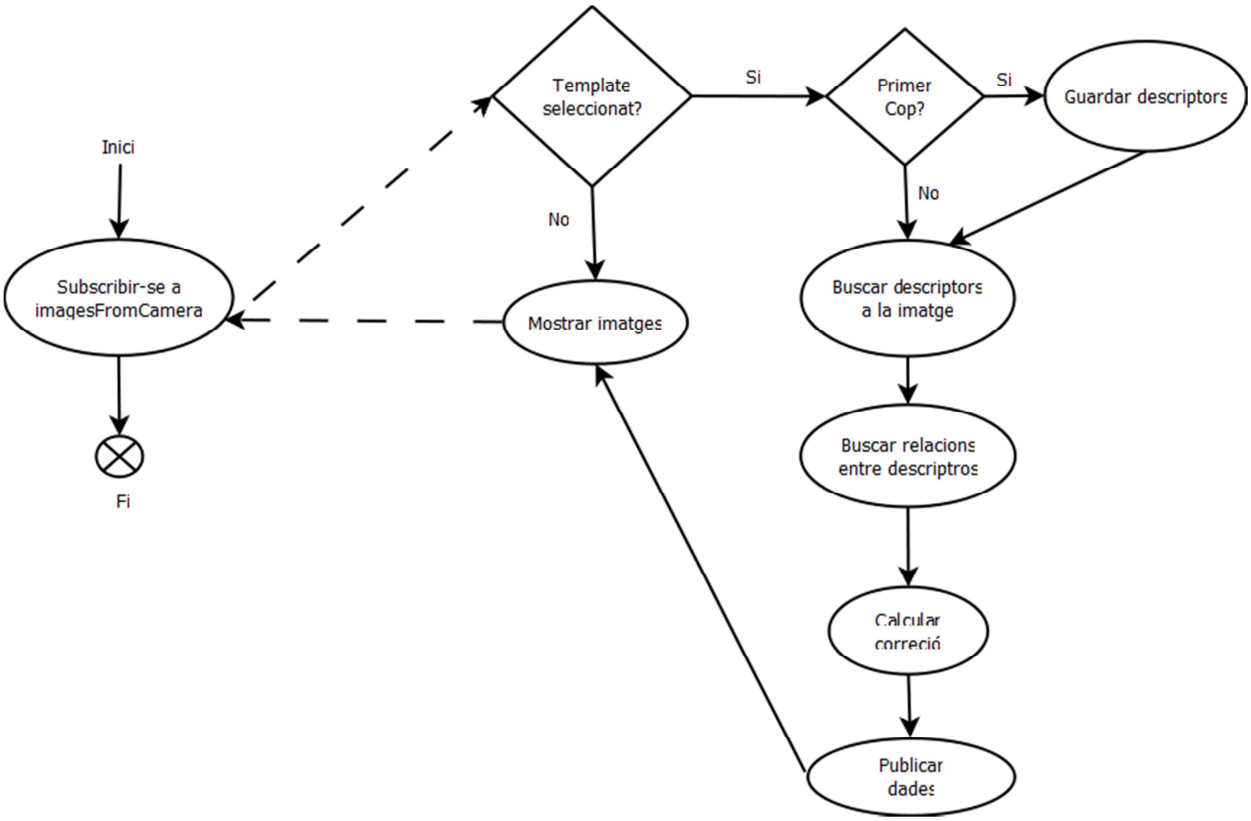


Figura 25: Diagrama Node SURF

Amb aquest node es poden fer correccions en x i y, z i yaw a diferència del de Template Matching. Encara que permet tenir un control de tots els graus de llibertat del robot, la z i yaw vindran controlades a traves de la informació de navegació del vehicle ja que l'angle i la profunditat són valors absoluts que venen donats per la brúixola i el sensor de pressió, que són sistemes més robustos.

8.6. Anàlisis i Disseny del Node de Processament d'imatges

Després de dissenyar els nodes de Template Matching i SURF, es va veure que els dos nodes desenvolupaven la mateixa tasca però amb algorismes per processar les imatges diferents. Així que, es poden pensar com dos nodes diferents, però finalment s'han implementat com a un de sol, que a l'hora d'executar-se, segons els seu fitxer de configuració, s'executarà amb un algorisme o l'altre.

Aquest node tindrà un main que es subscriurà al tòpic imagesFromCamera. Aquesta funció tindrà un Call Back que s'encarregarà de fer tota la feina. I el Call Back també tindrà un altre Call Back dintre, que aquest detectarà les interrupcions de mouse per el cas en que l'usuari esculli el template en temps d'execució.

Funció CallBackMouse

    Capturar Clicks

FFunció

Funció CallBackImatges

    Habilitar Interrupcions Mouse

    Processar Imatges

FFunció

Main

    Subscribir-se Tòpic Imatges

FMain

El Call Back del mouse el que farà és:

- Si es fa per primer cop un click esquerre, s'agafarà aquest punt com una cantonada del Template. Quan es fa un click esquerre per segon cop, serà la cantonada contraria del Template. En aquest moment la funció Call Back de la imatge és quan començarà a fer realment feina.
- Si es fa click dret, en qualsevol moment, es desseleccionarà el template i la funció Call Back de la imatge deixarà de fer feina fins que es torni a agafar un altre template.

La funció Call Back d'imatges farà les següents tasques:

- El primer cop es guardarà les dades necessàries del Template.
- Executar la part de codi dels algorismes que ubicaran el Template.
- Un cop ubicat el Template, calcular el desplaçament que s'hauria d'aplicar al Template per tal de que estigues on pertoca.
- Publicar les dades pel tòpic pixel_correction.

Durant tota l'estona s'aniran mostrant les imatges, la ubicació de Template i la resolució on s'està buscant-lo de manera de fer la interacció amb l'usuari més còmoda e intuïtiva.

Es pot observar el funcionament del node en el següent diagrama (Figura 26).

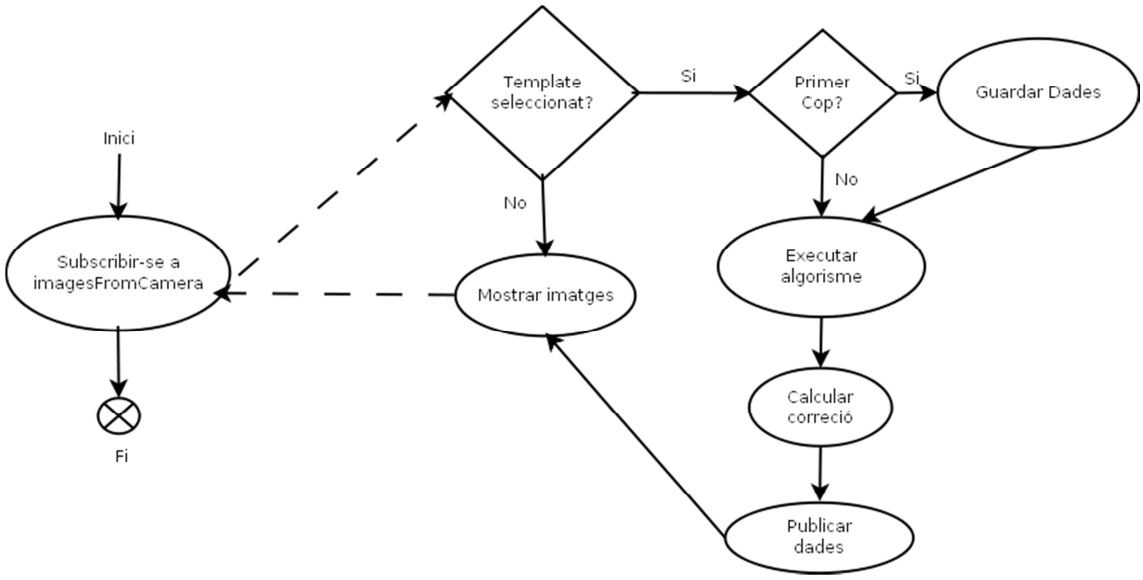


Figura 26: Diagrama Node Processament d'Imatges

8.7. Anàlisis i Disseny de tot el conjunt

Així doncs l'algorisme final seguirà els següents punts:

- Obtenir imatges de la càmera.
- Aquestes imatges arribaran al node de processament d'imatges, que les tractarà i obtindrà les correccions en píxels.
- Aquestes correccions en píxels arribaran al node Keep Pose, que calcularà les consignes per al sistema de navegació.
- El sistema de navegació i control dels motors agafarà les consignes i les aplicarà als motors del robot.

En el següent diagrama es mostrarà el funcionament de tot el conjunt (Figura 27).

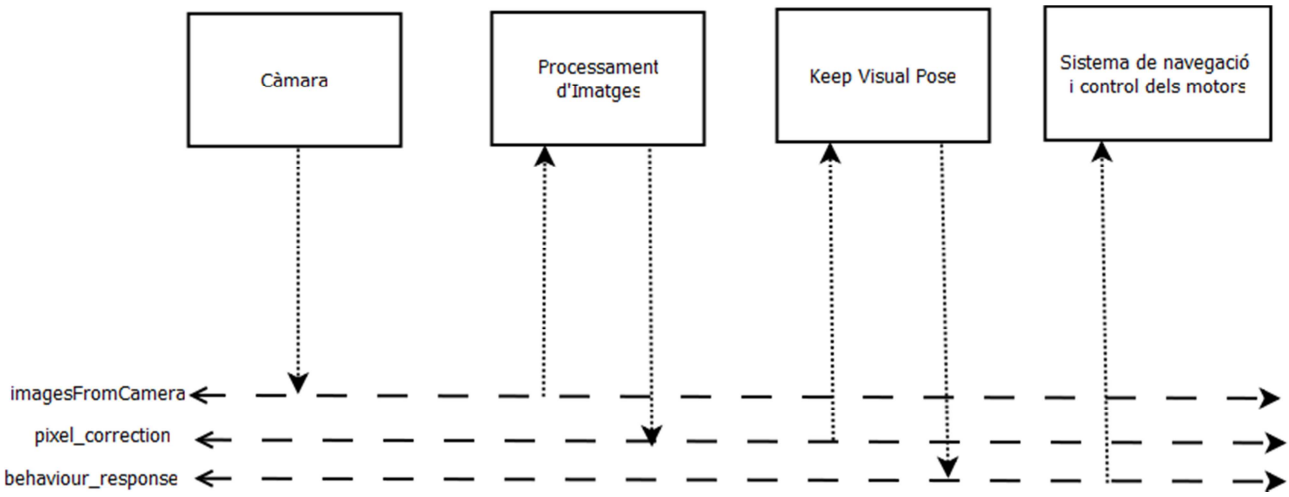


Figura 27: Diagrama funcionament del tot el conjunt

CAPÍTOL 9

9. Implementació i Proves

9.1. Implementació en ROS

Tots els nodes de ROS tenen un estructura genèrica que segueix mes o menys aquests esquemes [11]:

- Per als publicadors

```
#include "ros/ros.h"

#include "std_msgs/String.h"

#include <sstream>

int main(int argc, char **argv)

{

/* El Ros::init() ens arrencarà aquest main com a Node de ROS i li
donarà com a nom el tercer paràmetre que se li passi.   */

  ros::init(argc, argv, "talker");

  /* NodeHandle és el principal punt d'accés a les comunicacions amb
el sistema de ROS. El constructor NodeHandle inicialitzarà aquest
node, i NodeHandle el destructor tancarà el node.  */

  ros::NodeHandle n;

  / * La funció advertise() com li dius a ROS que vols publicar a un
Topic donat. Això invoca una crida al ROS Master que mante un
registre dels nodes i els tòpics. Després de fer l'advertise(), el
Master informarà a qualsevol node que es vulgui subscriure que ha de
negociar una connexió peer-to-peer amb aquest node. L'advertise()
retorna un objecte Publisher() que permet publicar missatges al
tòpic a traves de la crida publish. El segon paràmetre indica el
buffer del tòpic. */

  ros::Publisher chatter_pub =
n.advertise<std_msgs::String>("chatter", 1000);

/ * loop_rate() indicarà la freqüència a la que es publicarà * /

  ros::Rate loop_rate(10);

  int count = 0;

/ * el ros::ok() retornarà cert mentre no es tanqui el Master o se
li envií un cnt+c * /

  while (ros::ok())

  {

    /* Aquest es un objecte de missatges. Es on es fica l'
informació i es lo que publiquem. */

    std_msgs::String msg;

    std::stringstream ss;
```

```
    ss << "hello world " << count;

    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str())

    /* La funció publish() es la que permet enviar missatges. El
paràmetre es l'objecte missatge. El tipus de missatge a de coincidir
amb el declarat al advertise<>(). */

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();

    ++count;

 }

  return 0;

}
```

Bàsicament el que fa es declarar un publicador i mentre no s'enviï un cntr+c des de consola no pararà de publicar dades pel tòpic.

