Comportament de cerca d'objectes en una sala per a un robot de servei

Autor:
Gerard Canal Camprodon

Director:
Cecilio Angulo Bahón (ESAII)

Codirector:
Guillem Alenyà Ribas (IRI)

Data de defensa: 20 de Juny de 2013
Titulació: Grau en Enginyeria Informàtica
Especialitat: Computació
Centre: Facultat d'Informàtica de Barcelona (FIB)
Universitat: Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Juny de 2013

Agraïments

Voldria agrair al director, en Cecilio, i al codirector, en Guillem, tot el suport donat durant la realització del treball. També m'agradaria donar les gràcies als membres de l'equip REEM@IRI i a l'empresa PAL Robotics, amb especial menció per Ricardo Téllez, per proporcionar-me l'oportunitat de fer servir el REEM per al present treball.

Resum **(Català)**

En el present Treball de Final de Grau s'exposa un problema relacionat amb la robòtica de servei. Aquest consisteix en aconseguir un comportament per a un robot de tal manera que sigui capaç de cercar i trobar objectes en una sala de la qual en coneix el mapa. Aquest problema s'ha intentat resoldre des de diferents enfocaments i de manera incremental.

Primer es proposa una solució basada en localitzacions introduïdes per l'usuari, que el robot haurà de visitar per comprovar si hi ha objectes. En segon lloc es planteja una solució més autònoma, basada en l'assumpció de que els objectes que hi ha en entorns humans estan situats sobre superfícies planes. Finalment, es mostra una última solució basada en el coneixement del mobiliari de l'entorn. D'aquesta manera es pot indicar quin tipus de mobles pot contenir objectes i quin no, i fer que el robot utilitzi tot aquest coneixement per refinar la cerca.

Per tal d'obtenir solucions que minimitzin el temps de la cerca, s'ha combinat diferents algorismes. Primer s'han emprat per obtenir llocs del mapa des d'on es pot veure potencials localitzacions d'objectes i després s'ha fet ús de mètodes de cerca local per minimitzar la distància recorreguda al visitar aquests llocs. Així doncs s'ha obtingut diferents comportaments que intenten realitzar la cerca ràpidament i que poden usar-se indistintament segons les necessitats de l'usuari.

Resumen (Castellà)

En el presente Trabajo de Fin de Grado se expone un problema relacionado con la robótica de servicio. Este consiste en conseguir un comportamiento para un robot de forma que sea capaz de buscar y encontrar objetos en una sala de la qual conoce el mapa. Este problema se ha intentado resolver des de diferentes enfoques i de forma incremental.

Primero se propone una solución basada en localizaciones introducidas por el usuario, que el robot deberá visitar para comprovar si hay objetos. En segundo lugar se plantea una solución más autónoma, basada en la asunción de que los objetos que aparecen en entornos humanos estan coloados encima de superfícies planas. Finalmente se muestra una solución basada en el conocimiento de los mueles del entorno. De esta forma se puede indicar los tipos de muebles pueden contener objetos y los que no, y hacer que el robot use ese conocimiento para refinar la búsqueda.

Para obtener soluciones que minimizen el tiempo de búsqueda, se ha combinado diferentes algoritmos. Se han usado primero para obtener lugares del mapa des de los cuales se puede ver potenciales localizaciones de objectos. Posteriormente se han utilizado métodos de búsqueda local para minimizar la distáncia recorrida al visitar dichos lugares. Con esto se ha obtenido diferentes comportamientos que intentaran realizar la búsqueda de forma rápida, y que pueden ser usados indistintamente según las necesidades del usuario.

Abstract **(Anglès)**

In this Final Degree Project a problem related to service robotics is described. This problem consists in getting a behaviour for a robot such that it has the ability to search and find objects in a room where the map is known beforehand. The problem has been attempted to be solved in three approaches, implemented in an incremental way.

The first proposed solution is based on user defined locations, which the robot needs to visit and check if there are objects. Secondly, a more autonomous solution is suggested. This approach is based on the assumption that objects are placed on plane surfaces in human environments. Finally, an approach based on furniture recognition is shown. Using this skill, there is the possibility of indicating the robot what furniture of its environment can or can't contain objects, so the robot is able to use that information to improve the search.

In order to obtain solutions that minimize the search time, different algorithms have been combined. Some are first used to get locations in the map where potential object places can be seen. Then, some local search methods have been used to minimize the travelled distance when visiting all the locations. That way, different behaviours that try to search quickly have been implemented and either of them can be used depending on the user needs.

Índex

Índex de taules

Índex de figures

Índex d'algorismes i codis

1 Introducció

Des de sempre els humans pensem en els robots. Aquest fet que ha quedat palès en nombroses pel·lícules al llarg de la història i llibres com els d'Isaac Asimov. Sovint ens preguntem fins a quin punt podran arribar, o si podrem tenir-los algun dia a casa vivint entre nosaltres.

Tot i que avui en dia encara som una mica lluny de tenir algun robot com els que es veuen en el cinema, de mica en mica ens hi anem acostant. Però per tenir algun robot que faci feina útil per als humans, hi ha una sèrie de propietats que aquests robots han de tenir. Un exemple d'habilitat necessària és la capacitat de trobar objectes en un temps raonable, per tal de poder després manipular-los. Aquesta tasca, que pot semblar molt fàcil per als humans, no ho és tant per als robots artificials.

En el present treball de final de grau es proposa i compara diferents mètodes per tal d'intentar solucionar aquest problema i aconseguir que un robot trobi objectes dins d'una sala de manera satisfactòria.

1.1 Informació, context i descripció del treball

Aquest treball de final de grau està emmarcat dins del projecte REEM@IRI, una iniciativa de la mà de l'Institut de Robòtica i Informàtica Industrial (IRI) de la Universitat Politècnica de Catalunya, l'empresa PAL Robotics i l'associació AESS Estudiants.

L'objectiu d'aquesta iniciativa és formar un equip per tal de participar en una competició de robòtica anomenada Robocup[1], que es realitza cada any en una seu diferent. D'entre les diferents modalitats que ofereix la Robocup, la intenció de l'equip és participar en la modalitat Robocup@Home[2]. La finalitat d'aquesta modalitat de la competició és superar diferents proves basades en tasques casolanes amb un robot de servei, com poden ser netejar els objectes de damunt d'una taula i portar-los al seu lloc, o portar una beguda concreta a qui li hagi demanat. L'equip del REEM@IRI hi participaria amb el robot REEM proporcionat per l'empresa PAL Robotics.

Dins d'aquest projecte, el treball de final de grau consistirà en la implementació d'un comportament per al robot per tal de que sigui capaç de trobar

[1] www.robocup.org
[2] www.robocupathome.org

un objecte conegut que se li demani explícitament o no (un objecte qualsevol que pugui trobar), dins d'una habitació. Aquest comportament serà una estratègia o pla que el robot haurà de seguir per tal de trobar l'objecte en el menor temps possible. El robot disposa del mapa de la sala, però no té informació sobre els diferents mobles o obstacles que pot trobar (només té informació dels objectes que ha estat entrenat per a reconèixer i que ha de localitzar).

Aquest Treball de Final de Grau serà d'utilitat en diferents proves de les que es podran trobar a la competició: per portar una beguda a una persona, la beguda s'ha de trobar. També s'ha de trobar diferents objectes a l'hora de netejar una taula (traient els objectes que no hi pertoquin i portant-los al seu lloc).

Finalment, i allunyant-nos del món de la competició, en un futur (llunyà o no) on els robots de servei comencin a aparèixer en entorns casolans reals, sistemes com el que s'ha implementat en el present treball seran realment necessaris, doncs un robot de servei d'aquestes característiques, més sofisticats que els robots aspirador que es venen avui en dia, haurà de poder agafar objectes d'alguna manera i per tant, haurà d'haver estat capaç de trobar-los amb antelació (de la manera més eficient i ràpida possible).

1.2 Objectius

Els objectius marcats per a aquest Treball de Final de Grau són els següents:

- Desenvolupar un comportament d'exploració per a un robot de manera que trobi objectes coneguts dins d'una habitació. Els objectes poden estar en cadires, en taules, en estanteries, en sofàs...
- Que el robot es limiti a l'habitació que li ha tocat, sense sortir d'ella.
- Que els objectes siguin trobats en un temps raonable, sense perdre temps en llocs on no s'han observat obstacles, i sense fer recorreguts innecessaris per l'habitació, evitant repetir els llocs ja visitats mentre s'estan cercant els objectes.
- Que el robot sigui capaç de d'acomplir-ho de la forma més autònoma possible.

1.3 Competències tècniques

A continuació es mostra el conjunt de competències tècniques associades, amb el seu grau d'aprofundiment i una descripció de les parts on s'ha tractat.

- **Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts (CCO1.1, *En profunditat*)**: S'ha seleccionat i combinat diferents algorismes per tal de resoldre el problema que s'havia plantejat. Els subproblemes que han sorgit, com el problema del viatjant de comerç o el de la galeria d'art, són computacionalment difícils (NP-Complets). Per tal de fer-ho s'ha tingut en compte els diferents requisits del problema, com la rapidesa de la cerca. Aquesta competència s'ha treballat sobretot en el segon enfocament.

- **Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents, i analitzar, dissenyar i construir sistemes, serveis i aplicacions informàtiques que utilitzin aquestes tècniques en qualsevol àmbit d'aplicació (CCO2.1, *Bastant*)**: S'ha utilitzat algorismes sovint utilitzats en intel·ligència artificial, com són el Hill Climbing i el Simulated Annealing. A més, s'ha realitzat utilitzant un robot intel·ligent d'última generació.

- **Demostrar coneixement i desenvolupar tècniques d'aprenentatge computacional; dissenyar i implementar aplicacions i sistemes que les utilitzin, incloent les que es dediquen a l'extracció automàtica d'informació i coneixement a partir de grans volums de dades (CCO2.4, *Una mica*)**: S'ha emprat un classificador de mobles en el tercer enfocament.

- **Implementar codi crític seguint criteris de temps d'execució, eficiència i seguretat (CCO3.1, *Bastant*)**: El codi que s'ha implementat necessita temps d'execució petits, ja que s'executa en un robot que ha de realitzar accions en temps real. A més, el codi ha de ser segur en quant a que no pot comprometre la seguretat de les persones o mobiliari amb el robot. Per acabar, el codi ha de ser eficient per si es dona el cas en que s'executa en un robot que no disposi de grans recursos computacionals o hardware.

1.4 Recursos

Per realitzar aquest treball s'utilitzaran els següents components i recursos:

- Un robot de servei, REEM sèrie H, de l'empresa PAL Robotics[3]. La Figura 1.2 (esquerre) mostra el robot.
- El sistema de navegació del robot: permet seguir el mapa sense col·lidir amb els diferents obstacles.
- El sistema de reconeixement de veu: permet processar les ordres orals, com per exemple quin objecte ha de trobar.
- El sistema de reconeixement d'objectes ja entrenat: permet reconèixer els possibles objectes que se li poden demanar al robot. Aquest sistema reconeix objectes a 1.5 metres de distància, però és sorollós (pot donar falsos negatius). A la Figura 1.1 es mostren diferents objectes que el robot pot reconèixer.
- Simulador multi-robot en tres dimensions Gazebo[4]. La Figura 1.2 (dreta) en mostra una finestra.
- Framework per al desenvolupament de software en robots ROS (Robot Operating System[5])

Figura 1.1: Exemple d'objectes que el robot és capaç de reconèixer

[3] www.pal-robotics.com
[4] www.gazebosim.org
[5] www.ros.org

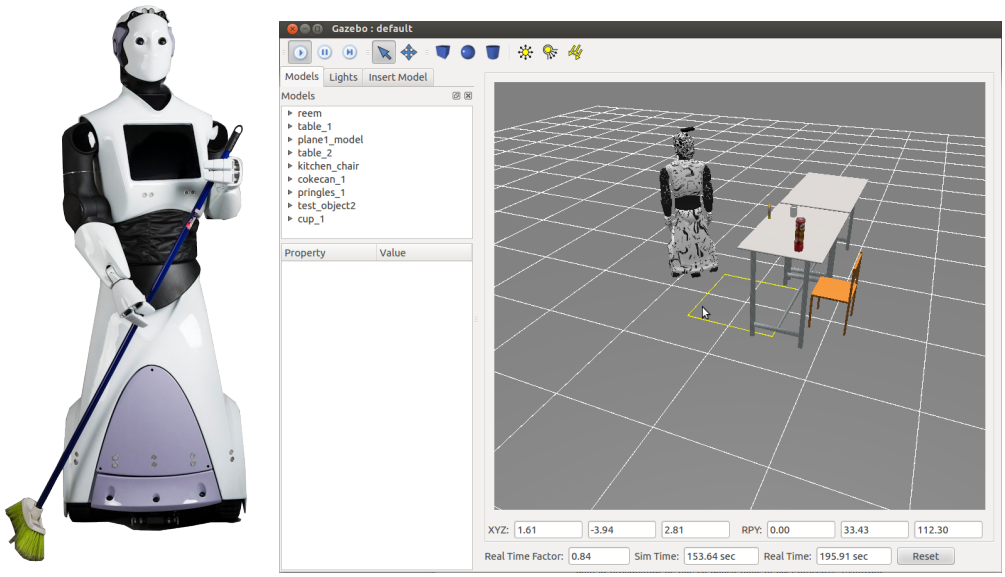


Figura 1.2: A l'esquerre, imatge d'un robot REEM sèrie H. A la dreta, una captura de la finestra del simulador Gazebo, durant algun dels tests realitzats

1.5 Abast

Aquest treball parteix de la base de que existeix un robot amb diferents característiques i funcions ja implementades, com el reconeixedor d'objectes o la navegació i no s'entrarà en aquests temes.

El treball s'encara a aconseguir els objectius ja exposats mitjançant diferents enfocaments, de més senzills a més elaborats, comprovant quin d'ells produeix uns millors resultats.

Així doncs, al termini del treball s'ha d'obtenir un algorisme que sigui capaç d'indicar el nom i la posició en el mapa d'un objecte reconegut pel robot en la sala on es troba. L'algorisme hauria d'aconseguir que aquest objecte sigui trobat pel robot en un temps raonable, intentant que sigui mínim, en una sala de la qual se'n coneix el mapa i on el robot hi està en qualssevol punt. Per tant, els plans que es crein han de tenir en compte diferents elements per tal que el temps sigui reduït, optimitzant els llocs on es cerca objectes (mirant de no repetir-los, per exemple) i tenint en compte els possibles problemes i limitacions dels sensors, com per exemple el seu rang de fiabilitat.

2 Metodologia de treball

2.1 Abordament del problema

El treball es desenvoluparà des de diferents enfocaments, que són els següents:

- Utilitzar un conjunt de punts definits amb anterioritat en el mapa, amb la probabilitat de que en aquell punt hi hagi objectes. Utilitzant aquestes dades, el robot s'hi acosta i comprova si hi ha objectes o no.
- Sense indicar-li on poden haver els objectes sobre el mapa, que el robot mateix mira en diferents localitzacions de la sala per saber on n'hi poden haver i llavors mira què hi ha. Aquest mètode es basa en diferents articles i tesis existents sobre el tema.
- Utilitzar coneixement sobre l'entorn. Fent que el robot sigui capaç de reconèixer el mobiliari que té al voltant, pot saber amb més precisió si pot haver objectes en algun d'aquests mobles o no.

2.2 Planificació temporal

2.2.1 Distribució de tasques

Les tasques es poden distribuir en els tres enfocaments amb els quals s'atacarà el problema.

Primer enfocament

Les diferents tasques en aquest enfocament són:

- **Crear un *service* que retorni una posició en el mapa on és possible que hi hagi objectes:** cada cop que sigui invocat se li passarà una cadena de caràcters indicant en quina habitació està. Retorna una cadena de caràcters indicant el lloc (per exemple, "aula"), que llavors es podrà transformar en una posició del mapa. Cada cop que és cridat retorna una posició diferent, ordenades per probabilitat (els llocs on és més probable que hi hagi objectes aniran abans). Recursos utilitzats: ROS (Robot Operating System).

- **Implementar un algorisme que utilitzi el *service*:** aquest ha de fer que el robot vagi fins a la posició indicada i provi de reconèixer objectes. Llavors indicarà quins objectes ha reconegut. Recursos utilitzats: ROS (Robot Operating System), SMACH (per implementar màquines d'estats), sistemes i algorismes de navegació i reconeixement d'objectes.
- **Provar el primer enfocament amb simulació:** utilitzant un entorn de simulació 3D comprovar que el sistema funciona, arreglant els possibles errors. Recursos utilitzats: ROS (Robot Operating System) i Gazebo Simulator.
- **Provar el primer enfocament sobre el robot:** instal·lar el programa en el robot i comprovar que aquest funciona, realitzant els arranjaments que siguin necessari i solucionant els errors que apareguin. Recursos utilitzats: ROS (Robot Operating System) i robot REEM H, amb els seus sistemes i algorismes de navegació i reconeixement d'objectes.

