Construcció de mapes per robots mòbils equipats amb sensors làser de profunditat

A. Checa, J. Andrade, A. Sanfeliu

IRI-DT-03-01

1400447265

Abril del 2003

Resum

Projecte Final de Carrera

Albert Checa Puimedon

Titulació: Enginyeria Industrial

Intensificació: Automàtica

Aquest projecte s'inclou dins de la investigació en el camp de la robòtica mòbil i queda englobat dins d'un altre projecte que rep el nom de Marco, que té per objectiu l'aprenentatge per part d'un robot de l'entorn que l'envolta. Per aconseguir aquesta fita, el robot ha de rebre dades captades pels seus sensors, saber interpretar-les i respondre en conseqüència. Concretament, aquest projecte tractarà les diferents tècniques per construir mapes d'entorns tancats, i se'n desenvoluparà una per tal d'assolir els objectius marcats. Finalment, es mostrarà el funcionament del programa realitzat i es comentaran els resultats obtinguts, extraient les pertinents conclusions.

Índex

Capítol 1

Introducció

Els robots mòbils tenen un bon nombre d'aplicacions en camps molt diversos, ja sigui ajudant o substituint als éssers humans. El que es pretén amb aquest tipus de robots és aconseguir aproximar el seu comportament a l'humà en la mesura del possible, i per aconseguir-ho, el robot ha de ser capaç de percebre el seu entorn i interactuar amb ell, és a dir, necessita una sèrie de sensors que captin dades de l'entorn, i un sistema d'intel·ligència artificial que les interpreti.

Dintre d'aquest camp de la robòtica mòbil, tal i com diu el seu nom, una part molt important és la mobilitat del robot. Un cop el robot ja ha decidit on vol traslladar-se, ha de saber quines són les trajectòries possibles per assolir la nova posició sense topar amb cap obstacle. Per això és de vital importància que el robot disposi d'un mapa de l'entorn en el que es troba.

A més, un altre problema que presenta la mobilitat del robot és que aquest no coneix la seva posició exacta, ja que les dades obtingudes dels seus sensors de desplaçament contenen error, que es va acumulant i que pot arribar a ser molt gran.

El present projecte abordarà aquest dos problemes, que en el camp de la robòtica mòbil s'engloben en un de sol, que rep el nom de *Concurrent Mapping & Localisation* (CML), i s'intentarà trobar un mètode que serveixi com a solució d'aquest.

1.1 Motivació i aplicacions

La robòtica mòbil i en concret el mapatge d'entorns tancats té un gran nombre d'aplicacions molt útils per a les nostres vides. A més, un cop superat aquest problema, s'obren les portes a molts altres camps nous per investigar. A continuació en citarem uns exemples:

1. Navegació totalment autònoma. Si el robot és capaç de crear un mapa del seu entorn sense necessitat d'ajut extern, significa també que serà capaç d'utilitzar qualsevol algoritme de planificació de trajectòries ja existents i desplaçar-se d'un punt a un altre sense topar amb cap obstacle. A més seria també capaç de trobar autònomament la millor trajectòria entre dos punts.

2. Aprenentatge i identificació d'habitacions. Un cop el robot ha fet el mapa de l'habitació o zona en la que es troba, aquest pot tractar d'identificar l'habitació on es troba, en cas que hi hagi estat anteriorment, o en el cas contrari, pot memoritzar el nou mapa per a què quan torni a estar al mateix lloc, el reconegui. D'aquesta manera es podran donar comandes al robot de més alt nivell, com "Vés al menjador" o semblants.

3. Repartidor. Per exemple, en una empresa amb moltes oficines, si el sistema de navegació autònoma va acompanyat d'un de reconeixement de cares i de reconeixement de veu, un robot pot fer de repartidor de cartes o paquets coneixent en quina oficina es troben habitualment cadascun dels treballadors.

4. Navegació per zones perilloses o ambients hostils. Un robot equipat amb un sistema de navegació autònom podria desplaçar-se i realitzar diferents tasques en entorns perillosos on l'home no té possibilitat de treballar, per exemple en entorns on es desprenguin gasos o algun tipus de substància tòxica.

1.2 Objectius i abast del projecte

L'objectiu principal del present projecte és el de desenvolupar un sistema capaç de crear un mapa de l'entorn del robot i que alhora sigui capaç d'estimar la posició exacta del robot a partir de les dades obtingudes d'un sensor de profunditat.

L'ideal d'un sistema de construcció de mapes per robots mòbils seria el que complís les següents condicions:

- Ha de construir un mapa de l'entorn del robot el més fidel possible a la realitat.
- Ha de poder corregir la posició del robot, a partir del mapa creat, i alhora evitar que el robot es perdi per l'espai en el què es troba, i es trobi en situació de *deriva*.
- Ha de ser capaç de tractar informació imprecisa, ja que d'una banda els sensors de moviment (encòders) i odometria del robot són imperfectes i per altra banda les mesures donades pel sensor làser també contenen error.
- Ha de permetre la integració de la informació de les dues fonts d'informació de les que disposa per tal de poder reduir la incertesa i possibles ambigüitats.
- *Autonomia*, és a dir que el robot ha de ser capaç de crear progressivament el mapa sense cap tipus d'ajuda externa.
- Robustesa davant de canvis temporals de l'entorn. Per exemple, una porta que s'obre, persones que es mouen al voltant del robot, etc.

Aquests requeriments són, com ja s'ha dit, els ideals. El que es pretén en aquest projecte és donar una solució tan propera a aquestes condicions com sigui possible.

1.3 Contingut del projecte

La memòria d'aquest projecte està estructruada de la següent manera:

- El capítol 1 és la introducció, objectius, abast i contingut del projecte.

- El capítol 2 té com a objectiu la revisió de l'estat de l'art. S'analitza la literatura existent en el camp de la mesura de distàncies, i sobre els diferents mètodes de tractament del problema de la construcció de mapes per robots mòbils, així com la seva pròpia localització.

- El capítol 3 descriu el hardware i el software utilitzat en el desenvolupament del projecte.

- En el capítol 4 es descriu el mètode utilitzat per a detectar landmarks.

- En el capítol 5 s'expliquen els passos seguits per al desenvolupament d'un mètode de construcció de mapes basat en un filtre de Kalman, a partir dels landmarks obtinguts amb el mètode d'extracció descrit en el capítol 4.

- Els resultats es reporten en el capítol 6.

- El capítol 7 descriu les conclusions i recomanacions per a futurs treballs.

- Finalment, els annexes inclouen la memòria econòmica i els llistats de codi.

Capítol 2

Estat de l'art

Aquest capítol pretén donar al lector una visió general de les tècniques més utilitzades per a la construcció de mapes. I degut a la seva importància en el desenvolupament d'aquest projecte, també s'explicaran els mètodes existents en l'actualitat per a poder mesurar la distància entre un robot i qualsevol dels obstacles que el rodegen.

2.1 Mesura de distàncies

L'element tractat en aquest apartat és la base per totes les tècniques que es descriuran posteriorment. Totes elles parteixen d'un sensor o sistema que ens dóna com a informació la distància a la que es troben els obstacles que envolten el robot, sense necessitat de contacte físic amb aquests obstacles. Per això és important tenir una idea de quins són els medis disponibles en l'actualitat per fer aquesta tasca, i també conèixer les bases del seu funcionament.

Cal distingir clarament entre sistemes que només detecten la presència d'obstacles dins d'una certa regió i els que mesuren la distància a la que estan aquests. En aquest document es parlarà només dels segons.

Els sistemes per mesurar distàncies més desenvolupats i més utilitzats són els següents:

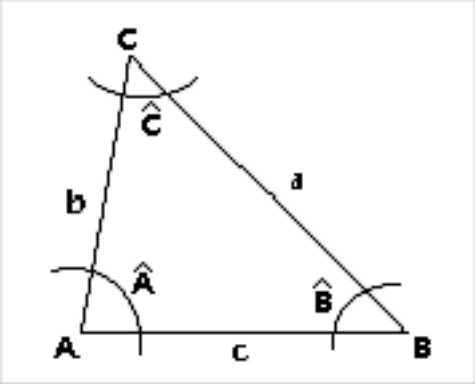

Figura 2.1: Teorema del sinus

- *Estereovisió.* Aquest mètode parteix de les imatges rebudes de dues càmeres situades cadascuna en una posició coneguda, i mitjançant la triangulació i càlculs trigonomètrics es pot saber la distància a la que està un objecte.

- SONAR (*Sound Navigation and Ranging*). Aquest tipus de sensors mesura distàncies de possibles obstacles a partir de la reflexió en aquests objectes de les ones d'ultrasons enviades pel propi sensor.

- LIDAR (*Light Direction and Ranging*). El funcionament d'aquests sensors és el mateix que el dels Sonar, però en comptes d'utilitzar ones sonores fan servir feixos de llum làser.

2.1.1 Sistemes d'estereovisió

L'estereovisió [13] és un sistema de mesura de distàncies a partir de les imatges rebudes de dues càmeres, de les quals es coneix la posició i l'orientació. A partir d'això, obté la distància mitjançant el conegut teorema del sinus, com s'indica a la figura 2.1.

