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

Joan Rodas Cusidó
14-04-2017

*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 mòbils, utilitzant imatges. S'utilitzarà la biblioteca OpenCV i diferents algorismes de visió per computador. El sistema final haurà de permetre al robot trobar i desplaçar-se cap a un punt de l'entorn, partint 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 environment designated by the user, selecting 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	25
4 Servidor	27
4.1 Estructura	27
4.1.1 Flask	28
4.1.2 Nginx	28
4.1.3 uWSGI	28
4.2 Instal·lació del <i>software</i>	29
4.3 Configuració	31
5 Tècniques de visió usades	34
5.1 Preprocessat digital d'imatges	34
5.2 Obtenció de <i>keypoints</i> en una imatge	35
5.3 Extracció de característiques	39
5.4 <i>Matching</i> de característiques	40
5.5 Homografia	41
6 Implementació	43
6.1 Programa principal en Python	43
6.1.1 Preprocessat de les imatges	43
6.1.2 Selecció de la regió d'interès	44
6.1.3 Obtenció de <i>keypoints</i>	45
6.1.4 Extracció de característiques	47
6.1.5 <i>Matching</i> i homografia	48
6.2 Aplicació web	50
6.2.1 Pujar imatge al servidor	50
6.2.2 Selecció de la regió d'interès	51
6.3 Aplicació mòbil	54
6.3.1 Capturar imatge	54
6.3.2 Selecció d'imatges de la galeria	54
6.3.3 Enviament de dades al servidor	54
7 Experiments i resultats	55
7.1 Experiments realitzats	55
7.1.1 Comparació detectors de <i>keypoints</i>	55
7.1.2 Comparació detecció i extracció de <i>keypoints</i>	56
7.2 Resultats i comparació d'algorismes	57
7.2.1 Detectors de <i>keypoints</i>	57
7.2.2 Detecció i extracció de <i>keypoints</i>	59
7.2.3 <i>Matching</i> i homografia	65

8 Gestió econòmica	66
8.1 Recursos de programari	66
8.2 Recursos humans	66
8.3 Recursos de maquinari	67
8.4 Costos indirectes	67
8.5 Imprevistos	68
8.6 Contingència	68
8.7 Costos totals	68
8.8 Control de gestió	68
9 Informe de sostenibilitat	69
9.1 Posada en producció	70
9.1.1 Consum del disseny	70
9.1.2 Factura	70
9.1.3 Impacte personal	70
9.2 Vida útil	71
9.2.1 Petjada ecològica	71
9.2.2 Pla de viabilitat	71
9.2.3 Impacte social	71
9.3 Riscos	72
9.3.1 Riscos ambientals	72
9.3.2 Riscos econòmics	72
9.3.3 Riscos socials	72
10 Conclusions	73
10.1 Conclusions tècniques	73
10.2 Conclusions personals	73
10.3 Treball futur	74
Bibliografia	75
Índex de taules	77
Índex de figures	78

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 projecte es divideix en 10 capítols. 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 detalla la planificació i els recursos utilitzats per realitzar el treball.

Els capítols 3 a 7 conformen el treball principal. Al tercer capítol s'explica el disseny i l'arquitectura del sistema desenvolupat. Al quart, l'estructura i configuració del servidor. Les tècniques de visió utilitzades s'expliquen al cinquè capítol, amb la seva implementació detallada al sisè. Al capítol 7 s'explicaran els experiments realitzats i els resultats obtinguts.

Al capítol 8 podeu trobar una anàlisi de la gestió econòmica del projecte, on es detallen els costos humans, de programari, de maquinari, indirectes i possibles imprevistos. I al novè capítol es presenta l'informe de sostenibilitat. Per acabar, hi haurà les conclusions del projecte, on es valorarà l'aportació del projecte en l'àmbit personal i si s'han complert 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 l'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 directament 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 persona, 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 és 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, 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 (prove, 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, per conèixer 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.

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'aniran realitzant a mesura que avanci el projecte. Algunes d'aquestes tasques seran:

- Obtenció de *keypoints* (Harris)
- Extracció de característiques (SIFT)
- *Matching* i homografia
- Algorismes alternatius (ORB[10], BRISK[11], LATCH...)
- Disseny de l'aplicació d'Android/web
- Creació de l'aplicació d'Android/web

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.

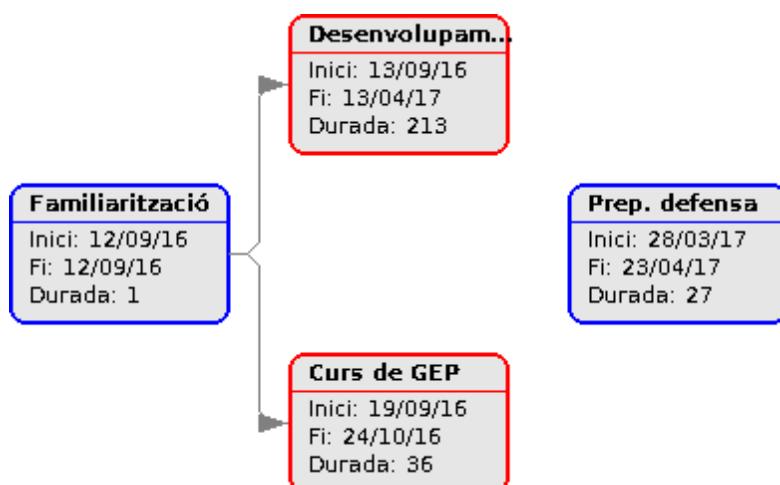


Figura 2.1: PERT del projecte

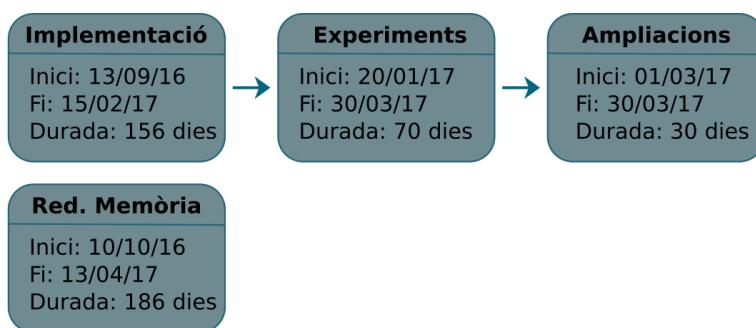


Figura 2.2: PERT - tasques desenvolupament

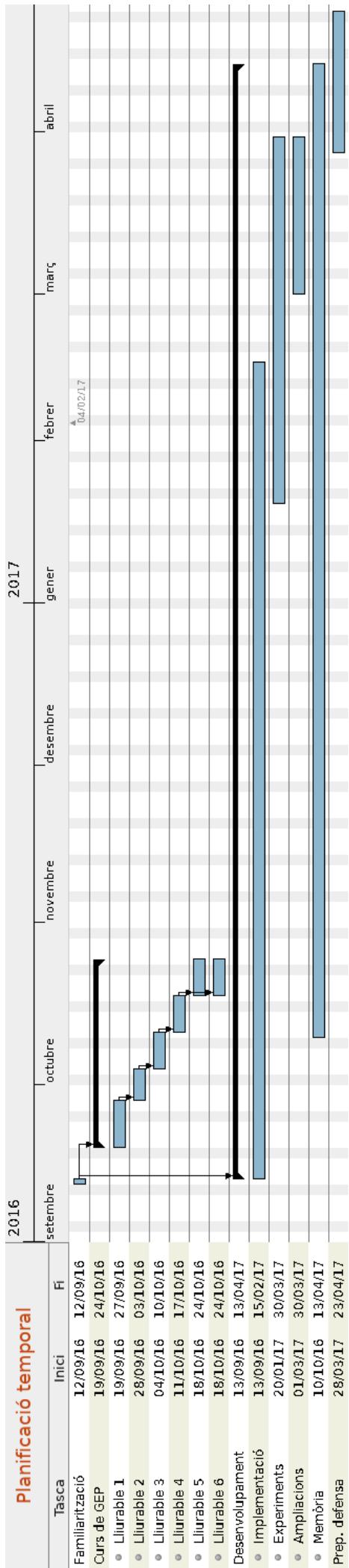


Figura 2.3: Gantt del proyecte

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.

Pel servidor s'ha decidit utilitzar una Raspberry Pi 2.

