
Sistema d'autolocalització per a robots mòbils mitjançant tècniques de visió per computador

Joan Rodas Cusidó
28-02-2017

*Treball final de grau en eng. informàtica
Tecnologies de la informació*



Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya
Director: Joan Climent Vilaró (ESAII)

Resum

Aquest treball final de grau es basa en la creació d'un sistema d'autolocalització per a robots basat en imatges. S'utilitzarà la biblioteca OpenCV i diferents algoritmes de visió per computador. El sistema haurà de permetre al robot trobar i desplaçar-se cap a un punt de l'entorn a partir de la selecció de l'usuari mitjançant una imatge.

This final project is based on creating an autolocation system for mobile robots using images. Different computer vision algorithms and techniques will be used with the help of the OpenCV library. The final system should be able to allow the robot to find and move to a point in the enviroment dessignated by the user selectiong a region on an image.

Agraïments

En primer lloc, m'agradaria donar les gràcies al director del projecte Joan Climent, per donar-me l'oportunitat de realitzar aquest treball de final de grau.

També m'agradaria donar les gràcies al meu amic Arnau, que em va fer descobrir el món de GNU/Linux i el programari lliure ja fa anys. Això va fer que m'interessés per la informàtica i possiblement va ser un factor important a l'hora d'escollar la carrera anys després. I evidentment, també vull donar les gràcies als meus pares i a la meva família per tot el seu suport.

Índex

1	Introducció i abast	7
1.1	Descripció del problema	8
1.2	Motivació	8
1.3	Actors implicats	8
1.4	Estat de l'art	9
1.4.1	Visió per computador	9
1.4.2	Robòtica	10
1.5	Objectius	11
1.6	Requeriments	11
1.7	Obstacles	12
1.8	Ampliacions	13
1.9	Metodologia	13
1.10	Eines de desenvolupament	13
1.10.1	OpenCV	14
1.11	Eines de seguiment	14
1.12	Mètode de validació	15
2	Planificació i recursos	16
2.1	Planificació temporal	16
2.1.1	Bloc 0: Preparació de l'entorn	16
2.1.2	Bloc 1: Curs de GEP	17
2.1.3	Bloc 2: Desenvolupament del projecte	17
2.1.4	Bloc 3: Preparació de la defensa	19
2.1.5	Diagrames	19
2.2	Recursos	21
2.2.1	Recursos humans	21
2.2.2	Recursos de maquinari	21
2.2.3	Recursos de programari	21
2.3	Desviacions i pla d'actuació	22

3 Disseny i arquitectura	23
3.1 Arquitectura del sistema	23
3.2 Aplicació de proves	24
3.3 Disseny de l'aplicació mòbil	26
4 Tècniques de visió usades	30
4.1 Pre-processat digital d'imatges	30
4.2 Obtenció de keypoints en una imatge	31
4.3 Extracció de característiques	31
4.4 Matching de característiques	32
4.5 Homografia	33
5 Implementació	34
5.1 Programa en python	34
5.1.1 Pre-processat de les imatges	34
5.1.2 Selecció de la regió d'interès	35
5.1.3 Obtenció de keypoints	36
5.1.4 Extracció de característiques	38
5.1.5 Matching i homografia	39
5.1.6 Angle de gir	40
5.2 Aplicació mòbil	41
5.2.1 Capturar imatge	41
5.2.2 Selecció d'imatges de la galeria	41
5.2.3 Selecció de la regió d'interès	41
5.2.4 Enviament de dades al servidor	42
6 Experiments i resultats	43
6.1 Experiments realitzats	43
6.1.1 Comparació detectors de keypoints	43
6.1.2 Comparació detecció i extracció de keypoints	43
6.2 Resultats i comparació d'algorismes	45
6.2.1 Detectors de keypoints	45
6.2.2 Detecció i extracció de keypoints	47
6.2.3 Matching i homografia	49
7 Gestió econòmica	51
7.1 Recursos de programari	51
7.2 Recursos humans	51
7.3 Recursos de maquinari	52
7.4 Costos indirectes	52
7.5 Imprevistos	53
7.6 Contingència	53
7.7 Costos totals	53

7.8 Control de gestió	53
8 Informe de sostenibilitat	54
8.1 Posada en producció	55
8.1.1 Consum del disseny	55
8.1.2 Factura	55
8.1.3 Impacte personal	55
8.2 Vida útil	56
8.2.1 Petjada ecològica	56
8.2.2 Pla de viabilitat	56
8.2.3 Impacte social	56
8.3 Riscos	57
8.3.1 Riscos ambientals	57
8.3.2 Riscos econòmics	57
8.3.3 Riscos socials	57
9 Conclusions	58
9.1 Conclusions tècniques	58
9.2 Conclusions personals	58
9.3 Treball futur	59
Apèndix	60
A Codi dels experiments	61
Bibliografia	65
Índex de taules	66
Índex de figures	67

1. Introducció i abast

Aquest projecte es desenvolupa com a treball final de grau dels estudis de grau en enginyeria informàtica, de l'especialitat en tecnologies de la informació. Es tracta d'un projecte de modalitat A, realitzat a la Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya) i proposat pel director Joan Climent, del departament d'ESAI (Enginyeria de Sistemes, Automàtica i Informàtica Industrial).

Els avenços tecnològics dels últims anys, han millorat la capacitat de les màquines per extreure informació i resoldre problemes de manera autònoma, imitant cada vegada millor el comportament humà. En aquest treball, es treballarà la visió per computador aplicada a un problema de robòtica.

El primer capítol serveix com a introducció del projecte, on s'explica l'abast, objectiu, motivació i estat de l'art de les tecnologies a tractar. En el segon capítol es detallen els recursos utilitzats per realitzar el treball.

Els capítols 3 a 6 conformen el treball principal. Al tercer capítol s'explica el disseny i l'aquitectura del sistema desenvolupat. Al quart, les tècniques de visió utilitzades, amb la seva implementació detallada al cinquè capítol. Al capítol 6 s'explicaràn els experiments realitzats i els resultats obtinguts.

Al capítol 7 podeu trobar un ànalisi de la gestió econòmica del projecte, on es detallen els costos humans, de programari, de maquinari, indirectes i possibles imprevistos. I al vuité capítol es presenta l'informe de sostenibilitat. Per acabar, hi haurà les conclusions del projecte, on es valorarà l'aportació del projecte a nivell personal i si s'han aconseguit els objectius inicials proposats.

1.1 Descripció del problema

El treball pretén resoldre un problema d'autolocalització de robots mòbils en un entorn variable, de tal manera que el robot sigui capaç de desplaçar-se d'un punt inicial a un punt final escollit per l'usuari. Per fer això, s'utilitzaran diverses tècniques de visió per ordinador.

1.2 Motivació

Visió per computador i Robòtica van ser sense cap mena de dubte dos de les assignatures més interessants que he cursat a la universitat, així que quan vaig veure la oferta del projecte vaig pensar que seria una bona idea per profunditzar els meus coneixements sobre la matèria.

1.3 Actors implicats

En aquesta secció es descriuen els actors implicats del projecte, és a dir, totes aquelles persones que es veuran beneficiades directa o indirectament amb la realització d'aquest.

- **Autor/Desenvolupador:** És el màxim responsable del projecte. En tractar-se d'un treball final de grau, l'autor del projecte serà també el màxim beneficiari, ja que la realització d'aquest li permetrà acabar la carrera d'enginyeria informàtica.
- **Usuaris:** Qualsevol persona qui ho desitgi, tindrà accés a tots els codis desenvolupats durant el projecte, ja que es llançaran sota una llicència de programari lliure que permetrà veure i adaptar el codi a les necessitats d'altres usuaris.
- **Altres beneficiaris:** Qualsevol empresa o institució interessada podrà utilitzar el sistema desenvolupat i adaptar-lo a les seves necessitats, com podria ser per exemple un sistema de transport d'equipatge basat en robots.

1.4 Estat de l'art

1.4.1 Visió per computador

La visió per computador[1] és una ciència que té com a objectiu dotar les màquines o ordinadors de la capacitat de “veure”. Es basa en l’extracció i anàlisi de dades obtingudes a partir d’imatges.

Algunes de les aplicacions de la visió per computador són:

- Vehicles autònoms
- Realitat augmentada
- Reconeixement facial
- Restauració d’imatges
- Inspecció industrial
- Robòtica

En aquest treball, ens interessa utilitzar la visió per computador en el camp de la robòtica, per aconseguir guiar a un robot mòbil cap a un objectiu determinat basant-se en la detecció d’un punt o regió en una imatge.

Nous algorismes

En els darrers anys, han aparegut nous algorismes d’obtenció de punts i extracció de característiques que suposen una alternativa als clàssics SIFT[2] (Scale Invariant Feature Transform) i SURF[3] (Speeded-Up Robust Features). Alguns d’aquests algorismes són BinBoost[4] o un dels més recents: LATCH[5]

En aquest projecte s’analitzarà si es adequat emprar algun d’aquests algorismes en la implementació del sistema d’autolocalització.

1.4.2 Robòtica

La robòtica és un camp de la tecnologia que estudia el disseny i la construcció de robots.

Que és, doncs, un robot? Al llarg de la història, s'han donat diverses definicions del concepte de robot, sense existir encara una definició exacta acceptada per tothom. I a mesura que passa el temps, cada vegada resulta més complicat determinar si una màquina és o no un robot. Per no complicar-nos massa, entendrem com a robot una màquina programable capaç de realitzar una sèrie de tasques concretes interactuant amb l'entorn, ja sigui de manera automàtica o dirigida.

Existeixen diversos tipus de robots, podent fer una classificació senzilla segons la seva arquitectura: robots mòbils, poliarticulats (industrials, mèdics, etc.), humanoides, zoomòrfics¹ i híbrids.

Els robots mòbils, que són els que ens interessen per aquest projecte, acostumen a tenir una sèrie de sensors i dispositius per permetre'n el desplaçament, la localització, esquivar obstacles i realitzar tasques concretes. Alguns exemples de sensors utilitzats per robots mòbils són:

- Odometria: S'utilitza la informació obtinguda amb sensors de moviment (*encoders* a les rodes, per exemple) per estimar la posició del robot respecte a la inicial.
- GPS (Global Positioning System): Es determina la ubicació del robot amb la xarxa de satèl·lits.
- Sensors de contacte: Permeten detectar si el robot està en contacte amb un altre objecte.
- Sensors d'ultrasons: Detecten objectes mitjançant ones ultrasòniques.
- Acceleròmetre: Determina l'acceleració del robot quan es mou.
- Càmera: Permet capturar imatges de l'entorn.