El triangle ABC de la figura 2.1 el formen les dues càmeres (punts A i B) i el punt vist per ambdues. A partir d'aquí i coneixent els angles $\hat{A}$ i $\hat{B}$ es poden calcular les distàncies del punt a cadascuna de les càmeres. La coneguda formulació del teorema del sinus és la

següent:

$$\frac{\sin(\hat{A})}{a} = \frac{\sin(\hat{B})}{b} = \frac{\sin(\hat{C})}{c} \tag{2.1}$$

El procés que segueix un sistema d'estereovisió consta dels següents passos:

1. Identificar un punt concret en la imatge d'una de les càmeres.
2. Identificar el mateix punt en la imatge de l'altra càmera.
3. Mesurar la posició horitzontal de cadascun dels punts respecte una referència comuna.
4. Calcular la profunditat del punt a partir de la disparitat dels punts, entenent com a disparitat, la diferència entre les posicions horitzontals del mateix punt en les dues imatges.

Els passos més complicats, i alhora més importants, d'aquest procés són el primer i el segon, però sobretot el segon ja que els altres dos són simples aplicacions de l'expressió 2.1. Aquest segon pas s'anomena també *matching*, ja que s'ha d'establir correspondència entre els punts d'una imatge amb els de l'altra. Per intentar trobar solució a aquest problema es sol fer només *matching* entre punts concrets, com poden ser cantonades o punts amb contorns molt marcats, però tot i així continua sent una tasca difícil, sobretot en alguns casos concrets com quan es produeixen oclusions, on l'associació es fa impossible perquè el punt només existeix en una de les imatges. Un altre dels principals problemes, són les reflexions de les imatges, ja sigui en vidres o miralls que poden donar lloc a càlculs totalment erronis de la profunditat.

Dins del camp de l'estereovisió hi ha alhora diferents tipus, dels que cal destacar sobretot la diferència entre la visió estereoscòpica i la llum estructurada. La diferència entre ambdues és que la primera utilitza la llum ambient i la segona utilitza una font de llum que permet tenir controlada la il·luminació, la qual és un altre dels grans obstacles en l'estereovisió.

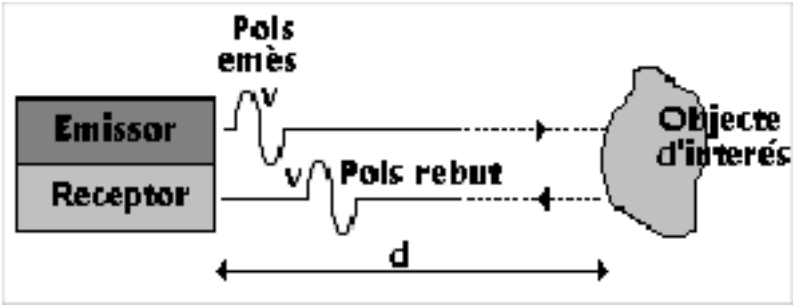

Figura 2.2: Recorregut de l'ona emesa pel *sonar*

2.1.2 Sensors d'ultrasons

Aquests sensors són coneguts més comunament amb el nom de *SONAR*, i formen part dels sistemes de mesura de distàncies catalogats com a sensors de *temps de vol*. Això és degut a que mesuren la distància a partir del temps que triga una ona en recórrer la distància entre el sensor i l'objecte, i la tornada fins al primer, com s'observa a la figura 2.2.

Si es coneix la velocitat de les ones emeses i el temps que triguen aquestes a retornar, es pot saber la distància a la que està l'obstacle aplicant la fórmula següent:

$$d = \nu \cdot \frac{t}{2} \tag{2.2}$$

on ν és, en el cas del *sonar*, la velocitat del so (340 m/s en el buit), ja que les ones emprades són ones sonores.

El principal avantatge d'aquests sensors respecte l'estereovisió és que no cal associar punts, i a més, l'error relatiu en la mesura és constant respecte la distància, mentre que en els sistemes d'estèreo l'error relatiu augmenta. Però per contra, té també una sèrie d'inconvenients com:

Variacions en la velocitat de propagació. La velocitat de les ones sonores depèn bastant de les condicions ambientals de pressió i temperatura. Si aquestes es prenen

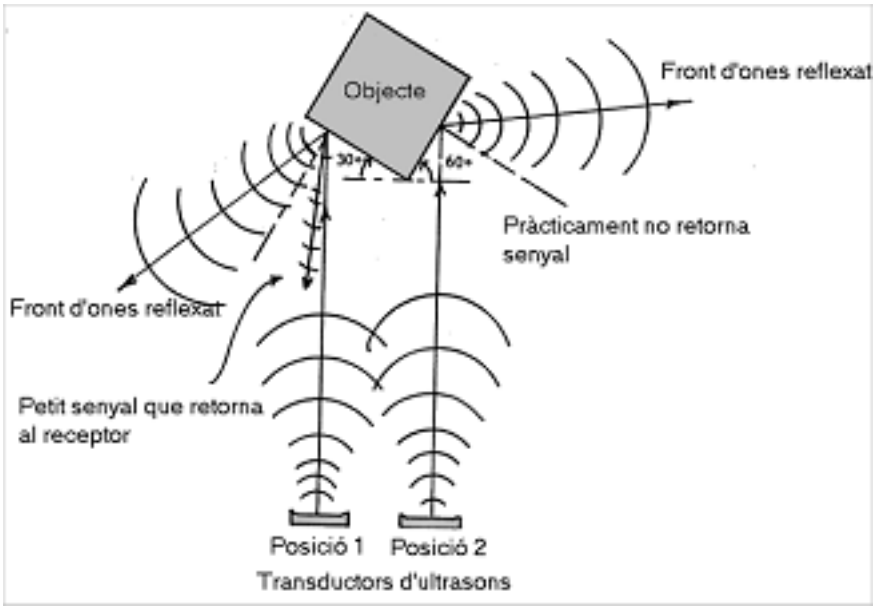

Figura 2.3: Reflexió de les ones sonores

com a una constant, l'error comès pot arribar a ser considerable. Per exemple, si la temperatura de l'ambient és 30ºC major que la de referència, l'error relatiu degut a aquest fet serà d'un 3% aproximadament.

Reflexió de les ones en l'obstacle. Si l'angle d'incidència de l'ona sobre l'obstacle amb el que xoca és major que un angle determinat, pot ser que la part d'ona reflectida no tingui prou potència per ser detectada pel receptor del sensor, amb la qual cosa aquest interpretarà que no hi ha obstacle en aquella direcció. Aquest efecte es pot observar a la figura 2.3

Reflexions múltiples. Algunes vegades, les ones reflectides en el cas anterior, poden tornar a xocar amb un altre obstacle i aleshores retornar cap al sensor, amb la qual cosa la mesura serà totalment errònia.

Imprecisions en la mesura del temps. Aquests són deguts a imprecisions del hardware que calcula el temps que triga l'ona en retornar al sensor. Òbviament, un error

en la mesura del temps té un efecte directe en l'error de la mesura de la distància, com es pot deduir de l'expressió 2.2.

Actualment, el *sonar* és l'element més utilitzat a l'hora de mesurar distàncies, degut bàsicament al seu preu relativament baix. Però la millora i l'abaratiment de l'electrònica, fan que cada cop siguin més els sistemes *sonar* que es vagin substituint per sensors làser.

2.1.3 Sensors làser

Els sensors làser tenen la mateixa base de funcionament que els sensors *sonar*, amb la única diferència que utilitzen ones làser en comptes d'ones sonores.

Això fa que l'electrònica necessària per aquests sensors sigui molt més precisa perquè la velocitat de les ones làser és la velocitat de la llum. Per tant, la variable ν de la fórmula 2.2 val 300.000 Km/s, i això vol dir que per obtenir resolucions de mil·límetres es necessita una circuiteria capaç de mesurar temps de l'ordre del picosegon (10^{-12} segons).

Aquest fet fa que aquest tipus de sensors siguin molt cars i només s'utilitzin en aplicacions on les prestacions que ofereix el *sonar* no siguin suficients.

Però malgrat aquesta diferència de preu, les millores dels sensors làser respecte dels *sonar* són considerables, bàsicament perquè els tres primers problemes esmentats en el cas del *sonar* no es presenten en el cas del sensor làser, i això és degut a les diferències de comportament de les fonts utilitzades, és a dir, ones sonores i electromagnètiques respectivament.

2.2 Construcció de mapes i localització

Els problemes de la construcció de mapes i de la localització [2, 3, 5, 7, 9], són dos problemes molt diferents que han estat molt estudiats i resolts amb bastant d'èxit de forma separada. En primer lloc cal explicar en què consisteix cadascun d'ells, i per això procedim a donar-ne la definició:

Construcció de mapes. Consisteix en crear un mapa de l'entorn del robot coneixent la posició absoluta d'aquest i informació de l'entorn proporcionada per algun sensor, ja sigui *sonar*, làser, sistema d'estereovisió o d'altres.

Localització. Té com a objectiu trobar la posició real del robot a partir de les dades de l'entorn que li proporcionen els sensors i el previ coneixement del mapa de la zona on el robot es troba.

Com ja s'ha dit, les solucions proposades per aquests problemes han obtingut bons resultats sempre que s'han tractat de forma separada [11, 6]. Però el cas que tracta aquest projecte, és el d'un robot totalment autònom i que per tant, no disposa d'informació prèvia sobre l'entorn, i només pot obtenir informació de la seva posició absoluta mitjançant la seva pròpia odometria, que com se sap, no és massa precisa.