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/Raspbian	Eina de desenvolupament	Execució del programari
Python	Eina de desenvolupament	Programació
Flask	Eina de desenvolupament	Programació (framework)
OpenCV	Eina de desenvolupament	Algorismes de VC
Nginx	Eina de desenvolupament	Servidor web / proxy
uWSGI	Eina de desenvolupament	Servidor uwsgi
Geany/Atom	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	Documentació	Redacció de la memòria
Zathura	Documentació	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: el client (aplicació web o per *smartphones*), un servidor i el robot. A continuació podeu veure un esquema de l'arquitectura i l'explicació de cada una de les parts.

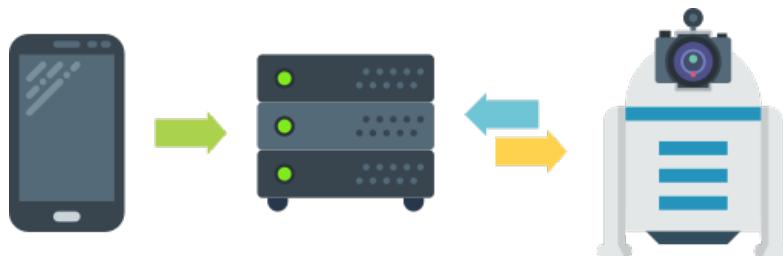


Figura 3.1: Arquitectura del sistema
Font: Madebyoliver i Pixel Buddha

- **Aplicació mòbil/web:** Permet a l'usuari seleccionar una regió en una imatge i l'envia al servidor.
- **Servidor:** Rep la selecció de l'aplicació i les imatges del robot. S'encarrega de fer el *matching* per obtenir el punt on s'haurà de desplaçar el robot.
- **Robot:** Envia les imatges capturades al servidor i espera rebre les ordres per desplaçar-se i girar.

El treball se 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[12], una GUI estàndard de Python.

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 l'opció de refer-la (simplement seleccionant de nou) o d'acceptar-la pulsant la tecla c.

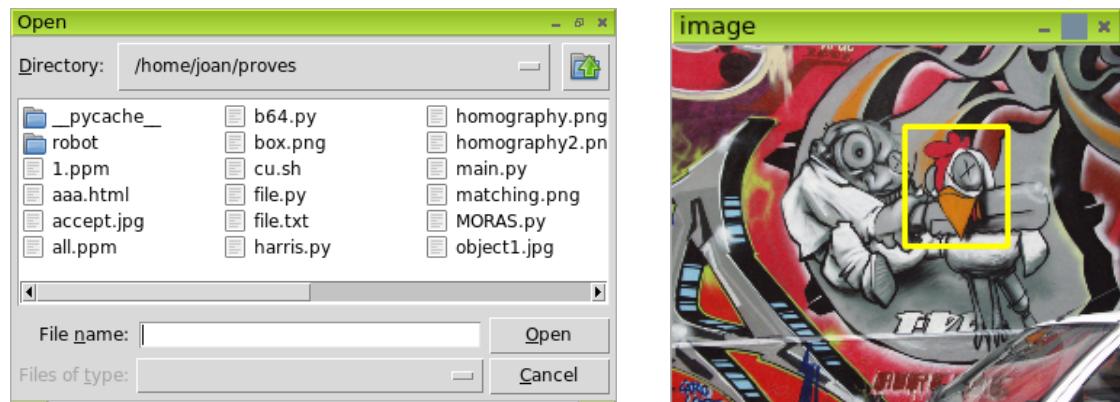


Figura 3.2: Aplicació de proves

3.3 Disseny de l'aplicació mòbil

La interfície de l'aplicació mòbil i de l'aplicació web ha de ser molt simple. L'usuari només hauria de veure una imatge, d'on podria seleccionar-ne una regió i acceptar la selecció.

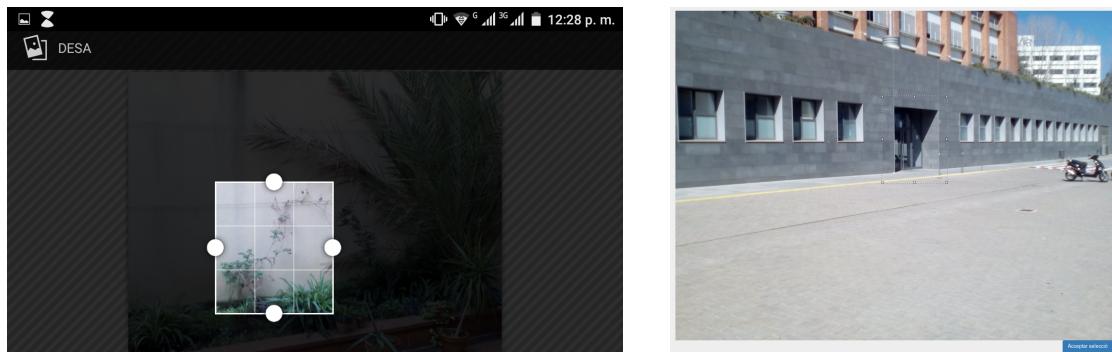


Figura 3.3: App/Webapp - Selecció de la regió d'interès

També s'hauria d'oferir una opció pels administradors, per tal de poder canviar la imatge a mostrar. El menú principal de l'aplicació mòbil hauria de tenir les següents opcions:

Fer una foto

Permet a l'usuari fer una foto amb la càmera del dispositiu, per després pujar-la al servidor.



Figura 3.4: App - Càmera

Seleccionar una imatge

Si ja es disposa d'una imatge de l'entorn a la memòria del dispositiu, aquesta opció permetria a l'usuari seleccionar-la.

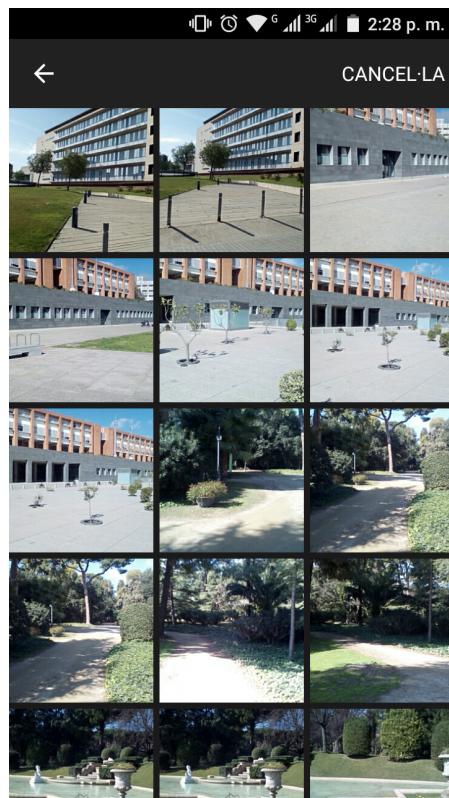


Figura 3.5: App - Galeria

Opcions

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

També s'haurien de poder escollir els algorismes de visió a utilitzar.

4. Servidor

4.1 Estructura

S'utilitzarà una Raspberry Pi 2 com a servidor de proves. La idea és que l'usuari pugui accedir a una aplicació web per utilitzar el sistema, ja sigui desde Internet o des d'una xarxa local.

L'estructura d'aquest servidor serà la següent:

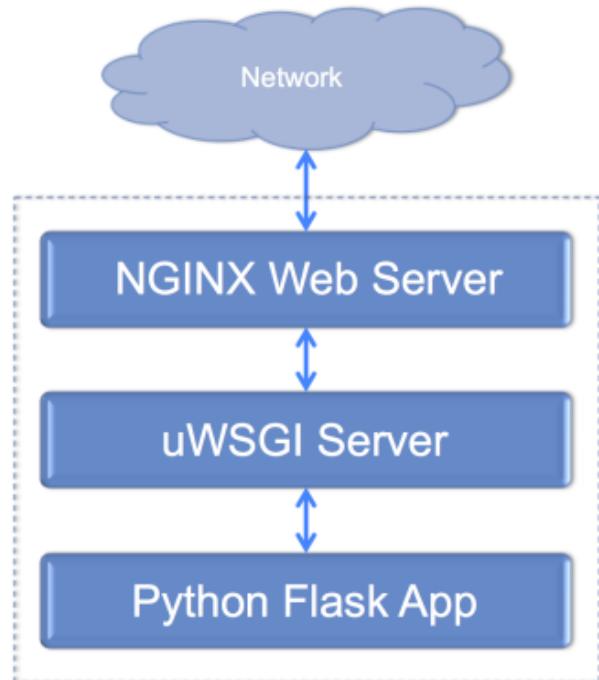


Figura 4.1: Estructura del servidor
Font: <https://iotbytes.wordpress.com>

4.1.1 Flask

Per desenvolupar l'aplicació web he decidit utilitzar Flask[13], un *micro-framework* de Python que permetrà utilitzar el codi desenvolupat del projecte amb facilitat. A més, també permet utilitzar un servidor per provar l'aplicació en la fase de desenvolupament sense necessitat d'inicialitzar el servidor web.