Segon enfocament

Les tasques per a aquest segon enfocament:

- **Fer recerca d'informació sobre feina feta al respecte:** trobar i analitzar informació sobre diferents tècniques utilitzades, per tal de veure quines ens poden ser útils i quines no. Recursos utilitzats: Repositoris d'informació acadèmica com per exemple Google Scholar.
- **Entendre i mirar com aplicar la informació trobada:** a partir dels diferents articles, tesis i altra informació, entendre-la i veure com es pot adaptar al problema. Recursos utilitzats: Documents (articles o *papers*) trobats.
- **Implementació dels algorismes:** Implementar els algorismes per realitzar la cerca d'objectes en diferents localitzacions, utilitzant informació trobada i adaptant-la si cal. Recursos utilitzats: ROS (Robot Operating System).
- **Crear una màquina d'estats que implementi aquest algorisme:** aprofitant o adaptant la màquina d'estats del primer enfocament, obtenir una nova solució que utilitzi aquest algorisme. Recursos utilitzats: ROS (Robot Operating System) i SMACH.

- **Test mitjançant simulació:** com en el primer enfocament, per solucionar errors. Recursos utilitzats: ROS (Robot Operating System) i Gazebo Simulator.
- **Test mitjançant el robot:** per veure com es comporta realment el programa en un entorn no simulat i arreglant els problemes que hi sorgeixin. Recursos utilitzats: ROS (Robot Operating System) i robot REEM H, amb els seus sistemes i algorismes de navegació i reconeixement d'objectes.

Tercer enfocament

En aquest últim enfocament, les tasques a realitzar són:

- **Implementar la màquina d'estats que utilitza la informació sobre el mobiliari per decidir on anar primer.** Recursos utilitzats: ROS (Robot Operating System) i SMACH.
- **Adaptar la màquina d'estats dels anteriors enfocaments:** canviar el mètode de selecció de llocs on anar, però mantenint la resta del programa (que realitza la navegació i comprova quins objectes hi apareixen). Recursos utilitzats: ROS (Robot Operating System) i sistemes i algorismes de navegació i reconeixement d'objectes.
- **Fer proves mitjançant el software de simulació:** com en els anteriors enfocaments. Recursos utilitzats: ROS (Robot Operating System) i Gazebo Simulator.
- **Fer proves mitjançant el robot:** de la mateixa manera, per veure els resultats finals i retocar els detalls que calgui. Recursos utilitzats: ROS (Robot Operating System) i robot REEM H, amb els seus sistemes i algorismes de navegació i reconeixement d'objectes.

Comparació dels tres enfocaments

Un cop els tes enfocaments hagin estat implementats i provats, es realitzarà una comparació per veure quins donen millors resultats en diferents escenaris, per tal de poder seleccionar el més adequat segons les diferents situacions on aquest problema ha de ser resolt per tal d'assolir alguna tasca amb el robot.

Redacció de la memòria

Redactar la memòria del projecte de manera incremental a mesura que el treball agafa forma i es va implementant, acabant de retocar alguns detalls un cop tot estigui finalitzat.

Preparació de la presentació

Preparar el material de suport a la presentació, així com fer diferents guions de què s'explicarà i en quin ordre. Realització de diferents assajos per tal de veure què es pot millorar i controlar-ne el temps de duració aproximat.

2.2.2 Planificació temporal de les tasques

En aquest apartat es mostra la previsió de temps assignat a cada tasca, així com possibles paral·lelitzacions entre diferents tasques. Aquesta planificació es mostra mitjançant el següent diagrama de Gantt:

	Nom	Durada	Inici	Fi	Predecessors	Recursos
1	**Primer enfocament**	**18.5d?**	**04/02/2013**	**28/02/2013**		
2	Crear un service que retorni una posició en el mapa on és possibl	2d	04/02/2013	05/02/2013		
3	Implementar una màquina d'estats que utilitzi el service	7d?	06/02/2013	14/02/2013	2	
4	Provar el primer enfocament amb simulació	4d?	15/02/2013	20/02/2013	3	
5	Provar el primer enfocament sobre el robot	5.5d?	21/02/2013	28/02/2013	4	
6	**Segon enfocament**	**51d?**	**18/02/2013**	**29/04/2013**		
7	Fer recerca d'informació sobre feina feta al respecte	5d	18/02/2013	22/02/2013		
8	Entendre i mirar com aplicar la informació trobada	8.5d?	25/02/2013	07/03/2013	7	
9	Implementació de l'algorisme	22d?	07/03/2013	08/04/2013	8	
10	Crear una màquina d'estats que utilitzi aquest algorisme	3.5d	08/04/2013	11/04/2013	9	
11	Test mitjançant simulació	5.5d?	12/04/2013	19/04/2013	10	
12	Test mitjançant el robot	6.5d?	19/04/2013	29/04/2013	11	
13	**Tercer enfocament**	**31.25d?**	**10/04/2013**	**23/05/2013**		
14	Anàlisis de l'estat d'altres components implementats dins de l'equ	0.25d?	10/04/2013	10/04/2013		
15	Implementació de la màquina d'estats que utilitza la informació se	14d?	12/04/2013	02/05/2013	14FS+2d	
16	Adaptar la màquina d'estats dels anteriors enfocaments	2d?	02/05/2013	06/05/2013	15	
17	Fer proves mitjançant el software de simulació	6d?	06/05/2013	14/05/2013	16	
18	Fer proves mitjançant el robot	8.5d?	10/05/2013	23/05/2013	16FS+4.5d	
19	Comparació dels tres enfocaments	3d?	22/05/2013	27/05/2013	18FS-1d	
20	Redacció de la memòria	80d?	11/02/2013	31/05/2013		
21	Preparació de la presentació	17d?	20/05/2013	11/06/2013		

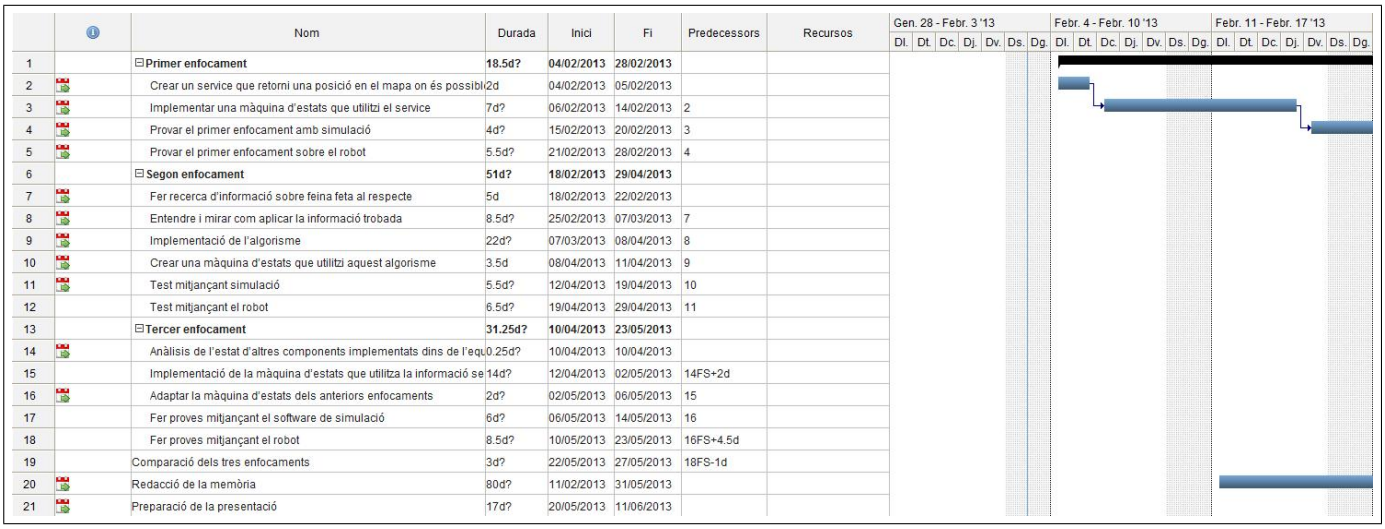


Figura 2.1: Diagrama de Gantt de Gener a mitjans de Febrer

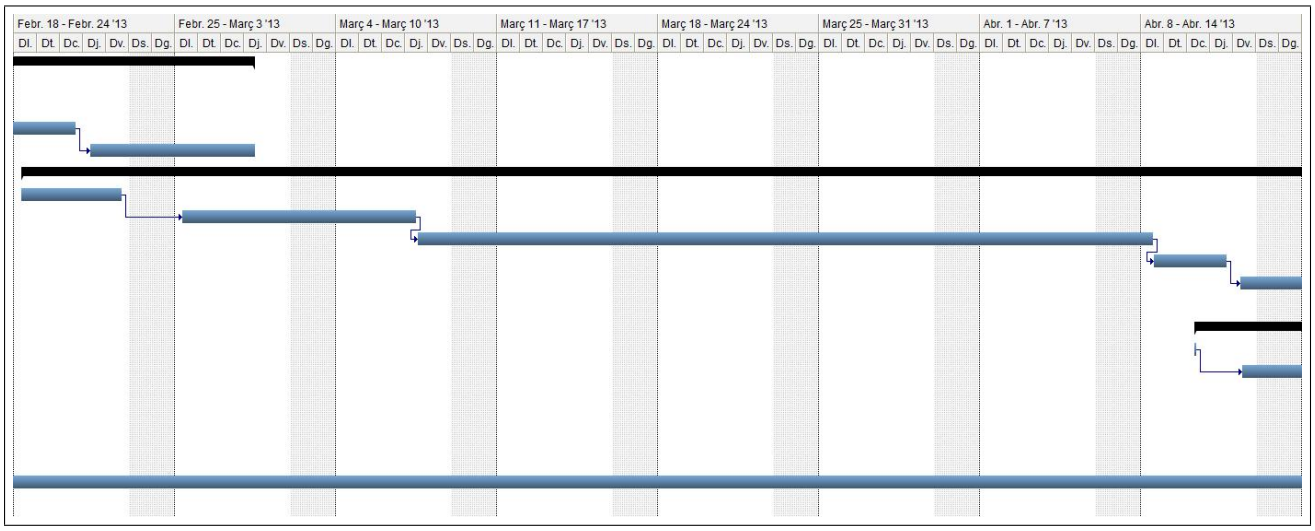


Figura 2.2: Diagrama de Gantt de mitjans de Febrer a mitjans d'Abril

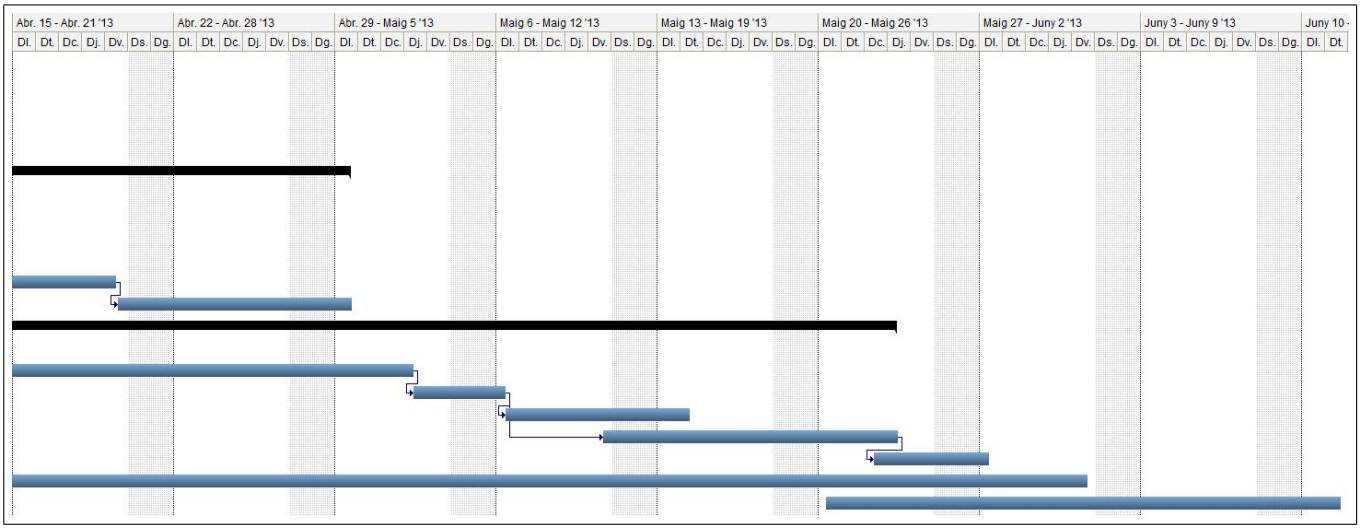


Figura 2.3: Diagrama de Gantt de mitjans d'Abril fins a la finalització del projecte

2.2.3 Planificació final

La planificació inicial[1], descrita en els anteriors apartats, s'ha seguit en gran majoria. Hi ha hagut diferents desviacions, detallades a continuació, que han suposat un retràs en algunes de les tasques. Malgrat tot, s'ha pogut complir amb la planificació inicial degut a que algunes feines s'han realitzat en menys temps del que era previst..

Les diferents desviacions han estat les següents:

- El simulador Gazebo no ha funcionat en les condicions esperades. Els sistemes de navegació per defecte en simulació no han operat de forma satisfactòria, cosa que fa que només es puguin provar petites parts, però no l'algorisme sencer. Això implica que les proves s'han hagut de fer majoritàriament amb el robot, el qual té una disponibilitat limitada. Això ha fet que el temps de validació sigui major i per tant ha suposat un retard.
- El temps de prova amb el robot, a part d'incrementar-se a causa de lo que s'ha indicat en el punt anterior, ha sigut també major del previst. Sovint hi ha hagut problemes diferents sistemes del robot. Cal notar que els sistemes del robot estan en continu desenvolupament i, per tant, és esperable que hi hagi problemes a l'hora d'utilitzar-los, però no hi ha manera de quantificar el temps que s'ha d'invertir en realitzar aquestes proves.
- S'ha hagut de dedicar més temps del previst per cercar una forma de realitzar el segon enfocament, doncs al principi no se sabia si es podria utilitzar el detector de plans i mentrestant es cercaven diferents alternatives. Durant la cerca de possibles algorismes es van trobar noves opcions que, tot i ser interessants, no van conduir a opcions satisfactòries però van retardar una mica el temps de consulta.
- En tercer enfocament es va haver d'invertir una gran quantitat de temps per tal de poder utilitzar el classificador. En concret, es va haver d'invertir grans esforços en la compilació, doncs s'utilitza una versió de la llibreria PCL no estable. En el moment de realitzar aquest treball, la versió 1.7 de PCL usada pel classificador havia estat modificada. Això va suposar fer vàries proves amb diferents revisions més antigues fins que se'n va trobar una funcional.

[1] Veure punt 2.2.2 Planificació temporal de les tasques

Tot i que hi havia plans alternatius que consistien en retallar la complexitat del tercer enfocament, no ha estat necessari aplicar-los.

2.3 Eines software utilitzades

Per tal de treballar s'utilitzarà *subversion* com a software de control de versions, per tal de compartir el codi entre tot l'equip i de tenir una còpia de seguretat en més d'un equip.

Per compartir arxius grans s'utilitzarà *Dropbox* i com a mecanisme de comunicació es farà servir el correu electrònic i *Skype*.

A continuació s'introduirà el funcionament d'algunes eines de programació emprades, així com la nomenclatura que utilitzen.

2.3.1 ROS: Robot Operating System

Aquest treball s'ha implementat utilitzant les llibreries i eines del *framework* ROS per al desenvolupament d'aplicacions per a robots. ROS es desenvolupa actualment en la fundació de robòtica *OSRF (Open Source Robotics Foundation)*[2].

ROS proporciona eines semblants a les d'un sistema operatiu, permetent executar les aplicacions de forma distribuïda. Permet al desenvolupador abstreure's del hardware, permetent una fàcil interconnexió entre els diferents processos, que formen una arquitectura en forma de graf i cada procés n'és un node. Per tal de que els diferents nodes puguin comunicar-se, cal que hi hagi un servei anomenat *Master* executant-se. Aquest servei proporciona als diferents nodes mitjans per registrar-se. Aquest servei també proporciona el servei de paràmetres, que és un servei al qual els nodes poden accedir per consultar diferents paràmetres en temps d'execució. Se sol utilitzar per introduir paràmetres de configuració, i se solen emmagatzemar el fitxers amb format *YAML*, que aquest servei carrega per posar els paràmetres del fitxer a disposició de qui els necessiti.

Les comunicacions entre els diferents nodes es realitza mitjançant pas de missatges, que es defineixen mitjançant tipus bàsics. Aquests missatges es defineixen en un fitxer individual amb extensió *.msg*, que llavors és compilat

[2] osrfoundation.org/

per donar lloc a classes en diferents llenguatges, facilitant-ne així el seu ús. El mecanisme més bàsic de pas de missatges es realitza per mitja dels anomenats *Topics*. Els nodes poden publicar missatges en un *Topic* o s'hi poden subscriure per llegir-ne els missatges publicats. Hom pot veure els *Topics* com si fossin una canonada (*pipe* en anglès) de Linux, amb la diferència que diferents processos s'hi poden connectar per llegir i escriure missatges. El servei *Master* és qui proporciona als nodes la informació dels que publiquen i llegeixen informació dels diferents *Topics*. Solen usar-se per sensors, ja que proporcionen dades contínuament.