En el nostre cas, només ens interessaran les dades obtingudes a través d'una càmera, és a dir, les imatges. El treball no se centrarà per tant en la part robòtica del sistema, i no es tindran en compte els sensors i algorismes necessaris per poder moure el robot.

¹**Robots zoomòrfics:** Robots que imiten característiques pròpies de determinats animals.

En cas d'aplicar el sistema desenvolupat en robots en un futur, aleshores s'hauran de tenir en compte altres sensors per permetre el moviment de la màquina i arribar a la destinació evitant obstacles.

1.5 Objectius

L'objectiu principal del projecte consisteix a dissenyar i desenvolupar un sistema d'autolocalització per a robots mòbils.

Aquest sistema estarà basat en tècniques de visió per computador i consistirà, bàsicament, a comparar dues imatges (una global i una altra capturada pel robot) i localitzar un punt o regió seleccionat per l'usuari.

Per arribar a aquest objectiu, es dividirà el treball en diverses fases:

- Estudi dels diferents algorismes de visió existents
- Obtenció de *keypoints* en una imatge
- Extracció de característiques
- *Matching* de dues imatges

1.6 Requeriments

El sistema d'autolocalització implementat ha de complir amb una sèrie de requeriments mínims presentats a continuació:

- L'usuari ha de poder seleccionar un punt o regió d'interès en una imatge donada.
- El sistema ha de ser capaç d'adaptar-se mínimament a diverses condicions de l'entorn (canvis de lluminositat, perspectiva, etc.).

1.7 Obstacles

Durant la planificació i realització del treball, s'hauran de tenir en compte els possibles obstacles que es trobaran. A continuació es detallen alguns dels problemes que es podran trobar.

Noves eines

Un dels principals obstacles serà el fet de treballar amb noves eines i algorismes. Per tal d'evitar problemes en aquest aspecte, caldrà fer una planificació acurada i documentar-se apropiadament. També serà important mantenir una bona comunicació amb el tutor en tot moment, per poder resoldre possibles dubtes referents als algorismes.

Calendari

Un altre obstacle important serà la falta de temps, ja que està previst realitzar el projecte en el transcurs d'un quadrimestre. Gestionar correctament el temps serà clau per aconseguir finalitzar el projecte sense problemes. Per tant, s'haurà de fer una planificació el més realista possible i escollir una metodologia de treball adequada i flexible.

Errors de programació

Com a qualsevol projecte on s'ha de programar, el codi serà una font important d'errors. Per això, caldrà realitzar diverses proves cada vegada que es realitzi una modificació en el codi o s'implementi una nova funcionalitat.

Condicions variables en les imatges

Les imatges capturades a través d'una càmera no presentaran sempre les mateixes condicions. La lluminositat, perspectiva o resolució de la imatge influiran a l'hora de processar les imatges i comparar-les.

Per intentar minimitzar aquests efectes, s'analitzaran diversos algorismes d'obtenció de punts i extracció de característiques. També s'estudiarà si és necessari realitzar un preprocessament o filtratge de les imatges abans d'aplicar els algorismes.

1.8 Ampliacions

Encara que el calendari és força estricta i no hi ha gaire marge d'ampliació, es podria estendre el projecte amb les següents ampliacions:

- Anàlisi del rendiment d'algorismes alternatius per l'obtenció de punts i característiques de les imatges.
- Creació d'una aplicació d'Android que permeti seleccionar un punt o regió d'una imatge.
- Execució del codi del sistema via servidor web, utilitzant les dades enviades per l'aplicació d'Android.

1.9 Metodologia

Per aquest projecte, s'utilitzarà una metodologia de treball àgil amb cicles de desenvolupament curts. Com que només hi ha un desenvolupador, no s'utilitzaran exactament les metodologies Scrum o XP[6] (Extreme Programming), però sí que s'aplicaran moltes de les pràctiques pròpies d'aquestes dues metodologies (proves, simplicitat, refacció de codi, etc.). Això ens donarà més flexibilitat a l'hora de fer canvis i adaptar-nos a una nova planificació.

Es començarà treballant amb imatges de prova (casos senzills) i algorismes coneguts com ara Harris[7] i SIFT. Més endavant, s'aniran introduint modificacions en el codi per intentar aconseguir un sistema capaç de funcionar amb fotografies “reals” i es provaran altres algorismes de visió per computador.

Per altra banda, s'utilitzarà el mètode en cascada per la realització del curs de GEP.

1.10 Eines de desenvolupament

El codi del projecte es desenvoluparà amb python i s'utilitzaran, sempre que sigui possible, eines de programari lliure o de codi obert.

En cas de crear una aplicació per a dispositius Android, es realitzarà mitjançant Android Studio (Java).

1.10.1 OpenCV

Per tal d'utilitzar algorismes de visió per computador en el codi amb relativa facilitat, s'utilitzarà la biblioteca de codi obert OpenCV[8] (Open Source Computer Vision Library), disponible per a python. La versió emprada serà la 3.1.

En concret, hi haurà tres passos indispensables que faran ús d'aquesta biblioteca:

- Obtenció de punts en una imatge
- Extracció de característiques
- *Matching* de dues imatges

1.11 Eines de seguiment

A continuació es detallen les eines de programari usades per fer el seguiment del treball final de grau:

LibreOffice Calc

Per fer un seguiment de les hores dedicades al projecte, es crearà un full de càlcul amb les hores diàries dedicades a cada tasca. S'utilitzarà LibreOffice Calc, inclòs en la *suite* ofimàtica LibreOffice.

Gantt Project

Per tal d'organitzar totes les tasques a realitzar i mirar si hi ha desviacions respecte el pla inicial, s'utilitzarà l'eina de *software* lliure Gantt Project[9]. Aquesta eina ens permetrà realitzar tant un diagrama de Gantt com un diagrama de PERT.

Git + Github

Tot i que no es tracta d'un projecte col·laboratiu (només hi ha un desenvolupador), s'ha decidit utilitzar el sistema de control de versions Git juntament amb la pàgina web Github. D'aquesta manera es facilitarà treballar amb diverses màquines i portar un control dels canvis realitzats. A més, permetrà compartir el codi amb el director amb facilitat.

1.12 Mètode de validació

Es faran validacions parcials durant la realització del projecte, fent proves del sistema amb diverses imatges.

Contacte amb el director

Hi haurà reunions presencials amb el director, així com comunicació via correu electrònic, per tal de resoldre dubtes i validar la feina realitzada. També es realitzarà una reunió de seguiment abans del 19 de desembre, per coneixer l'estat del projecte i poder escollir el torn de lectura.

2. Planificació i recursos

2.1 Planificació temporal

Inicialment s'esperava realitzar el treball entre els mesos de setembre i gener, però finalment s'ha optat per ampliar el termini fins a l'abril. La càrrega total serà d'unes 480 hores. La dedicació setmanal estimada serà d'unes 20 hores.

Es dividirà el projecte en quatre blocs, descrits a continuació:

Bloc	Descripció	Metodologia	Hores
Bloc 0	Preparació de l'entorn	-	5h
Bloc 1	Curs de GEP	Cascada	75h
Bloc 2	Desenvolupament del projecte	Àgil	355h
Bloc 3	Preparació de la defensa	-	45h

Taula 2.1: Blocs del projecte

2.1.1 Bloc 0: Preparació de l'entorn

Inicialment, s'instal·larà tot el programari necessari per començar a desenvolupar el projecte i es faran algunes proves bàsiques per familiaritzar-se amb el nou entorn de treball. Aquest primer bloc tindrà una durada aproximada de 5 hores.

Per poder començar a treballar en el projecte, caldrà instal·lar:

- **Desenvolupament:** Python, OpenCV, Geany i Git.
- **Curs de GEP:** Gantt Project.
- **Documentació:** L^AT_EX i Zathura.

2.1.2 Bloc 1: Curs de GEP

Aquest bloc correspon a la realització del curs de GEP, amb inici el dia 19/09/2016 i finalització el 24/10/2016 (amb una presentació final entre el 7 i l'11 de novembre). Té com a dependència el bloc 0.

Durant el curs s'entregarán 6 lliurables, detallats a continuació:

Descripció	Inici	Finalització	Durada	Hores
Introducció i abast	19/09/2016	27/09/2016	9 dies	16h
Planificació temporal	28/09/2016	03/10/2016	6 dies	9h
Gestió econòmica i sostenibilitat	04/10/2016	10/10/2016	7 dies	10h
Presentació en vídeo	11/10/2016	17/10/2016	7 dies	11h
Plec de condicions	18/10/2016	24/10/2016	7 dies	13h
Document final + presentació	18/10/2016	24/10/2016	7 dies	16h

Taula 2.2: Lliurables de GEP

2.1.3 Bloc 2: Desenvolupament del projecte

El bloc principal consistirà en el desenvolupament del projecte en si mateix: buscar informació, implementar el codi, redactar la memòria, etc.

Aquest bloc té com a dependència el bloc 0 i es dividirà en quatre tasques.

Tasca	Inici	Finalització	Durada	Hores
Implementació i proves	13/09/2016	15/02/2017	156 dies	225h
Experiments	20/01/2017	30/03/2017	70 dies	40h
Ampliacions (opcional)	01/03/2017	30/03/2017	30 dies	40h
Redacció de la memòria	10/10/2016	13/04/2017	186 dies	50h

Taula 2.3: Tasques desenvolupament

Recerca d'informació, implementació i proves

Una part molt important del projecte serà la cerca d'informació i l'estudi de les diverses eines i algorismes a utilitzar (com per exemple OpenCV i les seves funcions). Se cercarà informació contínuament i s'aniran fent proves a mesura que s'implementa el codi.

La fase d'implementació es dividirà en diverses tasques, que s'aniràn realitzant a mesura que avanci el projecte. Algunes d'aquestes tasques seràn:

- Obtenció de keypoints (Harris)
- Extracció de característiques (SIFT)
- Matching i homografia
- Altres algorismes (ORB, BRISK, BRIEF)
- Disseny de l'aplicació d'Android
- Creació de l'aplicació d'Android

Experiments

Un cop enllestida la implementació, es procedirà a realitzar diversos experiments amb el programa. Es compararan els resultats obtinguts amb diferents algorismes de visió i es faran proves del sistema amb diverses imatges.