Un exemple d'aplicació en Flask (el típic “Hello World”) seria així:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

4.1.2 Nginx

Nginx[14] és un servidor web i proxy invers lleuger i d'alt rendiment. Com que s'ha decidit utilitzar una Raspberry com a servidor de proves, Nginx és una bona opció, ja que no consumeix gaires recursos i ofereix molt bon rendiment.

Nginx, però, no és capaç de comunicar-se amb l'aplicació web per WSGI. Per tant, necessitarem instal·lar un altre servidor que faci d'intermediari.

4.1.3 uWSGI

uWSGI[15] és un servidor capaç de comunicar-se amb aplicacions web mitjançant el protocol uwsgi.

4.2 Instal·lació del *software*

El sistema operatiu a utilitzar serà la versió lite de Raspbian, el sistema oficial de la fundació de Raspberry. Es pot baixar la imatge des del web de Raspberry i instal·lar a la targeta SD amb la comanda dd.

Tot i que a continuació s'especifiquen les comandes utilitzades amb Raspbian, el mateix procés es podria replicar en qualsevol altre sistema basat en Linux amb poques diferències. I evidentment, el *hardware* tampoc ha de ser el d'una Raspberry Pi 2.

```
$ sudo dd bs=4M if=/home/joan/Downloads/raspbian.img of=/dev/mmcblk0
```

Un cop tenim el sistema operatiu instal·lat, abans de començar a fer res, actualitzarem els paquets del sistema.

```
$ sudo apt-get update && sudo apt-get upgrade
```

Python

Serà necessari instal·lar Python per poder executar el codi principal i l'aplicació web. També caldrà instal·lar altres dependències del projecte com numpy i matplotlib.

```
$ sudo apt-get install python3-pip python3-dev
$ sudo apt-get install python3-numpy
python3-matplotlib
```

OpenCV

Evidentment, com que el projecte està desenvolupat utilitzant la biblioteca OpenCV, serà necessari instal·lar-la al servidor. Abans, però, haurem d'instal·lar algunes eines i biblioteques per poder compilar OpenCV i tractar amb imatges.

```
$ sudo apt-get install cmake
$ sudo apt-get install libjpeg-dev libtiff5-dev
libjasper-dev libpng12-dev
$ sudo apt-get install libatlas-base-dev gfortran
```

La versió a instal·lar és la 3.2, que ens podem baixar des del repositori oficial d'OpenCV al GitHub.

```
$ wget -O opencv.zip https://github.com/Itseez/
opencv/archive/3.2.0.zip
$ unzip opencv.zip
```

Com que també utilitzem algorismes no lliures com SIFT i SURF, també serà necessari descarregar OpenCV Contrib.

```
$ wget -O opencv_contrib.zip https://github.com/
Itseez/opencv_contrib/archive/3.2.0.zip
$ unzip opencv_contrib.zip
```

Un cop obtinguts els arxius necessaris, ja podem preparar, compilar i instal·lar OpenCV al nostre sistema.

```
$ cd ~/opencv-3.2.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_C_EXAMPLES=ON \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=
~/opencv_contrib-3.2.0/modules \
-D BUILD_EXAMPLES=ON ..
$ make -j4
$ sudo make install
$ sudo ldconfig
```

Nginx

Per instal·lar el servidor web només cal utilitzar la comanda apt-get.

```
$ sudo apt-get install nginx
```

Flask

Podem instal·lar Flask a través del gestor de paquets pip.

```
$ sudo pip3 install flask
```

uWSGI

També es pot instal·lar el servidor uWSGI de la mateixa manera.

```
$ sudo pip3 install uwsgi
```

4.3 Configuració

Ara que tenim tots els programes necessaris al servidor, ja podem configurar-lo perquè s'executi la nostra aplicació web quan accedim a la IP del servidor.

S'ha de tenir en compte que la configuració següent ens permetrà accedir a l'aplicació web des de la xarxa local. En cas que es vulgui accedir des de l'exterior, s'haurà de configurar l'encaminador perquè redirigeixi el tràfic del port cap al servidor.

Suposant que l'aplicació web s'anomena “project.py”, el primer que haurem de fer serà crear un *SGI Entry Point* (wsgi.py) que executi l'aplicació i pugui ser utilitzar per uWSGI.

```
$ nano ~/moras/wsgi.py
```

```
from project import app

if __name__ == "__main__":
    app.run()
```

També necessitem l'arxiu de configuració que utilitzarà uWSGI.

```
$ nano ~/moras/moras.ini
```

```
[uwsgi]
module = wsgi:app

master = true
processes = 5

socket = /tmp/moras.sock
chmod-socket = 664
vacuum = true

die-on-term = true
```

Amb això ja podríem executar uWSGI sense problemes, però volem que el servei s'executi de manera automàtica quan iniciem el sistema. Per aconseguir això, afegirem una línia al fitxer ‘rc.local’ que s'encarregui d'executar l'arxiu .ini creat anteriorment.

```
$ sudo nano /etc/rc.local
```

```
[...]

/usr/local/bin/uwsgi --ini /home/pi/moras/moras.ini
--uid www-data --gid www-data
--daemonize /var/log/uwsgi.log

exit 0
```

Per tal que Nginx redirigeixi les peticions al servidor uWSGI, hem de crear un nou lloc a ‘/etc/nginx/sites-available’.

```
$ sudo nano /etc/nginx/sites-available/moras
```

```
server {
    listen 80;
    server_name localhost;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/tmp/moras.sock;
    }
}
```

Finalment, només caldrà activar el lloc web creant un *soft link* a ‘sites-enabled’ i reiniciar el servidor Nginx per aplicar la nova configuració.

```
$ sudo ln -s /etc/nginx/sites-available/moras
/etc/nginx/sites-enabled
$ sudo systemctl restart nginx
```

5. 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.

5.1 Preprocessat digital d'imatges

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

Algunes de les tècniques que es poden aplicar i s'han provat durant la realització del projecte són:

- **Suavitzat de la imatge i reducció de soroll:** S'han provat filtres senzills com la mediana.
- **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 irrelevants o fins i tot portar a errors.
- **Equalització de l'histograma:** Per tal de millorar el contrast de les imatges, s'ha provat d'utilitzar CLAHE (*Contrast Limited Adaptive Histogram Equalization*).
- **Operacions morfològiques:** Erode, dilate, open, close

Finalment, s'ha decidit no aplicar cap filtre i utilitzar les imatges tal com són, ja que o bé no feien cap efecte o funcionaven correctament només amb determinats tipus d'imatges i empitjoraven d'altres. El que sí que s'ha fet és reduir la mida i convertir les imatges a escala de grisos.

5.2 Obtenció de *keypoints* en una imatge

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

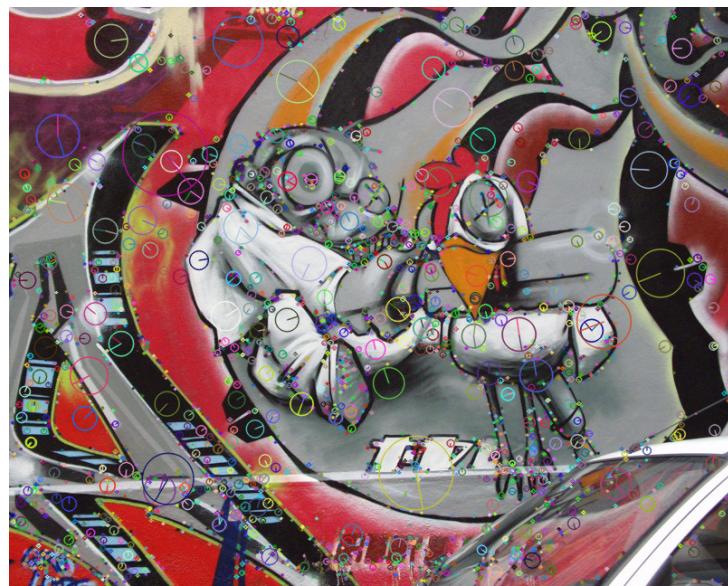


Figura 5.1: *Keypoints*

La detecció es pot classificar en:

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

Les principals tècniques d'obtenció de *keypoints* que s'han utilitzat són: SIFT, Harris i ORB. També s'han provat altres algorismes com FAST[16], SURF, MSER[17] o Shi-Tomasi[18].