Un altre mecanisme de comunicació són els serveis. Aquests serveis consisteixen en una mena de servidor al qual se li pot realitzar peticions, que són crides remotes a les operacions que defineixen (similars a crides RPC, *Remote Procedure Call*), tot i que aquestes sovint s'executen en la mateixa màquina que el client, encara que no ha de ser necessàriament així. El format de la petició i de la resposta es defineix en fitxers amb extensió *.srv*, i en ells s'hi especifica els tipus de missatges i paràmetres de la petició i els de la resposta, separats per tres guions. Els nodes que ofereixen un servei es connecten amb el *Master* i n'indiquen el nom i el tipus. Els nodes que vulguin utilitzar-lo, anomenats clients, han de realitzar una petició mitjançant les interfícies que proporcionen les llibreries de ROS, i esperar-ne la resposta. Els serveis són útils per tal de realitzar càlculs que es resolguin ràpid però que només es computaran quan sigui necessari, i per enviar missatges de gran mida, evitant així haver de publicar-los constantment, fet que generaria un col·lapse de la xarxa. Un cop invocats no se'n pot aturar l'execució fins que finalitzen.

Finalment, existeixen les accions. Aquestes són similars als serveis, però estan especialment dissenyats per a elements que triguen a donar una resposta. Permeten cancel·lar la petició i, a diferència dels serveis, retornen missatges indicant paràmetres que mostren l'estat de l'execució. Les accions es defineixen en fitxers amb extensió *.action* que indiquen els tipus de missatge de petició, els de resposta i els d'informació de l'estat.

Els diferents nodes poden executar-se de diferents formes, bé sigui en una sola màquina o en màquines distribuïdes. En aquest segon cas, un ordinador ha de proporcionar el servei *Master*, al qual els altres hi accediran a través de la xarxa. Aquests elements es poden aprofitar per exemple en robots que tenen poca potència de càlcul, permetent que els còmputs complexes es facin en un ordinador més preparat, rebent el robot els resultats utilitzant-los.

ROS proporciona llibreries per a llenguatges com Python i C++, permetent que nodes escrits en diferents llenguatges es comuniquin de forma senzilla i

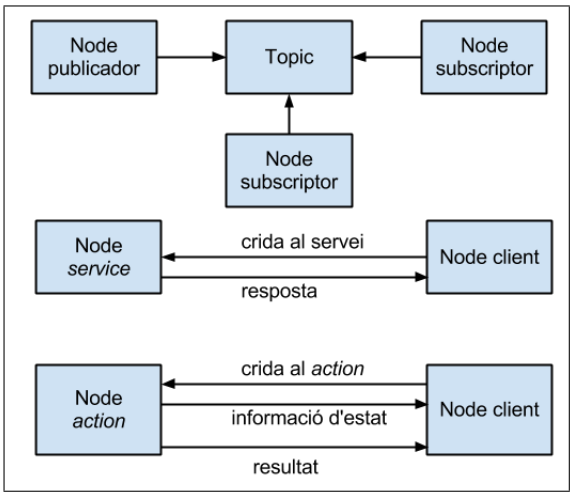


Figura 2.4: Esquema dels diferents tipus de nodes de ROS i les seves interfícies de comunicació

satisfactòria. També facilita la integració i estructuració del codi en diferents paquets, els quals faciliten la compilació del codi mantenint les dependències, així com la configuració de l'entorn. Aquests abstreuen al programador de la feina de configurar les rutes als fitxers importats. ROS també proveeix eines per desplaçar-se pels paquets que estan repartits en el sistema de fitxers de manera fàcil i ràpida.

2.3.2 SMACH

SMACH[3] és una potent llibreria per a escriure màquines d'estats jeràrquiques i complexes en Python. Permet una gran integració en l'ecosistema de ROS, proporcionant diferents estats preparats per a realitzar crides a serveis o accions. La seva facilitat d'ús permet la reutilització total dels estats programats individualment, així com la incorporació de sub-màquines d'estats en una màquina d'estats més gran. El seu funcionament es basa en la creació de classes que hereten d'una classe base que en proporciona els mètodes bàsics. Aquestes classes defineixen els estats de la màquina i quines transicions han de realitzar, depenent del resultat de la seva execució. També permet especificar quines dades es transmetran d'un estat a un altre, de manera que la informació que un estat genera la pot fer servir el següent a executar-se.

[3]El nom prove de *State MACHine*.

A part d'això, SMACH ofereix diferents eines per tal de visualitzar gràficament[4] les màquines d'estats i les seves transicions, facilitant-ne així la depuració.

2.4 Mètode de validació

La validació es realitzara en diferents fases: en primer lloc es provarà el codi en un simulador per tal de veure com seria l'execució en un entorn virtual i controlat, on el robot funciona de manera ideal. Un cop els resultats en la simulació siguin correctes o mínimament acceptables, es podran provar en el robot real i en un entorn també real (potser controlat d'alguna manera o limitat) per comprovar que els algorismes funcionen i en cas contrari, veure què falla, per quin motiu falla i com es pot solucionar. L'últim pas seria prova-lo en un entorn com el que ens trobaríem en alguna prova de la competició.

Aquest és el procediment que s'ha seguit durant tot el treball, de manera que s'ha anat provant les diferents parts que el formen independentment i de manera incremental, a mesura que el projecte ha anat prenent forma, ajuntant aquestes parts per composar parts més grans fins que s'ha provat amb la totalitat del treball acabat.

2.5 Lleis i regulacions

El present Treball de Final de Grau es pot trobar afectat per diferents lleis i/o regulacions, com poden ser les relacionades amb el software i les relacionades amb la robòtica de servei.

2.5.1 Software

Com és ja conegut, el software està regulat per diferents llicències. Aquestes llicències són les que determinen en quines condicions un usuari pot utilitzar el software, i quins drets té aquest usuari sobre el software que utilitza.

En el nostre cas, s'utilitza software de diferents fonts i que tenen diferents llicències:

[4] A A Diagrames de les màquines d'estats s'hi mostren diferents exemples.

- Codi de l'IRI: codi que conté diferents llibreries i utilitats que s'utilitzen en diferents parts del projecte REEM@IRI. Aquest codi està sota la llicència *GNU Lesser General Public License*
- Codi de PAL Robotics: codi per fer funcionar el robot i diferents sistemes com poden ser la navegació, el reconeixement de veu i el reconeixement d'objectes, entre d'altres. Aquest software és propietari. Tot i això, hi ha una part del codi en un repositori públic[5] de paquets per a ROS, on alguns paquets tenen llicència BSD modificada, i altres estan sota llicència *Creative Commons BY-NC-ND 3.0*.
- Codi de ROS: La majoria de llibreries de ROS estan sota llicència *BSD*, mentre que altres utilitats com el simulador Gazebo tenen una llicència *Apache License 2.0*.
- Codi del REEM@IRI: és l'apartat on s'inclou el codi desenvolupat en aquest treball. Es tracta de codi privat per a ser usat només per membres de l'equip REEM@IRI (format per l'IRI, AESS i PAL Robotics) per tal de participar a la Robocup@Home. Es tracta, doncs, del codi realitzat per superar les diferents proves i els diferents comportaments associats a aquestes que s'ha programat pel robot.

A continuació s'especificaran algunes característiques de cada llicència.

GNU Lesser General Public License

Llicència de software creada per la *Free Software Foundation*. Permet a les empreses i desenvolupadors utilitzar el codi que està sota d'aquesta llicència lliurement en el seu codi, sense que sigui necessari lliurar el seu propi software. Per tant, i al contrari de la GPL (per *General Public Licence*), permet que altres desenvolupadors de codi lliure o propietari utilitzin el codi.

Les modificacions del software original han de mantenir-ne la mateixa llicència i condicions, però les obres derivades poden utilitzar altres llicències.

Aquesta llicència no contempla activitats que no siguin còpia, distribució o modificació.

[5] github.com/pal-robotics/

BSD License

Llicència de software lliure i permissiva. Té menys restriccions que les llicències GPL. Per exemple, es permet l'ús del codi font en software no lliure i la distribució d'aquest en forma binària o de codi font.

La llicència obliga que el codi font o els binaris que es redistribueixin han de mantenir la nota amb el copyright i les condicions de la llicència.

L'usuari té el dret d'utilitzar lliurement el codi sense necessitat de mantenir-ne la llicència.

Hi ha diferents tipus de llicències BSD, que varien en nombre de clàusules i restriccions. Per exemple, la llicència BSD modificada que tenen alguns paquets del repositori públic de PAL difereix de la llicència BSD estàndard en la tercera clàusula, que no apareix en la versió modificada. Aquesta tercera clàusula fa referència a temes de publicitat, obligant a indicar que s'utilitza software desenvolupat a la *Universitat de Califòrnia, Berkeley* en tot el material publicitari on es mencioni algun software que estigui sota aquesta llicència.

Apache License 2.0

És una llicència de software lliure creada per la *Apache Software Foundation* (ASF). Aquesta llicència dóna a l'usuari la llibertat d'utilitzar el software amb qualsevol propòsit, distribuir-lo, modificar-lo i distribuir-ne versions modificades. A més, no requereix la redistribució del codi font quan es torna a distribuir una versió modificada del codi, ni que aquesta utilitzi la mateixa llicència ni tan sols que sigui software lliure, però si que exigeix que s'informi als receptors que s'ha utilitzat codi que ha estat sota la llicència d'Apache.

Llicències Creative Commons

Creative Commons és una organització sense ànim de lucre que té per objectiu facilitar la compartició legal de treballs creatius.

Aquesta organització ha creat diverses llicències de drets de còpia que poden ser utilitzades lliurament sense cap cost. Aquestes lli cències permeten que els creadors indiquin quins drets es reserven, i comuniquen als usuaris quins beneficis o ús poden fer de l'obra.

Per exemple, la llicència *Creative Commons BY-NC-ND 3.0* (utilitzada en alguns paquets del repositori públic de PAL Robotics) indica que si s'utilitza aquell paquet s'ha d'indicar l'autor i atribuir-li la feina de la manera que aquest hagi especificat, que no es permet l'ús de l'obra amb fins comercials i que no es permet derivar l'obra (és a dir, alterar-la, transformar-la o construir elements nous a partir d'ella).

Grans empreses d'Internet, com per exemple *Wikipedia*, utilitzen llicències *Creative Commons*.

Llicències de codi tancat

En forma part el codi propietari. No es permet la còpia o distribució no permesa ni la modificació del codi, i en els fins concrets pels quals es pot usar el codi.

En aquests casos, l'empresa o autor del codi és qui reté tots els drets sobre aquest.

2.5.2 Robòtica de servei

Existeixen diferents comitès que realitzen diferents tasques de normalització, en àmbits com el dels sistemes d'automatització, i la seva integració en el disseny, subministrament, fabricació, entrega, suport, manteniment i eliminació dels productes i serveis associats.

El comitè tècnic *ISO TC184/SC2* (robots i aparells robòtics) és el principal comitè d'ISO que desenvolupa estàndards sobre robots. Està format per diferents grups que treballen en elements com els següents:

- Vocabulari i característiques dels robots.
- Seguretat dels robots industrials.
- Seguretat dels robots de cura personal
- Equipament mèdic elèctric i sistemes que utilitzen tecnologia robòtica.
- Robots de servei

Aquest comitè ha desenvolupat diferents estàndards d'ISO referents temes com mètodes de testeig, interfícies mecàniques, vocabulari i requeriments de seguretat per a robots, entre d'altres.

En el moment d'escriure aquest treball s'estava desenvolupant l'estàndard *ISO/FDIS 13482*, on s'indiquen requeriments de seguretat per a robots no industrials.

2.6 Sostenibilitat i compromís social

El context d'aquest Treball de Final de Grau pot implicar un gran impacte social en els propers anys. Com ja s'ha anat indicant al llarg del document, els robot de servei d'aquest estil poden ser de gran utilitat en feines de la llar. Robots més sofisticats que els actuals robots aspirador poden dur a terme moltes més tasques que la de netejar el terra: poden ordenar una habitació, realitzar tasques com servir una beguda o el menjar que se li demani, o fins i tot detectar emergències, com un incendi o inundació i avisar a les autoritats corresponents. Totes aquestes oportunitats es podrien veure encara més augmentades incorporant elements com la connexió entre els robots i els diferents electrodomèstics, connectant el robot amb la domòtica, de manera que el propi robot sigui capaç de comunicar-se amb els electrodomèstics per fer coses com utilitzar la rentadora o baixar les persianes si fa molt sol.

És fàcil veure que aquests robots serien de gran utilitat en entorns de persones d'avançada edat, on la realització de diferents tasques quotidianes els suposa a vegades una gran dificultat. Persones que sovint estan soles, podrien trobar en el robot alguna mena de companyia, com la que podria suposar una mascota. També seria útil en entorns amb nens, on el robot podria vetllar per la seva seguretat i estar al seu servei com per exemple acostant-los menjar o begudes que puguin estar en llocs fora del seu abast. Tot plegat, pot suposar una millora substancial en la qualitat de vida de diferents persones.

El fet que els components d'aquest Treball de Final de Grau s'incorporin en robots de servei pot suposar un impacte econòmic, que podria implicar un increment en la compra de robots de la llar per part de particulars, entitats i empreses.

Pel que fa als aspectes ambientals, robots d'aquest estil es podrien utilitzar en diverses situacions de risc. Per exemple, podrien usar-se en tasques de recuperació de l'entorn després d'un desastre, com podria ser una central nuclear, on els robots no es veurien afectats per la radiació i podrien treballar sense haver exposar vides humanes. Un altre context on els robots podrien

ajudar a millorar l'entorn podria ser en plantes de tractament de residus, on sovint els diferents residus es separen per tal de reciclar. Un robot capaç de detectar i agafar objectes podria realitzar aquesta tasca, de forma igual o inclús més eficient que un ésser humà.

Finalment, un altre fet que podria ajudar a la sostenibilitat del medi ambient és la reutilització dels robots un cop finalitzada la seva vida útil. Es podrien arreglar, o reconstruir a partir d'elements d'altres robots, de manera que no es generessin més deixalles tecnològiques del compte.

3 Estat de l'art

El problema que s'intenta afrontar en aquest Treball de Final de Grau (a partir d'ara TFG) ha estat afrontat ja diferents vegades al llarg del temps. En alguns casos fins i tot basant-se en elements vists en la Robocup@Home [15], altres utilitzant Internet per aprendre els diferents objectes [7, 10]. S'esmenta a continuació aquelles solucions més properes a la proposta d'aquest TFG.

Hi ha aproximacions per solucionar el problema mitjançant cerca visual [11], utilitzant "atenció visual" per tal de simplificar el problema i optimitzar el procés de cerca d'imatges, atès que, com és obvi, processar totes les imatges que ens arriben és una tasca intractable. Altres enfocaments similars també fan ús del zoom i reconeixement d'objectes mitjançant l'algorisme d'extracció de característiques SIFT [13].

Algunes solucions utilitzen "ubicacions de detecció" des d'on el robot observa el seu entorn per veure si hi ha objectes. Dins d'aquest tipus d'enfocament, similar a l'utilitzat com a primer apropament del TFG, hi ha diferents mètodes per tal de generar aquestes ubicacions i l'ordre amb el que són recorregudes, bé sigui utilitzant heurístiques o diferents algorismes d'extracció de característiques. Una bona part d'aquests mètodes estan basats en assumpcions poc realistes, que serveixen de punt de partida per fer altres apropaments al problema considerant inconvenients reals com poden ser les limitacions en la visió dels robots, per exemple utilitzant una àrea de reconeixement [5], dins de la qual els objectes són reconeguts. A partir d'aquesta idea es poden basar diferents algorismes que generen millors ubicacions i camins respecte els algorismes existents.

També s'han proposat a la literatura enfocaments bastant diferents, representant el problema de cerca com un problema d'optimització on es maximitza la probabilitat de detectar un objectiu mentre es minimitza l'energia, distància i temps necessaris per aconseguir realitzar la tasca [9].

A part d'això, moltes solucions es basen en el problema de la galeria d'art, un problema difícil de resoldre que té per objectiu obtenir les posicions on guardes de seguretat o càmeres de vigilància s'haurien de col·locar dins del recinte per tal de tenir tota la galeria controlada. Un exemples n'és [1].

Enfocaments similars a la tercera aproximació plantejada en aquest TFG per solucionar el problema també són presents en el món de la investigació, intentant resoldre la cerca mitjançant diferents recursos, però tots relacionats en utilitzar diferents tipus d'informació semàntica. A [2] s'apropa el pro-

blema utilitzant coneixement semàntic de l'espai. A [6] utilitzen elements similars juntament amb SLAM (*Simultaneous Localization and Mapping*), i creen el mapa de l'habitació a mesura que es cerquen els objectes. Altres enfocaments més potents són capaços d'aconseguir que el robot cerqui de forma autònoma el web per tal d'adquirir coneixement del seu entorn físic, en lloc de fer-li una etiquetatge manual. Utilitzant Internet, aquesta solució és capaç d'inferir coses com que el cafè és probable que estigui a la cuina. Diversos enfocaments que utilitzen cerca al web són mostrats a [7, 10].

En [3] es parla de la possibilitat d'utilitzar detecció global d'objectes per tal d'obtenir candidats a ser reconeguts, que després seran descartats o reconeguts realment fent que el robot s'hi apropi i realitzi el reconeixement des d'una distància més fiable. El problema l'aproximen utilitzant Processos de Decisió de Markov (MDP, per *Markov Decision Processes*).

Una proposta interessant es mostra a [14]. Tot i que l'objectiu de l'article no és el mateix que té aquest TFG, es podria aprofitar el mètode que proposen. L'article es centra en la col·locació d'objectes en l'entorn, sense que el robot hagi vist mai aquest lloc, i ho fan tenint en compte les preferències humanes. Donat un conjunt d'escenes 3D amb objectes, proven de col·locar figures humanes per aprendre com els objectes es relacionen amb les posicions dels humans. Aprenent d'aquest conjunt de dades, l'algorisme és capaç de seleccionar les posicions humanes més significatives i les fan servir per determinar llocs on es podria deixar objectes. En el cas del present treball, aquesta informació es podria utilitzar per saber on pot haver-hi objectes.