Ampliacions i millores

Un cop realitzats els experiments bàsics, es faran ampliacions i millores en el programa.

En cas de patir un retard en la planificació del projecte, s'utilitzarà aquest temps per acabar l'etapa d'experimentació.

Redacció de la memòria

La memòria s'anirà redactant a mesura que es realitza el projecte. No hi ha per tant cap dependència, encara que es dedicarà més temps en l'etapa final del treball.

2.1.4 Bloc 3: Preparació de la defensa

En aquest bloc final es revisarà la memòria del projecte i es prepararà la presentació. Està previst dedicar unes 45 hores al bloc, que començarà el dia 28 de març i acabarà el 23 d'abril. La defensa del projecte es durà a terme entre els dies 24 i 28 d'abril.

2.1.5 Diagrames

Durant la fase de planificació del treball, s'han realitzat diversos diagrames. A continuació podeu trobar els diagrames de PERT i el Gantt del projecte.

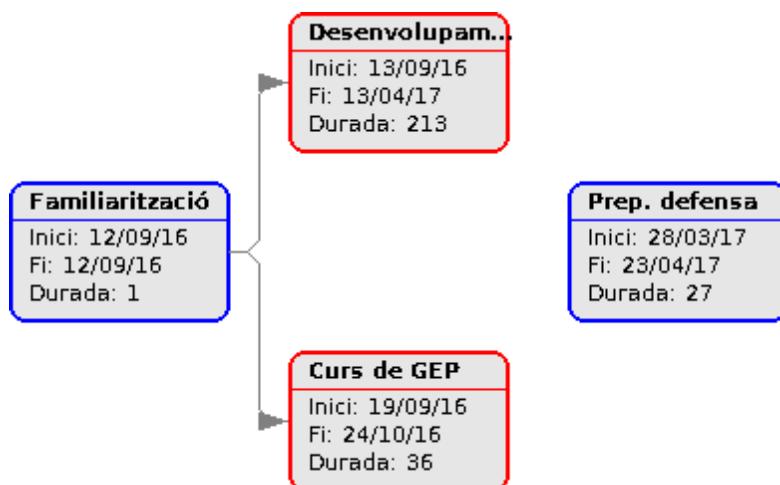


Figura 2.1: PERT del projecte

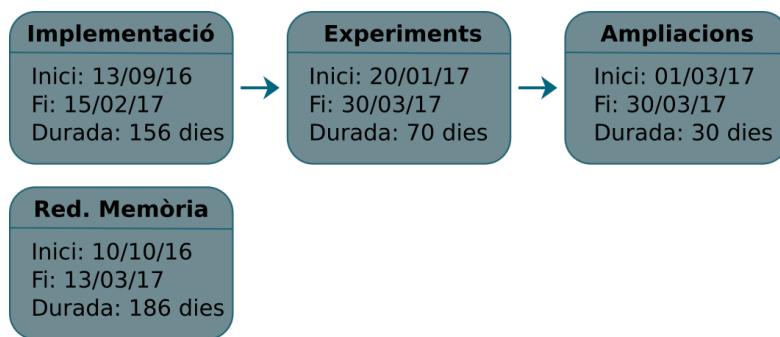


Figura 2.2: PERT - tasques desenvolupament

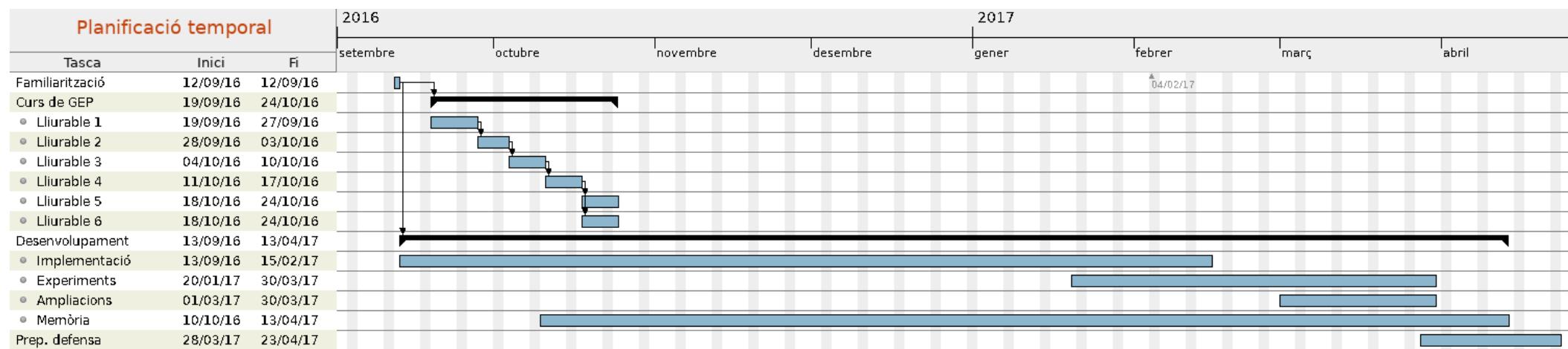


Figura 2.3: Gantt del projecte

2.2 Recursos

En aquesta secció es detallen els recursos necessaris per a la realització del projecte.

2.2.1 Recursos humans

El projecte el realitzarà una sola persona, que haurà d'assumir els rols de cap de projecte, analista, dissenyador, programador i *tester*. També es comptarà amb l'ajuda del director del projecte, que assumirà el paper de consultor/supervisor.

2.2.2 Recursos de maquinari

Per la realització del projecte no serà necessari adquirir cap mena de maquinari específic. Es podrà utilitzar un ordinador personal per treballar a casa i els ordinadors disponibles a la FIB per treballar des de la universitat.

Es treballarà principalment amb un ordinador equipat amb un processador AMD FX 6300 hexa-core 3.5GHz, 4GB de RAM i 250GB de disc dur SSD. També s'utilitzarà una càmera o smartphone qualsevol.

2.2.3 Recursos de programari

Durant la realització del projecte i el curs de GEP, s'utilitzaran diverses eines de programari, detallades a continuació:

Nom	Tipus	Ús
Arch Linux	Eina de desenvolupament	Execució del programari
Python	Eina de desenvolupament	Programació
OpenCV	Eina de desenvolupament	Algorismes de VC
Geany	Eina de desenvolupament	Programació del codi
Android Studio	Eina de desenvolupament	Programació del codi
Gimp/Inkscape	Eina de desenvolupament	Retocs i creació d'imatges
L ^A T _E X	Eina de desenvolupament	Redacció de la memòria
Zathura	Eina de desenvolupament	Visualització de pdf
Gantt Project	Eina de gestió	Creació diagrames de Gantt
LibreOffice Calc	Eina de gestió	Control de les hores
Git + Github	Desenvolupament i gestió	Control de versions

Taula 2.4: Recursos de programari

2.3 Desviacions i pla d'actuació

Mala planificació [Impacte: mig]

Hi haurà reunions amb el director i s'usaran eines de planificació per mirar de corregir la planificació i acabar el projecte a temps. També es reserven unes hores a l'ampliació del treball, que es podrien utilitzar en cas que una tasca s'allargués més del previst. Si fos necessari, es podria incrementar una mica la càrrega de treball setmanal.

Fallades de maquinari [Impacte: baix]

En cas de fallades en l'ordinador principal, no hi hauria cap problema en utilitzar-ne un altre. No hi ha dependències de hardware i es disposa d'altres ordinadors (a casa i a la FIB). Tampoc hi hauria una pèrdua de dades important, ja que es treballa amb Github i una còpia local.

3. Disseny i arquitectura

En aquest capítol es detalla l'arquitectura del sistema i el disseny de les aplicacions desenvolupades.

3.1 Arquitectura del sistema

L'arquitectura del sistema està formada per tres parts diferenciades: Un smartphone (aplicació usuari), un servidor i el robot. A continuació podeu veure un esquema de l'arquitectura i l'explicació de cada una de les parts.

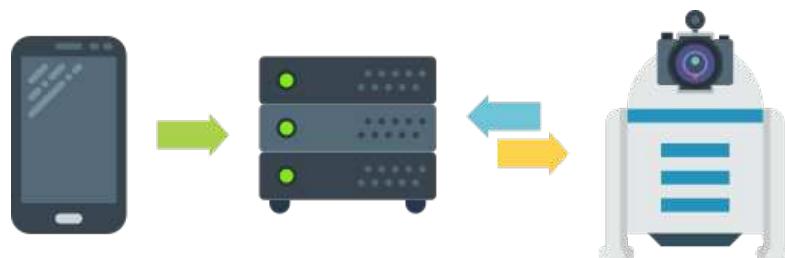


Figura 3.1: Arquitectura del sistema

- **Smartphone:** Permet a l'usuari seleccionar una regió en una imatge (de la càmera o la galeria) i l'envia al servidor.
- **Servidor:** Rep les imatges de l'smartphone i el robot i s'encarrega de fer el matching per obtenir el moviment necessari del robot.
- **Robot:** Envia les imatges capturades al servidor i espera rebre les ordres per desplaçar-se i girar.

El treball es centrarà en la part del servidor i segons el temps disponible es realitzarà la comunicació entre el servidor i l'aplicació per a smartphones.

3.2 Aplicació de proves

Inicialment es desenvoluparà una aplicació per realitzar les diverses proves sense interfície gràfica, que simplement mostrarà el resultat obtingut del matching. Més endavant, però, s'inclourà una interfície mínima que ens permetrà escollir les imatges a tractar i la regió d'interès (punt de destí).

Selecció de les imatges

Per seleccionar les imatges s'utilitzarà Tkinter, una GUI estàndard de Python.

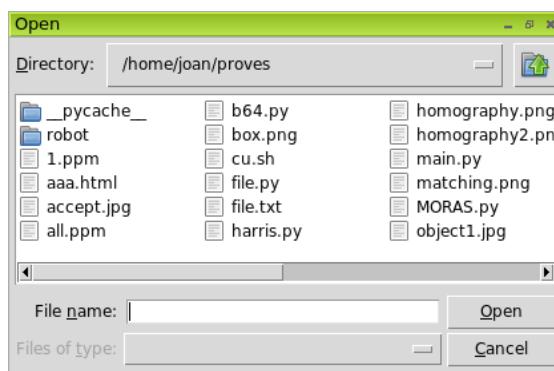


Figura 3.2: Selecció de les imatges

Selecció de la regió d'interès

Per poder escollir el punt de destí del robot, l'usuari haurà de seleccionar una regió d'interès en una imatge. Això es farà de manera molt senzilla, fent una selecció amb el ratolí. Un cop realitzada la selecció, l'usuari tindrà la opció de refer-la (simplement seleccionant de nou) o d'acceptar-la pulsant la tecla ç .