Harris

Harris és un detector que es basa a trobar cantonades. La idea bàsica és que mitjançant una finestra de NxM píxels, es recorre la imatge buscant els punts on hi ha canvis d'intensitat en diverses direccions.

Segons els canvis d'intensitat de cada píxel, es poden classificar els punts en *flat*, vores i cantonades.

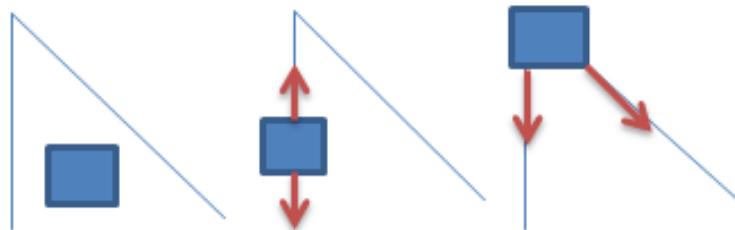


Figura 5.2: *Flat*, vora i cantonada

Font: Wikipedia

El problema principal de Harris és que no és invariant a l'escala. Com podeu veure en la figura 5.3, punts considerats cantonades en una escala, podrien convertir-se en vores en una altra. Per això, existeixen algunes solucions alternatives com Harris-Laplace.

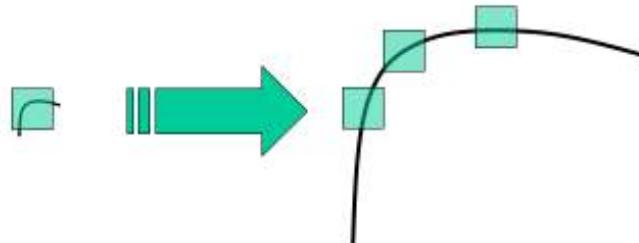


Figura 5.3: *Vores a diferent escala*

Font: OpenCV

En el nostre cas, s'ha optat per aplicar Harris en diverses escales, fent una piràmide de la imatge original.

SIFT

Localització multi-escala mitjançant una diferència de gaussianes (DoG), que s'utilitza com a aproximació d'una laplaciana de gaussianes.

TO-DO DoG s'obté com la diferència de gaussian blurring de la imatge amb diferents *. Aquest procés es realitza per diferents octaves de la imatge en la piràmide gaussiana, tal com es veu en la imatge següent:

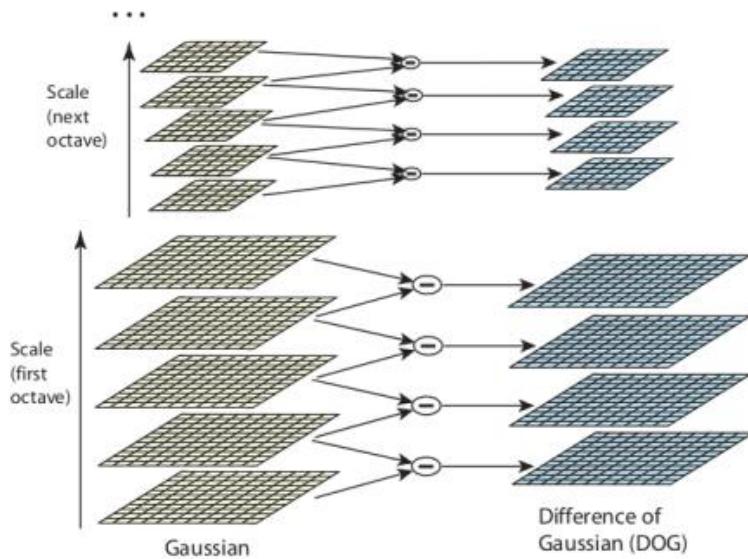


Figura 5.4: *SIFT - DoG*

Font: OpenCV

Un cop tenim les DoG, es busquen local extrema en l'espai i l'escala. Per cada píxel, es compara amb els seus 8 veïns, així com els 9 píxels de la següent escala i els 9 de les anteriors. Si és local extrema el considerem potencial *keypoint*.

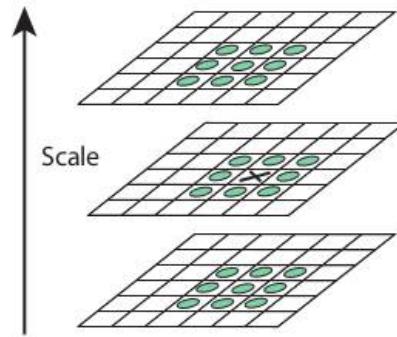


Figura 5.5: *SIFT - Local extrema*

Font: OpenCV

Un cop obtinguts els possibles *keypoints*, s'eliminen els menys robustos. S'utilitza [Taylor] per aconseguir una millor localització del [extrema], i si la seva intensitat

és menor a un llindar (0.03 segons Lowe) es rebutja. També s'han d'eliminar vores, utilitzant un mètode similar a Harris. S'agafa una matriu hessiana de 2x2 per calcular la principal corbatura. Sabent que les vores tenen un eigen-value major a l'altre, si la ratio dels eigen-values supera un llindar (10 per Lowe) es descarta el punt.

SIFT també assigna una orientació als punts d'interés per aconseguir ser invariant a la rotació. S'agafa un veinatge al voltant de la localització del *keypoint* segons l'escala i es calcula la magnitud i direcció del gradient. Es crea un histograma d'orientació de 36 *bins* que cobreix els 360 graus.

ORB

El detector d'ORB (*Oriented FAST and Rotated BRIEF*) utilitza el detector FAST amb modificacions per millorar el rendiment.

FAST és un detector de cantonades enfocat a aconseguir els punts de manera molt ràpida, a canvi d'empitjorar una mica l'eficàcia. S'agafa la intensitat d'un píxel i es compara amb el conjunt de N píxels veïns que el rodegen.

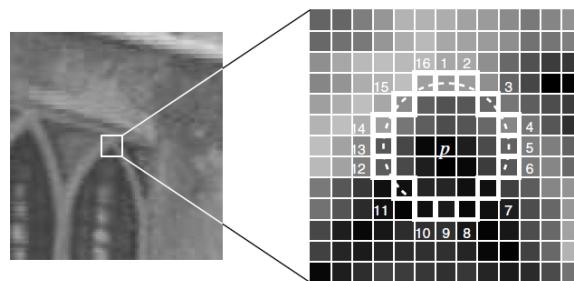


Figura 5.6: FAST, N=16

Font: -

Suposant que $N = 16$ i definint un llindar t , si 12 píxels veïns ($\frac{3}{4}$ parts) són majors a $I_p + t$ o bé menors a $I_p - t$, es considera que el punt és d'interès. El problema principal de FAST és que no té en compte l'orientació.

ORB utilitza FAST aplicant les següents millores:

- S'agafen els N millors punts després d'aplicar la mesura de Harris.
- Es fa una piràmide per fer multi-escala.

- S'utilitzen els moments per calcular l'orientació.

5.3 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

Principalment s'han utilitzat SIFT, ORB i BRISK, encara que també s'han provat d'altres com SURF, LATCH o DAISY[19].

SIFT

Per cada *keypoint*, s'agafa un veïnatge de 16x16 píxels al voltant del punt. Això es divideix en 16 blocs de mida 4x4. Per cada sub-block, es calcula l'histograma d'orientacions en 8 direccions, de manera que podem obtenir 128 valors, que es representaran en forma de vector.

A part d'aquest procés, s'aplicaran també mesures per fer més robust el descriptor contra canvis d'intensitat, rotació, etc.

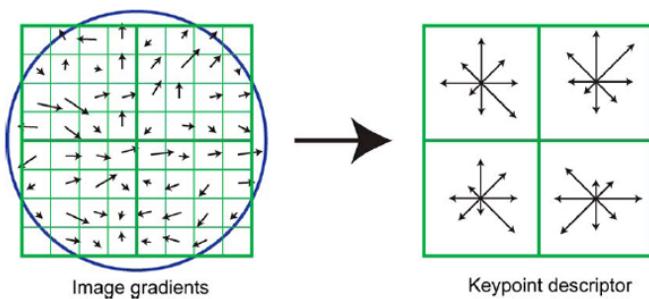


Figura 5.7: Descriptor SIFT

Font: -

ORB

El descriptor d'ORB és una modificació de BRIEF[20], un descriptor ràpid i senzill basat en *strings* binaris.