- Per als subscriptors

```
#include "ros/ros.h"

#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)

{

  ROS_INFO("I heard: [%s]", msg->data.c_str());

}

int main(int argc, char **argv)

{

  ros::init(argc, argv, "listener");

  ros::NodeHandle n;

  /* La crida subscirbe() es com li diu a ROS que vol rebre
els missatges d'un tòpic determinat. Això invoca un crida al
Master, que es qui s'encarrega del registre de publicadors i
subscriptors. Els missatges es passaran a una funció
callback, en aquest cas chatterCallback. El subscribe()
retorna un objecte que s'haura de guardar mentre no et
vulguis dessubscribir del tòpic. El segon paràmetre indica el
buffer del subscriptor.

  ros::Subscriber sub = n.subscribe("chatter", 1000,
chatterCallback);

  /*El ros::spin() el fa entrar en un loop, que llençarà els
callbacks. El ros::spin() acabarà quan es tanqui el Master o
se li envií un cnt+c */

  ros::spin();

  return 0;

}
```

El que fa el subscriptor és, al main, subscribir-se al tòpic desitjat de manera que cada cop que hi hagin dades fer un Call Back a una funció tractarà les dades rebudes.

Un cop fet aquesta pinzellada es pot passar a explicar tots els nodes desenvolupats.

9.2. Node Keep Pose

En aquest apartat es mostrarà la implementació i les proves del node Keep Pose.

9.2.1. Implementació Node Keep Pose

La principal tasca d'aquest node serà rebre dades de navegació o be desplaçaments en píxels sobre imatges i enviar consignes de correcció al sistema de navegació i motors del robot.

Per entendre millor el funcionament del node, primer s'explicarà el fitxer de configuració.

```
<keep_pose_config>
    <pid_config_x type="vector" value="1.0 0.0 0.0 "/>
    <pid_config_y type="vector" value="1.0 0.0 0.0 "/>
    <pid_config_z type="vector" value="1.0 0.0 0.0 "/>
    <pid_config_yaw type="vector" value="1.0 0.0 0.1 "/>
    <mode type="integer" value="0"/>
    <desired_pose type="vector" value="0 0 0 0 0 0"/>
</keep_pose_config>
```

Els camps que es poden veure per configurar són:

- Pid_config_? : Un vector amb la configuració de cada pid per a cada grau de llibertat.
- Mode type: Per indicar si el mode de treball serà amb una posició absoluta o amb correccions captades del tòpic píxel_correction.
- Desired_pose: Posició absoluta desitjada.

Un cop explicat el fitxer de configuració s'entendrà millor l'explicació d'aquest node.

Per treballar el node tindrà un main subscrit al tòpic pixel_correction i al navigation_data, que és per on ens arribaran les correccions en píxels o les posicions del robot.

Quan arribin dades del tòpic navigator_data, s'invocarà aquest Call Back:

```
Void navigationCallBack( const cola2_common::NavigationData&
navigationData )

{

    navigation = navigationData ;

    if( xmlParamMapper.data().mode == 1 ){

          cola2_common::BhResponse response;

          error = computeErrorPosition() ;

          response.twist.linear.x = -1.0 * roundf(
pidX_.action( error(0) ) * 10.0 ) / 10.0 ;

          response.activationLevels[0] = true ;

          response.twist.linear.y = -1.0 * roundf(
pidY_.action( error(1) ) * 10.0 ) / 10.0 ;

          response.activationLevels[1] = true ;

          response.twist.linear.z = roundf( pidZ_.action(
error(2)  ) * 10.0 ) / 10.0 ;

          response.activationLevels[2] = true ;

          response.twist.angular.x = 0 ;

          response.activationLevels[3] = false ;

          response.twist.angular.y = 0 ;

          response.activationLevels[4] = false ;

          response.twist.angular.z =  roundf( pidYaw_.action(
error(5) ) * 10.0 ) / 10.0  ;

          response.activationLevels[5] = true ;

          response.priority = 10 ;

          pub.publish( response ) ;

    }

}
```

El que aquest Call Back sempre farà es guardar-se les dades actuals de navegació. I en el cas de que el mode de treball sigui el 1, calcularà les consignes a partir de la posició actual de robot i la posició desitjada.

Per fer-ho es guardà al vector error l'error computat per la funció computeErrorPosition(). Aquesta funció calcularà els desplaçaments en x, y i z i la rotació en yaw, tenint en comte els canvis d'orientació.

```
Eigen::Matrix3d R = Eigen::Matrix3d::Zero() ;
R( 0, 0 ) =  cos( navigation.pose[5] ) ;
R( 0, 1 ) = -sin( navigation.pose[5] ) ;
R( 1, 0 ) =  sin( navigation.pose[5] ) ;
R( 1, 1 ) =  cos( navigation.pose[5] ) ;
R( 2, 2 ) =  1 ;
Eigen::Matrix3d rTrans = R.transpose() ;
```

El primer que farà la funció es preparar la matriu de rotació homogènia sobre l'eix Z (sobre l'únic eix en el que el robot pot rotar) i la transposarà per al futurs càlculs.

```
Eigen::Vector3d p ;
p( 0 ) = navigation.pose[0] ;
p( 1 ) = navigation.pose[1] ;
p( 2 ) = 0.0 ;
Eigen::Vector3d aux = -rTrans * p ;
```

Tot seguit es posa a un vector els punts x i y de la posició actual de navegació i es guarda a una variable auxiliar aquests la multiplicació d'aquests amb la matriu transposada negativa.

```
Eigen::Matrix4d T = Eigen::Matrix4d::Zero() ;
for( unsigned int i = 0; i < 3 ; i++ )
for( unsigned int j = 0; j < 3 ; j++ ) T( i, j ) = rTrans( i, j ) ;
for( unsigned int i = 0; i < 3 ; i++ ) T( i, 3 ) = aux( i ) ;
T( 3, 3 ) = 1 ;
```

Tot seguit es construeix la matriu de transformació homogènia del sistema.

```
Eigen::Vector4d pDesired = Eigen::Vector4d::Zero() ;

pDesired( 0 ) = xmlParamMapper.data().desiredPose( 0 );

pDesired( 1 ) = xmlParamMapper.data().desiredPose( 1 ) ;

pDesired( 3 ) = 1.0 ;

Eigen::Vector4d distance = T * pDesired ;

distance( 2 ) = navigation.pose[2] - xmlParamMapper.data().desiredPose( 2
);

distance( 3 ) = cola2::algebra::normalizeAngle( navigation.pose[5] -
xmlParamMapper.data().desiredPose( 5 ) ) ;
```

Un cop tenim la matriu de transformació homogènia, es calcularà els desplaçaments que s'han d'aplicar a cada grau de llibertat per obtenir la posició desitjada respecte la posició del robot. Per obtenir la x i la y, s'ha de multiplicar la matriu homogènia amb les x i y desitjades. I per obtenir la z i el yaw, com són valors absoluts, amb una resta és suficient.

Un cop fet això es retornarà el vector distance, que contindrà l'error computat. Amb aquest error computat i els pid's es calculen les consignes que s'hauran d'aplicar al sistema de control dels motors.

Quan arribin dades del tòpic píxel_correction s'executarà un altre Call Back.

En primer lloc, si arriba un missatge de stop de correccions, es pararà el procés.

Si no fos el cas de stop, el primer que farà és, el primer cop, guardar-se les dades de navegació que faran falta, la z i el yaw.

```
if ( firstTime_ ) {

    firstTime_ = false ;

    profunditat = navigation.pose[2] ;

    orientacio = navigation.pose[5] ;

}
```

I després es calcularan les consignes que s'hauran d'enviar. Com que les correccions són en píxels, es tindrà en compte la distància ja que no és el mateix un píxel a dos metres de distància que a 2 centímetres.

```
response.twist.linear.x = -1.0 * roundf( pidX_.action(
desviation.correction.position.x ) * 10.0 ) / 10.0 ;

response.activationLevels[0] = true ;

response.twist.linear.y = -1.0 * roundf( pidY_.action(
desviation.correction.position.y ) * 10.0 ) / 10.0 ;

response.activationLevels[1] = true ;

response.twist.linear.z = roundf( pidZ_.action( profunditat -
navigation.pose[2]  ) * 10.0 ) / 10.0 ;

response.activationLevels[2] = true ;

response.twist.angular.x = 0 ;

response.activationLevels[3] = false ;

response.twist.angular.y = 0 ;

response.activationLevels[4] = false ;

response.twist.angular.z =  roundf( pidYaw_.action( orientacio -
navigation.pose[5] ) * 10.0 ) / 10.0  ;

response.activationLevels[5] = true ;
```

El que es fa es anar emplenant el missatge de resposta obtenint les consignes fent servir pids. Els graus de llibertat que no es fan servir es deixen desactivats.

Un cop fets tots els càlculs, es dona una prioritat al paquet i s'envia.

```
response.priority = 10 ;

pub.publish( response ) ;
```

D'aquesta manera, a mesura que s'envien les correccions, s'anirà col · locant allà on es configuri al robot.

9.2.2. Proves Node Keep Pose

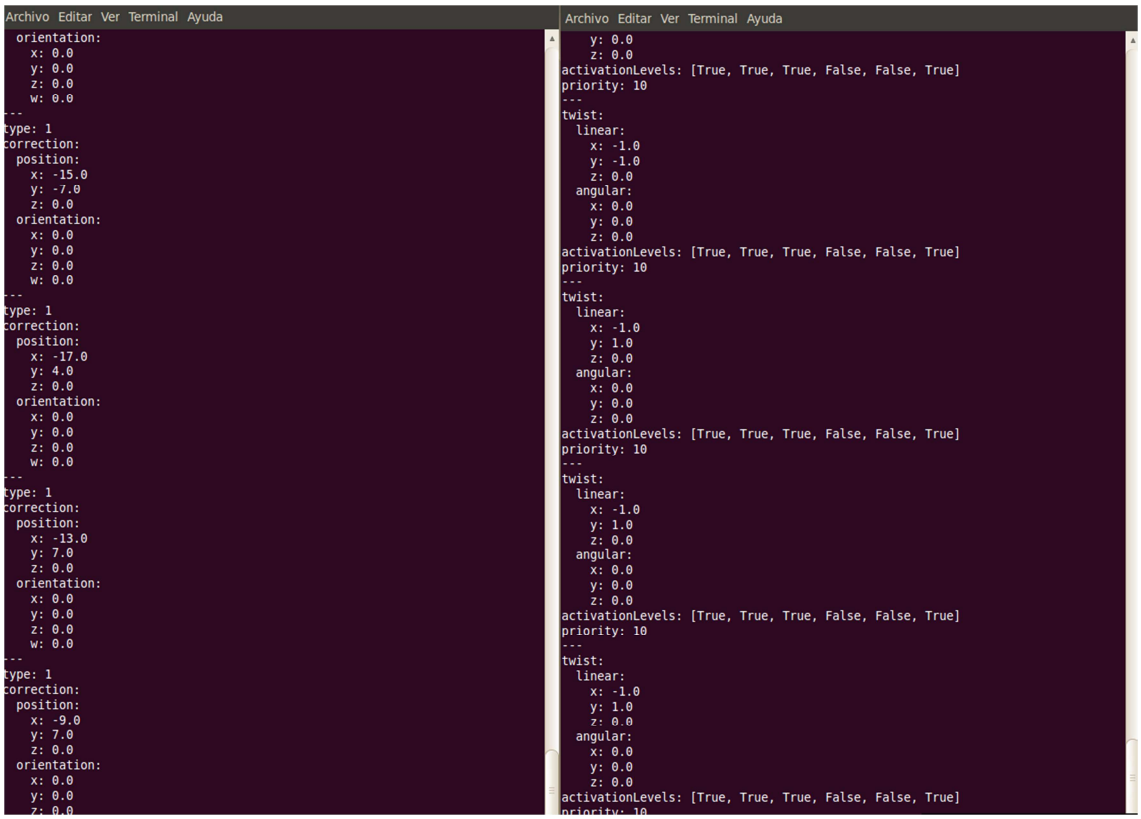


Figura 28: Exemple Keep Pose en marxa

Es pot observar (Figura 28) a la consola de l'esquerra les dades que estan arribant al node (el tòpic píxel_correction). A la consola de la dreta es pot observar les consignes que està enviant el node al sistema de navegació i control del robot.