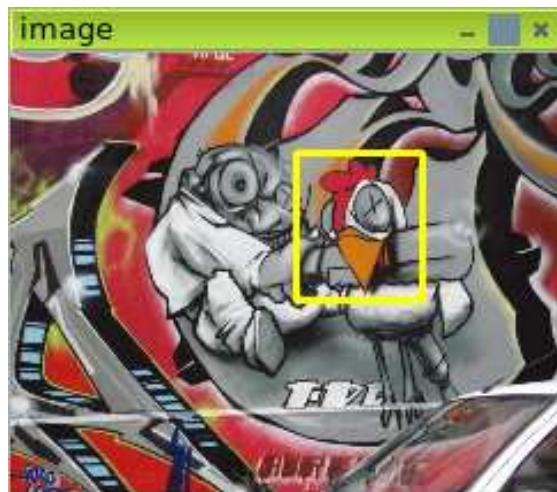


Figura 3.3: Selecció de la regió d'interès

3.3 Disseny de l'aplicació mòbil

La interfície de l'aplicació mòbil és també molt simple. Inicialment trobarem un menú que ens permetrà accedir a les funcionalitats del programa, donant-nos a escollir entre les següents opcions:

- **Fer una foto**
- **Seleccionar una imatge**
- **Opcions**

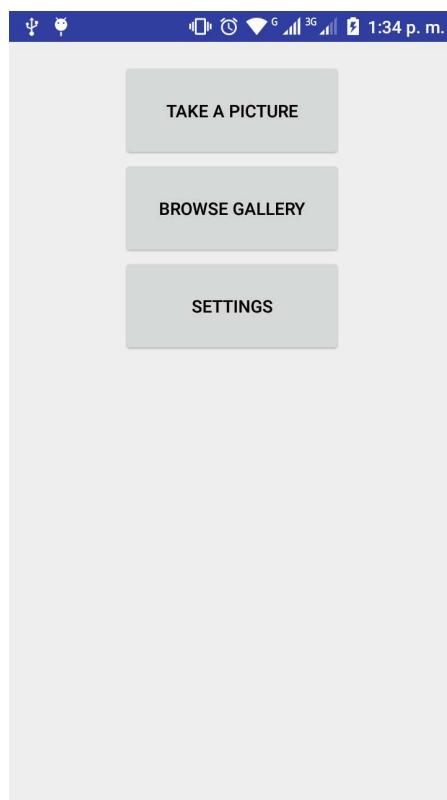


Figura 3.4: App - Menú

Fer una foto

Permet a l'usuari fer una foto amb la càmera del dispositiu, per després seleccionar la regió d'interès.



Figura 3.5: App - Càmera

Seleccionar una imatge

Si ja es disposa d'una imatge de l'entorn a la memòria del dispositiu, aquesta opció permet a l'usuari seleccionar-la. Un cop realitzada la selecció, l'usuari haurà de seleccionar la regió d'interès.



Figura 3.6: App - Galeria

Selecció de la regió d'interès

Després de realitzar una captura o seleccionar una imatge del dispositiu, el programa demanarà a l'usuari que seleccione una regió d'interès (destí) on vol que es desplaçï el robot.

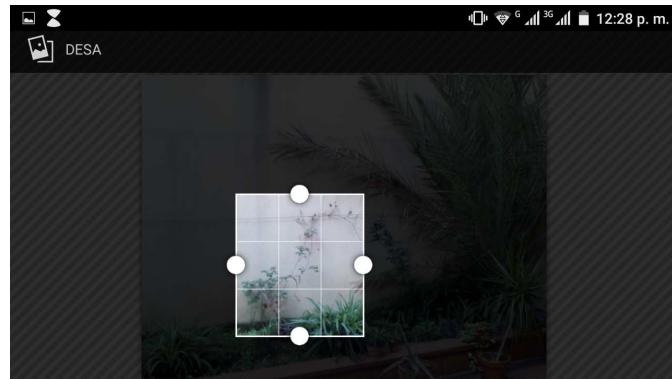


Figura 3.7: App - Selecció de la regió d'interès

Opcions

Des del menú d'opcions, l'usuari hauria de posar les dades per poder connectar-se al robot (direcció IP).

I pels usuaris avançats, existeixen opcions per canviar els algorismes de visió per defecte:

- Obtenció de keypoints: Harris, SIFT, SURF, ORB, BRISK, MSER
- Extracció de característiques: SIFT, SURF, ORB, BRISK, LATCH, DAISY

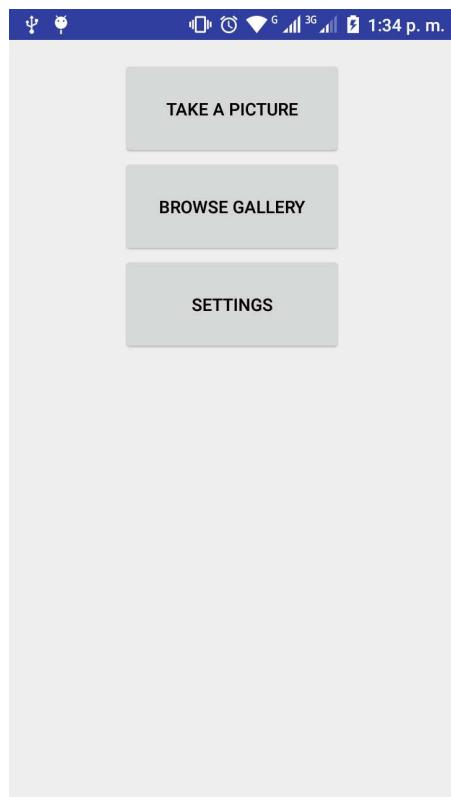


Figura 3.8: App - Opcions

4. Tècniques de visió usades

En aquesta secció es descriuen les tècniques de visió per ordinador i tractament d'imatges emprades durant la realització del projecte.

4.1 Pre-processat digital d'imatges

El pre-processament digital en una imatge, consisteix en aplicar diverses tècniques per tal d'aconseguir una imatge d'on poder obtenir la informació que necessitem més facilment. Es tracta d'eliminar distorsions o bé ressaltar determinades parts de la imatge.

Algunes de les tècniques que es poden aplicar són:

- Suavitzat de la imatge i reducció de soroll:
- Reducció de mides: Reduir la mida de la imatge ens permet millorar el temps d'execució.
- Escala de grisos: Els píxels de la imatge passen a tenir un valor en el rang 0-255. D'aquesta manera s'aconsegueix reduir el pes de la imatge. Encara que es perd la informació del color, moltes vegades pot ser irelevant o fins i tot portar a errors.
- Equalització de l'histograma: Per tal de millorar el contrast de les imatges. Clahe local adaptatiu.
- Ressaltar vores:

4.2 Obtenció de keypoints en una imatge

Consisteix en obtenir punts de la imatge amb característiques distintives, que ens puguin ser útils més endavant.

- Detecció de vores
- Detecció de cantonades
- Detecció de regions



Figura 4.1: Keypoints

4.3 Extracció de característiques

L'extracció de característiques és el que ens permetrà comparar els punts obtinguts en les dues imatges.

- Descriptors vectorials: SIFT, SURF
- Descriptors binaris: ORB, BRISK

4.4 Matching de característiques

Un cop tenim els keypoints i les característiques dels punts, necessitarem obtenir coincidències entre els punts de les dues imatges.

Bàsicament podem obtenir els matchs de dues maneres:

- Força bruta: Consisteix en provar totes les combinacions possibles per cada punt.
- Aproximació: ...

Com que el temps d'execució no es una factor essencial, s'aplicarà el mètode de força bruta, una mica més lent però més .

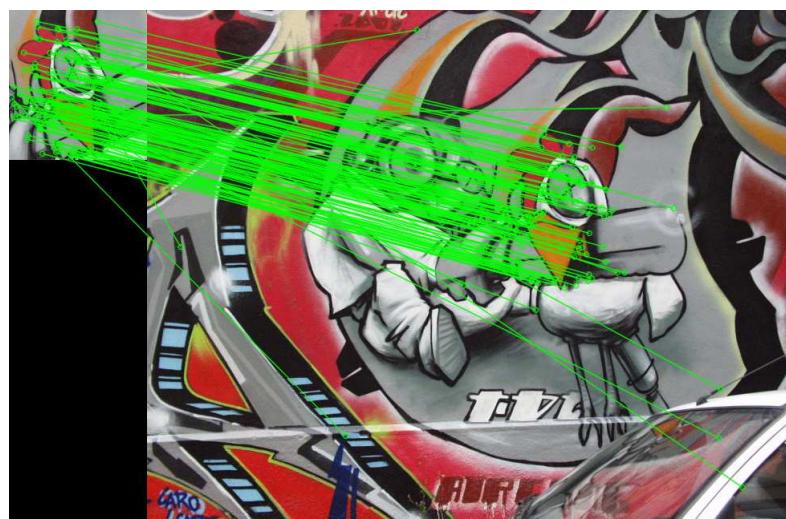


Figura 4.2: Matching

En la imatge anterior podem veure determinats punts on el "match" és clarament erroni. Aquest error es pot minimitzar escollint punts més significatius, característiques més distintives, aplicant el rati de Lowe o eliminant outliers.

4.5 Homografia

L'objectiu principal del programa es trobar una part d'una imatge en una altre imatge diferent i per fer això utilitzarem la homografia. Trobant la relació entre els píxels de les dues imatges podrem reprojectar el pla d'una imatge en l'altre i trobar el punt on volem dirigir el robot.

A l'hora de buscar la homografia aplicarem RANSAC (Random Sample Consensus), un algorisme que ens permetrà eliminar outliers dels matchs trobats.



Figura 4.3: Homografia

5. Implementació

5.1 Programa en python

Per facilitar la utilització del codi en possibles adaptacions o millores futures, s'ha decidit implementar una petita biblioteca en Python amb totes les funcions necessàries. El programa principal farà ús d'aquesta biblioteca, que permetrà:

1. Pre-processar les imatges
2. Seleccionar la regió d'interès
3. Obtenir els keypoints
4. Extreure les característiques
5. Fer matching de característiques
6. Homografia: Obtenir la regió/punt demanat
7. Obtenir l'angle de rotació pel robot

5.1.1 Pre-processat de les imatges

S'han provat diverses tècniques de pre-processat com ara filtres gaussians o canvis en el contrast, però els resultats obtinguts no han sigut satisfactoris i finalment s'ha optat per deixar les imatges tal com són.