Per cada *keypoint* s'agafen N punts veïns i d'aquests s'agafen parells de forma més o menys aleatòria. Per cada parell es compara la intensitat i es retorna un *string* binari de mida N amb '1' o '0' segons si la intensitat del primer punt és major a la del segon o no. Això ens permet comparar els descriptors amb una simple operació XOR.

És un mètode molt ràpid, però no és invariable ni a la rotació ni a l'escala. Per això, ORB intenta solucionar el problema de la rotació girant els patrons en funció de l'angle de la característica.

BRISK

És un descriptor binari que utilitza un patró de cercles concèntrics. S'agafen N punts del patró i per cada parell de punts es compara la intensitat del primer amb el segon. Si el valor del primer és major, es posa '1', si no '0'.

D'aquesta manera s'obté una cadena de N caràcters (binari) molt fàcil de comparar a l'hora de fer *matching*. Utilitzant el mateix patró i seqüència de parells, només caldrà comparar les cadenes binàries amb la suma del resultat d'una XOR.

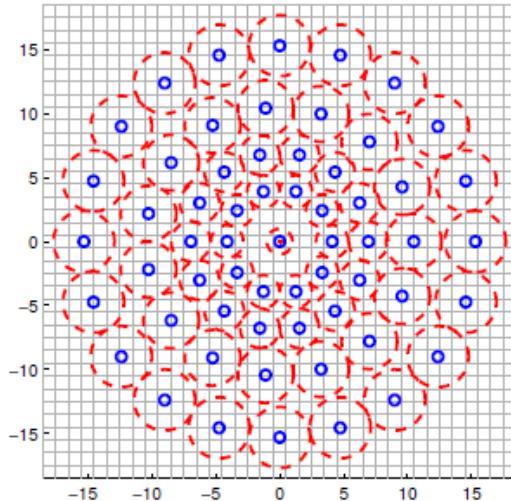


Figura 5.8: *Sampling pattern* BRISK

Font: -

5.4 *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 aparellaments de dues maneres:

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

Com que el temps d'execució no és un factor essencial, s'aplicarà el mètode de força bruta, una mica més lent. Pels descriptors binaris s'utilitzarà la distància de Hamming, mentre que pels vectorials s'utilitzarà l'euclidiana.

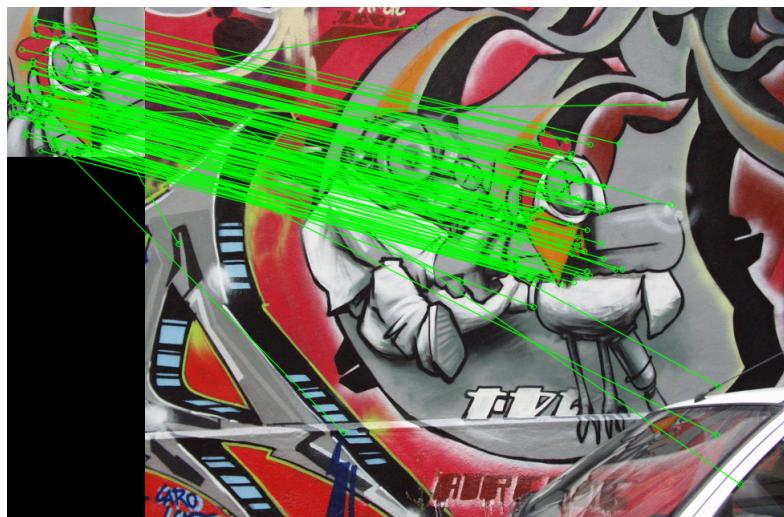


Figura 5.9: *Matching*

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 la ràtio de Lowe o eliminant *outliers*.

5.5 Homografia

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

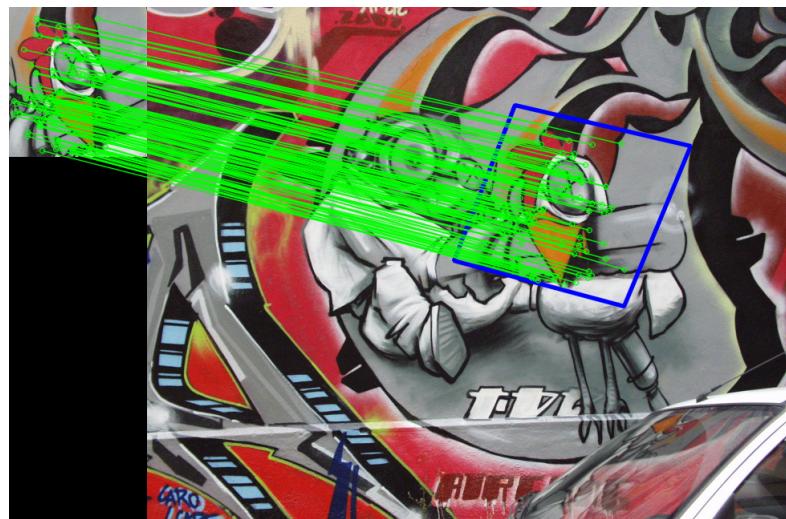


Figura 5.10: Homografia

A l'hora de buscar l'homografia aplicarem RANSAC (*Random Sample Consensus*)[21], un algorisme que ens permetrà eliminar *outliers* dels *match* trobats.

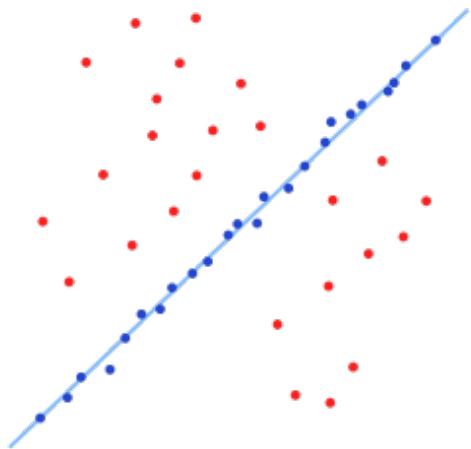


Figura 5.11: RANSAC

Font: Wikipedia

6. Implementació

6.1 Programa principal en Python

Per facilitar la utilització del codi en possibles adaptacions o millores futures, s'ha decidit implementar una petita biblioteca¹ en Python amb totes les funcions necessàries. El programa principal farà ús d'aquesta biblioteca, que permetrà:

1. Preprocessar 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

6.1.1 Preprocessat de les imatges

S'han provat diverses tècniques de preprocessat 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=0.5, fy=0.5)
    return img, cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

¹Codi disponible a <https://github.com/wildux/tfg>

6.1.2 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

```

6.1.3 Obtenció de *keypoints*

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

La funció **point_selection**(gray, alg) serà l'encarregada de cridar l'algoritme escollit per l'usuari (per defecte s'utilitzarà Harris). Els paràmetres necessaris seran 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. Modificarem el paràmetre *sigma* perquè funcioni millor amb les imatges utilitzades.

```
if alg == _SIFT:
    sift = cv2.xfeatures2d.SIFT_create(sigma=1.4)
    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 = 2500, 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 ($\frac{1}{2}$) 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 fàcilment i també té l'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=2000,
                                         qualityLevel=0.05, minDistance=4, 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*3)
            kp.append(k)
    
```

MSER

Per últim, tindrem l'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)
    
```

6.1.4 Extracció de característiques

De la mateixa manera en què 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)
```

6.1.5 *Matching* i homografia

La funció **matching()** retornarà els aparellaments trobats.

```
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 de Hamming, mentre que per la resta s'utilitzarà l'euclidiana. 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 aparellaments, aplicarem el *ratio test* per descartar-ne alguns.

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

Homografia

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

```
homography(img1, img2, des1, des2, kp1, kp2, good):
    ...
    draw_params = dict(matchColor = (0,255,0), singlePointColor = None,
        matchesMask = matchesMask, flags = 2)
    img3 = cv2.drawMatches(img1,kp1,img2C,kp2,good,None,**draw_params)
    return x, y, img3
```

Si hi ha prou coincidències (considerem acceptable com a mínim 10), es buscarà l'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)
    img2C = cv2.polylines(img2C,[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 mitjà.

```
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
img2C = cv2.circle(img2C,(int(x),int(y)), 5, (0,0,255), -1)
```

6.2 Aplicació web

L'aplicació web s'ha desenvolupat amb Flask (Python).