Com que les dades són coherents, les proves ratifiquen el correcte funcionament del node.

9.2.3. Experiments amb la UJI

Amb els companys de la UJI (Universitat Jaume I) es va fer un experiment en que la UdG (Universitat de Girona) s'encarregava del control del robot i la UJI del control d'un braç robot instal · lat al robot i una càmera (Figura 29).

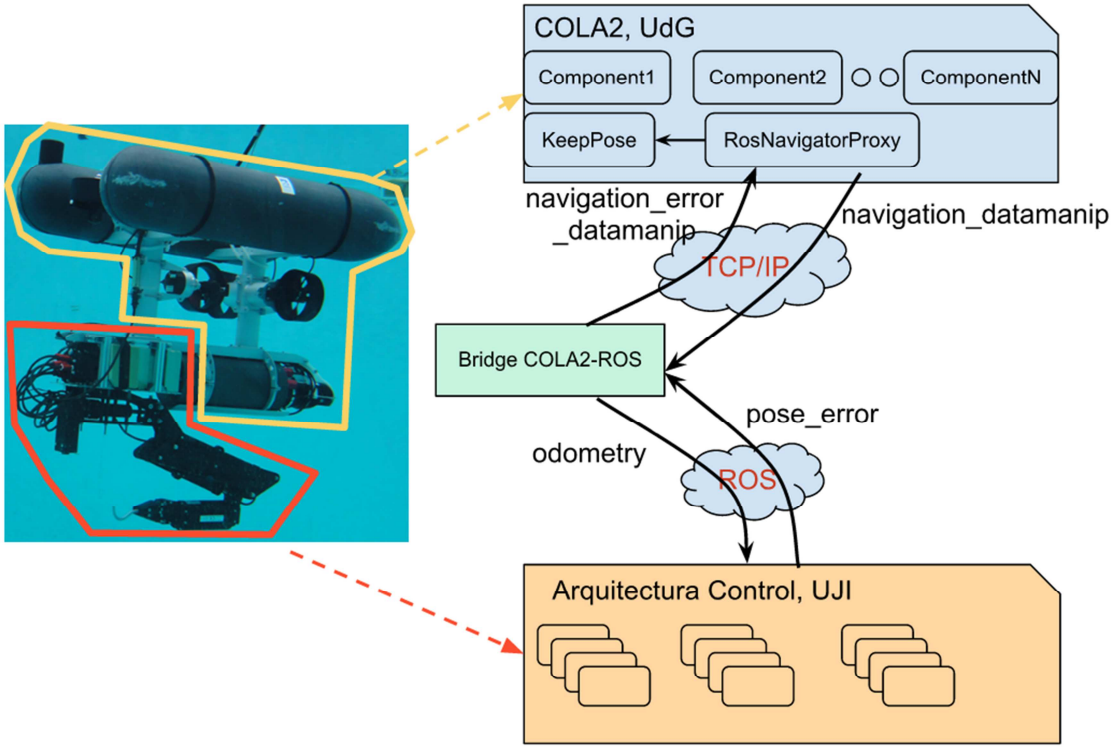


Figura 29: Esquema Experiment UDG-UJI

Del que tractava l'experiment era de que, per mitjà de l'Arquitectura de Control de la UJI, detectar amb una càmera i un sistema de control per visió per computador una caixa negra.

Després aquest sistema transmetia els errors en posició per tal que el robot es situés a sobre de la caixa. El sistema de la UdG rebia aquests errors, els computava i els transmetia al sistema de navegació i control de motors del robot per tal de posicionar el robot on el sistema de la UJI transmetia.

Per últim, quan el robot estava col · locat a la posició desitjada, el sistema de la UJI enviava la senyal al braç robot per a que recollís la caixa negra (Figura 30.)

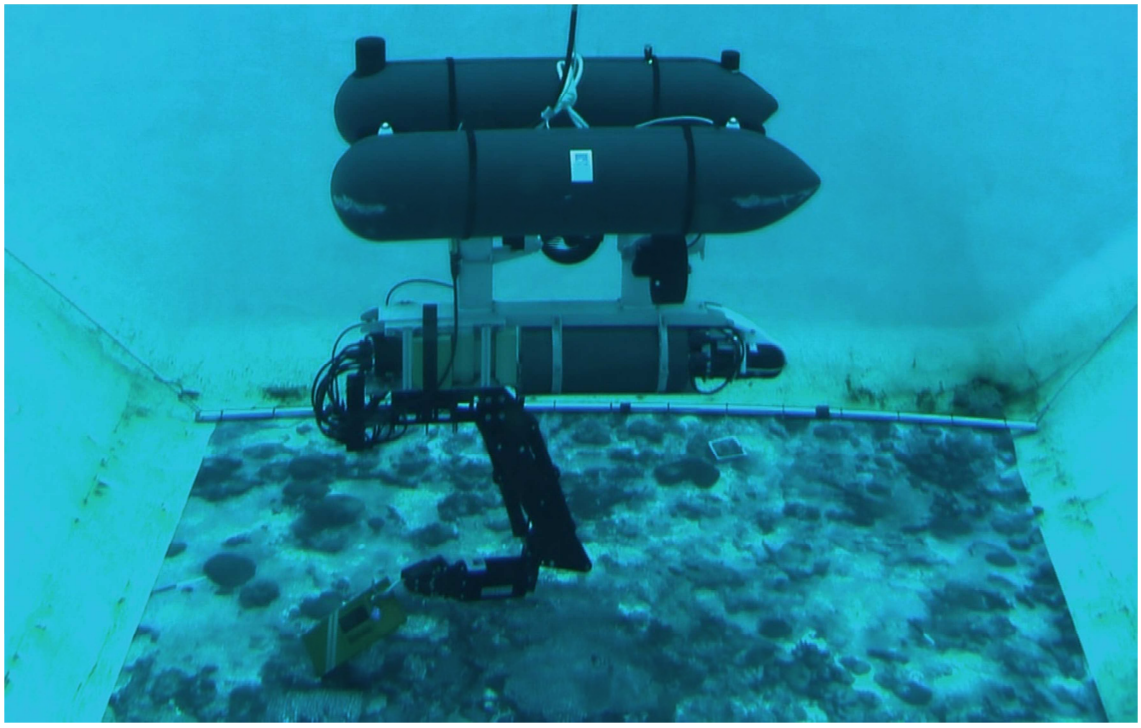


Figura 30: Girona600 recollint caixa negra

En aquest experiment es va fer servir tant el pont COLA2-ROS com el Keep Pose.

9.3. Driver de la càmera

En aquest apartat s'explicarà la implementació i es mostraran proves fetes amb la càmera.

9.3.1. Implementació del driver de la càmera

Per entendre i fer-nos una millor idea de com pot ser el codi del driver de la càmera, es mostrarà i s'explicarà primer el seu fitxer de configuració:

```
<camera_config>
     <video_device type="integer" value="0" />
     <window_name type="string" value="camera" />
     <width_crop type="integer" value="550" />
     <height_crop type="integer" value="450" />
     <enable_crop type="integer" value="0" />
     <enable_fps type="integer" value="1" />
     <enable_calibration type="integer" value="1" />
</camera_config>
```

Per ordre, els camps que ens hi es poden trobar són:

- Video_device: escull el dispositiu de video d'entrada.
- Windos_name: el nom de la finestra on es mostraran les imatges.
- Width_crop: l'ample de retall.
- Height_crop: l'altura de retall.
- Enable_crop: si s'habilita o no fer crops en la imatge amb l'ample i l'altura introduïts.
- Enable_fps: si s'habilita o no que es mostrin els frames per segon.
- Enable_calibation: si s'habilita o no que les imatges es vegin calibrades.

Un cop es sap quines dades ens fan falta per a la configuració, es passarà a repassar dels aspectes més importants del codi.

Primerament, dintre de declarar e inicialitzar totes les variables, caldrà que es declari al tòpic on publicarà el nostre node.

```
image_transport::ImageTransport it(n);

image_transport::Publisher cameraPub = it.advertise("imagesFromCamera", 1);
```

D'aquesta manera s'indicarà que tenim un publisher d'imatges que publicarà a imagesFromCamera.

Tot seguit es carregarà el fitxer de configuració.

```
if( ros::param::get("camera_config", xmlConfig) ) {  ROS_INFO("configfile =
%s", xmlConfig.c_str() );

}else { ROS_INFO("Couldn't get 'camera_config' param."); }

      // Map the XML configuration to an struct

cola2::xml::XmlParamMapper<DeviceConfig> xmlParamMapper;

xmlParamMapper.mapItem("video_device", &DeviceConfig::videoDevice);

xmlParamMapper.mapItem("window_name", &DeviceConfig::windowName);

xmlParamMapper.mapItem("width_crop", &DeviceConfig::widthCrop);

xmlParamMapper.mapItem("height_crop", &DeviceConfig::heightCrop);

xmlParamMapper.mapItem("enable_crop", &DeviceConfig::enableCrop);

xmlParamMapper.mapItem("enable_fps", &DeviceConfig::enableFPS);

xmlParamMapper.mapItem("enable_calibration",&DeviceConfig::enableCalibraion);

xmlParamMapper.loadXmlString(xmlConfig);
```

D'aquesta manera es tindran totes les dades dintre d'un struct i seran més accessibles.

Un cop es tenen les dades de configuració ja es pot seleccionar el dispositiu d'entrada de video.

```
VideoCapture capture(xmlParamMapper.data().videoDevice) ;
```

I també la finestra per on visualitzar les imatges.

```
cvNamedWindow( xmlParamMapper.data().windowName.c_str() );
```

Per a obtenir les dades, es redirigeix el del dispositiu cap a una matriu d'openCV.

```
capture >> cameraFrame ;
```

Si fos necessari retallar la imatge, és faria d'aquesta manera [12]:

```
cvSetImageROI( &IplFrame, cvRect( 75 , 25 ,
xmlParamMapper.data().widthCrop , xmlParamMapper.data().heightCrop )
) ;

IplImage *frameCropped = cvCreateImage( cvGetSize(&IplFrame),
IplFrame.depth, IplFrame.nChannels ) ;

cvCopy( &IplFrame, frameCropped , NULL ) ;

cvResetImageROI( &IplFrame ) ;
```

El que es fa és, fixar-se en una determinada regió de la captura original, copiar-la a una matriu auxiliar i deixar l'original intacta per si fes falta.

Si fos necessari calibrar la imatge, primer de tot seria necessari carregar la matrius intrínseca i la de distorsió de la càmera [13].

```
CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics.xml");

CvMat *distortion = (CvMat*)cvLoad("Distortion.xml");

mapx = cvCreateImage( cvGetSize(&IplFrame), IPL_DEPTH_32F, 1 );

mapy = cvCreateImage( cvGetSize(&IplFrame), IPL_DEPTH_32F, 1 );

cvInitUndistortMap(intrinsic,distortion,mapx,mapy);
```

Carregar els fitxers amb les matrius i crear els filtres.

```
IplImage *t = cvCloneImage(&IplFrame);

cvRemap( t, &IplFrame, mapx, mapy );

cvReleaseImage(&t);
```

I així s'apliquen a la imatge [14].

Un cop es té la imatge tractada, estarà preparada per ser publicada.

```
bridge.fromIpltoRosImage(&IplFrame , img , "bgr8" ) ;

cameraPub.publish(img) ;
```

Primer, amb el pont s'empaqueten la imatge dintre d'un paquet d'imatges ROS i després es publiquen[15].

A abans d'acabar es mostrà la imatge. Quan es vagin mostrant les imatges crearan la sensació de video amb la successió d'aquestes.

```
drawFPS( &IplFrame , IplFrame.width-180, IplFrame.height-10 ) ;
imshow(xmlParamMapper.data().windowName.c_str(), &IplFrame) ;
waitKey(5) ;
```

Primer s'imprimeixen els frames per segon, es mostra per la finestra creada i s'espera a que es refresqui la imatge.

Amb la repetició d'aquest procés s'anirà capturant, publicant i mostrant les imatges de la càmera.

9.3.2. Proves amb la càmera.

Simplement, s'ha de veure com captura imatges des de un dispositiu de video (Figura 31).

Figura 31: Exemple de Captura de la Càmera

9.4. Node de cal · libració

En aquest apartat es mostrarà la implementació del Node de cal · libració i les proves fetes per verificar el seu funcionament.