Simplement es transforma la imatge a escala de grisos per poder treballar amb tots els algorismes de visió i es redimensiona per agilitzar la selecció de keypoints, extracció de característiques i matching.

```
def prep(image):
    img = cv2.resize(image, (0,0), fx=0.3, fy=0.3)
    return img, cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

5.1.2 Selecció de la regió d'interès

Per tal de poder seleccionar la regió d'interès amb facilitat, s'ha definit la funció **selectROI(image)**, que permet a l'usuari seleccionar una regió rectangular de la imatge donada com a paràmetre. La imatge resultant serà aquesta selecció.

```

def click_and_crop(event, x, y, flags, param):
    global refPt, cropping, sel_rect_endpoint, img

    if event == cv2.EVENT_LBUTTONDOWN: # Initial coordinates. Cropping = true
        cropping = True
        refPt = [(x, y)]
    elif event == cv2.EVENT_LBUTTONUP: # End coordinates. Cropping = false (done)
        cropping = False
        refPt.append((x, y))
        clone = img.copy()
        cv2.rectangle(clone, refPt[0], refPt[1], (0, 255, 255), 2) # Draw a rectangle (ROI)
        cv2.imshow("image", clone)
    elif event == cv2.EVENT_MOUSEMOVE and cropping: # Update position (moving rectangle)
        sel_rect_endpoint = [(x, y)]


def selectROI(image):
    global img, refPt, sel_rect_endpoint
    img = image ###
    cv2.namedWindow("image")
    cv2.setMouseCallback("image", click_and_crop)
    cv2.imshow('image', img)

    while True:
        if not cropping:
            sel_rect_endpoint = []
        elif cropping and sel_rect_endpoint: # Display rectangle (moving)
            clone = img.copy()
            cv2.rectangle(clone, refPt[0], sel_rect_endpoint[0], (0, 255, 0), 1)
            cv2.imshow('image', clone)
        if (cv2.waitKey(1) & 0xFF) == ord("c"):
            break

    cv2.destroyAllWindows()
    if len(refPt) == 2:
        img = img[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
    return img

```

5.1.3 Obtenció de keypoints

Per obtenir els punts d'interès d'una imatge, s'utilitzaràn diversos algorismes de visió per computador.

La funció point_selection(gray, alg) serà l'encarregada de cridar l'algoritme desitjat segons el que vulgui l'usuari (per defecte s'utilitzarà Harris). Els paràmetres necessaris seràn una imatge en escala de grisos i l'algorisme desitjat. La funció retornarà un array amb els keypoints trobats.

```
def point_selection(gray, alg):
    kp = []
    ...
    return kp
```

SIFT

Una de les opcions serà utilitzar SIFT (DoG). Per fer això simplement cridarem la funció d'OpenCV detect() en la instància de SIFT creada.

```
if alg == _SIFT:
    sift = cv2.xfeatures2d.SIFT_create()
    kp = sift.detect(gray, None)
```

ORB

Per utilitzar ORB també cridarem la funció detect() d'OpenCV, però en aquest cas haurem de modificar una mica els paràmetres per defecte de la creació de l'objecte ORB, ja que els resultats obtinguts en primer moment no eren gaire bons.

```
elif alg == _ORB:
    orb = cv2.ORB_create(nfeatures = 5000, nlevels = 8, edgeThreshold = 8,
                           patchSize = 8, fastThreshold = 5)
    kp = orb.detect(gray, None)
```

Harris

En el cas de Harris, el detector no utilitza diferents escales i simplement detecta “corners” en la imatge. Per tant, utilitzarem la funció d’OpenCV pyrDown() per reduir la mida de la imatge (1/2) i aplicar Harris en diverses escales.

Per detectar els keypoints farem servir la funció goodFeaturesToTrack() en comptes del detector de corners de Harris, ja que permet obtenir els punts més facilment i també té la opció d’utilitzar Shi-Tomasi si ens interessés.

```

elif alg == _HARRIS:
    G = gray.copy()
    for i in range(5):
        if i != 0:
            G = cv2.pyrDown(G)
        scale = 2**i
        corners = cv2.goodFeaturesToTrack(image=G,maxCorners=1000,
                                           qualityLevel=0.01,minDistance=scale,useHarrisDetector=1, k=0.04)
        corners = np.int0(corners)
        for corner in corners:
            x,y = corner.ravel()
            k = cv2.KeyPoint(x*scale, y*scale, scale)
            kp.append(k)

```

MSER

Per últim, tindrem la opció d’utilitzar MSER. Tal com fem amb SIFT i ORB, també farem servir la funció detect().

```

elif alg == _MSER:
    mser = cv2.MSER_create()
    kp = mser.detect(gray,None)

```

5.1.4 Extracció de característiques

De la mateixa manera en que podem aconseguir keypoints, OpenCV també disposa de diversos algorismes d'extracció de característiques a partir dels keypoints d'una imatge.

La funció creada `feature_detection(image, kp, alg)` serà l'encarregada d'extreure les característiques utilitzant algun dels algorismes disponibles. Els paràmetres necessaris de la funció són: la imatge en escala de grisos, l'array de keypoints obtinguts i l'algorisme desitjat. Retornarà tant els descriptors com els keypoints.

```
def feature_extraction(image, kp, alg):
    des = []
    ...
    return kp, des
```

Els algorismes que podrà escollir l'usuari són:

- SIFT
- SURF
- ORB
- BRISK
- LATCH
- DAISY

S'utilitzaran els algorismes ja implementats en la biblioteca OpenCV de la següent manera:

```
elif alg == _LATCH:
    latch = cv2.xfeatures2d.LATCH_create()
    kp, des = latch.compute(image, kp)
```

5.1.5 Matching i homografia

La funció retornarà les coordenades de destí (x,y) i una imatge amb els matches i la regió de destí pintats.

```
def matching(img1, img2, des1, des2, kp1, kp2, fe):
    draw_params = dict(matchColor = (0,255,0), singlePointColor = None,
                         matchesMask = matchesMask, flags = 2)
    return x, y, cv2.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
```

Matching

En el cas dels descriptors binaris, utilitzarem la distància Hamming, mentre que per la resta s'utilitzarà... En els dos casos, s'utilitzarà la funció BFMatcher(), que aplicarà el matching per força bruta.

```
if fe == _LATCH or fe == _ORB or fe == _BRISK:
    bf = cv2.BFMatcher(cv2.NORM_HAMMING)
else:
    bf = cv2.BFMatcher()

matches = bf.knnMatch(des1, des2, k=2)
```

Ratio test

Un cop obtinguts els "matches", aplicarem el ratio test per descartar alguns "matches".

```
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append(m)
```

Homografia

Si hi ha prou coincidències (considerem acceptable com a mínim 10), es buscarà la homografia i es pintarà la regió trobada en la imatge. S'aplicarà Ransac.

```

if len(good) >= MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
    h,w,_ = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,M)
    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)

```

Coordenades de destí

Per obtenir el punt on volem que es dirigeixi el robot agafarem els valors de dst i retornarem el punt mig.

```

x1, y1 = np.int32(dst)[0].ravel()
x2, y2 = np.int32(dst)[1].ravel()
x3, y3 = np.int32(dst)[2].ravel()
x4, y4 = np.int32(dst)[3].ravel()
x = (x1+x2+x3+x4)/4
y = (y1+y2+y3+y4)/4

```

5.1.6 Angle de gir

Un cop obtingudes les coordenades de la imatge, s'obtindrà l'angle de gir necessari pel robot a partir de l'angle de visió de la càmera i les dimensions de la imatge.

```

def getAngle(aV, w, x):
    if x == -1:
        return 0
    else:
        return (aV*x / w) - (aV/2)

```

5.2 Aplicació mòbil

L'aplicació mòbil d'ha desenvolupat amb Android Studio i el llenguatge de programació Java (amb l'SDK d'Android).

5.2.1 Capturar imatge

Per poder utilitzar la càmera del mòbil s'ha hagut d'afegir el permís necessari.

Com que Android ja disposa d'una Activitat que ens permet llançar la càmera desde una altre aplicació, nomès ha sigut necessari fer l'Intent corresponent.

```
class MyClass(Yourclass):
    def __init__(self, my, yours):
        String = "String"
        String2 = '5 1 2 3 4'
        print String
    import numpy as np #Comment1
    # Comment2
```

5.2.2 Selecció d'imatges de la galeria

Per poder utilitzar la càmera del mòbil s'ha hagut d'afegir el permís necessari.

Com que Android ja disposa d'una Activitat que ens permet llançar la càmera desde una altre aplicació, nomès ha sigut necessari fer l'Intent corresponent.

```
class MyClass(Yourclass):
    def __init__(self, my, yours):
        String = "String"
        String2 = '5 1 2 3 4'
        print String
    import numpy as np #Comment1
    # Comment2
```

5.2.3 Selecció de la regió d'interès

Per poder utilitzar la càmera del mòbil s'ha hagut d'afegir el permís necessari.

Com que Android ja disposa d'una Activitat que ens permet llançar la càmera

desde una altre aplicació, nomès ha sigut necessari fer l'Intent corresponent.

```
class MyClass(Yourclass):
    def __init__(self, my, yours):
        String = "String"
        String2 = '5 1 2 3 4'
        print String2
    import numpy as np #Comment1
    # Comment2
```

5.2.4 Enviament de dades al servidor

Per poder enviar dades al servidor, serà necessari l'ús d'una connexió a Internet. Per tant, s'haurà d'habilitar el permís necessari.

Com que Android ja disposa d'una Activitat que ens permet llançar la càmera desde una altre aplicació, nomès ha sigut necessari fer l'Intent corresponent.