Finalment, es troben solucions híbrides a partir de diferents metodologies com en [15], on s'utilitzen sistemes basats en SLAM per obtenir un sistema d'atenció espacial. També s'utilitzen diferents tècniques de reconeixement d'objectes com poden ser un detector que utilitza les distribucions de color per aconseguir-ho, o màquines de vector suport (SVM, per *Support Vector Machines*).

Amb tot això, es fàcil veure que el problema que es vol resoldre en el present treball de final de grau és activament estudiat en diferents àmbits de recerca, amb una gran quantitat d'enfocaments que pel que sembla, donen resultats prou bons. En el TFG es centrarà l'atenció en els tres enfocaments ja explicats.

Així doncs, el treball es basarà en aquests documents per tal d'apropar-se als objectius, però adaptant les diferents solucions al problema concret que es vol resoldre.

4 Desenvolupament del treball

Aquest apartat mostrarà com s'ha desenvolupat el Treball de Fi de Grau, explicant en detall les tasques realitzades per a cada enfocament.

En tots els enfocaments plantejats s'ha fet servir les llibreries de ROS per al llenguatge de programació Python. A més, també s'ha utilitzat aquest llenguatge per a implementar els diferents algorismes. Aquests s'han estructurat en forma de màquina d'estats, fent ús de la llibreria de SMACH que inclou la distribució de ROS.

En cada cas, el que s'espera d'entrada i de sortida és el mateix: es creen les instàncies de la màquina d'estats indicant-li en quin sistema de coordenades es vol el resultat de la cerca d'objectes, i si s'està buscant un objecte en concret o qualsevol. Llavors se li indica en quina habitació ha de buscar objectes, i el nom de l'objecte a buscar en cas que es vulgui usar aquesta opció. Quan s'ha trobat un objecte, la màquina d'estats retorna informació d'aquest objecte trobat, com per exemple el nom i la posició de l'objecte en el sistema de coordenades que s'ha indicat al crear la instància de la màquina d'estats.

El desenvolupament s'ha realitzat de forma incremental, de manera que cada enfocament utilitza elements de l'anterior i en millora algun aspecte. La programació de les diferents solucions s'ha realitzat aprofitant el màxim del codi, que s'ha creat en forma d'estats individuals que es poden incloure en les diferents màquines d'estats implementades, facilitant-ne així la reutilització.

4.1 Primer enfocament

El primer intent de resoldre el problema de fer que un robot trobi objectes en una sala s'ha realitzat de manera que és un usuari humà qui indica quines són les localitzacions que poden tenir objectes en cada habitació. Per exemple, l'usuari podria introduir les posicions del mapa per a la taula, els fogons o les cadires de la cuina, de forma que el robot hi vagi i un cop s'hagi situat en el lloc indicat intenti reconèixer els objectes que pugui observar mitjançant els seus sensors.

Per tal d'aconseguir una major efectivitat i rapidesa, l'usuari també ha d'indicar, juntament amb les posicions dels diferents llocs de cada habitació, la

probabilitat de que en cada lloc s'hi trobin objectes. Utilitzant aquesta informació es pot fer que el robot vagi primer en els llocs on és més probable que hi hagi objectes.

Aquests paràmetres són introduïts en un fitxer amb format *YAML*, que posteriorment un sistema del robot llegirà i carregarà en el servei de paràmetres de ROS, per fer que siguin accessibles des dels diferents programes que s'executen i en concret per a aquest comportament de cerca d'objectes. En el Codi 4.1 es mostra un exemple d'aquest fitxer de configuració.

```
# Locations and its probabilities of having objects on top
# Syntax: Room_name/location_in_room: [probability, posX, posY,
    angle]

locations_prob:
  living:
      table:      [0.90, 0.888,  4.041, -1.047]
      chair:      [0.65, 1.878,  1.067,  1.000]
      sofa:       [0.40, 1.878,  3.007, -2.407]
      tv:         [0.05, 2.080,  2.046,  1.265]
      shelf:      [0.70, 3.902,  3.036,  3.146]

  bedroom:
      desk:       [0.85, 8.842, 11.315, -3.037]
      chair:      [0.70, 6.589, 10.910, -2.247]
      shelf:      [0.80, 5.985, 11.800,  1.589]
      nightstand: [0.75, 7.826,  9.458,  0.058]
      table:      [0.90, 3.423, 10.051,  3.014]
      bed:        [0.56, 4.129, 12.000, -1.512]

  kitchen:
      countertop: [0.95, 8.961,  4.666,  3.137]
      table:      [0.80, 9.646,  2.324,  1.482]
      dishwasher: [0.20, 9.895,  4.978, -1.632]
```

Codi 4.1: Exemple de paràmetres del fitxer de configuració del primer enfocament

L'algorisme s'implementa en una màquina d'estats[1]. Aquesta comença demanant una localització a un servei que s'ha creat de manera que cada cop que és cridat retorna una localització de l'habitació que se li indiqui, començant per la que té més probabilitat i seguint en ordre decreixent. Llavors s'envia una ordre al sistema de navegació per tal d'anar fins al lloc que ens ha retornat el servei i es prova de reconèixer objectes. En l'Algorisme 2 es

[1] A l'Annex A.1 Primer enfocament s'hi pot veure la seva representació gràfica.

mostra el comportament d'aquesta màquina d'estats en forma algorísmica.

Algorisme 2: Algorisme del primer enfocament

Entrada: Nom d'una habitació.
Entrada: (Opcional) Nom d'un objecte concret a cercar.
Sortida: Informació sobre un objecte trobat.

```
inici
    mentre cap objecte trobat fer
        lloc_cerca = get_next_probable_location()
        anar_a(lloc_cerca)
        objectes = reconèixer_objectes_del_davant()[2]
        si no objectes.és_buit() llavors
            objecte_sel = seleccionar_objecte(objectes)
            retornar objecte_sel
        fisi
    fimentre
fi
```

En els propers apartats es detallaran les parts i les característiques de la màquina d'estats.

4.1.1 El servei `get_next_probable_location`

El servei `get_next_probable_location` s'ha creat com a un node de ROS que defineix un `rosservice`. Per a definir els serveis s'ha de definir com serà la petició, és a dir, els tipus de missatges que tenen els paràmetres de l'operació remota que s'ofereix, i quina forma tindrà la resposta a la crida. En el Codi 4.3 es mostra el format dels missatges del servei. Primer hi ha els missatges que formen la petició, en aquest cas el nom de l'habitació, i separat per tres guions (− − −) hi ha el format del missatge de resposta, que en aquest servei és el nom de la localització dins l'habitació (per exemple 'taula'), i la posició d'aquesta en coordenades del mapa.

[2] Per simplicitat, es pot considerar que si s'ha indicat el nom d'un objecte concret a cercar, el reconeixement només retorna objectes als quals els correspongui el nom indicat.

```
string room
---
string location
geometry_msgs/Pose loc_position
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Codi 4.3: Missatge `NextProbableLocation` utilitzat pel servei del primer enfocament.

Quan el servei rep una petició, aquest retorna la següent localització en ordre de probabilitat. La primera vegada que és cridat, retorna la posició del lloc on és més probable que hi hagi objectes (basant-se en la configuració introduïda per l'usuari). A cada nova petició el servei respon amb la següent localització fins que se n'ha retornat la menys probable. Un cop ha arribat en aquest punt, quan el servei rep una nova petició torna a començar des del principi, retornant la localització amb major probabilitat associada.

Per tal d'implementar el servei s'ha creat una classe de Python anomenada `CircularOrderedList`, que és una estructura de dades en forma de llista circular, és a dir, un cop s'ha recorregut tota sencera el següent element és el primer de la llista. Tot i que s'ha implementat de forma que sigui una estructura iterable en Python, està pensada per ser una llista estàtica on l'operació principal és demanar el següent element, o els següents elements amb el mateix pes. Al crear-ne una instància se li proporciona una llista de parelles d'elements on el segon element n'és el pes amb el qual s'ha d'ordenar la llista. Llavors aquesta llista és ordenada i emmagatzemada de manera que sigui accessible de la forma ja descrita.

Utilitzant aquesta classe el servei s'implementa de forma senzilla. En la inicialització del servei s'obtenen els paràmetres de les localitzacions de cada habitació descrits per l'usuari (en un arxiu com el mostrat en el Codi 4.1). Amb aquests paràmetres es crea una `CircularOrderedList` per a cada habitació amb tots els llocs on pot hi haver objectes i la seva probabilitat, que s'emmagatzema en un diccionari utilitzant el nom de l'habitació com a clau.

Un cop inicialitzat, el servei només ha d'esperar rebre peticions amb el nom

d'una habitació. Quan això succeeix, s'accedeix a la llista de localitzacions d'aquella habitació, se n'obté la següent de la llista (que serà la pròxima amb més probabilitat d'aparició d'objectes) i es retorna la posició del lloc en coordenades del mapa. En cas que hi hagi diferents localitzacions amb la mateixa probabilitat, totes aquestes s'obtenen de la llista i el servei retorna aquella localització que està més a prop del robot. Per tal que la distància sigui el més real possible, es demana al planificador el pla de navegació entre la posició del robot i la posició de cerca d'objectes i es calcula la distància del camí retornat. Si per algun motiu el planificador no està disponible o no pot calcular el camí, s'utilitza la distància euclidiana entre els dos punts. Utilitzant la distància també s'aconsegueix proporcionar l'opció de fer que el robot vagi sempre al lloc de cerca més proper, inserint el mateix pes o probabilitat a totes les localitzacions que utilitza el servei.

Com a exemple de funcionament, es pot suposar un cas on s'utilitzin els paràmetres del Codi 4.1 i es vulgui cercar objecte en l'habitació anomenada *kitchen*. La `CircularOrderedList` tindria els elements *countertop, table* i *dishwasher*, en aquest ordre. La primera petició que el servei rebi retornarà les dades del *countertop*, i la tercera les del *dishwasher*. Quan es realitzi una quarta petició al servei, aquest tornarà a respondre amb la informació del *countertop*, i seguiria així en les invocacions successives. Si el *countertop* o la *table* fossin equiprobables, en la primera invocació es retornaria la localització més propera al robot d'entre les dues en el moment de fer la crida al servei. El mateix succeirà en la quarta petició, però en aquest cas potser es retornara la *table* encara que en la primera invocació s'hagués obtingut el *countertop*, tot dependrà de la posició del robot. Seguint aquest exemple, suposant les dues localitzacions equiprobables i el retorn de *table* en la quarta petició, la cinquena petició resultaria amb les dades del *countertop*.

El fet d'implementar l'obtenció de la següent posició a visitar en un servei ens proporciona diferents avantatges:

- S'inicia al principi de tot, creant les diferents llistes per a cada habitació i ordenant-les, de manera que quan s'ha d'accedir a les dades és molt més ràpid.
- És possible demanar localitzacions d'una altra habitació i després de la primera, per llavors tornar a la primera i continuar en la posició de la llista on s'estava.

4.1.2 Anar al lloc i reconèixer objectes

Un cop s'ha obtingut la posició on pot haver-hi objectes, que correspon a la variable `lloc_cerca` de l'Algorisme 2, el robot hi ha d'anar i cridar al sistema de reconeixement d'objectes.

D'això se n'encarrega la resta de l'algorisme. S'envia un objectiu al sistema de navegació per tal de que vagi fins al lloc i posteriorment s'utilitza el sistema de reconeixement d'objectes per veure què hi ha, si és hi ha alguna cosa. Per tal de fer el robot més "humà", s'utilitza el sistema de parla per tal de notificar les accions i els resultats a l'usuari. Per exemple, el robot diu coses com "Vaig a la taula a veure si hi ha objectes". A més a més, s'ha fet un estat que donat un conjunt de possibles frases en diu una aleatòriament (mirant de no repetir l'última frase dita), evitant així que el robot mostri un comportament repetitiu i mecànic. Aquest estat s'ha utilitzat en gairebé totes les accions de parla del programa. Seguint en la mateixa línia, s'ha utilitzat màquines d'estats concurrents per tal d'implementar aquesta petita màquina d'anar al lloc i reconèixer objectes, de tal manera que el robot parla mentre està anant al lloc, en comptes de que es faci primer la parla mentre el robot està quiet i posteriorment realitzant l'acció de moure's cap al lloc indicat. Així s'evita mostrar la forma seqüencial del programa, s'emula un comportament més proper al de les persones, i el més important, s'estalvia temps d'execució ja que la parla triga un temps en el qual no s'està fent res.

Posteriorment es tracten els resultats de la detecció d'objectes, comprovant si hi havia l'objecte buscat o si s'ha trobat algun objecte quan se'n busca qualsevol, i s'indica mitjançant una la parla que s'ha trobat un objecte i se'n diu el nom i el lloc on s'ha trobat, i la màquina d'estats acaba retornant la informació de l'objecte a qui l'ha invocat. En cas contrari, es diu que no s'ha trobat cap objecte (o no s'ha trobat l'objecte concret demanat) en aquell lloc, i en aquest cas es continua la cerca de la mateixa manera però a la següent localització que indica el servei `get_next_probable_location`.

En cas que per algun motiu el sistema de navegació no pugui portar el robot fins al l'objectiu que se l'hi ha indicat, el robot diu que no pot arribar al lloc i es continua amb la següent localització, de la mateixa manera que ho fa quan no ha trobat cap objecte (o l'objecte que es buscava).

4.1.3 Emmagatzematge de resultats

Per tal d'aprofitar els casos en què la màquina es invocada diverses vegades per tal de trobar més d'un objecte, s'emmagatzemen els resultats del reconeixement d'objectes en una variable.

Aquesta variable és consultada a l'inici de l'execució, de manera que si ja tenim informació d'objectes, el que fem és anar al lloc on hi havia aquells objectes, sense demanar una nova localització al servei, i es torna a fer un reconeixement d'objectes per certificar que els objectes encara hi són.

Amb aquesta petita comprovació s'aconsegueix que el robot vagi directament a un lloc on ja sap que hi ha objectes, en comptes d'anar cap a la següent localització on és menys probable que n'hi hagi, deixant-se enrere objectes que ja s'ha reconegut amb anterioritat.

4.1.4 Valoració de l'enfocament

Durant el desenvolupament d'aquest mètode es va valorar l'opció de fer que el robot visités alguna localització menys probable a la que tocaria però que estigués més a prop. Finalment es va desestimar degut al fet que si l'usuari indica unes probabilitats per als diferents llocs, és per què en visiti primer uns i després els altres. A més, si hi ha un lloc on és probable que hi hagi objectes, no val la pena parar-se en un lloc amb menys probabilitat, ja que el cost de reconèixer objectes per no trobar-hi res és més elevat que anar fins a la localització on sí que s'espera que n'hi hagi.

Les proves realitzades amb aquest enfocament han estat satisfactòries, on el robot feia el que s'ha descrit en els apartats previs.

En l'execució aquests proves s'ha observat diferents punts febles:

- Inserir totes les dades de les localitzacions és un procés costós i que s'ha de refer en cas que alguna localització canvii de lloc.
- Si alguna localització no s'ha inserit, el robot no cercarà en aquell indret i per tant no es trobaran els objectes que hi pugui haver.
- Si hi ha objectes en algun lloc on es creu que és poc probable que n'hi hagi, el robot trigarà més estona en trobar-los. Per tant, el pes que s'assigna a cada localització ha d'estar ben orientat.

Per contra a tot això, aquest enfocament ha resultat en una gran rapidesa en trobar els objectes quan les dades són correctes i precises.

4.2 Segon enfocament

Després de tenir una primera solució, senzilla però eficaç, l'objectiu és millorar-la en un segon enfocament del problema. En aquest cas, el que es pretén és fer que el robot mateix trobi els llocs on pot haver objectes, i llavors que miri si n'hi ha, d'una forma més autònoma i amb les mínimes indicacions per part de l'usuari.

Per tal de realitzar aquesta solució s'ha pres com a inspiració la proposta de [5]. En aquest article s'explica diferents mètodes per tal de resoldre un problema similar al que es presenta en aquest treball. Defineixen *ubicacions de detecció*, des d'on s'intenta reconèixer objectes, tenint en compte les limitacions sensorials del robot. Aquestes consideracions les aconsegueixen definint àrees de reconeixement, que són les zones on els sensors del robot donen resultats fiables. D'aquesta manera aconsegueixen algorismes més realistes en contraposició a diferents algorismes presentats on s'assumeix condicions de visibilitat infinites i perfectes. Llavors defineixen les *ubicacions de detecció*, i proposen tres algorismes per a fer-ho: el primer basat en triangulació dels polígons del mapa de l'habitació, el segon utilitzant una versió de l'algorisme Visib-PRM i finalment utilitzant l'algorisme ROMA (*Randomized points On the MAp*). Finalment computen una ruta que visita les diferents *ubicacions de detecció* per cercar objectes, fins que dins d'aquesta s'hi troba algun objecte.

En aquest enfocament s'ha realitzat un comportament de cerca similar a l'exposat en [5], però fent algunes suposicions que faciliten la cerca d'objectes. Per exemple, s'ha pres l'assumpció de que els objectes es troben sobre superfícies planes i horitzontals (com per exemple taules, cadires o llits, entre d'altres). Donada aquesta assumpció, la cerca d'objectes es pot considerar que es realitza en dues fases. En la primera s'utilitza l'algorisme Visib-PRM[3] per generar les *ubicacions de detecció* del present cas, que són els punts del mapa des d'on es cerquen els llocs on hi pot haver objectes. Aquests seran els llocs on s'hi hagi detectat superfícies horitzontals. Cal notar que aquesta és la informació que venia proporcionada per l'usuari en l'anterior enfocament. Un cop s'ha obtingut aquestes zones candidates a tenir objectes, comença la

[3] Veure 4.2.1 Generació de localitzacions de cerca de superfícies.

segona fase que consisteix en comprovar si en aquests punts candidats hi ha objectes o no.

Així doncs, es denotaran punts de detecció de superfícies (d'ara en endavant PDS) per a la primera fase, i punts de cerca d'objectes (a partir d'ara PCO), que seran les superfícies detectades en els PDS. Un cop obtinguts aquests punts en les diferents fases, s'han de visitar fent que el robot hi navegui. Aquestes visites als diferents punts es podria fer en qualsevol ordre, com per exemple l'ordre que tenen dins la llista que els emmagatzema, però això podria suposar donar moltes voltes innecessàries en el pitjor cas, perdent molt temps realitzant la navegació d'un punt a un altre. Per evitar aquest fet, abans de navegar fins a cada punt s'utilitza mecanismes de cerca local per tal de crear un ordre de visita dels punts que disminueixi la distància total recorreguda al visitar totes les localitzacions.

L'Algorisme 4 mostra l'algorisme dissenyat per a aquest enfocament. Aquest algorisme s'ha implementat en una màquina d'estats[4], de la mateixa manera que es va fer amb el primer enfocament.

[4]A l'Annex A.2 Segon enfocament s'hi pot veure la seva representació gràfica.

Algorisme 4: Algorisme del segon enfocament

Entrada: Nom d'una habitació.
Entrada: (Opcional) Nom d'un objecte concret a cercar.
Sortida: Informació sobre un objecte trobat.