Com sembla lògic, quan es va començar a plantejar el problema de la construcció de mapes i localització simultàniament, es pensava que tenint resolts els dos problemes per separat, només caldria fusionar-los i el problema estaria resolt. Però ràpidament es va veure que això no era possible ja que la construcció de mapes i la localització, tractats de forma separada, parteixen d'informació exacta i sense error sobre la posició del robot i l'entorn respectivament, mentre que quan es tracten de forma simultània, la informació sobre la posició del robot i l'entorn són només estimacions. I com en la majoria de casos on es tracta amb estimacions, aquestes van acompanyades d'un tractament probabilístic i estadístic per a poder quantificar d'alguna manera l'error màxim que es comet acceptant les estimacions com a valors reals.

De tot això es dedueix que cal plantejar nous mètodes per aconseguir tractar aquests dos problemes simultàniament. Per tant, a partir d'ara es parlarà d'aquest problema com a un de sol i se l'anomenarà **CML** (*Concurrent Mapping & Localisation*), que és el nom més utilitzat en el camp de la robòtica mòbil. També sol rebre el nom de **SLAM** (*Simultaneous Localisation And Mapping*).

Quan s'intenta trobar un mètode que resolgui el CML, es topa amb una sèrie de dificultats que s'han d'afrontar i tenir molt en compte, ja que són les causes principals que fan del CML un problema de difícil resolució. Aquestes són:

- **No es disposa de referències globals.** El robot parteix d'una posició inicial, i a partir d'aquí només disposa dels propis sensors imperfectes per a poder conèixer la seva posició absoluta en un moment donat. No hi ha cap referència global com podria ser, en el cas dels sistemes de navegació GPS, un satèl·lit. Es vol que el robot sigui totalment autònom, i per tant, ha de ser capaç de saber la seva posició mitjançant els medis dels que disposi.

- **L'error de "*Dead Reckoning*".** Quan el robot es mou per primera vegada, els sensors de moviment del robot cometen un cert error. I a mesura que aquest continua movent-se, si no hi ha cap mecanisme que ho corregeixi, l'error que cometen els sensors es va acumulant i al cap d'un cert període de temps, la posició real del robot i la que donen els sensors no tenen res a veure. També es diu que el robot està a la *deriva*. Aquest fenomen és especialment crític en la mesura de l'orientació, ja que un petit error en aquesta mesura pot donar lloc a grans errors en la posició. I precisament, en la majoria dels casos, l'orientació és la mesura que conté més error ja que els sensors del robot habitualment calculen el desplaçament en els eixos, mentre que el gir es calcula a partir d'aquestes dades, ja de per si amb un cert error.

- **Imprecisió dels sensors utilitzats per extreure informació de l'entorn.** Tots els sensors cometen errors en les seves mesures. En el cas que afecta aquest projecte, el valor de l'error depèn del tipus de sensor que utilitzem. Com ja s'ha comentat, els sonars poden tenir lectures provinents d'ones afectades pel fenomen de la reflexió múltiple, i en el cas de sistemes d'estereovisió es poden donar diferents resultats segons la il·luminació. A més, en el cas que tot funcioni correctament, sempre apareix un error que augmenta amb la distància.

- **Associació de les dades.** Les dades que es reben dels sensors, s'han d'intentar associar amb les del mapa ja existent. Aquesta tasca, no sol ser gens senzilla i en alguns casos es poden produir associacions errònies. Això s'ha d'intentar evitar al màxim ja que pot suposar una correcció de la posició del robot que l'allunyi de la posició real.

- **Canvis en l'entorn.** Sovint els possibles obstacles que hi poden haver en una habitació, estan en moviment. Això dificulta enormement la tasca de fer mapes i alhora la de la localització del robot ja que cada vegada s'haurà de tenir en compte que el mapa ja construït, pot haver canviat. També dificulta enormement la tasca d'associació de dades, ja que quan el robot *veu* un obstacle en una posició diferent de la que estava, no es pot saber si és perquè l'obstacle realment s'ha desplaçat o perquè els sensors de moviment del robot han comès un error en la mesura del desplaçament del robot.

- **Oclusions.** Són un fenomen molt difícil de manegar ja que en molts casos és molt difícil distingir entre la desaparició d'un obstacle o la seva oclusió darrere d'algun altre obstacle. En entorns dinàmics la dificultat de tractament del problema augmenta en gran mesura.

Per intentar salvar totes aquestes dificultats s'han creat dues vies de desenvolupament que enfoquen de forma diferent la possible resolució del CML. A partir d'aquests dos enfocaments s'han dissenyat una sèrie d'algoritmes i mètodes que parteixen d'alguna de les seves bases. Aquests dos enfocaments són els següents:

- **Mètodes de malla.** Aquests mètodes, també anomenats *grid*, divideixen l'espai en una malla i d'aquesta manera es creen subdivisions o parcel·les totalment independents entre si com s'observa a la figura 2.4.

  A cada parcel·la se li assigna una probabilitat de que estigui o no ocupada, i aquesta probabilitat es va actualitzant a mesura que es van rebent dades del sensor i

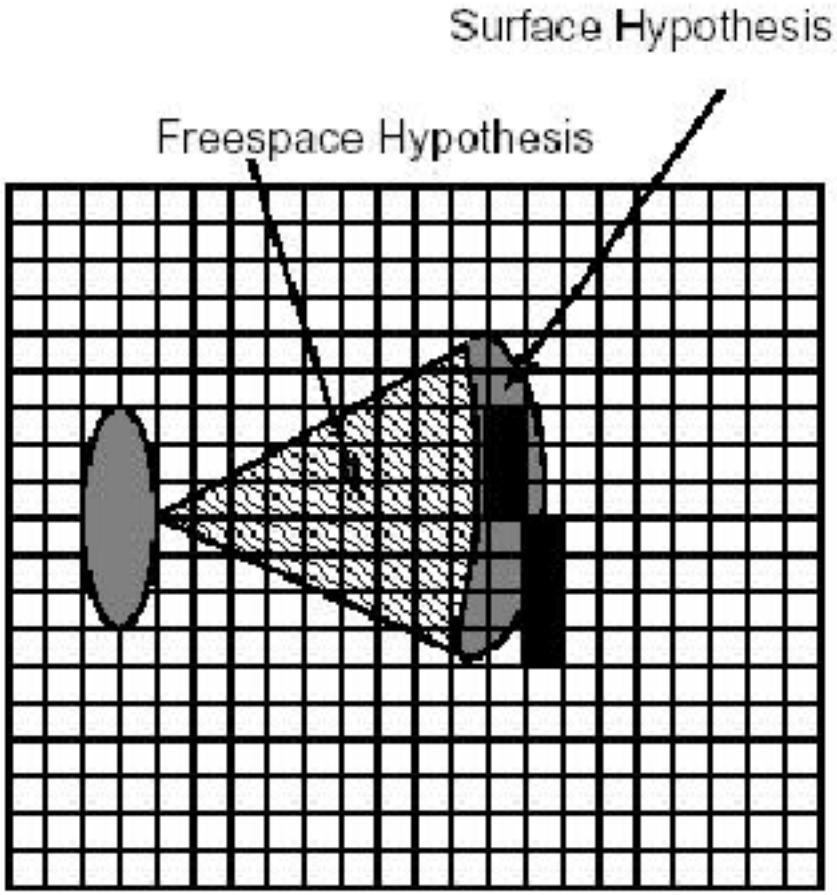

Figura 2.4: Divisió de l'espai en parcel·les en un mètode de malla

d'aquesta manera es va construint progressivament el mapa.

- **Mètodes basats en *landmarks*.** Aquests mètodes, contràriament als anteriors, no divideixen l'espai en parcel·les, sinó que el que fan és extreure de les dades del sensor el que habitualment s'anomena *landmark*, i els van col·locant dins el mapa, actualitzant-los o introduint-ne de nous segons les dades que arribin dels sensors. Cal comentar que a partir d'aquest punt, en aquest document s'utilitzarà molt freqüentment el terme *landmark*, i per tant, serà de vital importància conèixer la seva definició.

Un *landmark* és una característica o part de l'entorn fàcilment identificable i diferenciable de la resta. Habitualment s'utilitzen cantonades, parets, o en el cas d'utilitzar càmeres com a sensor, objectes amb colors ben diferenciats de l'entorn o amb contorns molt marcats.

S'utilitzi la filosofia que s'utilitzi de les anteriorment comentades, cal seguir els mateixos passos, i són els següents:

1. **Recepció de les dades del sensor.** En primer lloc, s'ha d'interpretar la informació que dóna el sensor sobre l'entorn, i en cas que sigui necessari, analitzar-la i extreure'n la informació necessària.

2. **Associació de dades.** En aquesta fase, s'ha d'associar la informació extreta en el punt anterior amb la ja existent en el mapa.

3. **Correcció de la posició del robot.** A partir de les associacions, cal fer una estimació de la posició real del robot i corregir-la, ja que la que es tenia fins aquest moment venia únicament donada pels sensors de moviment i la odometria del robot que, com ja s'ha comentat, cometen un cert error.

4. **Actualització del mapa.** Un cop s'ha corregit la posició del robot, ja s'està en condicions d'actualitzar el mapa existent, modificant-lo i ampliant-lo segons sigui convenient.