9.4.1. Implementació del Node de cal · libració

El main segueix la estructura general d'un subscriptor, en aquest cas d'imatges.

```
image_transport::ImageTransport it( nh ) ;

image_transport::Subscriber sub = it.subscribe( "imagesFromCamera" ,
1 , imageCallback ) ;
```

La funció imageCallback s'executarà cada cop que arribin imatges pel tòpic.

S'agafarà una de cada x imatges per tal de que no totes les imatges siguin iguals i es pugi anar canviant la perspectiva de les captures. Cada cop que es tracti una imatge, el que es farà es:

```
int found = cvFindChessboardCorners(image, board_sz, corners,
&corner_count, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS );

cvCvtColor(image, gray_image, CV_BGR2GRAY);

cvFindCornerSubPix(gray_image, corners, corner_count, cvSize(11,11)
,cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1
));

cvDrawChessboardCorners(image, board_sz, corners, corner_count, found);
```

Buscar els corners i en una imatge en blanc i negre comprovar si la captura dels cornes és prou precisa, ja que si no ho fos podria alterar el resultat de la cal · libració. Després pintarà la ubicació de cada corner dels quadrats del tauler [16].

Un cop fet això s'acumularan aquestes dades en dues matrius per a futur tractament de les dades.

```
for( int i=step, j=0; j<board_total; ++i,++j ) {

    CV_MAT_ELEM(*image_points, float,i,0) = corners[j].x;

    CV_MAT_ELEM(*image_points, float,i,1) = corners[j].y;

    CV_MAT_ELEM(*object_points,float,i,0) = (float) j/board_w;

    CV_MAT_ELEM(*object_points,float,i,1) = (float) (j%board_w);
    CV_MAT_ELEM(*object_points,float,i,2) = 0.0f;

}

CV_MAT_ELEM(*point_counts, int,successes,0) = board_total;
```

Un cop es tenen suficients captures de cantonades, es copien aquestes matrius a unes de noves d'acord amb el numero efectiu de dades obtingudes.

Després ve la part més important de la funció.

```
for(int i = 0; i<successes*board_total; ++i){

ROS_INFO("\n\ntest1\n");

    CV_MAT_ELEM( *image_points2, float, i, 0) = CV_MAT_ELEM(
*image_points, float, i, 0);

    CV_MAT_ELEM( *image_points2, float,i,1)   = CV_MAT_ELEM(
*image_points, float, i, 1);

    CV_MAT_ELEM(*object_points2, float, i, 0) = CV_MAT_ELEM(
*object_points, float, i, 0) ;

    CV_MAT_ELEM( *object_points2, float, i, 1)= CV_MAT_ELEM(
*object_points, float, i, 1) ;

    CV_MAT_ELEM( *object_points2, float, i, 2)= CV_MAT_ELEM(
*object_points, float, i, 2) ;

}

for(int i=0; i<successes; ++i){

    CV_MAT_ELEM( *point_counts2, int, i, 0)= CV_MAT_ELEM(
*point_counts, int, i, 0);

}
```

```
CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 ) = 1.0f;

CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 ) = 1.0f;

cvCalibrateCamera2(object_points2, image_points2,
point_counts2, cvGetSize( image ), intrinsic_matrix,
distortion_coeffs, NULL, NULL,0 );
```

Primer s'inicialitzen els valors intrínsecs de les dues longituds focals a 1 i després s'executa la funció que emplenarà la matriu intrínseca i la de distorsió [17].

Finalment es guarden aquestes matrius a dos fitxers .xml per a utilitzar-les posteriorment.

9.4.2. Proves node de cal · libració

A l'hora de provar-ho s'han fet unes quantes captures de pantalla per observar el seu funcionament.

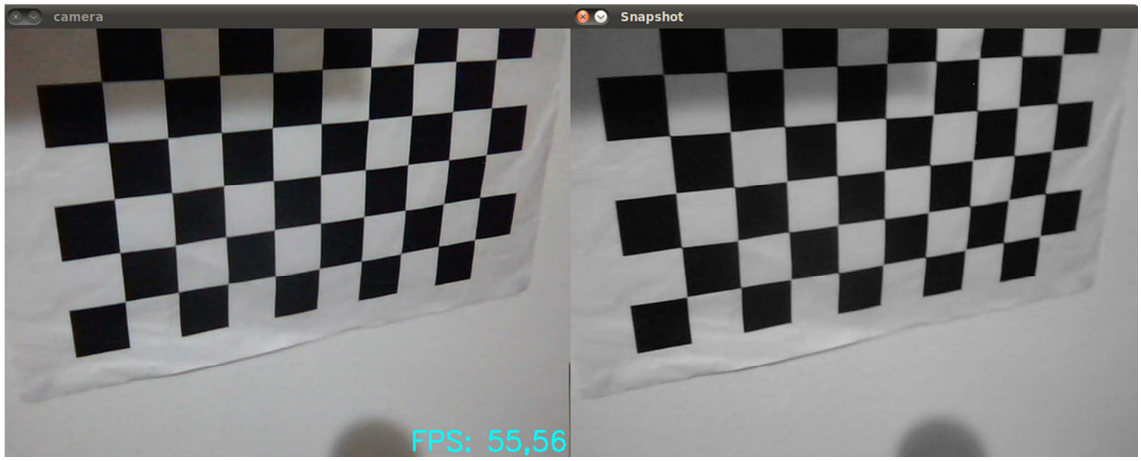


Figura 32: Exemple de cal · libració de la càmera no detectant cantonades

Es pot veure (Figura 32) com el node (finestra snapshot) obté les imatges de la càmera, i en aquest cas com no detecta totes les cantonades correctament en mostra la imatge en blanc i negre.

Figura 33: Exemple de cal · libració de la càmera detectant cantonades

Aquí (Figura 33), com que detecta les cantonades del tauler ens les pinta i interiorment va guardant la seva ubicació i les relacions de distancia entre elles.

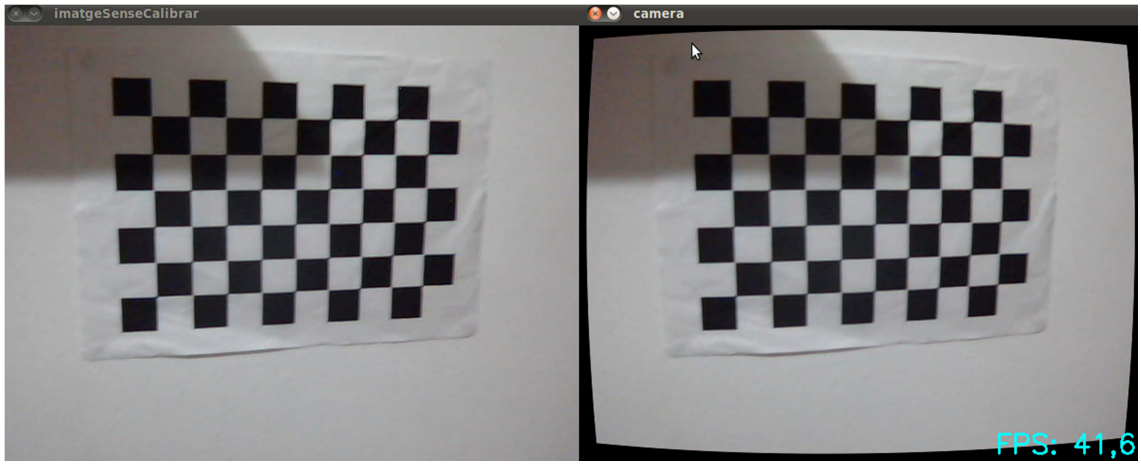


Figura 34: Exemple comparació càmera no calibrada i calibrada

I aquí (Figura 34) finalment es pot veure les imatges sense calibrar i la càmera calibrada.

9.5. Node de Processament d'Imatges

Després de fer el anàlisis dels Node de Template Matching i SURF, donat que la tasca que realitzaven és similar, tant d'entrada com de sortida, es va decidir a l'hora d'implementar-los fer-ho tot dintre d'un mateix node de manera que, segons el que es demanes al fitxer de configuració, utilitzar un mètode o un altre.

Així que a l'hora d'explicar aquest node, primer s'explicaran les parts compartides i després s'especificarà el que es fa en cada apartat segons l'algorisme.

9.5.1. Implementació Node de Processament d'Imatges

Per entendre millor el funcionament del node, primer s'explicarà el fitxer de configuració de manera que després es farà més entenedor.

```
<visual_servoing_config>
    <!-- mod 0 = SURF, mode 1 = template matching /-->
    <mode type="integer" value="0" />
    <algorisme type="integer" value="1" />
    <renovar type="integer" value="1" />
    <!-- 0 = from posicioActual, 1= Centre -->
    <respecte type="integer" value="0" />
    <templateDeFitxer type="integer" value="0">
</visual_servoing_config>
```

Els camps que es poden veure per configurar són:

- Mode: ens indicarà si es vol fer SURF o Template Matching.
- Algorisme: Dintre del Template Matching les OpenCV ens ofereixen 6 mètodes, que en el seu apartat s'explicaran degudament.
- Renovar: s'indicarà si en el Template Matching es vol anar renovant el template o no (es preferible fer-ho).
- Respecte: S'indicarà si es vol que el node ens mantingui el Template a la posició original o si es vol centrar-lo al centre de captura de les imatges.

- TemplateDeFitxer: Indica si es carregarà o no el Template d'un fitxer. Aquest fitxer haurà d'estar ubicat a la carpeta de d'on s'executi el Shell de comandes, amb el nom "template.jpg".

Un cop explicat això, s'explicarà a la part del codi.

Aquest node serà del tipus subscriptor amb publicacions, de manera que el main es subscriurà al tòpic imagesFromCamera i es publicaran dades al tòpic pixel_correction.

```
image_transport::ImageTransport it( nh ) ;

image_transport::Subscriber sub = it.subscribe( "imagesFromCamera" ,
1 , imageCallback ) ;

pub = nh.advertise<cola2_common::VisualServoingData>
("pixel_correction", 1);
```

Un cop definit això, cada cop que arribi una imatge es cridarà una funció Call Back que farà tota la feina.

Aquesta funció, el primer que farà quan l'usuari fixi un Template o es carregui de fitxer, serà guardar-lo.

```
firstTime = false ;

if( xmlParamMapper.data().templateDeFitxer == 0){

cvSetImageROI( img, cvRect( src_corners[0].x , src_corners[0].y ,
abs(src_corners[2].x - src_corners[0].x) , abs(src_corners[2].y-
src_corners[0].y) ) ) ;

tpl = cvCreateImage( cvGetSize(img), img->depth, img->nChannels ) ;

cvCopy( img, tpl , NULL ) ;

cvResetImageROI( img ) ;

}else{

tpl = cvLoadImage("template.jpg", CV_LOAD_IMAGE_COLOR);

}
```

O es fixa la imatge a la regió seleccionada, es copia el Template i es posa la imatge com estava originalment o es carrega directament del "template.jpg".

Després, cada algorisme farà la seva feina i quan tinguin les correccions en píxels calculades, es disposaran a enviar-les. Per fer-ho, cridaran la funció

templatefinded, que li enviaran els corners d'on ha trobat en aquella iteració el template i aquest calcularà les correccions. A cada apartat s'explicarà que fa cada algorisme per empaquetar les dades.

I d'aquesta manera s'aniran enviant les correccions per disposar l'escena com s'hagi indicat al fitxer de configuració.

9.5.2. Implementació Template Matching

L'algorisme del Template Matching està basat en una funció de les OpenCV, la cvMachTemplate [18], que pot fer servir 6 mètodes diferents per a trobar un Template dintre d'una imatge. Aquests mètodes són:

- Diferencia de quadrats: CV_TM_SQDIFF
- Diferencia de quadrats normalitzada: CV_TM_SQDIFF_NORMED
- Correlació creuada: CV_TM_CCORR
- Correlació creuada normalitzada: CV_TM_CCORR_NORMED
- Coeficient de correlació: CV_TM_CCOEFF
- Coeficient de correlació normalitzat: CV_TM_CCOEFF_NORMED

Per a més informació del funcionament d'aquest mètodes es pot trobar a la Wiki de Willow Garage.