```
inici
    mentre cap objecte trobat fer
        PDS = generar_punts_de_cerca_de_superfícies()
        rutaPDS = generar_ruta_per(PDS)
        per cada puntS de rutaPDS fer
            anar_a(puntS)
            PDO = detectar_superfícies_planes()
            rutaPDO = generar_ruta_per(PDO)
            per cada puntO de rutaPDO fer
                anar_a(puntO)
                objectes = reconèixer_objectes_del_davant()[5]
                si no objectes.és_buit() llavors
                    objecte_sel = seleccionar_objecte(objectes)
                    retornar objecte_sel
                fisi
            fiper
        fiper
    fimentre
fi
```

En els propers punts es detallaran les característiques de les diferents parts de l'algorisme.

4.2.1 Generació de localitzacions de cerca de superfícies

Per tal de cercar superfícies planes dins l'habitació, es podria fer que el robot navegués aleatòriament per aquesta fent deteccions de plans cada cert temps. Però una estratègia d'aquesta mena no té perquè trobar la majoria de les superfícies de l'habitació, ni fer-ho en un temps raonable. Per tal de seleccionar aquests llocs de detecció de superfícies, s'obté el mapa del sistema de navegació i s'hi cerquen punts tals que, tenint en compte les característiques dels sensors del robot, cobreixin gairebé la totalitat de l'espai

[5]Per simplicitat, es pot considerar que si s'ha indicat el nom d'un objecte concret a cercar, el reconeixement només retorna objectes als quals els correspongui el nom indicat.

de l'habitació. Trobar la configuració de punts òptima resulta en el conegut problema de la galeria d'art. Aquest és un problema NP-Complet on es busca el mínim nombre de guardes que s'ha de col·locar en una galeria d'art de manera que tot l'espai de la galeria sigui visible des d'algun dels guardes.

Donada la complexitat de trobar la solució òptima i el fet que, degut als possibles errors de localització i posicionament, aquesta pot no ser explotada amb la precisió necessària [1], s'ha optat per utilitzar un algorisme que ofereixi una solució prou bona pel present problema, i que sigui més senzilla de computar. Finalment s'ha utilitzat l'algorisme Visib-PRM, definit a [12]. Tot i que l'algorisme està dissenyat per construir *mapes de carreteres (RoadMaps)* per tal de fer plans de moviment, en aquest cas és útil per obtenir punts de guardes en el mapa, des d'on es puguin cercar localitzacions que puguin contenir objectes. Un *RoadMap* és un graf els nodes dels quals són punts en el mapa lliures de col·lisió, i on dos nodes són adjacents si existeix un camí lliure de col·lisions que els uneix. Aquest camí es pot calcular de diferents maneres, segons les capacitats o necessitats del robot. Els *RoadMaps* tradicionals són utilitzats per construir un camí entre dos punts del mapa. Per fer-ho, aquests s'uneixen al graf i s'hi cerca un camí entre l'inici i el destí, de manera que, si el camí existeix, el robot el pugui recórrer i arribar a l'objectiu.

L'algorisme genera punts aleatoris que estiguin lliures de col·lisió dins del mapa i els intenta unir. Per fer això utilitza la noció de visibilitat, incorporant dins del *RoadMap* únicament els punts que, o són visibles des de dos punts no connectats, o bé no són visible des de cap altre node. Als nodes generats de la primera forma els anomenen de connexió, i als altres guardes. D'aquesta manera s'aconsegueix *mapes de carreteres* amb un petit nombre de nodes en comparació a altres mètodes. En la Figura 4.1 es mostren exemples d'execució de l'algorisme.

En la implementació que s'ha realitzat, s'ha utilitzat el camí en línia recta com a noció de visibilitat per decidir si dos nodes poden ser adjacents o no. En altres paraules, es considera que un node és visible des d'un altre node si existeix una línia recta que els uneix i que no interseca amb cap obstacle o cel·la ocupada del mapa. Utilitzant aquest càlcul de visibilitat l'algorisme intenta trobar una cobertura de l'espai lliure del mapa, que és un conjunt de nodes guarda tal que la unió dels punts visibles des de cadascun dels elements equival a tot l'espai no ocupat del mapa.

Aquest algorisme no requereix el còmput explícit dels punts visibles des de cada node, cosa que fa que necessiti menys recursos computacionals. L'únic

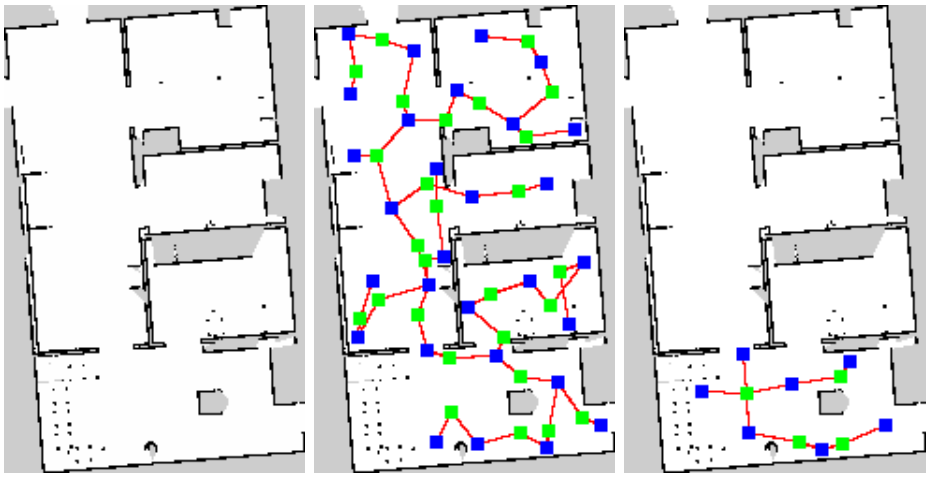


Figura 4.1: Diferents exemples de l'algorisme Visib-PRM. A l'esquerra es mostra el mapa tal i com el proporciona el sistema de navegació. Al centre, un exemple d'una execució del Visib-PRM en tot el mapa. A la dreta, una execució en l'habitació inferior (una sala d'estar). En blau es mostren els nodes guarda, mentre que en verd apareixen els nodes de connexió.

que necessita és poder saber si donat dos nodes, aquests es veuen entre ells. Donat que el mapa és una matriu d'enters on cada cel·la pot tenir diferents nombres indicant si la cel·la està lliure, ocupada o si és desconeguda, s'ha utilitzat una variant l'algorisme de línies de Bresenham[6] per tal de realitzar el càlcul de visibilitat. Tot i que aquest algorisme està més pensat per a ser usat en aplicacions gràfiques, també ens serveix per calcular la visibilitat entre dos punts del mapa. Això ho fa calculant l'error de precisió que hi ha a l'utilitzar una graella i la pendent de la recta. L'algorisme va incrementant els valors de x, incrementant el valor de y en el valor de la pendent de la recta i l'error de precisió. Llavors tenint en compte la diferència entre la nova y i la vella decideix si s'ha de moure cap a la posició $(x+1, y)$, $(x+1, y+1)$ o $(x, y+1)$. Addicionalment, es pren consideració de les característiques dels sensors a utilitzar. Per aquest motiu, s'ha afegit uns paràmetres a l'algorisme per indicar el rang mínim i el rang màxim dels sensors, de manera que previ al càlcul de visibilitat amb Bresenham es comprova si el nou punt està dins del rang de visibilitat del sensor.

En l'algorisme s'utilitza dues llistes de nodes, una per a cada tipus de node.

[6] S'ha adaptat la implementació de lifc.univ-fcomte.fr/~dedu/projects/bresenham/index.html, que està basada en [4].

La llista que conté els nodes guarda està formada per diferents llistes de nodes, on cada llista interna conté un conjunt de nodes guarda que han estat connectats entre ells, és a dir, estan en la mateixa component connexa del *RoadMap*. La llista de nodes de connexió conté tuples amb el node de connexió i els nodes que connecta. A l'inici, es crea un node aleatori q de tal manera que la regió de $N_x \times N_y$ cel·les centrada en el punt q estigui completament lliure d'obstacles o de nodes de l'algorisme, essent N_x i N_y el nombre de cel·les horitzontals i verticals que ocupa el robot. D'aquesta forma es crea un node vàlid on el robot hi cap. Posteriorment es recorren totes les llistes de components connexes de nodes guarda, mirant si el nou node és visible des d'algun element d'alguna llista. Si el nou node no és visible des de cap guarda existent, es converteix en una nova guarda (que estarà en una nova component del graf no connectada a cap altre node). En cas que sigui visible des dos o més nodes guarda de diferents components connexes, el node q es converteix en un node de connexió, i les components que uneix passen a fusionar-se en una sola component connexa del graf. Si q només és visible des d'un node guarda significa que únicament està dins de la zona de cobertura d'aquella guarda i no proporciona nova informació, sent així descartat. Posteriorment es crea un nou node aleatori i es repeteix aquestes comprovacions.

Per tal de que l'algorisme acabi, s'utilitza una condició relacionada amb el volum de l'espai lliure cobert. Per a fer això, s'empra un paràmetre que compta el nombre de nodes candidats descartats abans de trobar un nou node guarda. Un cop aquest nombre de fallades arriba a ser igual a un paràmetre M definit per l'algorisme, aquest acaba. Aquest paràmetre M aproxima el volum d'espai que cobreix el *RoadMap*, sent aquest volum probablement major que $1 - 1/M$. Una altra propietat de l'algorisme és que, un cop trobat un conjunt de guardes que cobreixen tot l'espai lliure, la mateixa execució amb un valor major per a M no faria que el resultat tingués més nodes, sinó que la seva mida seria la mateixa. Per tant, la grandària dels *Roadmaps* generats, tot i no ser òptima, depèn en gran mesura de la complexitat del conjunt de zones lliures de col·lisió. Un dels grans avantatges que tenen els *RoadMaps* que s'obtenen amb aquest mètode és que són petits, amb una poca quantitat de nodes en comparació amb altres mètodes.

Finalment, i degut al fet que el mapa del que disposa el robot sol ser de tota una planta de l'edifici i no només d'una sola habitació, s'ha de limitar la creació de punts aleatoris només a l'habitació on es vol cercar els objectes. Per fer-ho, s'ha creat un nou fitxer de configuració que emmagatzema les cantonades de cada habitació, que seran quatre punts que formin un rectan-

gle que contingui aproximadament tota l'habitació. A més, també incorpora un conjunt de "cantonades internes", que són aquelles cantonades que hi ha dins de l'habitació (dins del rectangle format per les primeres cantonades), com per exemple en casos on l'habitació no té forma del tot rectangular i alguna altra estància n'envaeix una part. Aquest fenomen és típic d'habitacions que tenen forma de "L", com la que es representa en Figura 4.2. En aquests casos, s'utilitzarien les quatre cantonades de l'habitació considerant que aquesta tingués forma rectangular, i després s'indicaria les coordenades de la cantonada interna, juntament amb un identificador que indica quina de les quatre cantonades externes està a dins de la regió que delimita, és a dir, de la que queda fora de l'habitació. Llavors, cada cantonada interna i la seva cantonada externa associada demarquen una regió rectangular, de la qual en són els vèrtexs oposats, que es considerarà com a ocupada en el mapa. D'aquesta manera no es generaran punts en aquella zona, evitant així tenir nodes fora de l'habitació on es cerca objectes. Tot i que sembli que aquest mètode només és vàlid per a habitacions rectangulars, també es pot fer servir per a les que tenen diferent geometria, ajustant-hi un rectangle que contingui el màxim espai de l'habitació. A més, la utilització de cantonades internes permet definir regions on no es vol que l'algorisme hi col·loqui guardes. Un exemple n'és una zona amb taules i cadires al costat d'una paret, on el mapa que genera el robot només en mostra les potes, sent possible que l'algorisme hi generi punts entremig que seran inaccessibles pel robot. La Figura 4.2 mostra un exemple dels tipus de cantonades. Suposant que es vol cercar objectes en l'habitació gran (en blanc), els punts vermells en són les cantonades externes, mentre que el punt blau mostra una cantonada interna, que tindria l'identificador "amunt-dreta". La zona acolorida serà la zona que es marcarà com a ocupada en el mapa. En el Codi 4.5 es mostra un exemple del fitxer de configuració creat per especificar la posició de les cantonades.