2.2.1 Mètodes de malla o *grid*

Aquests mètodes [14] discretitzen l'espai que rodeja el robot, dividint-lo en cel·les d'igual tamany, tal i com mostrava la figura 2.4. A cada cel·la de la malla, representada pel vector (x, y), se li assigna un valor que mesura la suposició subjectiva del robot sobre el fet que aquella cel·la estigui ocupada. A aquest valor se l'anomena probabilitat d'ocupació, i més concretament, significa la probabilitat de que el centre del robot pugui situar-se físicament

sobre el centre d'aquella cel·la. Un valor proper a 0 significa que el robot pot situar-se sobre la cel·la, ja que aquesta segurament està desocupada, mentre que un valor proper a 1 significa tot el contrari. Aquestes probabilitats es van actualitzant a mesura que es reben noves dades dels sensors.

El primer que cal fer en aquests mètodes és trobar la posició del robot. Això es fa comparant directament la forma del contorn de punts rebut del sensor amb el mapa que s'ha construit fins al moment, i d'aquesta manera, es pot trobar la posició on el robot té més possibilitats d'estar situat.

Un cop fet aquest pas, ja es pot procedir a actualitzar els valors de les probabilitats d'ocupació de les cel·les, augmentant-los si en aquella iteració el sensor hi ha detectat un obstacle, o disminuint-los en cas contrari.

Els algoritmes de resolució del CML que segueixen aquesta filosofia, solen basar-se en tècniques probabilístiques derivades del teorema de Bayes, i tenen el gran inconvenient que a mesura que el mapa creix en extensió, i per tant, augmenta el nombre de cel·les, l'algoritme es fa cada cop més lent degut a l'alt cost computacional. L'altre inconvenient que presenta, és provocat per la discretització de l'espai, ja que només es pot conèixer la probabilitat d'ocupació de cada cel·la globalment, no de qualsevol punt de l'espai. Tot i això, si el tamany de les cel·les és prou petit, l'impacte d'aquest segon inconvenient és pràcticament nul. Però alhora, com més petit sigui el tamany de cel·la, major nombre d'aquestes hi haurà per una mateixa extensió del mapa, i per tant, l'algoritme trigarà més temps. Per aquest motiu, cal trobar una solució de compromís entre ambdós factors.

A la figura 2.5 es pot observar un exemple del tipus de mapa que resulta d'aplicar un mètode d'aquest tipus.

2.2.2 Mètodes basats en *landmarks*

El model de mapa resultant d'aquests mètodes no és, com en el cas anterior, una malla de probabilitats sinó que l'objectiu d'aquests mètodes és construir un mapa de *landamarks*.

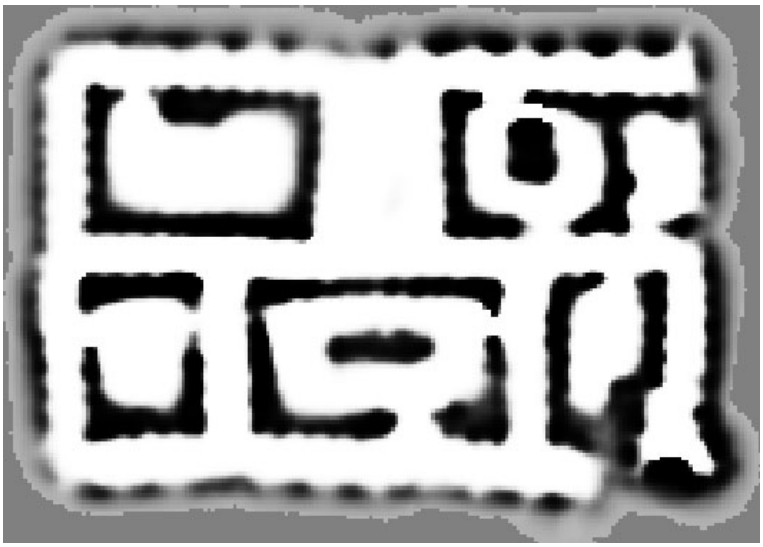

Figura 2.5: Exemple de mapa aplicant un mètode de malla obtingut a la *School of Computer Science, Pittsburgh*. Les zones més fosques són les que tenen probabilitats d'ocupació més elevades i viceversa.

En el cas que s'utilitzi un sensor làser o sonar, aquests *landamarks* solen ser parets o cantonades, i en el cas que s'utilitzin càmeres, solen ser objectes molt diferenciats de la resta de l'entorn, ja sigui perquè són d'un color molt diferent o tenen un contorn molt definit. A l'igual que els mètodes de malla, també hi ha un tractament estadístic del mapa, ja que habitualment, tant la posició del robot com la de cada *landmark* té associada una determinada zona d'incertesa.

La informació que dónen els sensors, ja siguin contorns en el cas del làser, o imatges en el cas de càmeres, no és útil directament, sinó que necessita ser tractada prèviament per a extreure els *landamarks* detectats. Per tant, és com si s'utilitzés un sensor de *landamarks* que acumula l'error propi dels sensors utilitzats i a més, el que pot afegir l'algoritme escollit per a l'extracció de *landmarks*.

Un cop s'han extret els *landamrks* detectats, es poden començar a comparar amb els ja existents en el mapa. Aquest procés d'associació de dades és força semblant al problema

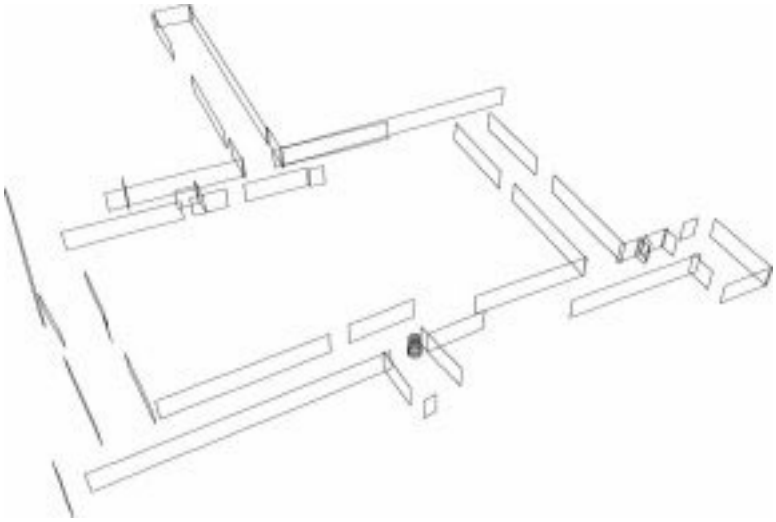

Figura 2.6: Exemple de mapa obtingut mitjançant un mètode basat en landmarks, al *Department of Ocean Engineering at Massachusetts Institute of Technology*

conegut amb el nom de *matching*, tot i que les oclusions sovint fan que aquest tractament es compliqui i s'hagi de rocórrer a quelcom més sofisticat que les tradicionals tècniques de *matching*.

Posteriorment, cal aplicar algun mètode que a partir de les associacions realitzades, pugui estimar la posició del robot i determinar la zona d'incertesa on aquest es troba, amb un cert nivell de confiança, i alhora, refinar el mapa a mesura que es van fent observacions de l'entorn, estimant les posicions dels *landmarks*, reduint la seva zona d'incertesa, i si es dóna el cas, afegir nous *landmarks* detectats. La eina més utilitzada per a fer aquesta tasca és el filtre de Kalman [1, 4, 9] que s'explicarà amb detall en el capítol 5.

El principal inconvenient d'aquests mètodes és que a mesura que augmenta el nombre de *landmarks*, el cost computacional creix excessivament ja que solen ser algoritmes d'ordre $O(n^2)$ o superior.

Un exemple de mapa obtingut amb aquest tipus de mètodes és el de la figura 2.6. Es pot observar clarament la diferència dels mapes obtinguts entre els dos mètodes.

Capítol 3

Hardware i software

3.1 Descripció general

Els components de hardware bàsics que s'han utilitzat per implementar els algoritmes desenvolupats són:

- Robot *Marco*, pertanyent a l'Institut de Robòtica i Informàtica Industrial (IRI).
- Un sensor de profunditat làser, model *Rotoscan RS4* de la companyia *Leuze Lumiflex*.
- Ordinador Pentium II, 300 Mhz, 128 MB RAM.
- Transmissors de ràdio Ethernet *Air EZY 2400*, de la companyia *OTC Telecom*.

Els productes de software que s'han utilitzat són:

- *Microsoft Visual C++ 6.0*: Usat per construir la interfície gràfica de l'usuari i programar el mètode proposat.
- Interfície de navegació RoboGUI, dissenyada per a robots mòbils Activmedia Pioneer, conjuntament entre l'IRI i l'IIIA.

- *Saphira ver 6.2*: Programa distribuït amb el robot amb múltiples funcions. En aquest projecte s'han utilitzat les seves llibreries per a conèixer la posició absoluta del robot segons les dades dels encòders, i també per a poder moure el robot.