I així es com es selecciona el mètode escollit en el fitxer de configuració.

```
IplImage *res = cvCreateImage( cvSize( rect.width - tpl->width + 1,
rect.height - tpl->height + 1 ), IPL_DEPTH_32F , 1 );

if( xmlParamMapper.data().algorisme == 0 )  cvMatchTemplate( img , tpl ,
res , CV_TM_SQDIFF ) ;

else if ( xmlParamMapper.data().algorisme == 1 ) cvMatchTemplate( img , tpl
, res , CV_TM_SQDIFF_NORMED ) ;

else if ( xmlParamMapper.data().algorisme == 2 ) cvMatchTemplate( img , tpl
, res , CV_TM_CCORR ) ;

else if ( xmlParamMapper.data().algorisme == 3 ) cvMatchTemplate( img , tpl
, res , CV_TM_CCORR_NORMED ) ;

else if ( xmlParamMapper.data().algorisme == 4 ) cvMatchTemplate( img , tpl
, res , CV_TM_CCOEFF ) ;

else if ( xmlParamMapper.data().algorisme == 5 ) cvMatchTemplate( img , tpl
, res , CV_TM_CCOEFF_NORMED ) ;

CvPoint minloc , maxloc ;

double minval , maxval ;

cvMinMaxLoc( res , &minval , &maxval , &minloc , &maxloc , 0 ) ;
```

Seguidament es crea un imatge de resposta d'un canal, es busca el template a la imatge actual i després es busquen els màxims i mínims valors d'ubicació calculats per el cvMatchTemplate amb la funció cvMinMaxLoc [19]. Quan es tracta dels dos primers mètodes, interessaran els valors mínims del resultat, per als altres mètodes interessaran els valors màxims. En aquesta posició es on estarà ubicat el template.

Si els valors mínims o màxims (depenent del cas) són 50 cops més alt que contrari, el valor serà fiable. Quan el valor sigui fiable s'aniran acumulant encerts i s'anirà renovant el template. Si el resultat no és tant fiable (50 cops o menys) indicarà que s'està anant de la regió de la imatge a on es troba el template, així que s'anirà augmentant la resolució de cerca.

Un cop controlat això, és pot cridar la funció templateFinded per a calcular les correccions en píxels i publicar les dades.

```
if( qualResultat >= 25 ){

     response.correction.position.x = posicioDesitjada.x  - (
corners[0].x + ( ( corners[2].x - corners[0].x ) / 2 ) ) ;

     response.correction.position.y = posicioDesitjada.y - (
corners[0].y + ( ( corners[2].y - corners[0].y ) / 2 ) ) ;

     response.type = 1 ;

     pub.publish ( response ) ;

}
```

Si la fiabilitat del resultat és major de 25, nomes s'haurà de restar la posició desitjada de l'actual per obtenir els desplaçaments ens x i y (graus de llibertat ens els que pot treballar aquest algorisme), indicar el tipus de resposta i publicar les dades.

9.5.3. Proves Template Matching

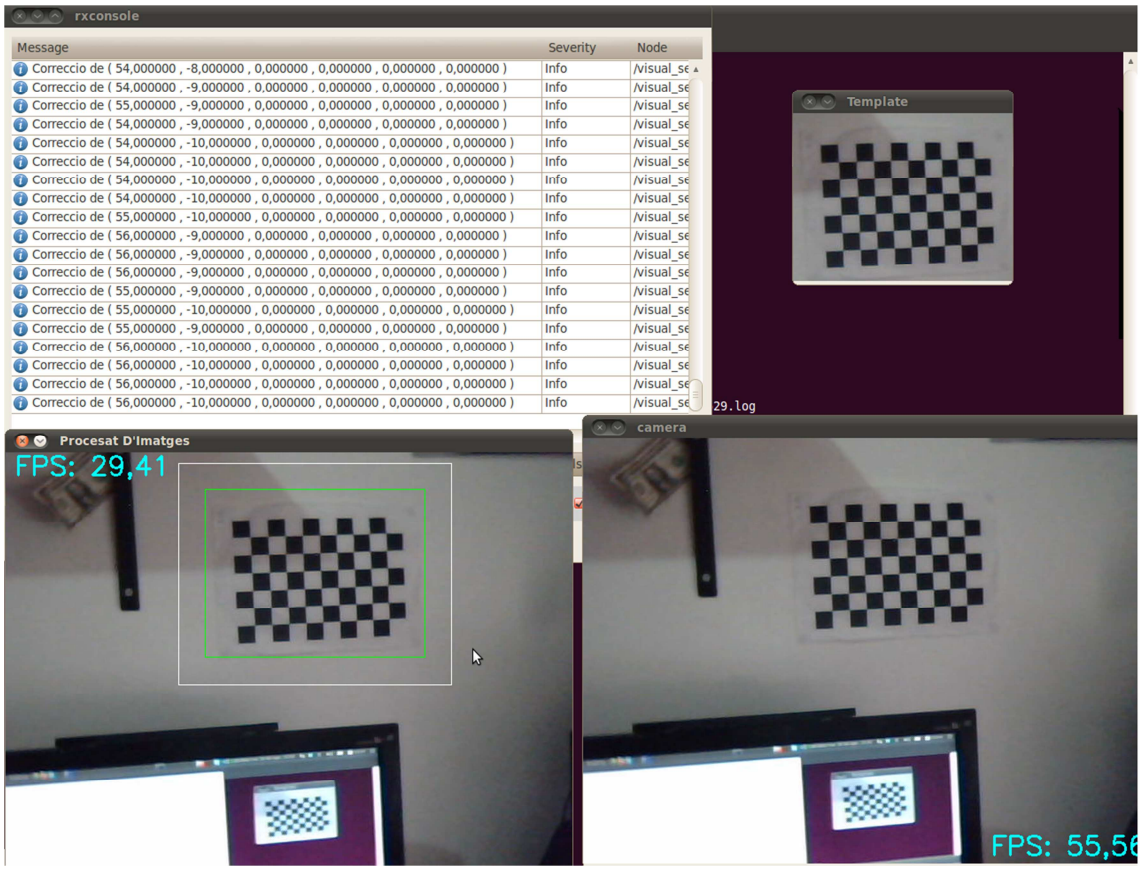


Figura 35: Exemple Template Matching en marxa

Es pot observar (Figura 35) a la finestra Template el template escollit, a la finestra de la càmera el que esta capturant la càmera i a la finestra de Processat d'Imatges on està trobant l'algorisme el template (al requadre verd) i la resolució on esta buscant (al requadre blanc). A la finestra de l'rxconsole es poden veure les correccions en píxels que està publicant el node.

Amb les proves que s'han fet (com es mostra a la Figura 35), es pot corroborar el correcte funcionament del node.

9.5.4. Implementació SURF

Donada la complexitat d'aquest algorisme i les facilitats que ens ofereixen les OpenCV, es va fer servir una llibreria anomenada OpenSURF desenvolupada per Cris Evans de La universitat de Bristol [20], que inclou les funcions necessàries per a la tasca que es vol portar a terme.

Primerament, quan es el cas de es fa servir l'algorisme de SURF, quan es guarda el Template s'ha de guardar el vector amb els descriptors d'aquest.

```
surfDetDes(tpl, ref_ipts, false, 3, 4, 3, 0.004f);
```

D'aquesta manera s'està indicant el vector (ref_ipts) on es volen guardar els punts d'interès del Template (tpl). El false indicarà que la rotació no es invariant. La resta de valors, per orde, indicaran el numero d'octaves per calcular, intervals per octava, step inicial i valor del thresold del blob [20].

Llavors, quan la funció Call Back arriba al punt on l'algorisme de SURF treballa, el que fa és obtenir el vector amb els punts d'interès de la imatge actual.

```
surfDetDes(img, ipts, false, 3, 4, 3, 0.004f);
```

I tot seguit buscar les coincidències entre el vector de referencia i l'actual [20].

```
getMatches(ipts,ref_ipts,matches);
```

Quan es tenen les coincidències es procedeix a ubicar a on es troba en la imatge actual el template [20].

```
CvPoint dst_corners[4];

nCornersMin = translateCorners(matches, src_corners,
dst_corners) ;

if ( nCornersMin )   {

     for(int i = 0; i < 4; i++ ){

           CvPoint r1 = dst_corners[i%4];

           CvPoint r2 = dst_corners[(i+1)%4];

           cvLine( img, cvPoint(r1.x, r1.y),   cvPoint(r2.x,
r2.y), cvScalar(255,255,255), 3 );      }
```

Si la funció translateCorners [20] troba els 4 corners serà capaç d'ubicar el template, i llavors s'emmarcarà en la imatge actual.

Després d'això es procedirà a calcular i enviar les correccions. Per fer-ho es cridarà la funció templateFinded.

Aquesta funció, quan es tracti de l'algorisme SURF, primer comprovarà que el rectangle sigui un rectangle ja que, la funció translateCorners, posiciona quatre punts en la imatge indicat a on ell identifica el template. Però aquest algorisme calcula transformacions afins que inclouen rotació, escalat i shear, i hi haurà casos en que detecti 4 cantonades però no siguin correctes, ja que a nosaltres només ens hauria de trobar rectangles, ja que els graus de llibertat que no siguin translacions van controlats per la brúixola i el sensor de pressió i no haurien de variar.

```
if (    nCornersMin  && variation( 2.0 , (src_corners[1].x -
src_corners[0].x) / (src_corners[3].y -
src_corners[0].y),(corners[1].x - corners[0].x) / (corners[3].y -
corners[0].y))  && variation( 2.0 , (src_corners[2].x -
src_corners[3].x) / (src_corners[2].y -
src_corners[1].y),(corners[2].x - corners[3].x) / (corners[2].y -
corners[1].y)) ){
```

El que fa aquest condicional, primer de tot comprovar que te les quatre cantonades i després que les cantonades que ens ha trobat l'algorisme siguin correctes (ja que pot trobar coincidències però no ubicar bé del tot la imatge, o que la qualitat de les coincidències no sigui prou bona). Per fer-ho fa servir la funció variació:

```
bool variacio( float tolerancia, float referencia, float valor){

     return ( (referencia*(1.0 + (tolerancia/100.0)) >= valor) &&
(referencia*(1.0 - (tolerancia/100.0)) <= valor) ) ;

}
```

Amb la que es comprovarà de que el rectangle respecti les proporcions de longitud entre els costats paral · lels.

Un cop s'ha validat el rectangle obtingut, procedeix a calcular les correccions en píxels.

```
response.correction.position.x = posicioDesitjada.x - ( ( (
corners[0].x + ( ( corners[1].x - corners[0].x ) / 2 ) ) + (
corners[3].x + ( ( corners[2].x - corners[3].x ) / 2 ) ) ) / 2 ) ;

response.correction.position.y = posicioDesitjada.y - ( ( (
corners[0].y + ( ( corners[3].y - corners[0].y ) / 2 ) )  + (
corners[1].y + ( ( corners[2].y - corners[1].y ) / 2 ) ) ) / 2 ) ;

response.correction.position.z = ((src_corners[1].x- src_corners[0].x)
-(corners[1].x - corners[0].x)+( src_corners[2].x - src_corners[3].x )
-(corners[2].x - corners[3].x )+(src_corners[3].y - src_corners[0].y )
-(corners[3].y - corners[0].y )+( src_corners[2].y - src_corners[1].y)
- ( corners[2].y - corners[1].y ) ) / 4 ;

response.correction.orientation.z = ( sin( ( src_corners[2].y -
src_corners[3].y ) / ( corners[2].x - corners[3].x ) )
+ sin( ( src_corners[1].y - src_corners[0].y ) / ( corners[1].x -
corners[0].x ) ) ) / 2;

response.type = 0 ;

pub.publish ( response ) ;
```

Per a fer-ho en x i y, resta a la posicion desitjada a la ubicació del rectangle en aquest grau de llibertat (fa mitja ja que amb aquest algorisme els costats no tenen per que mesurar el mateix).

Per a fer-ho en z calcula l'amplada de cada costat de l'original i li resta la del rectangle actual, de manera que sabrà si la imatge creix (s'allunya) o disminueix (s'apropa).

I per calcular la orientació calcula la desviació que hi ha respecte els 0 graus, ja que en una primera instancia, quan s'agafa el Template, aquest no te rotació.

Aquests dos últims càlculs es realitzen però al final és fan servir les dades de navegació del sensor de pressió i la brúixola, ja que aquests sistemes són mes robustos.

Un cop fet els càlculs, s'indica de quin tipus es el paquet i es publica.

Amb això s'aniran donant les correccions en píxels per a tots els grau de llibertat del robot.