```
# Values to identify inner corners (corner ids)
# Up-Left    = UL
# Up-Right   = UR
# Down-Left  = DL
# Down-Right = DR
corner_data:
 # Syntax: [[c1x, c1y], [c2x, c2y], [c3x, c3y], [c4x, c4y],
 #          [[x, y, corner_id], ...]]
 # The first 4 points [x, y] represent the corners of the room.
 # The fifth parameter is a list of lists where each list
     contain the x and y cordinate of an inner coorner and a
     corner id.
 kitchen: [[10.883, -5.320], [8.637, -5.252], [7.943, -1.831],
           [11.107, -1.888], []]
 living: [[0.83, 9.42], [1.016, 12.15], [8.05, 9.21],
          [8.1, 11.96], [[7.01, 10.71, UR], [5.51, 11.2, DR]]]
```

Codi 4.5: Exemple de paràmetres del fitxer de cantonades de les habitacions

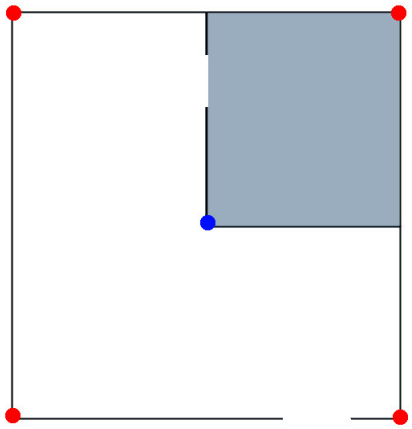


Figura 4.2: Exemple de cantonades internes (en blau) i externes (en vermell)

4.2.2 Cerca local per construir la millor ruta

Una vegada s'ha obtingut un conjunt de posicions del mapa a visitar, com les guardes obtingudes amb el Visib-PRM[7], aquestes s'han de recórrer. Per tal de minimitzar el temps de navegació, s'ha de crear una ruta que minimitzi

[7]Per exemple ls nodes de color blau de la Figura 4.1.

la distància recorreguda, cosa que també minimitzarà el temps que es triga en fer un recorregut de tots els punts.

Trobar la ruta òptima per un conjunt de punts a visitar és un problema NP-Complet, conegut amb el nom de *Problema del viatjant de comerç* (també anomenat TSP, de l'anglès *Travelling Salesman Problem*). Existeixen nombroses formes d'aproximar una solució al problema, i en el present treball s'ha utilitzat dos mecanismes de cerca local per fer-ho. Així s'aconsegueix obtenir una solució en un temps raonable que, tot i no ser la l'òptima, disminueix notablement la distància total que el robot hauria de recórrer en cas de no intentar optimitzar el camí.

Els algorismes de cerca local que s'ha emprat realitzen una cerca en un espai de solucions. Això vol dir que, d'entre totes les solucions possibles, se'n busca la millor sense importar el camí que ha conduït a trobar-la. El que importa és la solució en sí i, en aquests casos, trobar una solució vàlida no suposa una dificultat: el cost principal recau sobre l'optimització d'una solució arbitrària. Pel problema que es presenta en aquest apartat, qualsevol ordre dels punts a visitar en constitueix una ruta vàlida.

Aquests algorismes són útils en problemes on el nombre de solucions és tan gran que no és possible obtenir la millor solució, ja que realitzar una exploració exhaustiva, comprovant totes les combinacions possibles, no és viable computacionalment. A més, només tenen en compte un sol estat[8], anomenat estat actual de l'algorisme, i es solen moure només a estats o nodes veïns (també coneguts com a successors) de l'estat actual. D'aquí bé el nom de "cerca local", ja que els moviments que es realitzen són locals a l'estat que s'està utilitzant. Per tal de bellugar-se d'una solució a una altra usen operadors de transformació, que realitzen canvis en la ruta d'entrada per obtenir-ne de noves. Llavors, decideixen si una nova solució és la millor trobada depenent del valor d'aquesta en la funció de qualitat, també anomenada funció objectiu. Aquesta funció calcula quant bona és una solució, i és la funció que es vol optimitzar. En el cas del TSP, s'ha utilitzat la distància en metres d'una ruta com a funció de qualitat. En el present cas la cerca intenta trobar aquella ruta que sigui més curta i que, per tant, sigui un mínim en la funció objectiu.

La funció de qualitat calcula la distància entre cada node de la ruta i el següent, utilitzant l'expressió $\sum_{i=0}^{N-1} distància(r_i, r_{i+1})$ on N és el nombre de

[8] S'entén com a estat una solució al problema, pel fet que la cerca es realitza en un espai de solucions.

localitzacions a visitar i r_k és el k-èssim node del recorregut. Amb l'objectiu d'obtenir un valor de les distàncies que sigui el més real possible, tal i com es va fer en el servei del primer enfocament, s'utilitza la distància del pla de navegació des d'un punt a un altre. Si el planificador no retorna un pla correcte o no està actiu, llavors s'utilitza la distància euclidiana. Fent servir aquest càlcul per a la distància s'obtenen rutes que tenen en compte els diferents obstacles que hi pot haver en el camí, mentre que d'una altra forma la ruta òptima pot ser pitjor degut als obstacles que s'hagin d'esquivar. A canvi de tenir aquests valors més reals per a la funció de qualitat, s'ha d'assumir el cost de realitzar les crides al planificador amb el tràfic de xarxa que això implica, i també calcular la distància total del pla. Per minimitzar aquest cost, a l'inici es computen les distàncies entre cada parell de nodes i s'emmagatzemen en una matriu triangular, minimitzant l'ús de memòria aprofitant que la funció de distància es considera simètrica.

Per tal de resoldre aquest problema s'ha utilitzat dos algorismes de cerca local: el Hill Climbing i el Simulated Annealing. En ambdós casos s'utilitza l'ordre de la llista de guardes com a solució inicial, que s'intentarà millorar mitjançant operadors d'intercanvi de punts en la ruta.

Hill Climbing

L'algorisme de Hill Climbing (en català escalada de turons) parteix d'un estat inicial i, mitjançant els operadors de transformació, es va movent cap a nous estats solució que tinguin més qualitat. Només és necessari emmagatzemar la solució actual, i els nodes accessibles des d'aquesta s'anomenen nodes o estats successors. Aquest mètode es pot veure com una escalada en la funció objectiu, on la finalitat és fer-ne el cim més alt (un màxim global). Quan es vol minimitzar el valor, com és el cas, es pot veure com una maximització de la funció de qualitat canviada de signe, mantenint així vàlid el símil amb l'escalada.

Existeixen diferents variants de l'algorisme. Per exemple, el Hill Climbing amb ascens simple es mou cap al primer node que suposa una millora de qualitat, sense explorar la resta de nodes accessibles. En canvi, l'anomenat Hill Climbing amb ascens per màxima pendent (en anglès *steepest ascent Hill Climbing*) genera tots els successors del node actual i n'escull el millor.

Aquest algorisme també es conegut com a cerca local voraç pel fet que es mou cap al millor veí, sense pensar en què pot venir després. Tot i que sol

funcionar bé, el Hill Climbing es sol quedar encallat en una solució no òptima per diferents motius:

- **Màxims o mínims locals**: Són estats que no tenen cap veí que sigui millor, però no són l'òptim global. EL Hill Climbing no pot continuar endavant si es dóna aquesta situació, de manera que l'òptim local serà el resultat. La Figura 4.3 mostra aquest fenomen.
- **Crestes**: Formen una sèrie d'òptims locals des dels quals no es pot accedir al següent millor màxim, ja que tots els veïns tenen menys qualitat.
- **Altiplans**: Zones planes on tots els nodes tenen la mateixa qualitat. Poden esdevenir en un òptim local, o en una espatlla després de la qual continuï cap a l'òptim. En aquestes zones el Hill Climbing pot perdre's i no trobar el camí cap a l'òptim. També es mostren a la Figura 4.3.

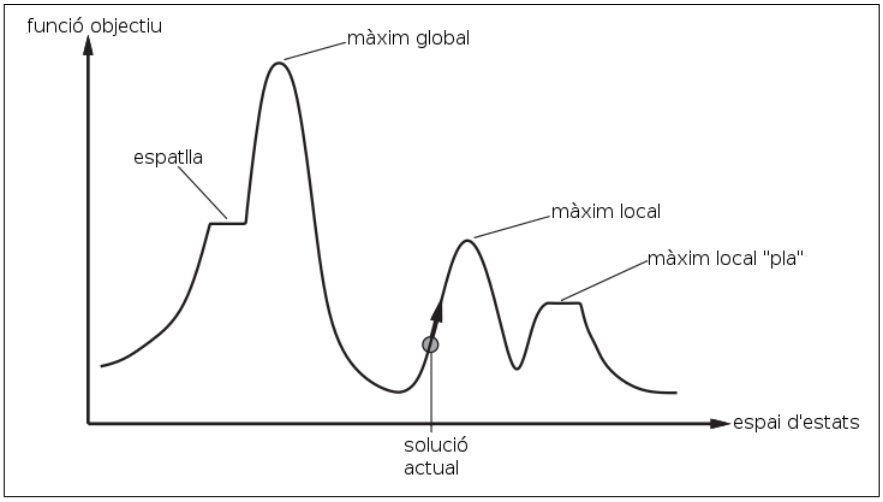


Figura 4.3: Representació d'una funció de qualitat on l'objectiu és trobarne el màxim global.També s'hi mostren diferents característiques topogràfiques. Extret de [8], pàgina 121.

Aquests fets mostren que arribar a l'òptim global depèn de quin sigui l'estat inicial. Per tal d'evitar aquests entrebancs que suposen trobar una pitjor solució, hi ha varies opcions. Els altiplans, es poden tractar permetent un moviment cap a un node que tingui la mateixa qualitat. Però això podria, per exemple, resultar en un bucle infinit. Els òptims locals també es poden tenir en compte, implementant el que s'anomena Hill Climbing amb reini-

ci aleatori. Aquesta variant consisteix en realitzar la cerca unes quantes vegades, començant cada cop amb un estat inicial generat de forma aleatòria. D'aquesta manera es realitza la cerca des de diferents punts de l'espai d'estats, incrementant la probabilitat de començar des d'un node que ens condueixi fins l'òptim global.

En la implementació del Hill Climbing que s'ha realitzat per al present treball s'ha utilitzat la versió d'ascens per màxima pendent amb reinicis aleatoris. L'únic paràmetre és el nombre de reinicis, que s'ha determinat de forma experimental, realitzant diferents proves amb problemes de diferents mides i seleccionant el valor que millors resultats ha donat en general. En la variant utilitzada, si algun dels reinicis de la cerca millora la solució el comptador de reinicis realitzats torna a zero. Així doncs, l'algorisme acabarà quan realitzi el nombre fixat de reinicis sense trobar una solució que sigui millor. Aquesta solució serà l'òptim global o un òptim local. Generar les noves solucions aleatòries sol ser un problema, però en aquest cas només ha calgut ordenar la llista de nodes de manera aleatòria per generar la nova ruta. Aquests nous punts de partida mantenen sempre el primer node com al punt inicial a visitar, doncs és la posició on està el robot i el lloc des d'on comença el recorregut.

L'operador de transformació que s'ha fet servir genera tots els successors d'un node donat. Per fer-ho, s'escull un node de la ruta aleatòriament i s'intercanvia amb la resta de nodes de la ruta (exceptuant el primer). El factor de ramificació[9] d'aquest operador és $\theta(N)$, essent N el nombre de localitzacions de la ruta. El millor d'aquests successors serà el següent node al que l'algorisme es mourà.

Simulated Annealing

El fet que el Hill Climbing només miri els successors que són millors fa que sovint no sigui possible arribar a l'òptim. Seria convenient poder realitzar salts a diferents punts tot i que aquests siguin més dolents. El Simulated Annealing (en català temperat simulat) pren el nom del procés físic utilitzat per endurir metalls i vidres que s'escalfen a altes temperatures i que llavors es refreden gradualment fent que l'estat final d'energia del material sigui petit i estable. Així s'eliminen impureses i es pot obtenir un producte millor, però l'estratègia de refredament hi juga un paper important. L'algorisme es va crear inspirant-se en aquest fenomen físic.

[9] Màxim nombre de successors d'un node.

En lloc de generar tots els successors d'un node i triar el millor, se n'escull un al atzar. Un cop obtingut el successor, si aquest millora la solució l'algorisme s'hi mou. En cas contrari, es decideix si es realitza el moviment cap al successor en funció de diferents paràmetres, que són la funció de qualitat[10], la temperatura (que actua com a paràmetre de control) i l'estratègia de disminució de la temperatura. Aquests tres elements defineixen la probabilitat de moure's a un estat pitjor segons l'expressió $e^{\Delta E/T}$, essent ΔE la diferència entre l'energia del node actual i del successor, i T la temperatura. D'aquesta expressió se n'extreu que, com més alta sigui la temperatura, més probable serà realitzar el moviment a un estat pitjor, i que com més gran sigui la diferència d'energies, menys probable serà realitzar tal moviment. Cal notar que $\Delta E = E_{actual} - E_{successor}$ i que, si el successor té pitjor qualitat (distància més llarga), llavors ΔE tindrà valor negatiu. Respecte l'estratègia de refredament, aquesta consta en un nombre fixe d'iteracions, on cada cert nombre d'iteracions la temperatura disminuirà des d'una temperatura inicial fins a ser zero al final de l'algorisme. Si aquesta estratègia va prou a poc a poc, l'algorisme és capaç de trobar l'òptim amb una alta probabilitat.

El node successor es genera de forma similar que en el Hill Climbing però, com que no és necessari obtenir tots els successors d'un node, l'operador retorna una nova ruta on dos elements escollits a l'atzar s'han intercanviat. Els paràmetres de l'algorisme com la temperatura inicial, el nombre d'iteracions prèvies a un descens de la temperatura i el seu factor de descens s'han determinat de forma experimental, realitzant diferents proves fins que s'ha arribat a resultats satisfactoris.

Comparació de resultats

A continuació es mostra una taula amb els resultats d'executar cent vegades l'algorisme Visib-PRM sobre un mapa. Per a cada execució, es cercava la ruta òptima sobre el conjunt de nodes resultant mitjançant els dos algorismes de cerca local emprats, ambdós començant des del mateix estat inicial. Les distàncies mostrades són en metres.

El problema amb menys nodes en tenia quinze, el que en tenia més trenta-dos i en mitjana n'hi havia vint-i-set. S'ha pogut comprovar que el Hill Climbing ha donat millors resultats en mitjana, i que ha estat millor que el Simulated Annealing moltes més vegades. Els paràmetres del Simulated Annealing es

[10] En aquest mètode també s'anomena energia.

Algorisme	Millor distància	Pitjor distància	Distància mitjana	Nombre de vegades millor
Simulated Annealing	34.001	120.699	82.833	21
Hill Climbing	32.10	76.611	64.618	79

Taula 4.1: Resultats dels algorismes de cerca local

van cercar mitjançant execucions d'aquesta prova amb diferents paràmetres, seleccionant els que millors resultats van donar.

4.2.3 Cerca de superfícies

Una vegada s'ha obtingut una ruta que passi per tots els PDS, que són els punts de guarda del Visib-PRM, aquesta s'ha de recórrer. Quan el sistema de navegació ens hagi portat a la següent localització de la ruta, el robot ha de detectar les superfícies visibles des d'aquest punt.

Per tal d'acomplir aquesta tasca, s'ha creat un estat que es pot usar en diferents màquines d'estats. L'estat utilitza un paquet de ROS anomenat *tabletop_ object_ detection*[11] i que ha estat adaptat per PAL Robotics. Aquest paquet, entre d'altres opcions, ofereix un sistema de detecció de superfícies. Utilitzant informació d'un sensor de profunditat[12], aquest paquet obté un núvol de punts al qual li aplica el mètode RANSAC per trobar el pla dominant de l'escena. Aquest és un mètode iteratiu emprat per estimar paràmetres d'un model matemàtic a partir d'un conjunt de dades d'entrada que pot tenir bastant soroll. En la detecció de plans les dades d'entrada són el núvol de punts, i el model matemàtic que es vol ajustar és el d'un pla. El procés de detecció del paquet retorna les característiques del pla detectat, en concret les coordenades de cada cantonada del pla i la seva orientació.

Com que l'objectiu de l'estat és maximitzar el nombre de deteccions possibles, aquest realitza tres crides al detector de plans, cascuna amb el cap mirant mirant a una direcció diferent, primer dreta, després endavant i finalment a l'esquerra. D'aquesta manera obtenim tots els plans que hi ha al davant del robot, però abans de retornar-los se'ls aplica un petit filtre. En primer lloc, es comprova que la superfície sigui horitzontal, descartant així alguna detecció d'una paret. En segon lloc, es calcula la distància que hi