- Llibreries gràfiques *Open GL*

3.2 Descripció del Hardware

3.2.1 Robot *Marco*

Marco és un robot mòbil dissenyat a l'Institut de Robòtica i Informàtica Industrial. Està format per una base comercial de la marca Activmedia, model Pioneer 2 DX, un tronc cilíndric i dues càmeres sostingudes per un sistema de Pan & Tilt que permeten llur movilitat, com s'observa en la figura 3.1

La base comercial Pioneer 2DX té una mobilitat que ve donada per dues rodes motrius i una tercera que marca la direcció. Cadascuna disposa dels seus encòders digitals per mesurar la distància recorreguda pel robot. A més, també disposa d'un PC Pentium II a 300 MHz, 16 sensors d'ultrasons *Polaroid* disposats al voltant de la base per detectar obstacles , i de 10 *bumpers* que serveixen de paraxocs i es poden utilitzar per a parar els motors del robot en cas de col·lisió.

3.2.2 Sensor làser de profunditat *Leuze Lumiflex Rotoscan RS4*

La figura 3.2 mostra el sensor làser utilitzat per desenvolupar aquest projecte. El sensor és de tipus òptic i fa mesures de distància bidimensionals, és a dir, en un sol pla.

En el seu interior hi ha un deflector rotatiu que envia periòdicament pulsos de llum dins de l'epai de treball del sensor, que és de 190º. Quan aquests pulsos xoquen contra un obstacle, el sensor capta la llum reflectida i mitjançant la seva electrònica calcula la distància de l'obstacle al sensor, a partir del temps de viatge, o de *vol*, de la llum reflectida.

Figura 3.1: Robot *Marco*

A la figura 3.3 es pot veure el principi de funcionament del sensor, així com la composició del deflector. Aquest està format a partir d'un díode làser i un sistema òptic. El primer és l'encarregat de generar els polsos del làser, mentre que el sistema òptic, que està format per tres miralls, un d'ells rotatiu, s'encarrega de distribuir aquests polsos al llarg dels 190º que pot escombrar el làser, en un temps de 40 ms.

Amb aquetes característiques descrites, el sensor és capaç de detectar obstacles fins a 65 metres, però a partir dels 15 m, l'error relatiu comès en la mesura del sensor deixa de mantenir-se constant i passa a augmentar amb la distància, cosa que fa menys creïble la informació del sensor a partir d'aquesta distància.

Per altra banda, l'angle mínim entre dues mesures consecutives al llarg dels 190º és de 0.36º, cosa que permet obtenir 529 mesures del pla de treball en un sol escombrat. Alhora, això implica també que per exemple, als 15 metres sigui capaç de detectar objectes de tan

Figura 3.2: Sensor làser Leuze Lumiflex Rotoscan RS4

sols 5 cm d'amplada.

Tot el que s'ha dit fins ara fa que aquest sensor reuneixi les condicions suficients per a poder ser utilitzat per a la construcció de mapes amb robots mòbils. No obstant, el fet que només prengui mesures bidimensionalment, fa que només es pugui obtenir informació de l'entorn en un sol pla. En la majoria de casos això és suficient, ja que si per exemple, s'intenten detectar parets, la informació obtinguda en un pla és fàcilment extrapolable a qualsevol altre pla. Però hi ha alguns casos concrets, com poden ser les taules, en els que ens podem trobar amb el problema que el sensor només sigui capaç de detectar les potes i no trobi cap obstacle enmig, amb la qual cosa es poden produir col·lisions que poden causar danys al robot.

L'altre inconvenient, tot i ser molt més irrellevant, és la zona de treball de 190º. Tot i ser bastant ampla, no permet tenir cap informació sobre el que hi ha als 170º restants, cosa que fa que el robot no es pugui moure cap enrere amb la seguretat de que no xocarà, i a més, es perden una sèrie de referències que en molts casos podrien ajudar bastant en la resolució del CML. Una solució fàcil per a aquest problema, és posar dos sensors en sentits oposats, però això exigiria una inversió encara més elevada que caldria valorar si val la pena.

A l'actualitat, aquests sensors es comercialitzen i s'utilitzen industrialment únicament

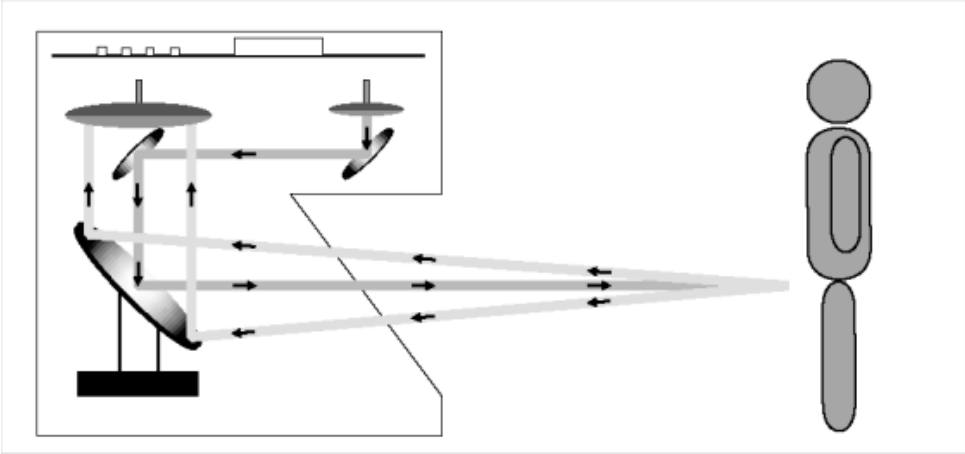

Figura 3.3: Principi de funcionament del Rotoscan RS4

amb la finalitat de proporcionar seguretat. Per exemple, una aplicació molt típica la podem trobar en els AGVs (*Automatic Guided Vehicles*). Aquests no utilitzen el sensor per a navegar, ja que la trajectòria a recórrer ja la tenen prefixada, sino només per detectar si s'interposa quelcom en el seu trajecte i evitar d'aquesta manera col·lisions, per exemple, amb un operari, que podrien comportar conseqüències greus.

3.2.3 Transmissors de ràdio Ethernet

L'aparell mostrat a la figura 3.4 serveix per establir una connexió de xarxa Ethernet entre dos o més ordinadors. En definitiva, és com si les targetes de xarxa de cada ordinador estessin connectades mitjançant cable, però sense la necessitat d'aquest. Òbviament, cada ordinador connectat ha de disposar d'un transmissor.

En el cas d'un robot mòbil, l'absència de cables és essencial, ja que l'objectiu és la total autonomia d'aquest, i per tant, qualsevol connexió amb un ordinador extern ha d'establir-se mitjançant la tecnologia anomenada comunament *Wireless*.

En aquest apartat es parlarà només de les característiques d'aquests dispositius i més endavant s'explicarà la seva utilitat dins el projecte.

Figura 3.4: Transmissor de ràdio Ethernet OTC Telecom Air EZY 2400

Les característiques tècniques principals són les mostrades a la taula 3.1.

Dimensions	120 x 60 x 27 mm (sense antena)
Longitud antena	57 mm
Velocitat màxima de transmissió	2 Mbps
Distància màxima de transmissió sense obstacles	150 m
Distància màxima de transmissió en entorn d'oficines	De 40 a 100 m
Nombre màxim de canals simultanis	3
Freqüència de les ones de transmissió	2.4 GHz
Alimentació	5 Volts

Taula 3.1: Especificacions tècniques dels transmissors de ràdio Ethernet

3.3 Descripció del Software

3.3.1 Saphira 6.2

Saphira és un entorn dissenyat específicament per a les bases de robot Pioneer, i desenvolupat al *SRI International's Artificial Intelligence Center*. Aquest software s'entrega juntament amb la base comercial, i consisteix en una aplicació per comunicar i donar ordres al robot, i també en unes llibreries per a realitzar aquestes operacions. En aquest projecte només s'utilitzaran les llibreries, ja que són les que permetran que l'aplicació creada en aquest projecte, pugui interaccionar amb el robot.

Les llibreries del Saphira contenen múltiples funcions per a aplicacions programades en C++, que permeten a l'usuari realitzar diferents operacions amb el robot de forma molt senzilla. Les utilitzades en aquest projecte són:

- **Connexió.** Connecta un PC extern amb el robot . És el pas previ a qualsevol de les altres operacions.

- **Encesa dels motors.** Prepara els motors per actuar quan arribin comandes de moviment.

- **Moure el robot.** Permeten desplaçar el robot una certa distància o fer-lo girar sobre el seu propi eix de rotació. També hi ha funcions de més alt nivell que permeten posicionar el robot en el punt que l'usuari desitgi.

- **Obtenir informació dels sensors.** Aquestes funcions dónen la informació dels encòders, dels sonars o dels bumpers. Com que el Saphira disposa del model dinàmic del robot, també hi ha funcions que extreuen la posició i orientació del robot, en coordenades globals, a partir de la succeció de lectures dels encòders.