9.5.5. Proves SURF

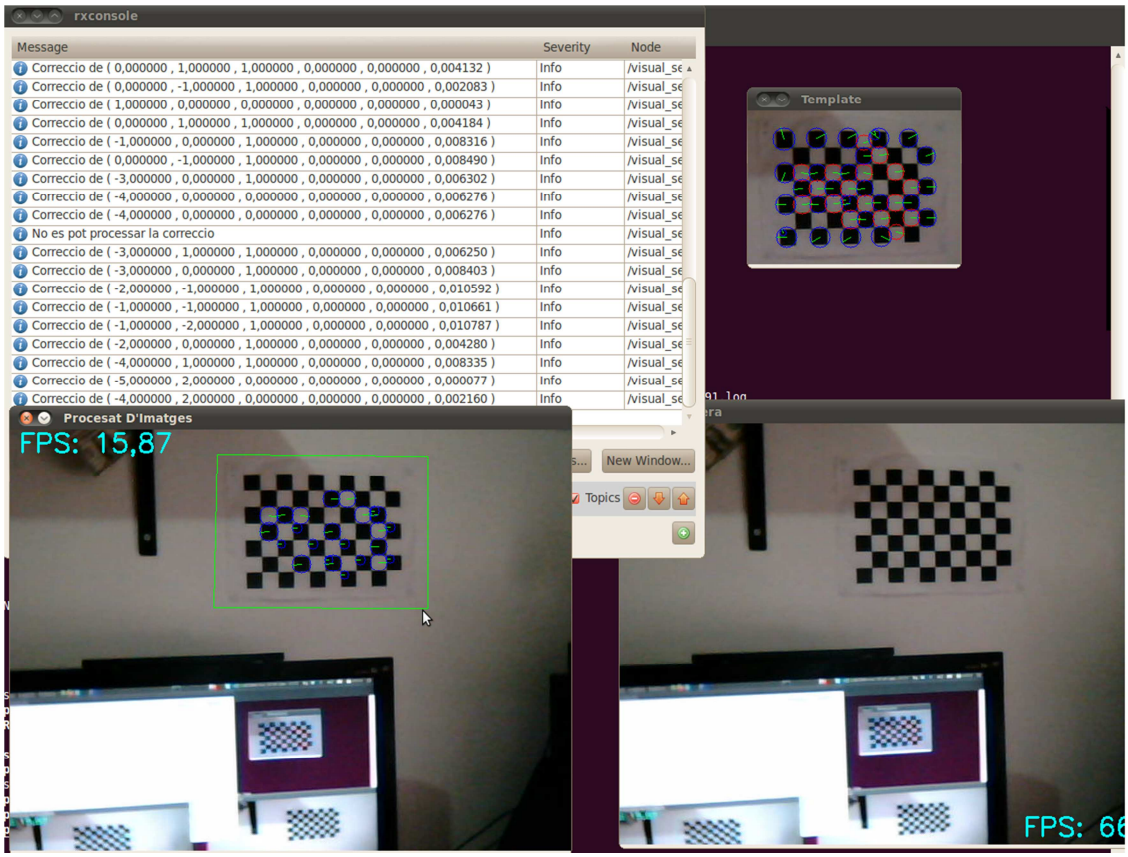


Figura 36: Exemple SURF en marxa

Es pot observar (Figura 36) a la finestra Template el template escollit i els punts d'interès que ha trobat l'algorisme i s'ha guardat a la taula de referencia, a la finestra de la càmera el que esta capturant la càmera i a la finestra de Processat d'Imatges on està trobant el Template al requadre verd. A la finestra de l'rxconsole es poden veure les correccions en píxels que està publicant el node.

Amb les proves que s'han fet (com es mostra a la Figura 36), es pot corroborar el correcte funcionament del node.

CAPÍTOL 10

10. Resultats

10.1. Resultats obtinguts

Per a obtenir resultats més seriosos i estudiar el comportament del sistema, és van capturar resultats fent exactament les mateixes proves amb l'algorisme de Template Matching i tots els seus possibles mètodes i amb l'algorisme de SURF, de tal manera que els resultats poguessin ser comparables.

S'han fet servir 3 .bag diferents amb un video d'uns 30 segons i sempre fent servir el mateix template per a tots els algorismes.

Per fer les proves també s'han fet servir dos equips diferents, un Intel Atom (el model de processador del robot) i un Quad Core (connectat en xarxa amb el Master de ROS i únicament processant imatges). D'aquesta manera es podrà repensar l'opció de si es vol alliberar de la feina costosa que suposa tractar imatges al processador de robot per a que tingui més recursos per als altres mòduls.

10.2. Video 1

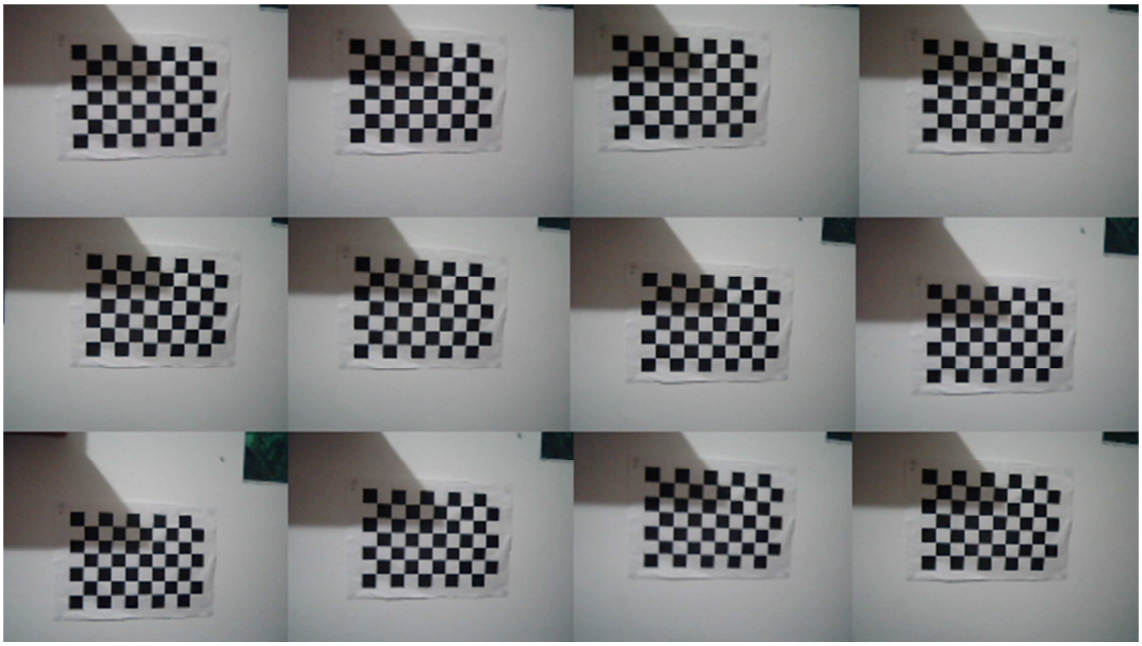
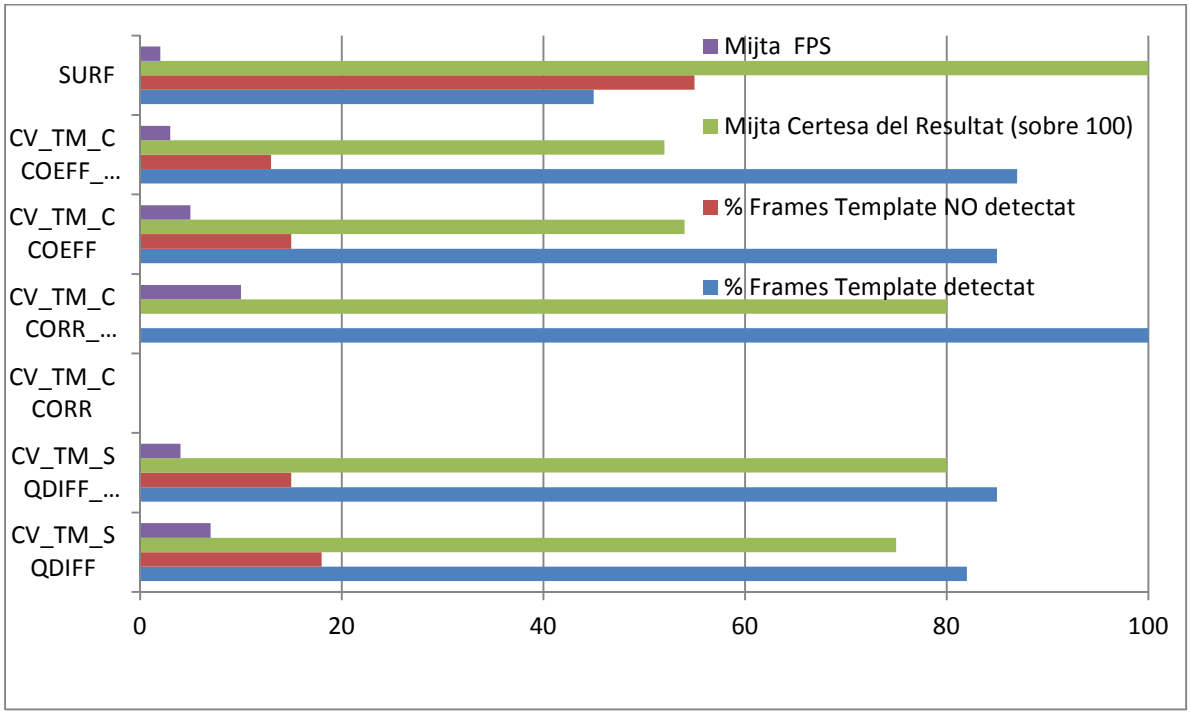


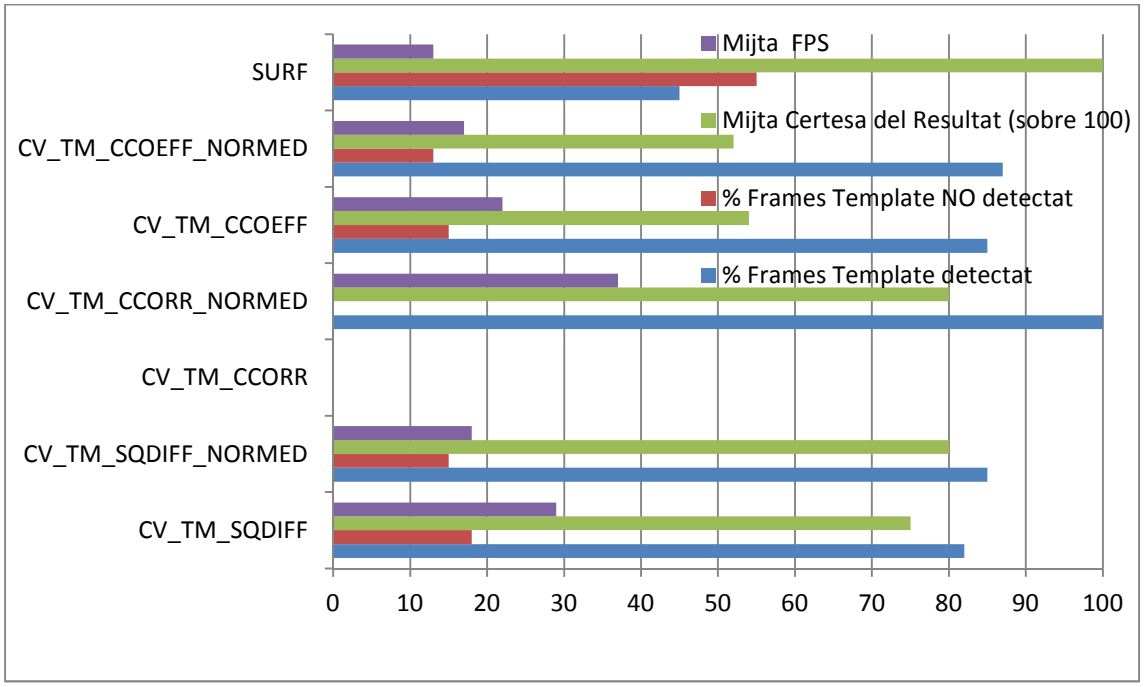
Figura 37: Mostra video 1

En aquesta prova es farà un seguiment d'una porció del tauler d'escacs (Figura 37), el mateix emprat per a la cal · libració de la càmera. Serà un video al qual hi hauran pocs color (la majoria sobre l'escala de grisos) i les imatges tindran moltes cantonades.