[11] www.ros.org/wiki/tabletop_object_detector

[12] Per exemple una càmera de tipus Microsoft® Kinect ™ o Asus® Xtion.

ha entre el pla i la resta de plans detectats, descartant-lo si aquest està a menys d'una certa distància d'alguna de les altres deteccions. La distància és un paràmetre variable de l'estat, especificat al instanciar-lo, i es calcula des d'un centre de la superfície a l'altre. Aplicant aquest segon filtre s'evita que possibles repeticions de taules es prenguin com a dues superfícies diferents, cosa que provocaria una doble visita en aquell pla.

Com que en un PDS s'ha de cercar totes les superfícies del voltant del robot, s'ha creat també una sub-màquina d'estats que utilitza l'anterior estat per fer-ho. Aquesta realitza una execució de l'estat anterior i llavors gira sobre el seu eix vertical, tornant a executar l'estat. S'ha inclòs paràmetres per especificar la quantitat de gir i el nombre de vegades que s'ha de girar. S'ha provat diferents opcions, com realitzar quatre execucions de la màquina i tres girs de noranta graus, cobrint així tot el voltant. Una altra opció és realitzar només dos girs de cent vuitanta graus, aprofitant al màxim el moviment del cap per cobrir també les zones laterals. Aquesta última manera és més ràpida però pot no detectar algun pla, pre exemple si el detector troba una paret com a pla predominant enlloc de la taula. De la mateixa forma que es fa en l'estat que realitza la detecció, les superfícies detectades es filtren cada vegada per evitar resultats repetits, descartant les deteccions la distància entre les quals no superi un cert llindar. Al finalitzar, la màquina d'estats retorna les posicions dels plans detectats. Aquestes posicions corresponen al centre de cada superfície, però traslladades a fora de la taula, a la distància que s'especifiqui al instanciar la màquina. Això permet enviar un objectiu de navegació cap al punt, deixant el robot davant la taula. En cas de no fer-ho així, el robot mai podria col·locar-se en el lloc.

Les posicions que retorna aquesta màquina són els punts que abans hem denominat PDO. Donat que n'hi poden haver uns quants, i per minimitzar el temps de recorregut, es torna a executar el càlcul de la ruta mínima que passa per tots els punts, igual com s'ha fet amb els punts PDS. Calculant aquesta ruta s'aconsegueix que, si diferents superfícies no contenen objectes, el temps i la distància que s'inverteixen en navegar d'un lloc a l'altre siguin menors.

4.2.4 Altres particularitats

Com ja s'ha indicat, aquest enfocament ha pres de base el primer que s'ha realitzat. Per exemple, es segueixen emmagatzemant els resultats de les deteccions d'objectes, de manera que la següent vegada que s'executi la cerca

s'anirà directament a aquell punt del mapa.

Si no queden objectes pendents de reconèixer, es comprova l'esta de la ruta de PDOs i, en cas que no s'hagi recorregut sencera, es continuarà des d'on s'havia deixat l'últim cop. Tot i això, la ruta es tornarà a calcular ja que no es pot saber si el robot s'ha mogut de lloc o no entre una execució i la següent, i la ruta era bona des del lloc on s'havia acabat l'execució. Des d'un altre punt de la sala poden existir rutes més curtes per recórrer els punts restants. En cas contrari, si la ruta de PDOs és buida, es realitza la mateixa comprovació per a la ruta de PDSs, aplicant els mateixos raonaments.

Finalment, la part de reconeixement d'objectes també és idèntica a la realitzada pel primer enfocament.

4.2.5 Valoració de l'enfocament

Aquest enfocament funciona de forma correcte, tot i que té alguns avantatges i alguns inconvenients respecte l'anterior enfocament.

Com a principal millora, no és necessari que l'usuari indiqui les zones. S'ha aconseguit un comportament bastant autònom, on l'únic que es necessita per part de l'usuari és una indicació de quines són les cantonades de cada habitació, tal i com s'ha explicat. Obtenir aquestes dades és una feina una mica feixuga i avorrida, però només cal fer-ho una vegada ja que són uns paràmetres que no canviaran, a no ser que es canvii el mapa o el lloc on està el robot. En canvi, les dades necessàries pel primer enfocament era més probable que patissin modificacions. A més, la generació de punts mitjançant l'algorisme Visib-PRM ha donat resultats satisfactoris la majoria de les vagades, amb uns resultats que cobrien la major part de l'espai de la sala.

Per contra, al no tenir informació del tipus de lloc on s'ha detectat un objecte, no és possible indicar que l'objecte ha estat trobat sobre la taula o sobre la cadira. Per tant, per indicar on està l'objecte, s'han d'utilitzar expressions indeterminades, com "allà", o simplement només dir que s'ha trobat.

En aquesta versió de l'algorisme, un cop s'ha detectat superfícies horitzontals en un punt PDS s'avorta la cerca d'altres superfícies, i es visiten les superfícies detectades. S'ha implementat d'aquesta manera per les necessitats del moment, que requerien trobar objectes en el mínim temps possible. Una altra opció podria ser fer primer el recorregut de tots els PDS, emmagatzemant totes les superfícies detectades i recorrent llavors la ruta amb totes les

superfícies de l'habitació. Aquesta versió es podria implementar a partir de la primera versió amb pocs canvis i sense gaire esforç. En la Figura 4.4 es mostra un exemple d'una prova de l'algorisme simulada, on es provaven els .

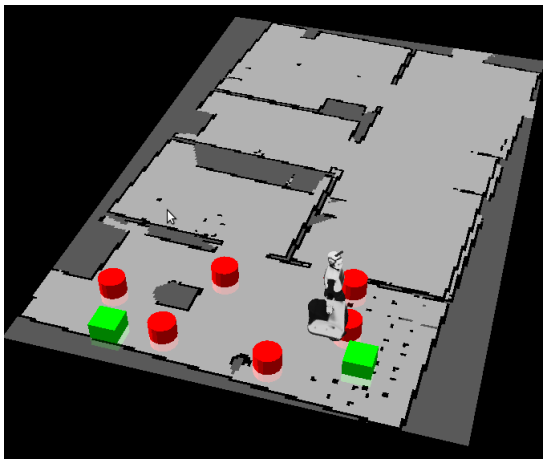


Figura 4.4: Execució del segon enfocament en simulacó. En el visualitzador es mostren els punts que s'ha de visitar (els cilindres vermells) i les superfícies detectades (els cubs verds).

4.3 Tercer enfocament

Una vegada ja s'ha aconseguit implementar una solució al problema que sigui més autònoma, aquesta es pot intentar refinar per veure si es pot obtenir millors resultats. Per fer-ho, en aquest tercer enfocament s'ha incorporat més coneixement sobre l'entorn.

En l'anterior solució, es pren com a assumpció que els objectes es troben sobre superfícies planes i horitzontals. Tot i ser una afirmació bastant certa, no es distingeix entre taules, llits, cadires ni cap altre tipus de mobiliari. Donat que no és igual de probable que hi hagi objectes en un sofà que en una prestatgeria, s'ha afegit coneixement sobre el mobiliari de manera que l'ordre de visita de les superfícies detectades depengui del tipus de superfícies que hi ha.

Aquest reconeixement del mobiliari s'ha dut a terme mitjançant l'ús i adaptació del paquet de ROS anomenat *furniture_ classification*[13]. El paquet fa

[13] www.ros.org/wiki/furniture_classification

ús de la llibreria PCL[14], que proporciona moltes utilitats per tractar amb núvols de punts, com per exemple els obtinguts amb un sensor de profunditat.

En el moment d'escriure aquest document, el paquet *furniture_ classification* no es troba actualitzat des de fa un temps. Això ha suposat un problema degut al fet que utilitza una versió inestable de PCL, que rep actualitzacions diàries. Per aquest motiu s'ha hagut d'utilitzar una versió més antiga de la llibreria, doncs hi havia bastants problemes de compilació. També ha estat necessari adaptar una mica el codi i els paràmetres de compilació a mesura que s'anava provant de fer-lo funcionar, ja que d'altra forma apareixien problemes en temps d'execució.

Les eines que proporciona el paquet són un classificador, una eina d'entrenament d'aquest i un programa que genera fitxers amb núvols de punts a partir de models CAD de l'objecte, que poden servir d'entrada a l'entrenador. En aquest enfocament només s'usarà l'entrenador i el classificador.

Com que les eines per fer classificació de mobles només es poden usar des d'una terminal, sense proporcionar quelcom que es pugui cridar des de la màquina d'estats SMACH, s'ha hagut de crear un servei que realitzi el mateix que aquestes eines.

Finalment, i per tal de poder obtenir informació per l'entrenador de manera senzilla, s'ha creat una eina per a terminal que captura instantànies del sensor de profunditat, i les emmagatzema en diferents arxius organitzats en directoris de la manera que siguin acord al format que espera l'eina que entrena el classificador. Aquest programa fa captures quan l'usuari ho indica, i permet capturar un mateix objecte des de diferents punts de vista.

Tant el servei com l'eina que fa les captures s'han implementat en C++, per facilitar l'ús de la llibreria PCL i del codi del paquet. Per la màquina d'estats[15] s'ha utilitzat Python, com en els altres enfocaments.

4.3.1 El servei `furniture_detector_service`

És un servei de ROS que utilitza el classificador. Per tal d'invocar-lo, cal fer-li una petició, sense cap paràmetre, i el servei prendrà una instantània del sensor de moviment i invocarà el classificador. Un cop s'ha obtingut resultats

[14] *PointCloud Library*: www.pointclouds.org/

[15] A l'Annex A.3 Tercer enfocament s'hi pot veure la seva representació gràfica.

de classificació, retorna al client el conjunt de deteccions d'aquests resultats. En aquesta resposta s'hi inclou una llista amb els noms dels mobles detectats, i una llista de posicions d'aquests mobles. Aquestes posicions corresponen a les coordenades del punt del núvol de punts obtingut del classificador que està més a prop del robot.

Per tal d'implementar-lo, s'ha creat una classe que utilitza el codi del paquet *furniture_ classification*, de manera similar a com ho fan les eines que aquest proporciona.

Per tal de fer-lo funcionar, s'ha hagut de fer algunes modificacions al codi del paquet, així com una adaptació de les crides que fa el classificador de la terminal, ja que en cas contrari hi havia excepcions en temps d'execució que no es tenien en compte en el codi original.

4.3.2 Diferències en l'algorisme respecte el segon enfocament

La màquina d'estats utilitzada per aquest enfocament és idèntica a la utilitzada pel primer, amb excepció d'alguns estats i de la informació que aquests es comuniquen.

En primer lloc, s'ha hagut de crear una nova màquina d'estats que escanegi l'entorn quan el robot es situa sobre algun dels punts de guarda rebuts del Visib-PRM[16]. Aquesta nova màquina és essencialment la mateixa, però en lloc d'utilitzar l'estat que detecta superfícies planes, fa ús d'un nou estat que detecta mobles. Aquest estat també és similar a l'implementat per a l'anterior enfocament, però s'hi ha modificat els moviments del cap per millorar les deteccions dels mobles, i en lloc de realitzar una petició al servei de detecció de plans, la realitza al servei `furniture_detector_service`. A part, ambdós elements retornen, a part de la posició de les deteccions, el nom dels mobles detectats.

Un cop obtinguts els mobles, s'ofereix dues opcions a l'hora de recórrer-les, mitjançant un paràmetre de la màquina d'estats d'aquest enfocament:

- La primera es comporta de la mateixa manera que en l'anterior, creant una ruta pels diferents mobles que minori la distància a recórrer, però amb l'afegit de que es coneix el nom del lloc on es va a mirar.

[16] Veure 4.2.1 Generació de localitzacions de cerca de superfícies.

- L'altra crea una ruta d'una manera diferent, utilitzant una probabilitat de que hi hagi objectes sobre un moble. Aquests paràmetres, són introduïts per l'usuari d'una manera quasi idèntica a com es fa pel primer enfocament, mitjançant un fitxer de configuració on s'hi indica el valor de probabilitat per a cada moble que s'ha entrenat. Llavors, es recorreran els llocs en funció d'aquesta probabilitat, anant primer als que siguin més probables. En cas de que dos elements tinguin la mateixa probabilitat, s'anirà al que estigui més a prop. També s'ofereix la possibilitat d'especificar mobles on no es vol que es cerqui objectes, ja que els elements amb probabilitat zero no es tenen en compte dins la ruta.

4.3.3 Valoració de l'enfocament

Aquest enfocament aprofita els punts forts del primer i del segon, i en millora alguns aspectes. Per exemple, a diferència del segon enfocament, en aquest enfocament es pot dir que l'objecte trobat està sobre una cadira, ja que aquesta s'ha pogut reconèixer.

Tot i que un punt feble del primer enfocament era el fet d'afegir els diferents llocs amb la seva probabilitat, en aquest cas no és un inconvenient. El fet de no haver d'afegir les coordenades dels punts simplifica la tasca, i al no haver-hi aquestes coordenades, l'arxiu només s'ha de crear un cop amb cada moble i les seves probabilitats, sense necessitat de canviar-lo si es canvia algun element de lloc, sempre i quant el moble es segueixi anomenant igual.

Un dels principals problema d'aquest enfocament és la necessitat d'entrenar els mobles. Això implica capturar informació de tots els tipus de mobles des de diferents angles de visió. Aquesta tasca, tot i que només seria necessària una vegada, és bastant costosa en temps i paciència de l'usuari, tot i que l'eina que s'ha fet per prendre captures simplifiqui en gran part la feina. L'altre problema, i el que el fa menys usable, és el temps que triga el classificador. Aquest temps s'incrementa amb el nombre de models entrenats, fent que tot l'enfocament sigui molt lent a l'hora de detectar els mobles de les diferents zones, mentre que la detecció de plans utilitzada en el primer enfocament era molt més ràpida.

Aquests problemes fan que, de moment, l'enfocament sigui poc aprofitable per a entorns reals. Malgrat això, sembla que és una bona opció i que mostra un camí per on es podria continuar, potser amb un classificador diferent o més

ràpid, o realitzant una adaptació més profunda de l'existent, refaccionant el codi d'aquest per tal que satisfaci exactament les necessitats d'aquest treball.

4.4 Simplificació de la utilització dels enfocaments

Per tal de que es pugui utilitzar els diferents enfocaments de manera que es pugui canviar d'un a un altre realitzant els mínims canvis possibles en el codi que en fa ús, s'ha creat una funció que encapsula la creació d'instàncies de les diferents màquines d'estats.

D'aquesta manera, per utilitzar un comportament de cerca d'objectes només s'ha d'importar aquesta funció, anomenada `ObjectFindingBehaviour`, rep un paràmetre de tipus cadena de caràcters que indica si es vol el primer, el segon o el tercer enfocament. Per aquest últim, també s'especifica si es vol utilitzar la ruta de cerca d'objectes amb les probabilitats o mitjançant la solució al TSP resultant.

Aquesta solució maximitza i simplifica la canviabilitat del codi, minimitzant l'acoblament entre els diferents enfocaments i les màquines d'estats que els incorporen. La seva creació ha estat inspirada en els coneguts patrons d'arquitectura del software Factoria i Estratègia.

5 Resultats