Totes aquestes rutines faciliten molt la tasca del programador, ja que eviten haver de tractar amb el hardware directament. Per exemple, donar una ordre de moviment al

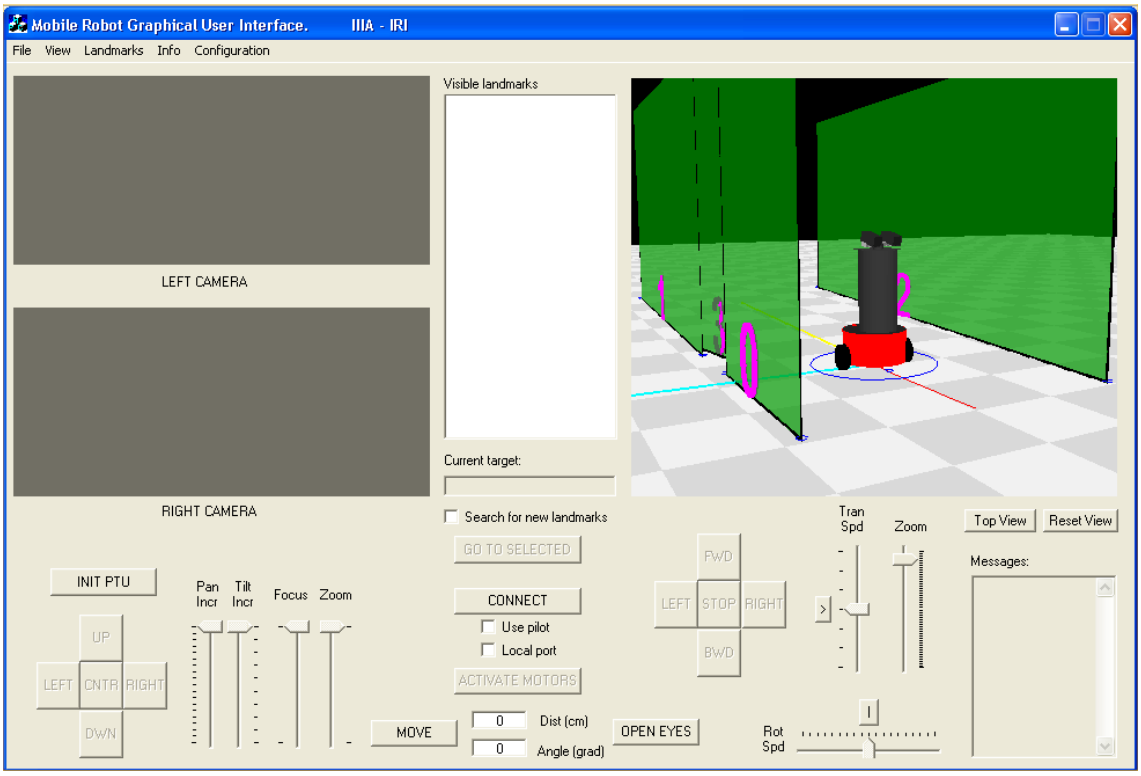

Figura 3.5: Interfície de navegació RoboGUI

robot implicaria enviar senyals directament als motors, amb la qual cosa caldria molt més codi i conèixer molt més a fons el hardware per a poder fer el que es pot fer en una línia, gràcies a aquestes llibreries.

3.3.2 Interfície de navegació RoboGUI

Aquesta interfície ha estat dissenyada conjuntament entre l'IRI i l'IIIA, pertanyents a la UPC, per a diferents tipus de plataformes mòbils, entre les que s'inclouen les bases comercials Activmedia Pioneer 2, per tal de poder aprofitar al màxim les possibilitats que el robot permet. La pantalla principal d'aquest programa es pot observar a la figura 3.5.

El principal objectiu d'aquesta interfície és poder intercanviar i visualitzar informació del robot i dels seus sensors, ja siguin càmeres, sonars, làsers o encòders. Les principals

funcions que disposa el programa són:

1. **Visualitzar la posició del robot i el seu entorn.** La interfície disposa d'una representació en 3 dimensions del robot i el seu entorn, en el que es poden dibuixar els *landmarks* que el robot vagi detectant. No obstant, el programa no disposa de cap mètode de detecció de landmarks.

2. **Moure el robot.** Hi ha quatre botons que permeten desplaçar el robot endavant i enrere, i fer-lo girar cap a l'esquerra i cap a la dreta. La velocitat d'execució d'aquests dos tipus de moviments és constant al llarg d'un moviment en concret però pot ser modificada.

3. **Visualitzar el vídeo de les càmeres en temps real.** Per aquest projecte, aquesta opció ha estat inhabilitada, ja que l'ús d'aquest tipus de sensors no entra dins l'abast d'aquest projecte. La dificultat de visualitzar imatges procedents de les càmeres és abordada en altres projectes.

4. **Control·lar la posició del Pan & Tilt.** El Pan & Tilt és el suport mòbil de les càmeres, i movent-lo s'aconsegueix que les càmeres puguin moure's i enfocar diferents zones. Tampoc és necessari el seu ús en aquest projecte.

Per aprofitar aquesta sèrie d'avantatges, els mètodes utilitzats en aquest projecte per a poder fer mapes de l'entorn s'implementaran a partir del codi d'aquest programa. D'aquesta manera es podran visualitzar els mapes resultants en 3D.

3.3.3 Llibreries *Open GL*

Són una sèrie de llibreries gràfiques desenvolupades en comú entre unes quantes de les companyies informàtiques més importants, com per exemple *Microsoft*. El seu objectiu és posar a disposició dels programadors una sèrie de comandes que serveixin per crear

entorns tridimensionals i que facilitin enormement les tasques de dibuixar poligonalment en 3D. Aquestes llibreries han estat desnvolupades per als llenguatges C i C++.

Permeten fer una gran quantitat d'operacions, que es poden resumir en les següents:

- Crear entorns tridimensionals en un diàleg de Windows i mostrar-los per patalla. Dins d'aquests entorns, que habitualment tenen forma cúbica, és on posteriorment es fan la resta d'operacions. Per a poder mostrar els entorns per la pantalla, les llibreries d'*Open GL* han de transformar aquest entorn tridimensional, en una representació d'aquest en dues dimensions.

- Introduir cossos amb formes geomètriques definides dins d'entorns 3D, des de punts o línies fins a cilindres o cons. Tots ells poden ser de diferents tamanys, colors, grau de transparència, etc., segons la voluntat de l'usuari.

- Visualitzar aquest entorn des de diferents punts, amb diferens camps de visió i també diferents tipus de perspectives, que poden arribar a diferir molt de la de l'ull humà. Els paràmetres aportats en aquestes operacions són els necessaris per passar l'entorn 3D creat, a una determinada representació d'aquest en 2D. Alhora, aquestes funcions també permeten fer operacions com les rotacions, ja que només són canvis del punt des d'on s'observa l'escena.

- Introduir focus de llum de intenstitat configurable, en els punts desitjats. Aquestes operacions creen ombres i diferents graus d'il·luminació en l'escena, que depenen fonamentalment del tipus de superfície sobre el que incideix la llum, que també es pot configurar.

Com es pot comprovar són una eina molt útil, i per això no és d'extranyar que sigui l'eina més utilitzada en l'actualitat pels programadors d'aplicacions gràfiques en 3D.

Figura 3.6: Disposició del hardware i del software

3.4 Disposició del hardware i software

El sistema requerit per dur a terme aquest projecte està dividit bàsicament en dues parts. La primera, és el robot mòbil i la segona és l'ordinador de sobretaula. Tots els elements descrits en aquest capítol estan distribuits entre aquestes dues parts.

L'objectiu, com ja s'ha comentat és poder manipular el robot i veure el mapa de l'entorn que es va generant des de l'ordinador de sobretaula, sense que quedi limitada la mobilitat del robot. Això obliga a establir una comunicació entre robot i PC sense cap cable, ja que de no ser així, la mobilitat del robot estaria limitada a la longitud del cable.

La disposició del software i hardware és el que es pot observar a la figura 3.6.

L'ordinador de sobretaula és el que realitza totes les tasques de visualització i de càlcul dels algoritmes de construcció de mapes. Però per a fer-ho necessita les dades de posició del robot i el contorn mesurat pel làser. Quan l'algoritme de construcció de

mapes necessita aquesta informació, envia al robot un senyal i el robot comença a llegir el contorn del làser o la seva posició i ho envia a l'ordinador de sobretaula. Una cosa semblant succeeix amb les comandes de moviment, ja que si l'algoritme de construcció de mapes vol moure el robot, només ha d'enviar-li la comanda i el robot executa el moviment demanat.

Tota aquesta transmissió d'informació es realitza mitjançant un socket amb el sistema de comunicació client/servidor. L'ordiandor de sobretaula té el client, que envia peticions al servidor, allotjat a l'ordinador del robot, i aquest servidor respon retornant la informació sol·licitada o realitzant la tasca encomenada.

Capítol 4

Detecció de landmarks

A l'apartat 2.2 s'explica que hi ha bàsicament dos tipus de mètodes per a la construciió de mapes: els basats en malles i els basats en landmarks. Per a desenvolupar el mètode presentat en aquest document, s'ha escollit el segon tipus, és a dir, el que concep un mapa com una sèrie de landmarks distribuïts en l'espai. Com s'explica al citat apartat, un landmark es pot definir com una característica o part de l'entorn fàcilment identificable i diferenciable de la resta.

Per aquest motiu és de vital importància dissenyar un algoritme que extregui landmarks a partir de les dades proporcionades pels sensors, quen en el cas que ens ocupa, es tracta d'un contorn donat per un sensor làser de profunditat.