10.2.1. Atom



10.2.2. Quad



10.2.3. Comentaris Video 1

Comentar que l'única diferencia entre els processadors es que la quantitat de frames per segon que es tracten, amb el quad, es pràcticament quatre vegades més, cosa que pot ser molt important per a tenir en compte.

La primera cosa que s'ha de comentar es que el tercer i el quart algorisme (els CCORR), donen resultats incongruents, per tant no es tindran en compte en futurs experiments. Els resultats que donaven, comprovant-ho visualment amb la interfície d'usuari, eren incorrectes.

El primer i el segon algorisme (SQDIFF), detecten el template a més del 80% dels casos i amb una certesa del resultat d'un 75%, però quan perden el template es costa massa tornar-lo a identificar.

El cinquè i el sisè (CCOEFF) són també molt bons, identifiquen el template sobre el 85% dels cops, però quan perden el template donen per bona l'àrea que més se li assembla al template i això són encerts que es compten i no s'haurien de tenir en compte, encara que quan el template torna a l'escena l'identifiquen al moment.

El SURF es poder el que té una taxa d'encerts més baixa, sobre el 45%, però sempre que identifica el template ho fa amb un 100% de certesa, cosa molt important a valorar. També s'ha de tenir en compte que es l'únic capaç de detectar els errors en z i yaw (encara que no ho es faci servir).

10.3. Video 2

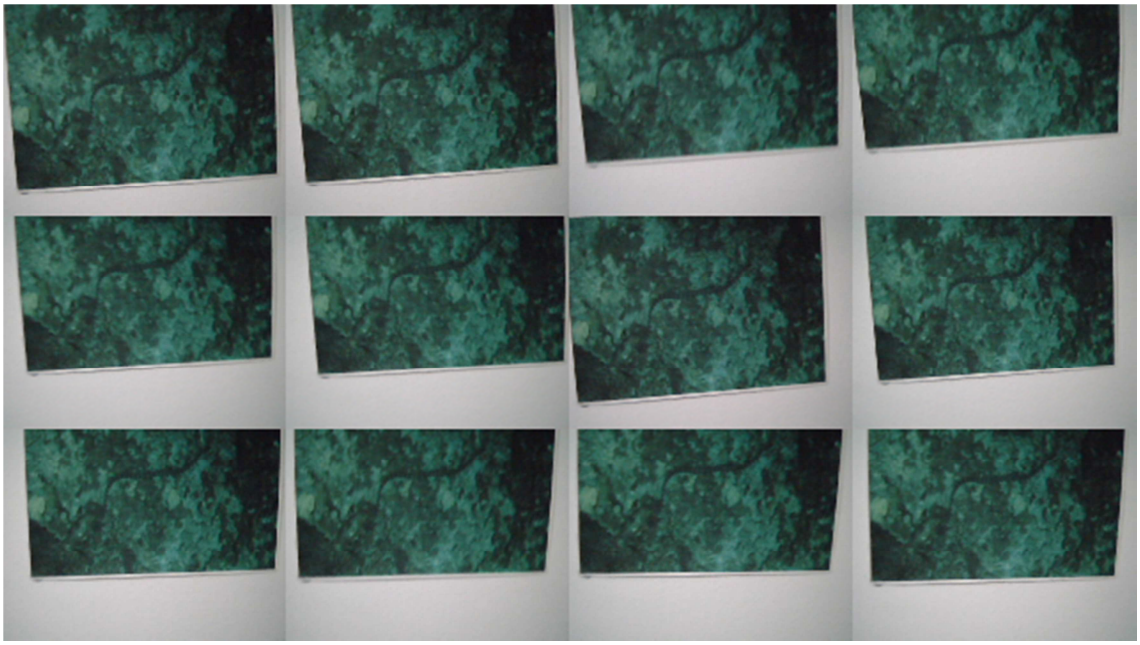
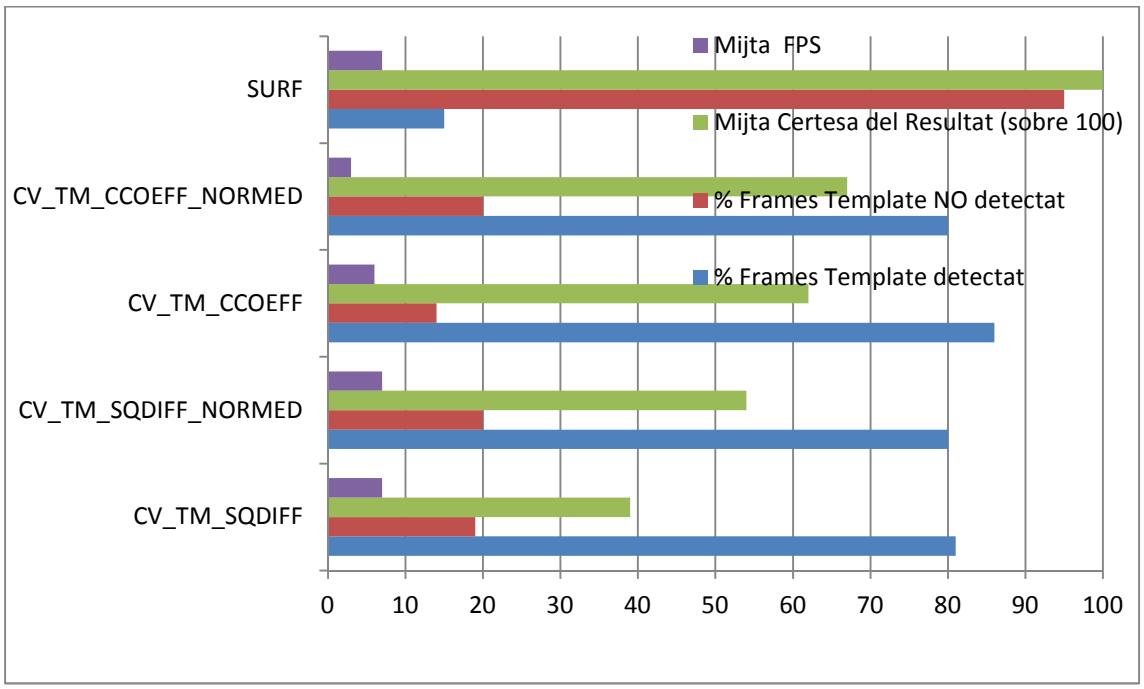


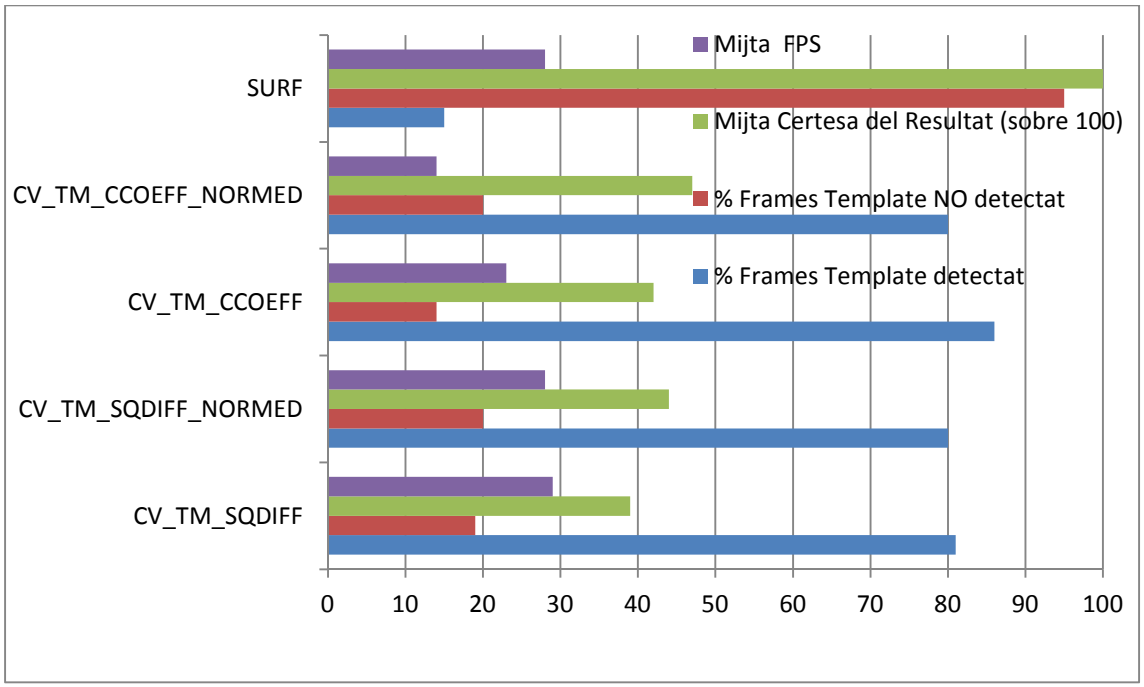
Figura 38: Mostra video 2

En aquesta prova es farà un seguiment d'una porció d'un pòster d'un fons marí (Figura 38). Serà un vídeo al qual hi hauran molts colors, una repetitivitat que farà que quan s'agafi una porció de Template, inclús fent servir la vista humana, ens costaria detectar de quina regió es tracta.

10.3.1. Atom



10.3.2. Quad



10.3.3. Comentaris Vídeo 2

En aquesta prova, tots els algorismes de Template Matching han donat més o menys els mateixos resultats, bones taxes d'encerts (sobre el 80%), encara la certesa del resultat no és gaire bona (sobre el 40%).

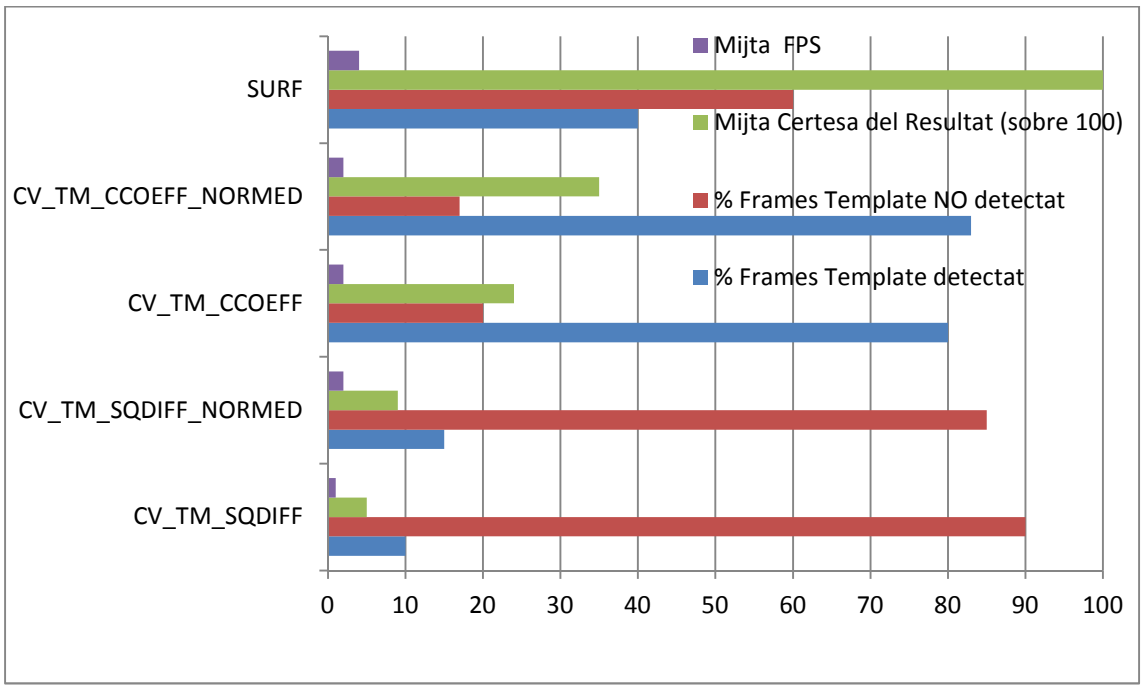
El que ha tingut més dificultats ha sigut el SURF ja que, al ser una imatges amb pocs pics i les regions molt semblants, no ha pogut capturar masses punts d'interès i li ha sigut bastant difícil localitzar el template.