```
class MyClass(Yourclass):
    def __init__(self, my, yours):
        String = "String"
        String2 = '5 1 2 3 4'
        print String2
    import numpy as np #Comment1
    # Comment2
```

6. Experiments i resultats

En aquest capítol es detallen els experiments realitzats i els resultats obtinguts a partir d'aquests.

6.1 Experiments realitzats

Podeu trobar el codi dels scripts realitzats a l'anex del treball.

6.1.1 Comparació detectors de keypoints

Tot i que la velocitat d'execució no és un factor determinant pel projecte, s'ha realitzat una comparació inicial de la velocitat d'execució dels algorismes de detecció de keypoints.

Es comparen els següents algorismes:

- Harris
- SIFT
- SURF
- ORB
- MSER

Per mesurar el temps d'execució, s'agafa la mitja de 5 execucions de la funció d'obtenció de keypoints utilitzada.

6.1.2 Comparació detecció i extracció de keypoints

En aquest experiment s'analitzen els diversos algorismes de detecció i extracció de keypoints, tant en la velocitat d'execució com en fiabilitat, que serà el més important pel sistema desenvolupat.

Es compararan els següents algorismes:

- Harris + SIFT
- Harris + ORB
- SIFT + SIFT
- SIFT + LATCH
- ORB + ORB
- ORB + BRISK
- MSER + SIFT
- MSER + DAISY

6.2 Resultats i comparació d'algorismes

6.2.1 Detectors de keypoints

En primer lloc s'han realitzat proves amb imatges típiques de visió per computador.



Figura 6.1: Imatges graff, boat i ubc

Algorismes	Graff		Boat		Ubc	
	Punts	Temps (s)	Punts	Temps (s)	Punts	Temps (s)
Harris	1285	0.0123	1666	0.0124	1329	0.0101
SIFT	1096	0.0675	1599	0.0755	1284	0.0639
SURF	1467	0.0182	1768	0.0213	1533	0.0181
ORB	2500	0.0102	2500	0.0128	2500	0.0116
MSER	1044	0.2323	924	0.1586	286	0.0565

Taula 6.1: Detectors de keypoints - comparació

Amb aquestes imatges, podem veure com Harris, ORB i SURF són força més ràpids que el detector de keypoints de SIFT (DoG). MSER sembla ser el més lent en general i el que menys punts obté.

S'ha de tenir en compte que el detector ORB s'ha limitat a 2500 punts.

També s'ha provat el sistema amb imatges reals d'entorns coneguts (campus nord i jardins de palau reial).



Figura 6.2: Imatges UPC i jardins

Algorismes	Campus		Jardins		Campus 2	
	Punts	Temps (s)	Punts	Temps (s)	Punts	Temps (s)
Harris	2522	0.1108	1971	0.1086	2589	0.1095
SIFT	4621	0.7358	9690	0.8172	4750	0.7405
SURF	7676	0.1761	14888	0.2332	10720	0.1990
ORB	2500	0.0515	2500	0.0978	2500	0.0618
MSER	2135	0.6790	1141	0.3930	1798	0.6397

Taula 6.2: Detectors de keypoints - comparació 2

SURF és el que més keypoints detecta, seguit de SIFT. Pel que fa al temps d'execució, SIFT és el més lent, seguit de MSER (el que menys keypoints obté). El més ràpid amb diferència és ORB, però la limitació a 2500 punts probablement farà que no s'obtinguin tan bons resultats.

6.2.2 Detecció i extracció de keypoints

Tal com s'ha fet en l'experiment anterior, es prova el sistema amb les imatges de visió i després amb fotografies pròpies.



Figura 6.3: Imatges graff i jardins

Algorismes	Graff			Jardins		
	Matches	%	Temps (s)	Matches	%	Temps (s)
Harris + SIFT	71	14.08%	0.1289	108	34.26%	0.5980
Harris + ORB	221	04.07%	0.0437	320	10.00%	0.2816
SIFT + SIFT	48	43.75%	0.4469	138	69.56%	7.9518
SIFT + LATCH	0	00.00%	0.2879	21	66.67%	3.9541
ORB + ORB	7	04.09%	0.0771	243	16.05%	0.2952
ORB + BRISK	26	61.54%	0.9651	57	96.49%	1.1579
MSER + SIFT	-	-	-	-	-	-
MSER + DAISY	-	-	-	-	-	-

Taula 6.3: Matching - comparació

Els resultats obtinguts són bastants dolents. En el cas de la imatge del graffiti sembla ser que el canvi de perspectiva i rotació de la imatge és massa gran. Els únics algorismes que aconsegueixen superar el 50% d'encerts són ORB+BRISK, però només troben 26 matches. Latch, amb els punts obtinguts amb SIFT (DoG) no troba cap coincidència.

Pel que fa a la imatge dels jardins, els resultats són una mica millors. ORB+BRISK aconsegueix un 96.49% d'encerts, encara que amb nomès 57 matches i bastant focalitzats, pel que podria tenir problemes per trobar la homografia. SIFT encerta en el 69.56% dels casos i troba 138 matches. ORB sembla ser el pitjor.

També s'han realitzat proves amb les següents fotografies:



Figura 6.4: Imatges UPC

Algorismes	Campus 1			Campus 2		
	Matches	%	Temps (s)	Matches	%	Temps (s)
Harris + SIFT	77	29.87%	0.5308	461	%	0.6783
Harris + ORB	212	10.38%	0.2514	617	20.10%	0.2881
SIFT + SIFT	150	60.67%	2.6028	520	%	3.6935
SIFT + LATCH	19	10.53%	1.7193	351	%	2.2701
ORB + ORB	414	11.84%	0.1942	580	%	0.2200
ORB + BRISK	143	54.55%	1.0620	328	%	1.0876
MSER + SIFT	-	-	-	-	-	-
MSER + DAISY	-	-	-	-	-	-

Taula 6.4: Matching - comparació 2

SIFT i ORB+BRISK continuen sent els algorismes amb la taxa d'encert més elevada i SIFT+LATCH segueix obtenint molt pocs matches.

En els dos conjunts d'imatges, els algorismes confonen en molts casos les finestres, ja que realment són iguals i els canvis de zoom i perspectiva fan molt complicat diferenciar-les. En el primer cas, la moto i la porta es detecten força bé.

6.2.3 Matching i homografia

Aquí podeu veure alguns dels resultats obtinguts a l'executar el programa seleccio-
nant una regió més concreta:

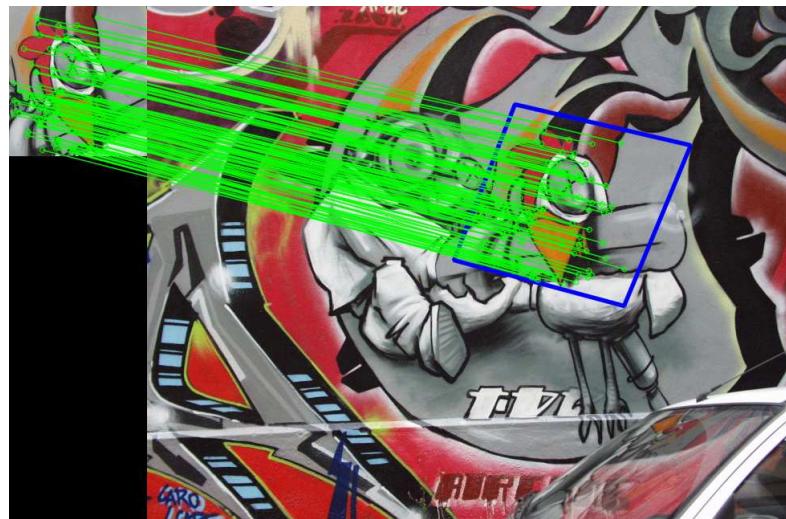


Figura 6.5: Homografia graff - SIFT

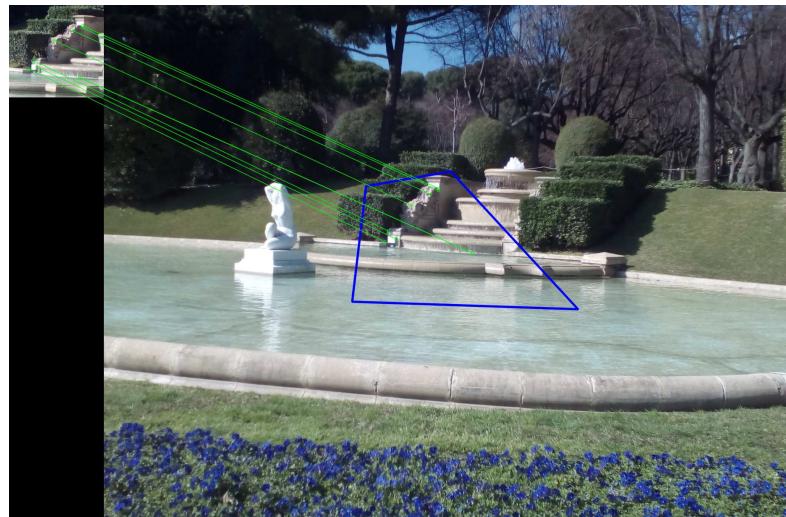


Figura 6.6: Homografia jardins - ORB+BRISK

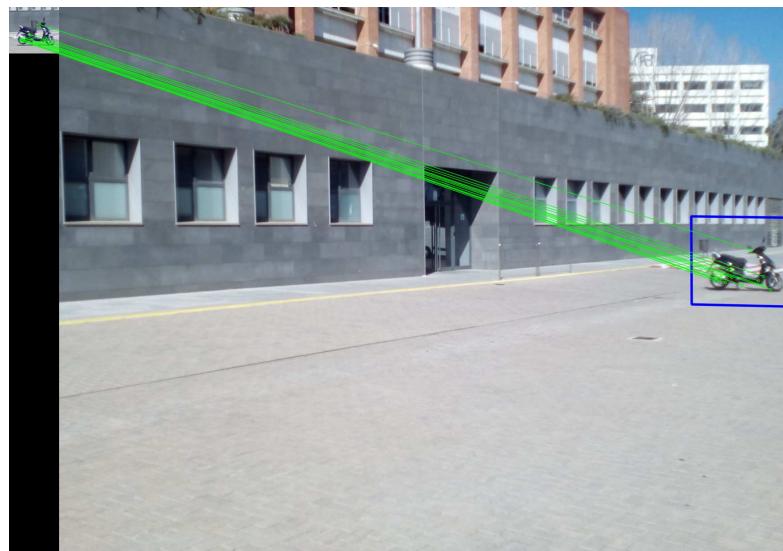


Figura 6.7: Homografia campus - SIFT



Figura 6.8: Homografia campus 2 - SIFT

7. Gestió econòmica

7.1 Recursos de programari

Tot el *software* utilitzat en aquest projecte és gratuït i de codi obert. Per tant, el programari no suposarà cap despesa. Podeu trobar el llistat del programari utilitzat a la taula 2.4 (Recursos de programari).

7.2 Recursos humans

El projecte el desenvoluparà una sola persona, que assumirà diversos rols durant la realització d'aquest. Tenint en compte les tasques descrites a la secció 2.1, les hores de treball queden repartides de la següent manera:

Tasca	Cap	Analista	Programador
Preparació de l'entorn	3h		2h
Curs de GEP	75h		
Implementació i proves		30h	195h
Experiments			40h
Ampliacions		10h	30h
Redacció memòria	50h		
Preparació defensa	45h		
Total	173h	40h	267h

Taula 7.1: Recursos humans (hores)

Suposem uns costos de 25€/h pel cap de projecte, 20€/h per l'analista i 15€/h pel programador/*tester*.

Rol	Hores	Cost/hora	Cost total
Cap de projecte	173h	25€/h	4325€
Analista	40h	20€/h	800€
Programador	267h	15€/h	4005€
Total			9130€

Taula 7.2: Recursos humans (costos)

7.3 Recursos de maquinari

El *hardware* necessari per a la realització del treball serà només un ordinador (usat durant tot el projecte) i una càmera (per la fase d'implementació/proves).

Producte	Preu	Ús	Vida útil	Amortització
Ordinador personal	500€	7 mesos	5 anys	58,33€
Smartphone	39€	1 mes	3 anys	1,08€
Total				59,41€

Taula 7.3: Recursos de maquinari (costos)

7.4 Costos indirectes

També es tindran en compte els costos indirectes més importants: la connexió a Internet i el consum elèctric. La connexió a Internet costarà 40€ al mes (considerem 240 hores) i l'electricitat 0,141033€/kWh (considerem la potència 0,2kW).

Tipus	Temps	Cost	Cost total
Electricitat	480h	0,028€/h	13,44€
Accès a Internet	480h	0,17€/h	81,6€
Total			95,04€

Taula 7.4: Costos indirectes

7.5 Imprevistos

Es podria donar el cas que el projecte ocupa més temps de l'esperat, pel que es considerarà un extra de 30 hores de treball, que es dividirien entre el programador i el *tester*. Això suposaria un increment de 600€ en el pressupost.

No es tindran en compte possibles fallades de maquinari, ja que l'ordinador principal amb què es treballa està en garantia i també es disposa d'altres ordinadors.

7.6 Contingència

Com a mesura de contingència, s'estableix un marge del 5%.

7.7 Costos totals

Tipus	Cost estimat
Recursos humans	9130€
Recursos de programari	0€
Recursos de maquinari	59,41€
Costos indirectes	95,04€
Imprevistos	600€
Contingència (5%)	494,22€
Total	10378,67€

Taula 7.5: Costos totals

7.8 Control de gestió

Després de cada tasca es farà una valoració del pressupost i es revisarà si és necessari. També es durà a terme un control de les hores de treball per cada rol mitjançant un full de càlcul, que s'anirà actualitzant cada dia de treball.

Es calcularà la desviació en mà d'obra, programari, maquinari i altres costos (cost estimat - cost real).

8. Informe de sostenibilitat

En aquest capítol es farà una anàlisi de la sostenibilitat del projecte, que es divideix en tres parts, identificades per les columnes de la matriu:

- El projecte posat en producció (PPP), que inclou la planificació, el desenvolupament i la implantació del projecte.