6.2.1 Pujar imatge al servidor

Per poder canviar la imatge que mostrerà el sistema, es defineix la ruta '/update', que ens mostra un formulari per penjar un arxiu (imatge).

```

app.config['UPLOAD_FOLDER'] = 'static/'
app.config['ALLOWED_EXTENSIONS'] = set(['png', 'jpg', 'jpeg', 'gif'])

def allowed_file(filename):
    return '.' in filename and
       filename.rsplit('.', 1)[1] in app.config['ALLOWED_EXTENSIONS']

@app.route("/update")
def update():
    return render_template('upload.html')

@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']
    if file and allowed_file(file.filename):
        ext = filename.rsplit('.', 1)[1]
        filename = "scene." + ext
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    return redirect(url_for('uploaded_file', filename=filename))

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
  
```

El formulari de l'arxiu 'upload.html' és el següent:

```
<form action="upload" method="post"
    enctype="multipart/form-data">
    <div class="col-lg-6 col-sm-6 col-12">
        <label class="input-group-btn">
            <span class="btn btn-primary">
                Browse... <input type="file"
                name="file" style="display: none;">
            </span>
        </label>
    </div>
    <div class="col-lg-6 col-sm-6 col-12">
        <input class="btn btn-primary" type="submit"
            value="Pujar imatge">
    </div>
</form>
```

6.2.2 Selecció de la regió d'interès

Per seleccionar la regió d'interès s'ha utilitzat jQuery i el *plugin imgAreaSelect*, essent també compatible amb dispositius mòbils.

En la pàgina principal es mostra una imatge i un botó per acceptar la selecció. El *plugin imgAreaSelect* s'encarrega de permetre a l'usuari fer una selecció i actualitzar els camps amagats del formulari.

```

<form action="send" method="post" >
    <input type="hidden" name="x1" value="" />
    <input type="hidden" name="y1" value="" />
    <input type="hidden" name="x2" value="" />
    <input type="hidden" name="y2" value="" />
    <input type="hidden" name="width" value="" />
    <input type="hidden" name="height" value="" />
    <input class="btn btn-primary" style="float:right"
        type="submit" value="Acceptar sel." />
</form>

<script type="text/javascript">
    jQuery(document).ready(function () {
        var img = document.getElementById('scene');
        document.getElementById('width')
            .value = img.clientWidth
        document.getElementById('height')
            .value = img.clientHeight

        jQuery('#img#scene').imgAreaSelect({
            handles: true,
            persistent: true,
            x1: 50, y1: 50, x2: 200, y2: 200,
            OnInit: function ( image, selected) {
                jQuery('input[name=x1]').val(selected.x1);
                jQuery('input[name=y1]').val(selected.y1);
                jQuery('input[name=x2]').val(selected.x2);
                jQuery('input[name=y2]').val(selected.y2);
            },
            onSelectEnd: function ( image, selected) {
                jQuery('input[name=x1]').val(selected.x1);
                jQuery('input[name=y1]').val(selected.y1);
                jQuery('input[name=x2]').val(selected.x2);
                jQuery('input[name=y2]').val(selected.y2);
            }
        });
    });
</script>

```

El formulari ens porta a la ruta ‘/send’, que executarà el codi principal de visió per computador. Finalment podrem veure el resultat obtingut del *matching* a ‘result.html’.

```

imgPath = 'static/scene.jpg'
imgRobotPath = 'static/uni.jpg'

@app.route("/")
def index():
    return render_template('index.html')

@app.route('/send', methods=['POST'])
def send():
    x1 = int(request.form['x1'])
    x2 = int(request.form['x2'])
    y1 = int(request.form['y1'])
    y2 = int(request.form['y2'])
    height = int(request.form['height'])
    width = int(request.form['width'])

    img = cv2.imread(imgPath)
    h,w,_ = img.shape
    scale_h = h/height
    scale_w = w/width
    x1 = int(x1*scale_w)
    x2 = int(x2*scale_w)
    y1 = int(y1*scale_h)
    y2 = int(y2*scale_h)

    img = img[y1:y2, x1:x2]
    imgRobot = cv2.imread(imgRobotPath)
    imgRes, x, y = vc.getResult(img, imgRobot, vc.SIFT)
    resultPath = "static/" + strftime("%Y-%m-%d-%H:%M") + ".png"
    cv2.imwrite(resultPath, imgRes)

    return render_template('result.html', x=x, y=y, image=resultPath)
  
```

6.3 Aplicació mòbil

L'aplicació mòbil s'està desenvolupant amb Android Studio i el llenguatge de programació Java (amb l'SDK d'Android).

6.3.1 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 des d'una altra aplicació, només ha sigut necessari fer l'Intent corresponent.

```
if (v.getId() == R.id.capture_btn) try {
    Intent captureIntent = new
        Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    startActivityForResult(captureIntent,
        CAMERA_CAPTURE);
} catch (ActivityNotFoundException anfe) {...}
```

6.3.2 Selecció d'imatges de la galeria

Android també disposa d'una activitat que ens permet obrir la galeria des d'una altra aplicació, així que s'ha seguit el mateix procés que per obrir la càmera.

```
else if (v.getId() == R.id.browse_btn) try {
    Intent galleryIntent = new
        Intent(Intent.ACTION_PICK,
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    startActivityForResult(galleryIntent,
        RESULT_GALLERY);
} catch (ActivityNotFoundException anfe) {...}
```

6.3.3 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.

Inicialment s'havia pensat d'enviar la imatge seleccionada, codificada en base 64, però només funcionava amb imatges molt petites. Per tant, s'ha considerat millor opció utilitzar alguna biblioteca d'Android com Volley o Retrofit.

7. Experiments i resultats

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

7.1 Experiments realitzats

S'han realitzat diversos experiments per comprovar tant la detecció de *keypoints* com la fiabilitat del *matching*. Els paràmetres utilitzats en tots els algorismes són els descrits al Capítol 6: Implementació.

Podeu trobar el codi dels scripts utilitzats per la realització dels experiments al GitHub del projecte.

7.1.1 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*.

Els algorismes a comparar són:

- Harris
- SURF
- MSER
- SIFT
- ORB

Per mesurar el temps d'execució, s'ha fet la mitjana de 5 execucions de la funció d'obtenció de *keypoints* utilitzada.

Un cop obtinguts els temps d'execució, s'ha analitzat la repetibilitat dels detectors a ull, utilitzant imatges similars i subimatges.

7.1.2 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 dels aparellaments, que serà el més important pel sistema desenvolupat.

Es compararan els següents algorismes:

- Harris + SIFT
- SIFT + SIFT
- ORB + ORB
- ORB + BRISK

En primer lloc es provaran els algorismes utilitzant subimatges. Després es provarà el sistema amb imatges amb poques variacions (canvis d'il·luminació, *blur...*). L'última prova serà amb fotografies del campus i els jardins de palau reial, que presenten canvis de perspectiva i zoom entre d'altres.

Per comprovar els aparellaments, s'hauran de mirar manualment un per un, ja que les imatges no ens permeten utilitzar cap mecanisme per automatitzar el procés.

7.2 Resultats i comparació d'algorismes

7.2.1 Detectors de *keypoints*

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



Figura 7.1: Imatges graff, boat i ubc

Algorismes	Graff		Boat		Ubc	
	Punts	Temps (s)	Punts	Temps (s)	Punts	Temps (s)
Harris	587	0.0128	686	0.0071	535	0.0060
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 7.1: 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.

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 7.2: Imatges UPC i jardins

Algorismes	Campus		Jardins		Campus 2	
	Punts	Temps (s)	Punts	Temps (s)	Punts	Temps (s)
Harris	1553	0.0789	1282	0.0779	1302	0.0785
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	0.0878	2500	0.0618
MSER	2135	0.6790	1141	0.3930	1798	0.6397

Taula 7.2: 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. Els més ràpids amb diferència són ORB i Harris. MSER i Harris són els que obtenen menys *keypoints*.

7.2.2 Detecció i extracció de *keypoints*

En primer lloc es prova el sistema amb imatges similars i subimatges, per comprovar que funciona correctament en els casos més senzills.



Figura 7.3: Imatges campus

Algorismes	Campus 1 (subimatge)				Campus 2 (subimatge)			
	Kp1	Kp2	Parells	t	Kp1	Kp2	Parells	t
Harris + SIFT	1133	1554	835	1.463s	265	1308	187	1.070s
SIFT + SIFT	3518	5769	2784	2.357s	1269	6168	1134	1.910s
ORB + ORB	2500	2500	793	0.181s	2500	2500	243	0.182s
ORB + BRISK	2500	2500	1012	1.073s	2500	2500	301	1.061s

Taula 7.3: Extracció - Subimatges

Algorismes	Campus 1 (subimatge)		Campus 2 (subimatge)	
	Correctes	Erronis	Correctes	Erronis
Harris + SIFT	816	19	184	3

Taula 7.4: Matching - Subimatges

En general, quan s'utilitzen subimatges el sistema funciona correctament. Es detecten bastants aparellaments i la majoria són correctes. Com es pot veure en la taula anterior, Harris+SIFT encerta en la majoria dels aparellaments, tot i que falla en alguns casos.