En aquest capítol s'explica el mètode utilitzat en aquest projecte per a detectar landmarks. En aquest cas concret, els landmarks que es buscaran són parets. En realitat, el que es busca són línies rectes estàtiques i d'una longitud considerable, que en la major part dels casos seran parets físiques reals, i en altres casos seran altres obstacles que a efectes de construir un mapa podran ser considerats com a tal. En altres paraules, es busca més aviat el contorn de l'habitació que no pas objectes concrets que hi puguin haver en un moment determinat.

El principal motiu d'aquesta elecció és la gran quantitat de formulació matemàtica i

estadística existent per a les rectes. Això fa que es disposi d'una sèrie d'eines força potents que poden ser molt útils per al tractament posterior dels landmarks, un cop detectats.

Un altre factor important, són les dades que arriben del sensor. Com s'ha comentat a l'apartat 3.2.2, aquestes són un contorn de punts en un arc de 190º. És evident que amb aquestes dades, els únics landmarks que es poden buscar són formes geomètriques conegudes, i d'aquestes, la més fàcil i ràpida de trobar és la línia recta. En el cas que el sensor que proporcionés la informació de l'entorn fossin càmeres, llavors s'hauria d'optar per altres tipus de landmarks com objectes amb colors molt vius, amb gradients de color elevats, etc.

I per últim, en el cas que es vulgui fer un posterior reconeixement d'habitacions per tal que el robot sàpiga a on es troba, és molt més fàcil fer-lo a partir del contorn que no pas a partir d'objectes concrets, molt més fàcilment variables que no pas les parets.

El procés d'extracció de parets està dividit en dos passos:

1. **Transformació de contorn de punts a contorn poligonal**. En primer lloc, un algoritme agrupa els punts en rectes de forma que el contorn passa a ser un polígon com mostra la figura 4.1. La mínima agrupació es produeix quan una recta només està formada per dos punts, com és el cas de les rectes 2 i 4 de la figura.

2. **Determinar les rectes que es poden considerar com a parets**. S'han d'establir una sèrie de criteris per saber quines de les rectes en les que s'ha dividit el contorn, són landmarks vàlids pel robot per tal de localitzar-se. Per exemple, les rectes 2 i 4 no poden ser considerades com a parets, ja que no hi ha lectures en aquella zona i per tant, no es pot saber què hi ha.

A continuació s'explicarà amb més detall cadascun d'aquests dos passos.

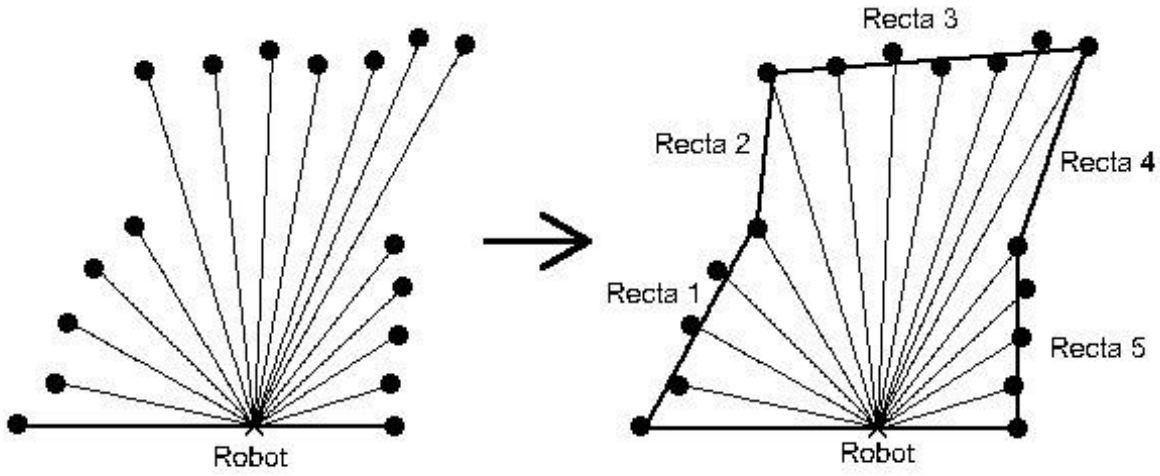

Figura 4.1: Pas de contorn de punts a contorn poligonal

4.1 Agrupació de punts en línies rectes

Per a dur a terme aquesta tasca s'ha escollit un conegut algoritme anomenat *IEPF (Iterative End-Point Fit)*. Està basat en l'estructura anomenada *Split & Merge*, que consisteix en partir el problema en trossos més petits fins que es compleix una certa condició, i posteriorment intenta ajuntar alguns d'aquests trossos, en cas que sigui convenient. Quan es vegi l'estructura de l'algoritme es veurà clarament el perquè d'aquesta nomenclatura.

Aquest algoritme va ser diessenyat originalment per Douglas i Peucker [10], amb l'objectiu de comprimir dades amb soroll. D'aquesta manera, es podien guardar una sèrie de punts alineats com un segment, amb el conseqüent estalvi de memòria. Posteriorment, Hershberger i Snoeyink [12] el van millorar fins a presentar la versió que s'utilitza en aquest projecte.

Les principals característiques d'aquest algoritme són la recursivitat i que és de tipus heurístic. La primera propietat permet que l'algoritme tingui un ordre logarítmic ($O(n\ log$

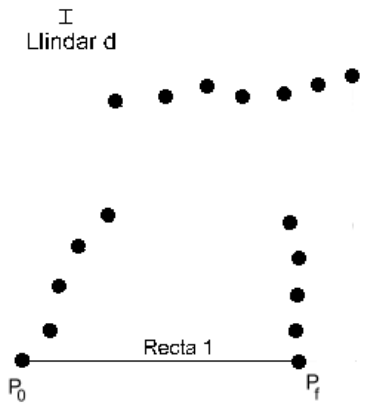

Figura 4.2: Primer pas de l'IEPF: agrupació de tots els punts en una recta

n)), cosa que permet que dóni la solució molt ràpidament. Pel que fa a la segona propietat, com ja se sap, implica que la solució donada no és l'òptima sino una aproximació a aquesta. La proximitat de la solució donada per l'*IEPF* i l'òptima dependrà de cada cas concret, així com de l'ajust dels parèmtres de l'algoritme.

Hi ha diversos algoritmes que desenvolupen la mateixa tasca. Alguns d'ells agrupen els punts en rectes de forma que l'ajust de les regressions sigui màxim, i en aquest sentit, òptim. Per això, aquests algoritmes habitualment dónen resultats millors que l'*IEPF*, però tenen un cost computacional molt més elevat, cosa que els fa inservibles per a aquesta aplicació concreta.

El problema a resoldre es podria definir de la següent manera: Donada una sèrie de n punts que formen un polígon P de n costats, trobar la millor aproximació a P que tingui el menor nombre de costats possible.

Per resolde'l el mètode seguiex els següents passos:

1. **Agrupar tots els punts en una sola recta**. La recta inclou els n punts però només passa pel primer, p_0, i per l'últim, p_f. Un exemple es pot veure a la figura 4.2

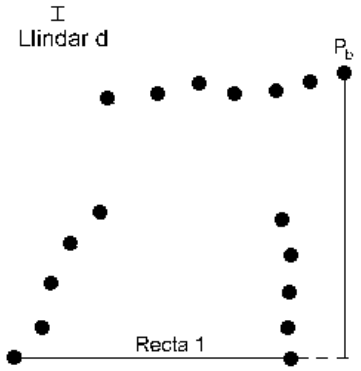

Figura 4.3: Segon pas de l'IEPF: buscar el punt més llunyà a la recta

2. **Buscar el punt més llunyà a la recta**. S'han de calcular les distàncies a la recta de cadascun dels punts continguts en ella. Al final ens quedem només amb el punt més llunya p_b. Aquest pas es mostra a la figura 4.3

3. **Partir la recta**. Havent fixat prèviament una distància llindar d_s, l'algoritme comprova si la distància a la recta del punt més llunyà és major que d_s. En aquest cas, la recta inicial es parteix en dos, una que va des de p_0 fins a p_b, i una altra que va des de p_b fins a p_f, com es pot observar a la figura 4.4. En cas contrari, l'algoritme acaba.

En el cas que s'hagi partit la recta, l'algoritme torna a començar des del pas 1 per a cadascuna de les rectes resultants de la partició. Al final, quan tot l'algoritme acaba, el resultat és un conjunt de rectes que formen un polígon, on els punts continguts a cada recta, no estan mai a una distància d'ella superior a d_s.

Fins aquí arriba la part de *Split* de l'algoritme, és a dir, anar partint progressivament la recta inicial en altres rectes fins que compleixin la condició del llindar de distància. A partir d'aquí comença la part de *Merge*, l'objectiu de la qual és unir rectes reals que l'algoritme hagi partit. Per fer-ho, uneix dues rectes consecutives i comprova la condició

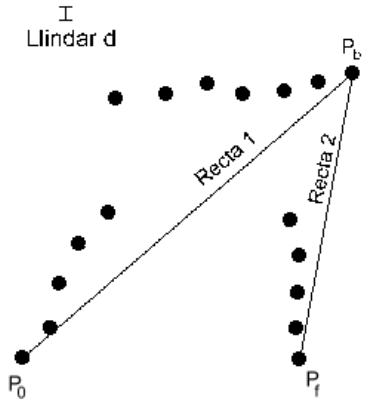

Figura 4.4: Tercer pas de l'IEPF: partir la recta

del llindar de distància com s'explica en el pas 2. Si la distància màxima és inferior a d_s, uneix les dues rectes, i en cas contrari, les deixa tal com estan. Aquest procés es realitza per cada parell de rectes consecutives. Amb aquest procés, s'aconsegueix pal·liar en gran mesura els pocs errors que comet l'algoritme de *Split*.