- La vida útil del projecte, que comença un cop implantat el sistema i finalitza amb el seu desmantellament.
- Els riscos inherents al propi projecte, considerant tota la construcció i la vida útil del mateix.

Cadascuna de les columnes s'analitzarà des dels punts de vista ambiental, econòmic i social, les tres dimensions de la sostenibilitat. A continuació podeu veure la matriu de sostenibilitat del projecte:

Sostenibilitat	PPP	Vida útil	Riscos
Ambiental	Consum del disseny 8 [0:10]	Petjada ecològica 15 [0:20]	Riscos ambientals 0 [-20:0]
Econòmica	Factura 7 [0:10]	Pla de viabilitat 10 [0:20]	Riscos econòmics 0 [-20:0]
Social	Impacte personal 8 [0:10]	Impacte social 5 [0:20]	Riscos socials 0 [-20:0]
Valoració total		53 [-60:90]	

Taula 8.1: Matriu de sostenibilitat

8.1 Posada en producció

En aquesta secció es detalla la sostenibilitat del sistema desde la seva planificació fins a la possible implantació. Es tindrà en compte el consum del disseny, la factura i l'impacte personal que ha suposat la realització d'aquest treball.

8.1.1 Consum del disseny

Els recursos necessaris per al desenvolupament d'aquest projecte són mínims. No és necessari comprar cap tipus de maquinari addicional i tot el *hardware* utilitzat ha estat comprat a la unió europea, pagant una taxa pel correcte reciclatge dels residus. A més, el maquinari seguirà essent funcional un cop acabat el projecte. L'impacte ambiental del projecte serà mínim, ja que només es consumirà l'energia necessària per utilitzar un ordinador personal. No es generarà cap tipus de residu durant el desenvolupament o el desmantellament.

8.1.2 Factura

Tal com podeu veure a l'apartat "Gestió econòmica", per analitzar la viabilitat del projecte s'ha realitzat un pressupost tenint en compte els costos directes, indirectes i possibles imprevistos. Com es pot veure, els costos de *software* i *hardware* són mínims, pel que el projecte resulta econòmicament viable. L'única manera de rebaixar costos seria incrementant el nombre d'hores diàries (baixarien els costos de llum i Internet) o contractant a algú amb més experiència. No està prevista cap col·laboració amb altres projectes, però s'utilitzaran eines i algorismes existents que no caldrà programar de nou.

El cost inicial ha sigut aproximadament el mateix que el final, ja que no han hagut modificacions en els recursos de programari o maquinari.

8.1.3 Impacte personal

Aquest projecte m'ha permès profunditzar els meus coneixements sobre les tècniques de visió per computador actuals i la seva possible aplicació en sistemes robòtics.

Amb la realització d'aquest treball, també he hagut d'utilitzar una metodologia de treball àgil, que fins ara no havia utilitzat i he hagut de realitzar la planificació i la gestió dels recursos d'un projecte real, fet que estic segur que m'ajudarà en un futur a l'hora d'emprendre altres projectes a nivell professional.

8.2 Vida útil

A continuació es descriu la sostenibilitat del projecte desde la implantació fins al desmantellament. Es tindrà en compte la petjada ecològica, la viabilitat i l'impacte del projecte en la societat.

8.2.1 Petjada ecològica

Es tracta d'un projecte de programari, que a més es publicarà sota una llicència de *software* lliure, de manera que qualsevol usuari se'n podrà beneficiar i podrà reutilitzar el codi per a futurs projectes. No suposaria un augment en la petjada ecològica ni tampoc una disminució, encara que amb la utilització del programari desenvolupat es necessitarien menys recursos materials pel control d'un robot.

8.2.2 Pla de viabilitat

En principi, no està prevista la implantació o la comercialització del sistema en un futur. El codi estarà disponible al repositori de Github i podrà ser utilitzat i adaptat lliurement. Per tant, seran els propis usuaris els que s'hauran de preocupar dels costos en cas d'utilitzar el sistema.

Si es volgués implantar el sistema, s'haurien de considerar els costos de la utilització i possible manteniment d'un servidor on allotjar el codi desenvolupat (ja sigui un servidor local o extern). I en cas de voler millorar o actualitzar el codi amb noves funcionalitats, s'haurien de tenir en compte els recursos humans necessaris.

8.2.3 Impacte social

El projecte no suposarà cap canvi important en la situació social o política del país ni té intenció de canviar substancialment la vida de les persones. Actualment, existeixen múltiples maneres de controlar un robot, ja sigui manualment o amb altres mètodes de localització, com podria ser utilitzant les coordinades GPS. També hi ha sistemes que utilitzen marques visuals (punts de referència) o que operen en un entorn conegut pel robot. El proposit d'aquest projecte serà oferir una alternativa, un sistema d'autolocalització barat (no serà necessari dotar el robot de molts sensors) i disponible per a tothom.

Qualsevol usuari podrà beneficiar-se del sistema, ja que el codi serà publicat sota una llicència de *software* lliure en un repositori de Github. I evidentment, la realització d'aquest TFG no perjudicarà cap col·lectiu de cap manera.

8.3 Riscos

Finalment, s'analitzaran els riscos inherents del projecte en les tres dimensions de la sostenibilitat: ambientals, econòmics i socials.

8.3.1 Riscos ambientals

En principi la petjada ecològica del projecte té un marge molt limitat. Un cop desenvolupat el codi, s'allotjarà a Github. El que altres usuaris facin després ja no depen de la feina de l'autor.

8.3.2 Riscos econòmics

En cas d'implantar el sistema desenvolupat, l'únic risc econòmic serien les possibles fallades del sistema a nivell de servidor. Els costos del maquinari o del programari no podrien suposar cap problema, ja que el programari és gratuït i el sistema no depen d'un maquinari específic. Un cop implementat el codi, el sistema només depen d'un servidor que pugui executar Python i OpenCV.

8.3.3 Riscos socials

Aquest treball no perjudicarà en cap cas a algun sector de la població, ni en la posada en producció, ni durant la possible implantació o posterior desmantellament. El sistema desenvolupat es un programari inofensiu, que només podria ser perjudicial si algun usuari en fes un mal ús en un servidor extern, fet que ja no seria responsabilitat de l'autor.

La única dependència del programa principal es la biblioteca OpenCV, però tractant-se d'una biblioteca de *software* lliure no hauria de suposar cap problema. S'utilitzen els algorismes SIFT i SURF, que estan patentats i per tant no es poden utilitzar gratuïtament en productes comercials. Tot i que no està previst comercialitzar el sistema desenvolupat de cap manera, també s'utilitzen algorismes alternatius com ORB que no suposarien cap problema si algú usuari volgués comercialitzar un producte derivat d'aquest projecte.

9. Conclusions

En aquest apartat es descriuen les conclusions extretes després de realitzar aquest treball final de grau. En primer lloc es descriuràn les conclusions tècniques, seguides de les conclusions personals i finalment es detallarà els plans de treball futurs i les possibles ampliacions i millors del sistema.

L'objectiu principal del projecte, crear un sistema d'autolocalització, s'ha complert.

9.1 Conclusions tècniques

Com es pot veure en els experiments realitzats, la taxa d'encert dels match obtinuts amb els diversos algorismes de detecció i extracció de característiques és molt baixa.

9.2 Conclusions personals

Aquest treball m'ha permés profunditzar els meus coneixements sobre visió per computador.

9.3 Treball futur

En principi no està previst continuar amb el treball en un futur, però el codi serà públic i no es descarta continuar amb el projecte més endavant (personalment o a través d'altres persones).

En tot cas, hi ha una sèrie de possibles millors i ampliacions que caldria mencionar:

- Provar el sistema en un entorn real i un robot: Com que no es disposava del temps necessari per fer les proves amb un robot, no s'ha pogut experimentar amb el sistema en casos reals. Per tant, considero necessària l'execució de proves amb robots.
- Comparació i anàlisi d'algorismes: És necessari fer un anàlisi més exhaustiu dels diversos algorismes de detecció i extracció de característiques.
- Pre-processat: S'hauria d'investigar amb més profunditat quines tècniques de pre-processat podrien ajudar als algorismes a detectar millors keypoints i característiques.
- Servidor: El codi s'hauria de posar en un servidor amb el que l'aplicació mòbil es connectaria.

Apèndix

A. Codi dels experiments

A.1 Velocitat obtenció de keypoints