Analitzant els casos on hi ha falsos aparellaments, podem veure que les característiques són molt similars, fallant sobretot en la finestra correcta.

Pel que fa al temps d'execució, ORB és amb diferència el més ràpid, mentre que SIFT és el més lent.



Figura 7.4: Imatges motos i cotxes

Algorismes	Motos				Cotxes			
	Kp1	Kp2	Parells	t	Kp1	Kp2	Parells	t
Harris + SIFT	126	242	111	0.162s	135	136	102	0.148s
SIFT + SIFT	1195	1237	515	0.374s	612	523	302	0.209s
ORB + ORB	2500	2500	1132	0.096s	2500	2500	907	0.061s
ORB + BRISK	2500	2500	1250	0.994s	2500	2500	1161	0.983s

Taula 7.5: Extracció - imatges similars

Algorismes	Motos		Cotxes	
	Correctes	Erronis	Correctes	Erronis
Harris + SIFT	110	1	98	4

Taula 7.6: Matching - imatges similars

Amb canvis d'il·luminació i *blur*, els algorismes també presenten una bona taxa d'encert. ORB continua sent el més ràpid, però en aquest cas ORB+BRISK és més lent que SIFT, ja que es detecten més aparellaments.

També s'han realitzat proves amb les següents fotografies, que presenten canvis de perspectiva i zoom:



Figura 7.5: Imatges campus i jardins

Algorismes	Campus				Jardins			
	Kp1	Kp2	Parells	t	Kp1	Kp2	Parells	t
Harris + SIFT	417	1099	71	1.714s	757	1276	66	0.917s
SIFT + SIFT	1342	5921	150	2.595s	5148	12582	120	4.328s
ORB + ORB	2500	2500	180	0.221s	2500	2500	86	0.272s
ORB + BRISK	2500	2500	143	1.093s	2500	2500	87	1.145s

Taula 7.7: Extracció - canvis de perspectiva i zoom

Algorismes	Campus		Jardins	
	Correctes	Erronis	Correctes	Erronis
Harris + SIFT	47	24	64	2
SIFT + SIFT	94	56	91	29
ORB + ORB	112	68	62	24
ORB + BRISK	78	65	81	6

Taula 7.8: Matching - canvis de perspectiva i zoom

Quan utilitzem escenes diferents, amb canvis de zoom i de perspectiva, el nombre d'aparellaments disminueix considerablement. La taxa d'encert també disminueix, però sempre superant el 50%.

En les imatges de la universitat, el millor algorisme és Harris+SIFT (tot i que troba menys aparellaments), mentre ORB+BRISK seria el pitjor. En el cas dels jardins, tots els algorismes presenten una bona taxa d'encert, especialment Harris+SIFT i ORB+BRISK.

Pel que fa als temps d'execució, SIFT és bastant més lent, encara que troba molts més *keypoints*. Harris en comparació troba molts menys punts però acaba funcionant millor.

Algorismes	Campus			Jardins		
	RANSAC	OK	Erronis	RANSAC	OK	Erronis
Harris + SIFT	30	30	0	24	24	0
SIFT + SIFT	41	41	0	31	31	0
ORB + ORB	60	60	0	33	33	0
ORB + BRISK	42	42	0	41	41	0

Taula 7.9: RANSAC - canvis de perspectiva i zoom

Un cop aplicat RANSAC, el nombre d'aparellaments restants es redueix un altre cop, però en aquest cas tots els aparellaments són correctes.

També s'ha provat el sistema amb regions més concretes, que és són realment el tipus d'imatges que utilitzarà.



Figura 7.6: Imatges campus i jardins (objectes)

Algorismes	Campus (moto)				Jardins (font)			
	Kp1	Kp2	Parells	t	Kp1	Kp2	Parells	t
Harris + SIFT	67	417	30	0.790s	304	757	38	0.375s
SIFT + SIFT	125	1342	16	1.058s	570	5148	51	1.110s
ORB + ORB	830	2500	15	0.077s	2405	2500	62	0.130s
ORB + BRISK	830	2500	19	0.940s	2405	2500	77	1.037s

Taula 7.10: Extracció - objectes

Algorismes	Campus (moto)		Jardins (font)	
	Correctes	Erronis	Correctes	Erronis
Harris + SIFT	30	0	38	0
SIFT + SIFT	15	1	49	2
ORB + ORB	14	1	52	10
ORB + BRISK	19	0	71	6

Taula 7.11: Matching - objectes

En les imatges més petites, com era d'esperar, s'obtenen menys *keypoints*. Per tant el nombre total d'aparellaments trobats també serà menor.

Harris+SIFT funciona molt bé amb aquests tipus d'imatges, essent tots els aparellaments trobats correctes. La resta d'algorismes també obtenen bons resultats, essent ORB el pitjor (amb un 93% i 84% d'encert).

Algorismes	Campus (moto)			Jardins (font)		
	RANSAC	OK	Erronis	RANSAC	OK	Erronis
Harris + SIFT	18	18	0	23	23	0
SIFT + SIFT	13	13	0	26	26	0
ORB + ORB	12	12	0	33	33	0
ORB + BRISK	18	18	0	49	49	0

Taula 7.12: RANSAC - objectes

Un cop aplicat RANSAC, tots els aparellaments considerats *inliers* són correctes.

Finalment, s'ha provat el sistema utilitzant parells d'imatges totalment diferents, per comprovar que el nombre d'aparellaments trobats no sigui elevat.



Figura 7.7: Imatges campus i jardins (diferents)

Algorismes	Campus 1 (subimatge)				Campus 2 (subimatge)			
	Kp1	Kp2	Parells	t	Kp1	Kp2	Parells	t
Harris + SIFT	67	757	0	0.331s	304	417	2	0.839s
SIFT + SIFT	125	5148	1	0.987s	570	1342	5	1.113s
ORB + ORB	830	2500	2	0.840s	2405	2500	7	0.113s
ORB + BRISK	830	2500	1	0.942s	2405	2500	7	1.003s

Taula 7.13: *Matching* - imatges diferents

Analitzant els resultats es pot veure que no es troben gaires aparellaments, però tot i així se'n troben alguns. Harris+SIFT sembla ser el que en troba menys.

7.2.3 Matching i homografia

Aquí podeu veure alguns dels resultats obtinguts en executar el programa seleccióant una regió més concreta:

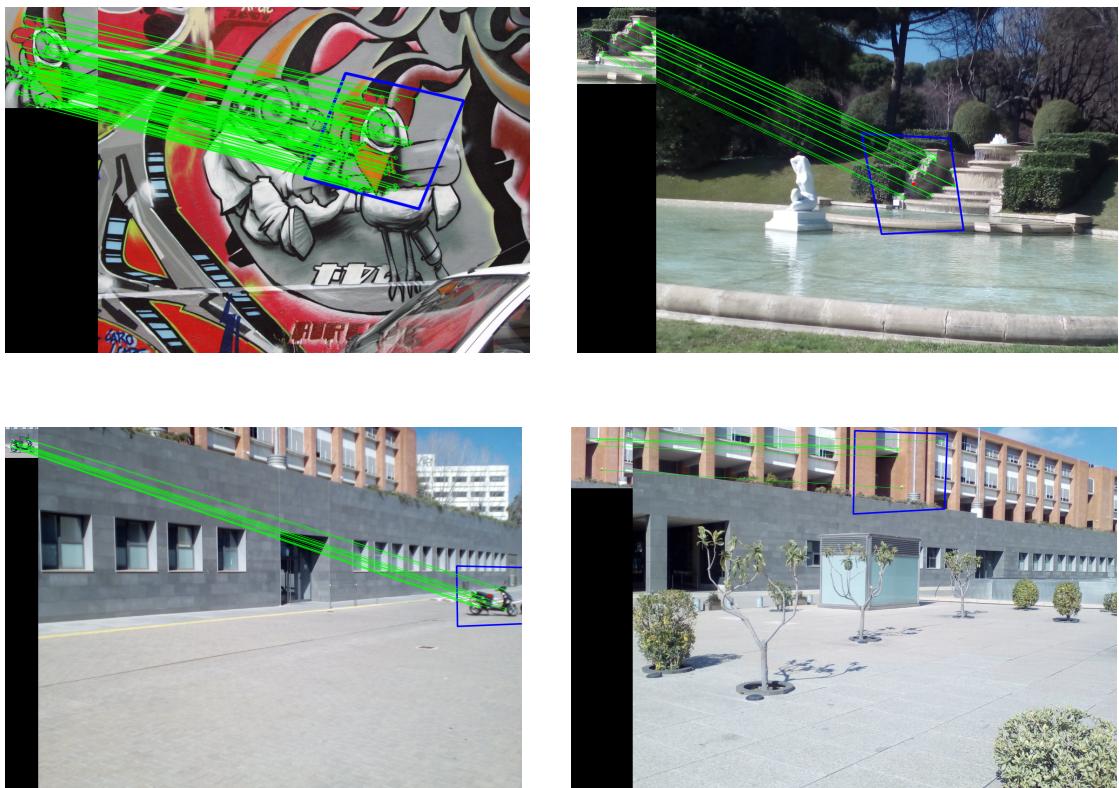


Figura 7.8: Homografia - Resultats

8. Gestió econòmica

8.1 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).