10.4. Video 3

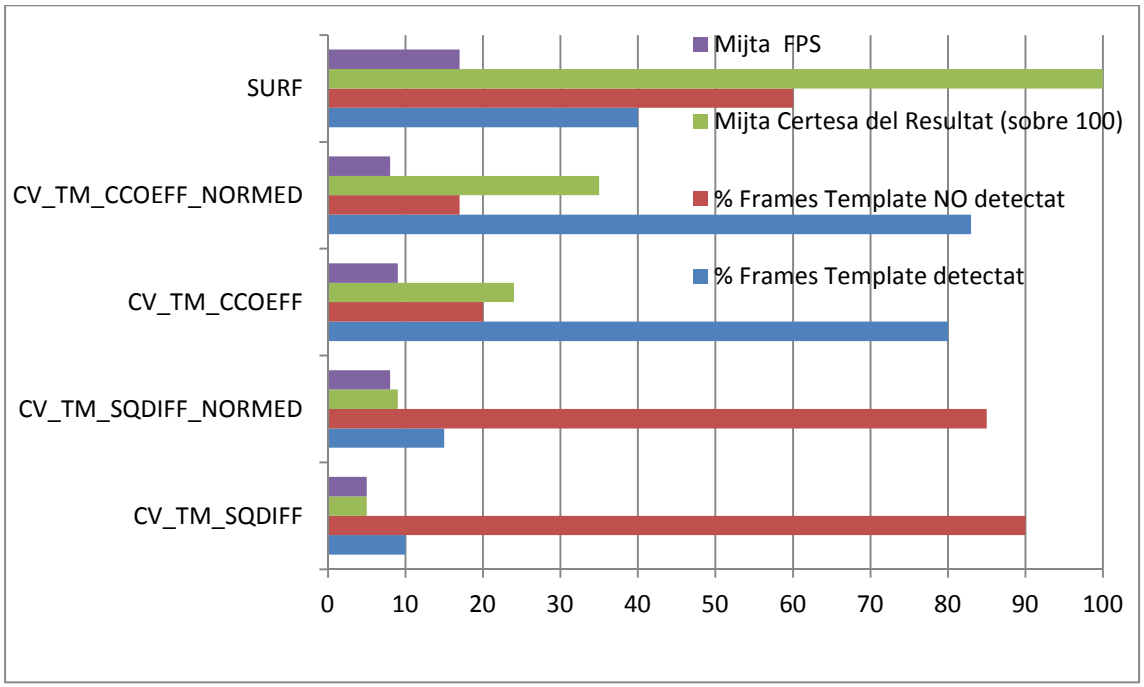
Figura 389: Mostra video 3

La tercer prova es va fer seguint objectes d'una prestatgeria (Figura 39). Es una imatge amb molts colors, molts objectes i per afegir-li més dificultat, a la càmera se li van fer moviments bruscos perquè els algorismes tinguessin més dificultats a l'hora de ubicar el template.

10.4.1. Atom



10.4.2. Quad



10.4.3. Comentaris Vídeo 3

Es pot veure que en aquests casos, el CV_TM_SQDIFF i el CV_TM_SQDIFF_NORMED no ens serveix ja que pràcticament no troben el template, a més de que tracten molts pocs frames per segon-

El CV_TM_CCOEFF i el CV_TM_CCOEFF_NORMED tenen molt bona taxa d'encerts (d'un 80%), però la certesa del resultat i els frames per segon tractats deixen molt que desitjar.

I el SURF és el millor algorisme per aquesta prova. Ha trobat el template al 40% dels frames tractats i en tots l'ha identificat. A més, en aquesta prova, tractat més del doble de frames per segon que els altres algorismes.

CAPÍTOL 11

11. Conclusions

L'objectiu principal del projecte s'ha assolit satisfactòriament, s'ha implementat un comportament per a robot Girona500 en la versió ROS de l'arquitectura COLA2 que ens permet realitzar control en posició del robot utilitzant la informació tant de navegació com provinent d'una càmera.

Aquest objectiu, s'ha subdividit en tres parts.

La primera part del projecte era la desenvolupar un driver per a la càmera del robot, cosa que s'ha aconseguit, a més d'haver desenvolupat un mòdul extra que permet calibrar la càmera. Amb les proves realitzades s'ha corroborat l'assoliment de l'objectiu a més de verificar el seu correcte funcionament.

La segona part del projecte i segurament la que més feina i importància se li ha donat, ha sigut implementar el software que processa les imatges de la càmera. Per fer-ho s'han fet servir dos mètodes:

- El Template Matching, que ha consistit en agafar una imatge com a referencia i buscar on coincidia aquesta referencia en les imatges que han anat arribant de la càmera per tal d'ubicar la imatge de referència.
- El SURF d'imatges, que ha consistit en agafar una imatge i obtenir els seus punts d'interès per a fer-los servir com a referencia. Després s'ha anat buscant els punts d'interès de referencia a les imatges que han anat arribant de la càmera, per tal de trobar les coincidències i ubicar la imatge de referència.

Aquests dos mètodes, després de fer experiments i proves, s'han arribat a les següents concussions:

- El Template Matching és molt útil en escenes en les que el robot no es veu molt afectat per les corrents d'aigua (això implicarà que hi no hagi molt de moviment) i ens dona molt bones taxes de frames per segon. Aquest algorisme te de dolent que, quan no troba correctament el template, sempre buscarà la coincidència més semblant i possiblement en alguns casos acabi per enganyar amb les correccions. També una cosa molt important que s'ha de tenir en compte es que aquest algorisme només podrà donar correccions els graus de llibertat x i y del robot, i per a corregir el de z i yaw fa falta ajuda d'altres mòduls externs al nostre, cosa que amb SURF no és necessari.

- El SURF es molt útil en escenes amb molts objectes i moviments bruscos, no dona tan bones taxes de frames per segon com el Template Matching (encara que en els casos en els que el Template Matching falla, el SURF pot doblar les taxes de FPS del Template Matching) però això es veu compensat per el fet de que mai falsos positius, o troba la referencia o no la troba, a més dona correccions en tots els graus de llibertat del robot.

A l'hora de fer servir aquest comportament s'ha de ser crític i saber quin algorisme pot anar millor segons l'escena que es necessiti tractar. Això serà fàcil i ràpid ja que escollir un mètode o l'altre es simplement canviar una dada en el fitxer de configuració abans d'executar el mòdul.

També s'ha de tenir en compte que les tècniques de tractament i processament d'imatges constitueixen un cost computacional molt important, però una facilitat que ofereix ROS es tenir més d'una maquina en marxa comunicada per xarxa, de manera que aquestes feines es poden portar a terme en ordinadors més potents, tal i com s'ha fet en les proves. El Girona500 porta un processador adequat per a un AUV ja que és suficientment potent per tenir el marxa tots els mòduls que un vehicle de les seves característiques ha de tenir i consumir el mínim possible. Les imatges per segon que pot processar amb tota l'arquitectura en funcionament, en el pitjor dels casos son de dos o tres imatges per segon, mes que suficient per a poder oferir una qualitat de funcionament mínima.

La tercera part i última, ha estat desenvolupar un comportament que faci moure el robot en funció de l'error en posició detectat per mòdul de tractament d'imatges o les dades de navegació. Amb les proves realitzades es pot comprovar que aquest mòdul tracta les dades de correccions i envia les consignes als motor d'acord amb l'error a corregir i ens els graus de llibertat que pertoca.

CAPÍTOL 12

12. Treball futur

A continuació es proposen diversos treballs futurs que poden dur-se a terme un cop finalitzat aquest projecte, a fi i efecte de donar continuïtat la feina que s'ha fet:

- Proves en mar obert: Les proves realitzades s'ha fet en un entorn artificial com és el de una piscina, en la que els corrents d'aigua són pràcticament inexistents, i seria molt positiu per al projecte poder provar-lo en condicions més reals com en mar obert.

- Seguiment d'objectes: Un pas més en el projecte podria ser no només mantenir una escena sino que, poder seguir objectes en moviment dintre de l'aigua.

- Augment de FPS tractats: Un aspecte que podria millorar es fer servir algorismes menys costosos a nivell computacional o més optimitzats.

CAPÍTOL 13

13. Bibliografia

[1] – Documentation: ROS Wiki – http://www.ros.org/wiki/

[2] – OpenCV Wiki – http://opencv.willowgarage.com/wiki/

[3] – Wikipedia, Software, Modelo Iterativo incremental – http://es.wikipedia.org/wiki/Software#Modelo_iterativo_incremental

[4] – Web CIRS – http://eia.udg.es/~pere/Pere_Ridao_Home_Page/CIRS.html

[5] – C++ Reference – http://www.cplusplus.com/reference/stl/

[6] – The Eclipse Foundation open source community website – http://www.eclipse.org/

[7] – Git: Fast Version Control System – http://git-scm.com/

[8] – Gantt Project – http://www.ganttproject.biz/

[9] – Dia a drawing progam – http://projects.gnome.org/dia/

[10] - SURF Wikipedia - http://en.wikipedia.org/wiki/SURF

[11] – ROS Tutorials Writing Publisher Subscriber(c++) – http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

[12] – OpenCV Region of Interest – http://nashruddin.com/OpenCV_Region_of_Interest_%28ROI%29

[13] – CameraCalibration and 3d Reconstruction: cvInitUndistortMap – http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html?highlight=undistort#initUndistortRectifyMap

[14] – Geometric Image Transformations OpenCV: cvRemap – http://opencv.willowgarage.com/documentation/cpp/geometric_image_transformations.html#cv-remap

[15] – CvBridge Class Reference – http://www.ros.org/doc/api/cv_bridge/html/c++/classsensor___msgs_1_1CvBridge.html

[16] – Camera Calibration and 3d Reconstruction: cvFindChessboardCorners – http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html?highlight=chessboard#findChessboardCorners

[17] – Camera Calibration and 3d Reconstruction: cvCalibrateCamera2 – http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html?highlight=calibrate#calibrateCamera

[18] – Object Detection: cvMatchTemplate –

http://www.larmor.com/projects/JavaOpenCVMatchTemplate/doc/com/larmor/opencv/MatchTemplate.html
http://opencv.willowgarage.com/documentation/cpp/imgproc_object_detection.html?highlight=matchtemplate#matchTemplate

[19] – Operations on Arrays: cvMinMaxLoc – http://opencv.willowgarage.com/documentation/cpp/core_operations_on_arrays.html#minMaxLoc

[20] – OpenSURF – The Official Home of the Image Processing Library - http://www.chrisevansdev.com/computer-vision-opensurf.html

[21] – rxconsole - ROS Wiki – http://www.ros.org/wiki/rxconsole

CAPÍTOL 14

14. Annexos

En el CD adjunt s'hi podran trobar els següents annexos del projecte:

- Stack de ROS amb els nodes implementats.
- PDF de la memòria.
- PDF del resum.

CAPÍTOL 15

15. Manual d'usuari i/o instal · lació

Per a d'instal · lació del projecte, primer farà falta instal · lar ROS. Per fer-ho, directament s'ha de seguir el manual de la Wiki de ROS (l'entorn on s'ha d'instal · lar ha de ser un Ubuntu de la versió 10.4 o superior):

http://www.ros.org/wiki/electric/Installation/Ubuntu

Un cop es té ROS instal · lat, s'ha de posar sistema de fitxers amb el codi adjuntat al CD dintre del Path de ROS.

Després, s'hauria de recompilar. Per fer-ho, desde la carpeta COLA2/cola2_dev executem:

> rosmake –-pre-clean

Un cop es té tot això, es pot procedir a executar el fitxer COLA2.launch ubicat a COLA2/base_config de la carpeta copiada prèviament.

> roslaunch COLA2.launch

Dintre d'aquesta carpeta també es troben els fitxers de configuració per a cada node. Estan ubicats a COLA2/base_config/config. Dintre de cada fitxer està indicada la utilitat de cada paràmetre de configuració.

Un cop el sistema està en marxa, s'han de tenir en compte les següents instruccions:

- Si es vol fer servir un Template des de fitxer, aquest haurà d'estar a la mateixa carpeta des d'on el Shell executi el fitxer COLA2.launch.
- Si es vol fer servir un Template en temps d'execució de les imatges capturades per la càmera, aquest haurà de ser seleccionat de la finestra Processat d'Imatges.
- Si es vol calibrar la càmera, s'haurà d'executar el fitxer CalibrateCamera.launch. Està ubicat a la mateixa carpeta que el fitxer COLA2.launch.
- Si és vol aturar el sistema (sigui quin sigui el .launch executat), s'haurà de fer un CNTR+C al Shell d'es d'on s'hagi executat.

Si hi ha qualsevol dubte és pot contactar amb l'autor d'aquest projecte al correu electrònic u1063888@correu.udg.edu.