```
from timeit import default_timer as timer
import numpy as np
import MORAS as vc
import cv2

algs = [vc._SIFT, vc._SURF, vc._ORB, vc._HARRIS]
algsName = ["SIFT", "SURF", "ORB", "HARRIS"]
paths = ['images/graff/img1.png', 'images/bikes/img1.png', 'images/ubc/img1.png', 'images/all..']

for imgPath in paths:
    img = cv2.imread(imgPath, 0)
    img = cv2.resize(img, (0,0), fx=0.5, fy=0.5)
    print(imgPath)
    for i in range(len(algs)):
        psAlg = algs[i]
        name = algsName[i] + ":"
        times = np.zeros(10)
        numKp = 0
        for j in range(len(times)):
            start = timer()
            kpROI = vc.point_selection(img, psAlg, True)
            end = timer()
            times[j] = end - start
            numKp = len(kpROI)
        print(name, np.mean(times), "ms", "Keypoints:", numKp)
print("\n")
```

A.2 Velocitat Obtenció + extracció + match

```

from timeit import default_timer as timer
import numpy as np
import MORAS as vc
import cv2

imgs1 = ['images/light/img1.png', 'images/boat/img5.png', 'images/graff/img1.png']
imgs2 = ['images/light/img6.png', 'images/boat/img6.png', 'images/graff/img3.png']
algs1 = [vc._HARRIS, vc._SIFT, vc._ORB]
algs2 = [vc._SIFT, vc._SIFT, vc._ORB]
algsName = ["HARRIS + SIFT", "SIFT", "ORB"]

for z in range(len(imgs1)):
    print(imgs1[z], imgs2[z])
    imgROI = cv2.imread(imgs1[z])
    imgROI = cv2.resize(imgROI, (0,0), fx=0.5, fy=0.5)
    imgROIGray = cv2.cvtColor(imgROI, cv2.COLOR_BGR2GRAY)

    imgRobot = cv2.imread(imgs2[z])
    imgRobot = cv2.resize(imgRobot, (0,0), fx=0.5, fy=0.5)
    imgRobotGray = cv2.cvtColor(imgRobot, cv2.COLOR_BGR2GRAY)

    for i in range(len(algs1)):
        psAlg = algs1[i]
        feAlg = algs2[i]
        name = algsName[i] + ":"

        times = np.zeros(5)
        numKp = 0
        good = 0
        for j in range(len(times)):
            start = timer()

            kpROI = vc.point_selection(imgROIGray, psAlg, True)          # Get Keypoints
            kp1, desROI = vc.feature_extraction(imgROIGray, kpROI, feAlg) # Get features

            kpRobot = vc.point_selection(imgRobotGray, psAlg, True)          # Get points
            kp2, desRobot = vc.feature_extraction(imgRobotGray, kpRobot, feAlg) # Get features

            x, y, img3, good = vc.matching(imgROI, imgRobot, desROI, desRobot,
                kp1, kp2, feAlg, True) # Find matching point

```

```

        end = timer()
        times[j] = end - start
        cv2.imwrite("test"+algsName[i]+" "+str(z)+".png", img3)
        print(name, np.mean(times), "Good matches: "+str(good))
        print("\n")
    
```

A.3 Encert matches

```

from timeit import default_timer as timer
import numpy as np
import MORAS as vc
import cv2
import scipy as sp

imgss1 = ['images/light/img1.png', 'images/boat/img5.png', 'images/graff/img1.png']
imgss2 = ['images/light/img6.png', 'images/boat/img6.png', 'images/graff/img3.png']
algs1 = [vc._HARRIS, vc._SIFT, vc._ORB]
algs2 = [vc._SIFT, vc._SIFT, vc._ORB]
algsName = ["HARRIS + SIFT", "SIFT", "ORB"]

def printMatches(img1, img2, k1, k2, sel_matches, name):
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    view = sp.zeros((max(h1, h2), w1 + w2, 3), sp.uint8)
    view[:h1, :w1, :] = img1
    view[:h2, w1:, :] = img2
    view[:, :, 1] = view[:, :, 0]
    view[:, :, 2] = view[:, :, 0]
    cp = view.copy()

    num = 0
    for m in sel_matches:
        color = tuple([sp.random.randint(0, 255) for _ in range(3)])
        cv2.line(view, (int(k1[m.queryIdx].pt[0]), int(k1[m.queryIdx].pt[1])) ,
                 (int(k2[m.trainIdx].pt[0] + w1), int(k2[m.trainIdx].pt[1])), color)
    cv2.imwrite("resultats/"+name+"_"+str(num)+".png", view)
    view = cp.copy()
    num = num + 1

for z in range(len(imgss1)):

```

```

print(imgs1[z], imgs2[z])
imgROI = cv2.imread(imgs1[z])
imgROI = cv2.resize(imgROI, (0,0), fx=0.5, fy=0.5)
imgROIGray = cv2.cvtColor(imgROI, cv2.COLOR_BGR2GRAY)

imgRobot = cv2.imread(imgs2[z])
imgRobot = cv2.resize(imgRobot, (0,0), fx=0.5, fy=0.5)
imgRobotGray = cv2.cvtColor(imgRobot, cv2.COLOR_BGR2GRAY)

for i in range(len(algs1)):
    psAlg = algs1[i]
    feAlg = algs2[i]
    name = algsName[i] + ":""

    kpROI = vc.point_selection(imgROIGray, psAlg, True)          # Get Keypoints
    kp1, desROI = vc.feature_extraction(imgROIGray, kpROI, feAlg) # Get features

    kpRobot = vc.point_selection(imgRobotGray, psAlg, True)          # Get points
    kp2, desRobot = vc.feature_extraction(imgRobotGray, kpRobot, feAlg) # Get features

    good = vc.matching2(imgROI, imgRobot, desROI, desRobot,
                        kp1, kp2, feAlg) # Find matching point

    printMatches(imgROI, imgRobot, kp1, kp2, good, algsName[i]+"_"+str(z))
print("\n")

```

Bibliografia

- [1] Richard Szeliski. “Computer Vision: Algorithms and Applications”. 2010. URL: http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.
- [2] David G. Lowe. “Object recognition from local scale-invariant features”. A: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 2. 1999, pàg. 1150-1157. DOI: 10.1109/ICCV.1999.790410. URL: <http://www.cs.ubc.ca/~lowe/papers/iccv99.pdf>.
- [3] Herbert Bay et al. “Speeded-Up Robust Features (SURF)”. A: *Comput. Vis. Image Underst.* 110.3 (juny de 2008), pàg. 346-359. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.014. URL: <http://dx.doi.org/10.1016/j.cviu.2007.09.014>.
- [4] V. Lepetit T. Trzcinski M. Christoudias i P. Fua. “Boosting Binary Key-point Descriptors”. A: *Computer Vision and Pattern Recognition*. 2013. URL: <https://infoscience.epfl.ch/record/186246/files/top.pdf>.
- [5] Gil Levi i Tal Hassner. “LATCH: Learned Arrangements of Three Patch Codes”. A: *Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016. URL: <http://www.openu.ac.il/home/hassner/projects/LATCH>.
- [6] Yani Dzhurov, Iva Krasteva i Sylvia Ilieva. “Personal Extreme Programming—An Agile Process for Autonomous Developers”. A: (2009). URL: <https://www.researchgate.net/publication/229046039>.
- [7] Chris Harris i Mike Stephens. “A combined corner and edge detector”. A: *In Proc. of Fourth Alvey Vision Conference*. 1988, pàg. 147-151. URL: www.bmva.org/bmvc/1988/avc-88-023.pdf.
- [8] Intel Corporation, Willow Garage i Itseez. *OpenCV*. 2000. URL: <http://opencv.org>.
- [9] Alexandre Thomas i Dmitry Barashev. *Gantt Project*. 2003. URL: <http://www.ganttpoint.biz/>.

Índex de taules

2.1	Blocs del projecte	16
2.2	Lliurables de GEP	17
2.3	Tasques desenvolupament	17
2.4	Recursos de programari	22
6.1	Detectors de keypoints - comparació	45
6.2	Detectors de keypoints - comparació 2	46
6.3	Matching - comparació	47
6.4	Matching - comparació 2	48
7.1	Recursos humans (hores)	51
7.2	Recursos humans (costos)	52
7.3	Recursos de maquinari (costos)	52
7.4	Costos indirectes	52
7.5	Costos totals	53
8.1	Matriu de sostenibilitat	54

Índex de figures

2.1	PERT del projecte	19
2.2	PERT - tasques desenvolupament	19
2.3	Gantt del projecte	20
3.1	Arquitectura del sistema	23
3.2	Selecció de les imatges	24
3.3	Selecció de la regió d'interès	25
3.4	App - Menú	26
3.5	App - Càmera	27
3.6	App - Galeria	27
3.7	App - Selecció de la regió d'interès	28
3.8	App - Opcions	29
4.1	Keypoints	31
4.2	Matching	32
4.3	Homografia	33
6.1	Imatges graff, boat i ubc	45
6.2	Imatges UPC i jardins	46
6.3	Imatges graff i jardins	47
6.4	Imatges UPC	48
6.5	Homografia graff - SIFT	49
6.6	Homografia jardins - ORB+BRISK	49
6.7	Homografia campus - SIFT	50
6.8	Homografia campus 2 - SIFT	50