8.2 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 8.1: 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 8.2: Recursos humans (costos)

8.3 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 8.3: Recursos de maquinari (costos)

8.4 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 8.4: Costos indirectes

8.5 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.

8.6 Contingència

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

8.7 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 8.5: Costos totals

8.8 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).

9. 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 mateix projecte, considerant tota la construcció i la vida útil d'aquest.

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 9.1: Matriu de sostenibilitat

9.1 Posada en producció

En aquesta secció es detalla la sostenibilitat del sistema des de 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.

9.1.1 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.

9.1.2 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 hi ha hagut modificacions en els recursos de programari o maquinari.

9.1.3 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 en l'àmbit professional.

9.2 Vida útil

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

9.2.1 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.

9.2.2 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 mateixos 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 (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.

9.2.3 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, sigui manualment o amb altres mètodes de localització, com podria ser utilitzant les coordenades GPS. També hi ha sistemes que utilitzen marques visuals (punts de referència) o que operen en un entorn conegut pel robot. El propòsit 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.

9.3 Riscos

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

9.3.1 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 depèn de la feina de l'autor.

9.3.2 Riscos econòmics

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

9.3.3 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 és 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.

L'única dependència del programa principal és 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 algun usuari volgués comercialitzar un producte derivat d'aquest projecte.

10. 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.

En general, l'objectiu principal del projecte, crear un sistema d'autolocalització, s'ha complert.

10.1 Conclusions tècniques

Veient els resultats dels experiments realitzats, considero que la taxa d'encert del sistema és acceptable. Tot i així, crec que encara hi ha molt marge de millora.

Analitzant els resultats, considero que Harris és el detector de punts d'interès que obté punts més robustos i juntament amb el descriptor de SIFT aconsegueix bons aparellaments. Per altra banda, ORB sembla una bona alternativa en casos en què el temps d'execució sigui crític.

10.2 Conclusions personals

Aquest treball m'ha permès profunditzar els meus coneixements sobre visió per computador. També m'ha ajudat en poder planificar projectes, gestionant recursos i establint objectius. Considero que de cara al món laboral la realització d'aquest projecte ha estat una experiència molt positiva.

10.3 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:

- **Comparació i anàlisi d'algorismes:** És necessari fer una anàlisi més exhaustiva dels diversos algorismes de detecció i extracció de característiques.
- **Diferents imatges:** El sistema s'hauria de provar amb imatges diverses. Imatges d'entorns diferents, condicions diferents i capturades amb diverses càmeres.
- **Pre-processat:** S'hauria d'investigar amb més profunditat quines tècniques de pre-processat podrien ajudar als algorismes a detectar millors *keypoints* i característiques.
- **Aplicació mòbil:** S'hauria d'acabar l'aplicació d'Android.
- **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.

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.openv.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/>.

- [10] Ethan Rublee et al. “ORB: An Efficient Alternative to SIFT or SURF”. A: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pàg. 2564 - 2571. ISBN: 978-1-4577-1101-5. DOI: 10.1109/ICCV.2011.6126544. URL: <http://dx.doi.org/10.1109/ICCV.2011.6126544>.
- [11] Stefan Leutenegger, Margarita Chli i Roland Y. Siegwart. “BRISK: Binary Robust Invariant Scalable Keypoints”. A: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pàg. 2548 - 2555. ISBN: 978-1-4577-1101-5. DOI: 10.1109/ICCV.2011.6126542. URL: <http://dx.doi.org/10.1109/ICCV.2011.6126542>.
- [12] Steen Lumholt i Guido van Rossum. *Tkinter*. URL: <http://tkinter.unpythonic.net/wiki/>.
- [13] Armin Ronacher. *Flask*. 2010. URL: <http://flask.pocoo.org/>.
- [14] Igor Sysoev. *Nginx*. 2004. URL: <https://nginx.org/>.
- [15] *uWSGI*. URL: <https://uwsgi-docs.readthedocs.io>.
- [16] Edward Rosten i Tom Drummond. “Machine Learning for High-speed Corner Detection”. A: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. ECCV’06. Graz, Austria: Springer-Verlag, 2006, pàg. 430 - 443. ISBN: 3-540-33832-2, 978-3-540-33832-1. DOI: 10.1007/11744023_34. URL: http://dx.doi.org/10.1007/11744023_34.
- [17] J. Matas et al. “Robust Wide Baseline Stereo from Maximally Stable Extremal Regions”. A: *Proc. BMVC*. doi:10.5244/C.16.36. 2002, pàg. 36.1 - 36.10. ISBN: 1-901725-19-7.
- [18] Jianbo Shi i Carlo Tomasi. *Good Features to Track*. Inf. tèc. Ithaca, NY, USA, 1993.
- [19] E. Tola, V. Lepetit i P. Fua. “DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo”. A: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.5 (2010), pàg. 815 - 830.
- [20] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. A: *Proceedings of the 11th European Conference on Computer Vision: Part IV*. ECCV’10. Heraklion, Crete, Greece: Springer-Verlag, 2010, pàg. 778 - 792. ISBN: 3-642-15560-X, 978-3-642-15560-4. URL: <http://dl.acm.org/citation.cfm?id=1888089.1888148>.
- [21] Martin A. Fischler i Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. A: *Commun. ACM* 24.6 (juny de 1981), pàg. 381 - 395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: <http://doi.acm.org/10.1145/358669.358692>.

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
7.1	Detectors de <i>keypoints</i> - comparació	57
7.2	Detectors de <i>keypoints</i> - comparació 2	58
7.3	Extracció - Subimatges	59
7.4	<i>Matching</i> - Subimatges	59
7.5	Extracció - imatges similars	60
7.6	<i>Matching</i> - imatges similars	60
7.7	Extracció - canvis de perspectiva i zoom	61
7.8	<i>Matching</i> - canvis de perspectiva i zoom	61
7.9	RANSAC - canvis de perspectiva i zoom	62
7.10	Extracció - objectes	63
7.11	<i>Matching</i> - objectes	63
7.12	RANSAC - objectes	63
7.13	<i>Matching</i> - imatges diferents	64
8.1	Recursos humans (hores)	66
8.2	Recursos humans (costos)	67
8.3	Recursos de maquinari (costos)	67
8.4	Costos indirectes	67
8.5	Costos totals	68
9.1	Matriu de sostenibilitat	69

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	Aplicació de proves	24
3.3	App/Webapp - Selecció de la regió d'interès	25
3.4	App - Càmera	25
3.5	App - Galeria	26
4.1	Estructura del servidor	27
5.1	<i>Keypoints</i>	35
5.2	<i>Flat</i> , vora i cantonada	36
5.3	<i>Vores a diferent escala</i>	36
5.4	<i>SIFT - DoG</i>	37
5.5	<i>SIFT - Local extrema</i>	37
5.6	FAST, N=16	38
5.7	Descriptor SIFT	39
5.8	<i>Sampling pattern BRISK</i>	40
5.9	<i>Matching</i>	41
5.10	Homografia	42
5.11	RANSAC	42
7.1	Imatges graff, boat i ubc	57
7.2	Imatges UPC i jardins	58
7.3	Imatges campus	59
7.4	Imatges motos i cotxes	60
7.5	Imatges campus i jardins	61
7.6	Imatges campus i jardins (objectes)	62
7.7	Imatges campus i jardins (diferents)	64
7.8	Homografia - Resultats	65