En aquesta secció es mostren alguns dels resultats obtinguts, a títol d'exemple. Només s'ha realitzat proves completes amb el primer i el segon enfocament. El tercer enfocament, tot i ser funcional, no ha acabat de donar bons resultats a l'hora de classificar els mobles.

Aquestes proves s'han realitzat en diferents entorns casolans reals. Un exemple d'entorn es mostra en la Figura 5.1. Per tal de realitzar-les, s'ha configurat diferents paràmetres de la zona de proves, com les cantonades o les localitzacions on hi pot haver objectes. Llavors s'ha col·locat diferents objectes que el robot pot reconèixer. Posteriorment s'ha executat les diferents màquines d'estats tot calculant-ne el temps que trigava el robot a trobar el primer objecte.

Figura 5.1: Exemple de sala de proves

Per obtindre aquests resultats s'ha realitzat cinc execucions de cada algorisme, totes començant des del mateix punt de partida. Durant l'execució s'ha mesurat el temps que ha trigat el robot a trobar el primer objecte, des de l'inici de l'algorisme. Aquests valors només són orientatius, doncs la cerca dependrà, entre d'altres elements, de les dimensions de la sala. Els resultats en mitjana han estat els següents:

- Temps mitjà del primer enfocament: 1 minut i 47 segons.
- Temps mitjà del segon enfocament: 4 minuts i 53 segons.

El primer enfocament, com ja era d'esperar, ha estat notablement més ràpid que el segon. El motiu principal n'és el fet que en el primer el robot ja coneix

els llocs on ha d'anar. En el segon, en canvi, el robot comença més a cegues i s'inverteix una gran quantitat de temps en cercar els llocs on és possible que hi hagi objectes. Tot i això, hi ha casos on el segon ha anat millor que el primer. Per exemple, si els paràmetres del primer enfocament s'han configurat erròniament, el robot no és capaç de trobar els objectes. També és més lent si no hi ha objectes als llocs on més probabilitat hi ha, com es va comprovar en algunes execucions. Per altra banda, si el segon visita primer punts propers a superfícies planes on hi ha objectes, els trobarà en poca quantitat de temps. A més, aquest segon enfocament és més autònom, cosa que fa que no s'hagi d'invertir gaire temps en introduir i mantenir els paràmetres.

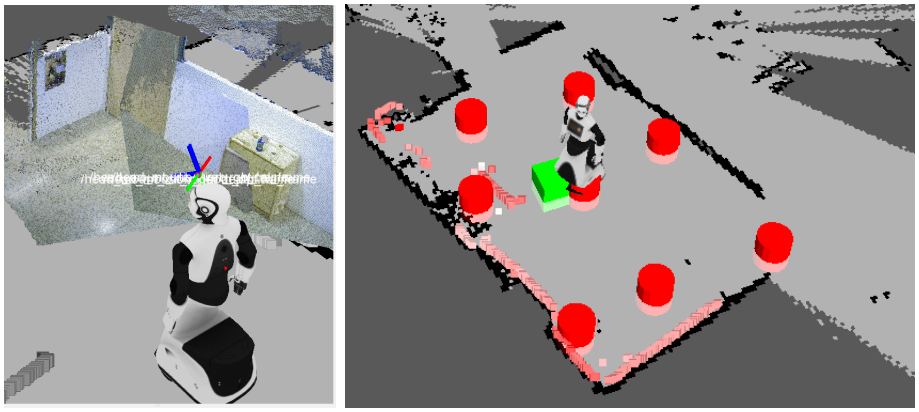


Figura 5.2: Captura de finestres del visualitzador del robot durant els tests. A l'esquerra, imatge on es veu el núvol de punts que utilitza el robot per a fer reconeixement. A la dreta, imatge del robot sobre el mapa durant el segon enfocament. Els cilindres vermells són els llocs a visitar (PDS), mentre que el cub verd és la detecció d'una superfície (PDO, en aquest cas una taula).

6 Conclusions i treball futur

Dels resultats del projecte se'n deriven algunes conclusions.

Per començar, la tria d'un enfocament o un altre depèn de per a quina raó es vol fer ús del comportament de cerca d'objectes. En entorns on el més important sigui trobar un objecte en el menor temps possible, com és el cas de la Robocup, el primer enfocament és més apropiat ja que els llocs on pot haver objecte estan ben definits i el temps del que es disposa és limitat. Per altra banda, en un entorn casolà seria preferible utilitzar el segon o el tercer enfocament, donat que es vol que els robots siguin autònoms, sense necessitat de configurar-los.

Realitzant aquest treball s'ha pogut observar com és possible dotar als robots de comportaments típicament humans. Cercar objectes en una sala és una tasca que depèn d'una gran quantitat d'habilitats, innates per als humans, però que costen de simular. N'és el cas, per exemple, de la navegació del robot per la sala i del reconeixement dels objectes. No obstant també és important poder veure tots els punts d'interès, intentant evitar deixar alguna localització que pugui tenir objectes sense comprovar. Aquests elements són els que s'ha intentat tenir en compte a l'hora de dissenyar i escollir els algorismes que s'ha utilitzat.

Tot i això, és clar que s'està lluny d'assolir uns nivells d'eficiència i naturalitat que siguin comparables amb els de les persones. Per tant, encara queda camí per recórrer en aquest tema, intentant augmentar la rapidesa de les cerques i millorant elements com la detecció dels objectes, per fer-los més robustos als canvis de lluminositat o fins i tot de forma. Es considera que s'ha obtingut resultats que acompleixen els objectius del treball, però encara hi ha diferents elements d'aquest que es podrien millorar en un futur.

En primer lloc, els enfocaments segon i tercer necessiten informació sobre les cantonades de cada habitació. Obtenir aquesta informació, encara que nomes s'hagi de fer una vegada, és poc agradable. Per aquest motiu es va plantejar l'opció d'obtenir aquestes dades del mapa utilitzant tècniques de visió per computador.

S'ha realitzat un intent de resoldre aquest problema mitjançant els instruments que proporciona la coneguda llibreria OpenCV[1]. Es va intentar resoldre mitjançant la transformada de Hough per a línies. Aquest mètode

[1] opencv.org/

detecta línies rectes sobre una imatge. Per a cada píxel de la imatge, representa totes les línies rectes hi que passen, en coordenades polars, en el pla $r - \theta$. Aquesta representació forma una gràfica sinusoïdal en aquest plan. Llavors l'algorisme superposa les línies de tots els píxels transformades al pla $r - \theta$ de manera que les diferents corbes que intersequin en un mateix punt indicaran que els píxels que les han generat estaven sobre una mateixa línia recta. Previ a l'aplicació de la transformada de Hough, es va realitzar diferents filtrats a la imatge, per eliminar-ne el soroll i deixar-ne només les arestes, facilitant la detecció. Després de provar amb diferents valors per als paràmetres que ofereix la funció de OpenCV, no es va trobar una combinació d'ells que proporcionés una detecció de totes les línies suficientment bona per tal de trobar-ne les cantonades. A més, les combinacions que anaven millor en un mapa fallaven per a un mapa diferent. Altres intents van comportar l'ús de detectors de cantonades, com l'operador de Harris, però tampoc es va arribar a resultats prou bons. Per tant, aquesta és una via interessant de seguir per tal de millorar aquests enfocaments i fer-los més autònoms.

Finalment, un altre punt on s'hauria de treballar és en el classificador de mobles del tercer enfocament, bé sigui trobant-ne un de millor, adaptant-ne el que s'ha usat o implementant-ne un des de zero. Aconseguint un classificador més ràpid i amb millors resultats suposaria una gran millora de l'enfocament, fent que el robot es mostri més intel·ligent tant a l'hora de dir on està l'objecte com en decidir en quin moble és més probable trobar-hi alguna cosa.

7 Pressupost

Tot seguit es farà una valoració dels costos que suposa el projecte. Com que el projecte l'ha realitzat una sola persona, es repartirà la feina de les diferents tasques en rols, de manera que cada rol tindrà un preu per hora. Per fer aquest càlcul es considera que els dies tenen aproximadament unes 4,5 hores de treball i es basarà en la planificació feta.

7.1 Rols escollits

Els rols que s'ha triat són els següents:

- Cap de projecte. El preu per hora serà de 40€/hora. La seva feina es comprovar que el projecte segueix el seu curs i planificació, fer reunions i valorar resultats.
- Programador. El preu per hora serà de 20€/hora. La seva feina és implementar els diferents programes, fer recerca d'informació necessària i corregir errors, entre d'altres.
- Provador. El preu per hora serà de 15€/hora. La seva feina és comprovar que el programa funciona correctament en el simulador i en el robot, informant dels problemes per tal de que siguin solucionats.

7.2 Despeses de personal

La distribució de feina per tasques amb els seus preus es mostra en la següent taula:

Enfocament	Rol	Hores	Preu
Primer	Cap de projecte	3,5	140,00€
	Programador	40,5	810,00€
	Provador	42,75	641,25€
	Total	**86,75**	**1.591,25€**
Segon	Cap de projecte	10,0	400,00€
	Programador	175,5	3.510,00€
	Provador	54,0	810,00€
	Total	**239,5**	**4.720,00€**
Tercer	Cap de projecte	5,5	220,00€
	Programador	72,0	360,00€
	Provador	65,25	978,75€
	Total	**142,75**	**1.558,75€**
Comparació	Cap de projecte	2,0	80,00€
	Programador	13,5	270,00€
	Total	**15,5**	**350,00€**
Total acumulat		**484,5**	**8.220,00€**

Taula 7.1: Pressupost de personal

7.3 Despeses de hardware i software

Pel que fa al pressupost per a hardware, tenim el següent:

Ítem	Unitats	Preu unitari	Cost real (6 mesos)
Ordinador amb Ubuntu. Quad core amb 4GB de RAM	1	450,00€	75,00€
Ordinador amb Ubuntu. Quad core amb 8GB de RAM i gràfica Nvidia	1	670,00€	111,67€
Robot REEM H2	1	0,00€	0,00 €
Total	**3**	**1.120,00€**	**186,67€**

Taula 7.2: Pressupost de hardware

El robot té cost nul ja que és cedit per l'empresa PAL Robotics. L'ordinador

amb la targeta gràfica més potent i més memòria serà destinat al provador, doncs necessitarà utilitzar software de simulació en 3D i haurà de menester una bona targeta gràfica. L'altre ordinador és pel programador, i en aquest cas no serà necessari un ordinador d'altes prestacions.

En aquest pressupost de hardware, s'identifica com a cost real el cost imputable al projecte, doncs els equips informàtics s'amortitzen en tres anys, però el projecte només ha durat sis mesos.

Pel que fa a les despeses de software, s'utilitzen els programes ROS i Gazebo, ambdós de llicència gratuïta.

7.4 Despeses generals

Les despeses que s'originen mentre es realitzen les activitats del treball. En aquest apartat s'inclouen les despeses de lloguer del local, Internet i l'aigua i l'electricitat consumides.

Concepte	Preu	Temps	Total
Lloguer (inclou electricitat i aigua)	850€/mes	6 mesos	5.100,00€
Internet ADSL	49,90€/mes	6 mesos	299,40€
Total acumulat			**5.399,40€**

Taula 7.3: Despeses generals

7.5 Cost total

Sumant les despeses de personal, hardware i software durant els sis mesos de duració del projecte, el cost total d'aquest seria de 13.806,07€. Aplicant-hi un I.V.A. de tipologia general del 21%, ens resulta en un cost final de 16.705,34€.

7.6 Seguiment del pressupost

Donat que la planificació s'ha pogut acomplir, tot i que amb lleugeres desviacions, no es considera que hi hagi hagut alteracions en el pressupost d'aquest projecte.

7.7 Viabilitat del projecte

Aquest treball, tot i ser relacionat amb una empresa, no està destinat a formar un producte final sinó que serveix per a un projecte més gran (el REEM@IRI) i per tant, en si mateix no tindrà viabilitat econòmica.

Es podria considerar com un projecte d'investigació, on a priori no hi ha viabilitat en termes de beneficis econòmics, però si que és viable en termes de coneixement: es comparteix el codi amb la resta d'equips de la competició per tal d'avançar en l'estat de l'art dels robots de servei, i a canvi es rep també el codi i coneixements aportats per la resta d'equips.

Llavors, empreses com PAL Robotics (que cedeix el robot REEM), es poden valer d'aquests coneixements per tal d'incorporar noves característiques i funcionalitats als seus robots de cara a la venda dels mateixos al públic. D'aquesta manera, aquest projecte seria econòmicament viable per a aquestes empreses.

Bibliografia

[1] A. Aydemir. *An Approach to Efficient Object searching for mobile robots.* Master Thesis. Mälardalen University, 2008.

[2] A. Aydemir, A. Pronobis, K. Sjöö, M. Göbelbecker, and P. Jensfelt. *Object search guided by semantic spatial knowledge.* RSS'11 Workshop on Grounding Human-Robot Dialog for Spatial Tasks, Los Angeles, CA, USA, July 2011.

[3] M. Boussard, and J. Miura. *Observation planning for object search by a mobile robot with uncertain recognition.* Intelligent Autonomous Systems 12. Advances in Intelligent Systems and Computing Vol. 194, 2013, pp 101-110.

[4] J. E. Bresenham. *Algorithm for computer control of a digital plotter.* IBM Systems Journal, vol.4, no.1, pp.25,30, 1965

[5] J. Cabanillas, E. F. Morales, and L. E. Sucar. *An efficient strategy for fast object search considering the robot's perceptual limitations.* Advances in Artificial Intelligence – IBERAMIA 2010. Lecture Notes in Computer Science Vol. 6433, 2010, pp 552-561.

[6] S. Ekvall, P. Jensfelt, and D. Kragic. *Integrating active mobile robot object recognition and SLAM in natural environments.* IEEE/RSJ International Conference on Robotics and Automation (IROS'06), 2006.

[7] T. Kollar, M. Samadi, and M. Veloso. *Enabling robots to find and fetch objects by querying the web.* 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '12) Vol. 3, Valencia, Spain, June 2012, pp 1217-1218.

[8] S. J. Russell, P. Norvig and E. Davis. *Artificial Intelligence: A Modern Approach.* 3rd ed. Upper Saddle River, N.J.: Prentice Hall, 2010. ISBN 9780136042594.

[9] F. Saidi, O. Stasse, K. Yokoi, and F. Kanehiro. *Online object search with a humanoid robot.* IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007 (IROS 2007), San Diego, CA, USA, October-November 2007, pp 1677-1682.

[10] M. Samadi, T. Kollar, and M. Veloso. *Using the web to interactively learn to find objects.* 26th Conference on Artificial Intelligence, AAAI 2012, Toronto, Canada, July 2012.

[11] K. Shubina, and J. K. Tsotsos. *Visual search for an object in a 3D environment using a mobile robot.* Computer Vision and Image Understanding, Vol. 114, Issue 5, May 2010, pp 535-547.

[12] T. Siméon, J.-P. Laumond, and C. Nissoux. *Visibility-based probabilistic roadmaps for motion planning.* Advanced Robotics, Vol. 14, No. 6, 2000, pp. 477-493

[13] K. Sjöö, D. Gálvez López, C. Paul, P. Jensfelt and D. Kragic. *Object search and localization for an indoor mobile robot.* Journal of Computing and Information Technology, Vol. 17, No. 1, 2009.

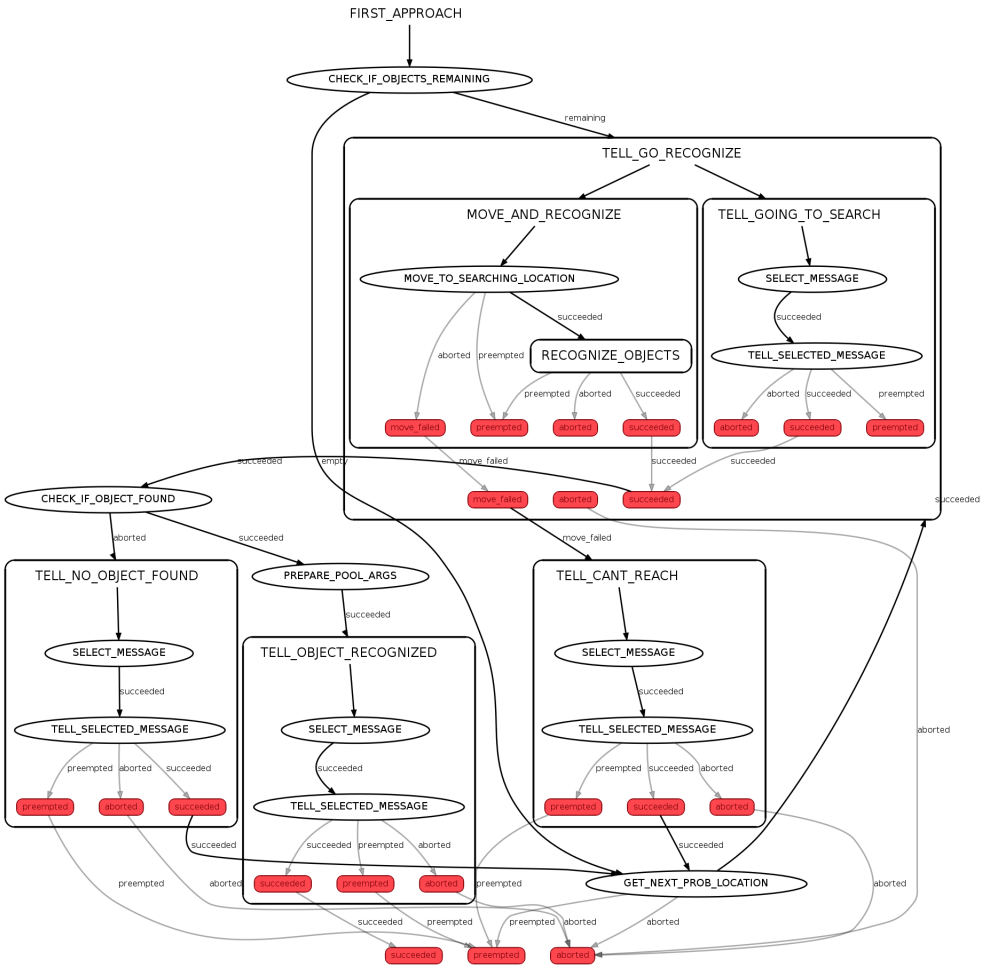
[14] Y. Jiang, A. and Saxena. *Hallucinating Humans for Learning Robotic Placement of Objects.* International Symposium on Experimental Robotics (ISER), May 2012.

[15] L. Ziegler. *Developing a vision-based object search behavior for a mobile robot.* Master Thesis. Applied Informatics Group, Bielefeld University, 2010.

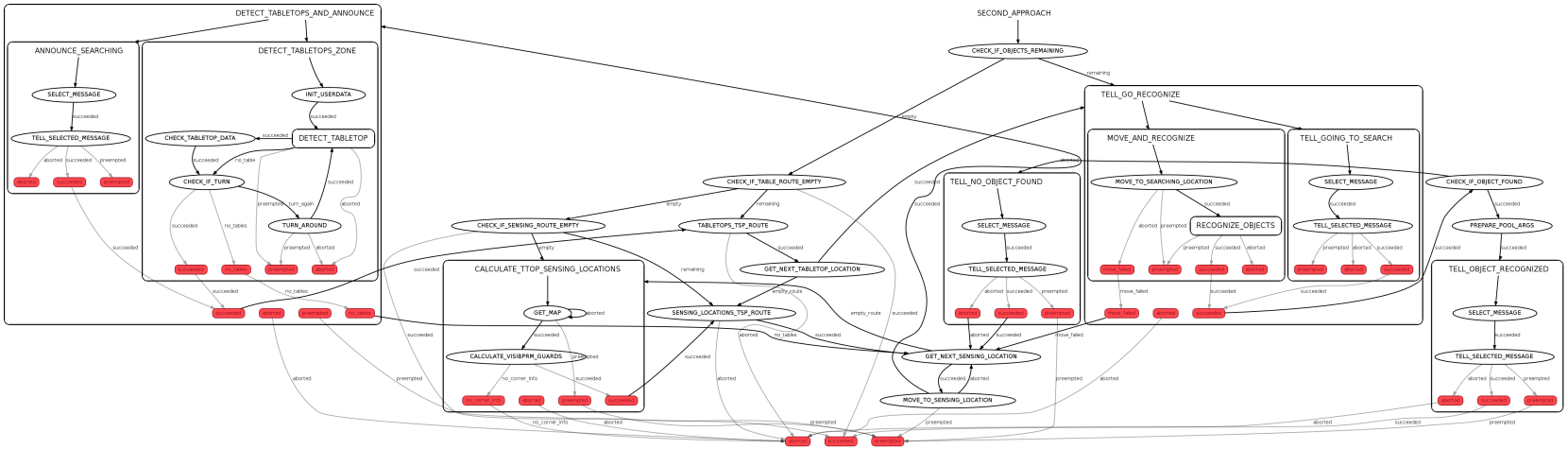
A Diagrames de les màquines d'estats

En aquest annex es mostren les diferents màquines d'estats de forma gràfica.

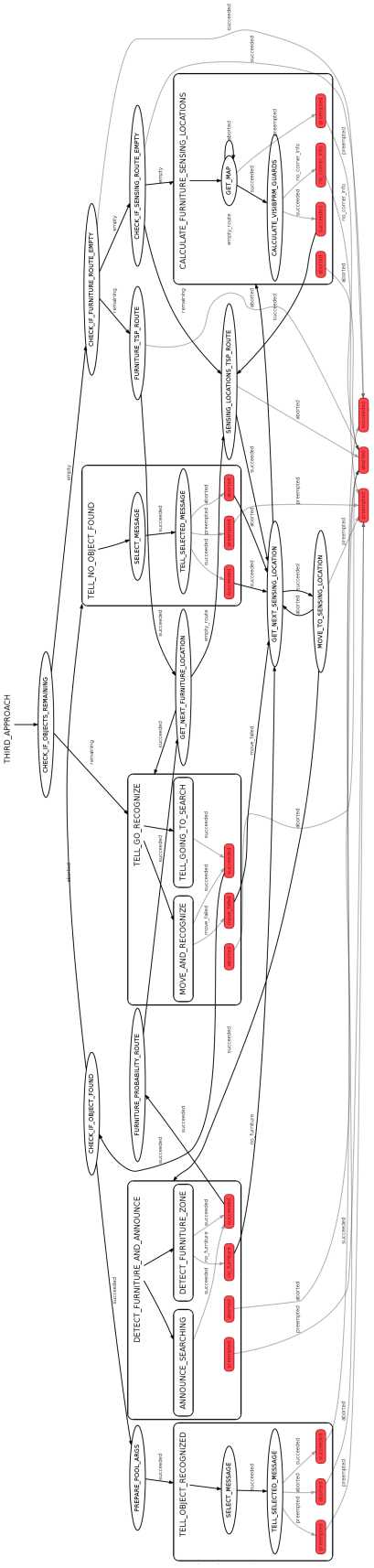
A.1 Primer enfocament



A.2 Segon enfocament



A.3 Tercer enfocament



B Glossari d'abreviatures

IRI: Institut de Robòtica i Informàtica Industrial
OSRF: Open Source Robotics Foundation
PCL: PointCloud Library
PDS: Punts de Detecció de Superfícies
PDO: Punts de Detecció d'Objectes
RANSAC: RANdom SAmple Consensus
ROS: Robot Operating System
SMACH: State MACHine
TFG: Treball Final de Grau
YAML: "YAML Ain't Markup Language", anteriorment considerat "Yet Another Markup Language"