A la figura 4.5, es pot veure la continuació de l'algoritme IEPF per l'exemple mostrat a les figures 4.2, 4.3, 4.4. Al final s'observa que l'algoritme, per aquest cas concret, funciona correctament i acaba deixant un polígon de 6 costats, cosa que vol dir que troba 5 rectes. Posteriorment caldrà comprovar quines es poden considerar realment com a landmarks.

A continuació, es pot veure el mètode IEPF en llenguatge algorísmic

Funció IEPF

punt_particio=Trobar_punt_mes_llunya(*C,i,j*)
si *distancia_punt_mes_llunya*>*llindar_d*$_s$
IEPF(*C,i,punt_particio*)
IEPF(*C,punt_particio,j*)
sino
retornar($\overline{\nu_i \nu_j}$)

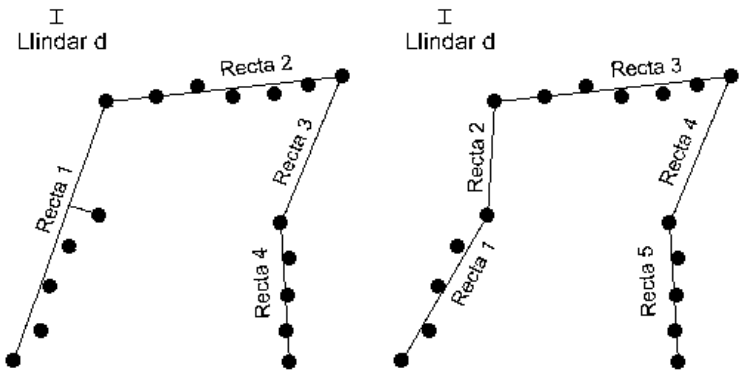

Figura 4.5: Continuació de l'algoritme IEPF

fsi

Funció Trobar_punt_mes_llunya

distancia_punt_mes_llunya=0

per k **entre** i+1 **i** j-1

$distancia\_punt\_k = \frac{|\tilde{\nu_i}\tilde{\nu_j}\tilde{\nu_k}|}{\|\nu_i - \nu_j\|^2}$

si *distancia_punt_k>distancia_punt_mes_llunya*

distancia_punt_mes_llunya=distancia_punt_k

punt_particio=k

fsi

fper

retornar(*punt_particio*)

Es pot comprovar la seva propietat recursiva, ja que la funció IEPF es crida a si mateixa en el seu interior.

4.2 Validació de les rectes com a landmarks

Un cop es tenen els punts agrupats en línies rectes, cal discernir quines d'aquestes es poden acceptar com a landmarks. Aquest pas és fonamental per no afegir obstacles inexistents en passos posteriors del mètode de construcció de mapes, ja que, com es veurà, hi ha moltes rectes que dóna l'IEPF que no corresponen a cap obstacle real.

Per fer la discriminació entre rectes vàlides i no vàlides, s'han establert una sèrie de condicions que una determinada recta ha de complir per ser considerada com a landmark. Aquestes condicions tenen dos objectius: el primer i principal és eliminar les rectes que no corresponen a landmarks reals, mentre que el segon és rebutjar landmarks que segurament sí que existeixen però són poc clars o difícils d'identificar. Aquest segon punt serveix per a fer més fàcil el treball posterior amb aquests landmarks, com es podrà comprovar en el següent capítol.

Les condicions fixades per a què una recta sigui considerada com a landmark, són les següents:

1. **Ha de contenir un nombre determinat de punts**. L'objectiu d'aquesta condició és evitar incloure dins la llista de landmarks detectats, línies rectes sobre les que la informació és escassa o pràcticament nul·la. L'exemple més clar és el mostrat en les rectes 2 i 4 de la figura 4.1, on les rectes només contenen dos punts, que són els seus extrems, i clarament es veu que no es té cap tipus d'informació del que hi ha en el seu interior. Podria ser que realment fos una paret però no es pot assegurar de cap manera. Altres casos on la informació és escassa són quan la recta conté només tres punts o quatre. A partir de cinc, els resultats que s'obtenen ja comencen a ser bons i es pot prendre aquest valor com a llindar de punts continguts en una recta per tal que aquesta sigui considerada com a landmark.

2. **La distància entre punts consecutius ha de ser inferior a un cert valor**. Aquesta condició evita, com en el cas anterior, posar obstacles en el mapa que no

existeixen realment. En aquest cas, però, no es descarta un landmark sencer, sino només una part d'ell. En el cas que la distància entre dos punts consecutius sigui major que un cert llindar, es considera que no es pot saber què hi ha entre ambdós punts, i per tant, no es pot considerar com a obstacle fins que el robot s'apropi a aquella zona i tingui una millor visibilitat.

Per a aconseguir l'objectiu d'aquesta condició cal fer un procés previ de partició de les rectes resultants de l'IEPF, entre els punts consecutius entre els que hi hagi massa distància. Això evita que descartem un landmark pel simple fet de tenir dos punts molt separats. Dos exemples es poden observar en la figura 4.6. En la primera s'observa com entre els dos últims punts de la recta hi ha una distància excessiva com per confirmar que hi ha un obstacle i no una porta, per exemple. Per això cal partir la recta per aquella zona i quan el robot s'acosti, hi haurà més informació sobre la zona.

En el segon cas, es veu com un punt que clarament no correspon a la paret de la dreta, està alineat amb aquesta i aquest fet provoca que l'IEPF ho consideri com a una sola recta. La solució passa per partir per aquesta zona i d'aquesta manera s'obté el landmark real.

Amb els experiments realitzats, s'ha comprovat que un bon llindar pot ser una distància de 35 centímetres.

3. **La distància de la recta ha de ser major que un cert valor**. Aquesta discriminació es fa pensant en la fase posterior a la detecció de landmarks. Les parets, com més llargues són, serveixen millor com a referència per després poder corregir la posició del robot, ja que habitualment contenen més punts, i per tant, la informació sobre elles és més gran i per tant, es poden extreure landmarks amb més precisió. A més, són també les més importants a l'hora de construir mapes. Per altra banda, els landmarks petits són més inexactes i més difícils de tractar en la part final de l'algoritme.

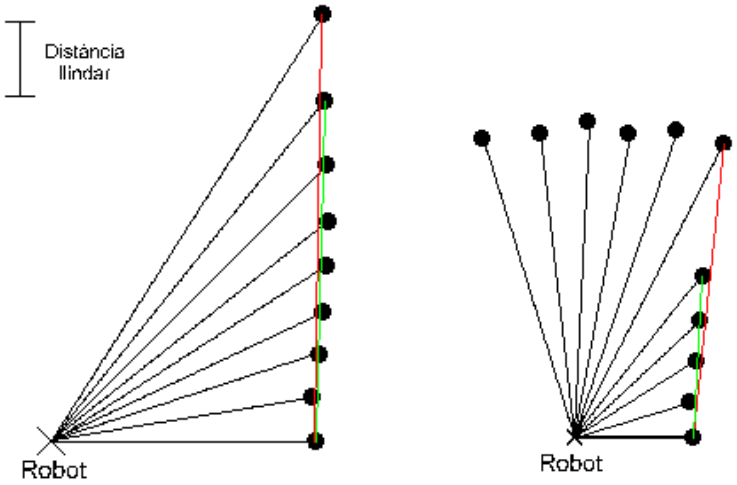

Figura 4.6: Dos casos en que cal partir una recta degut a la distància entre punts consecutius. En vermell: recta abans de partir; en verd: recta resultant de la partició

Cal afegir també que, com ja es veurà, el temps que empra l'algoritme del CML, és força depenent del nombre de landmarks, pel que és convenient que n'hi hagin els mínims possibles.

Per tot això, s'ha optat per descartar les rectes curtes, de més baixa qualitat, i quedar-nos només amb les que tinguin una certa distància. En els experiments fets, s'ha determinat que una bona solució de compromís pot ser una distància mínima de 40 centímetres.

4. **Ha d'estar com a màxim a una certa distància del robot**. Com s'explica a l'apartat 3.2.2, els sensors làser cometen un error relatiu proporcional a la distància, però a partir d'una certa distància aquest comença a augmentar de forma més gran que la corresponent a la linealitat. En el cas del sensor emprat en aquest projecte, s'observen en algunes ocasions, comportaments anòmals per a distàncies superiors als 7 o 8 metres. Per tant, per evitar la mala col·locació d'un landmark, que posteriorment podria ser una mala referència per a la localització del robot, es

decarten les rectes que estiguin més lluny d'aquesta distància. En tot cas, a mesura que el robot s'hi vagi apropant, ja seran col·locats amb una precisió molt més alta.

4.3 Acondicionament dels landmarks

Fins aquest punt del procés, les rectes que són acceptades com a landmarks simplement són rectes que van des del punt inicial fins al punt final que conté cadascuna, sense tenir en compte en cap moment els punts intermitjos.

Per a poder aprofitar d'alguna manera tota aquesta informació, i no dependre únicament de la qualitat de la mesura dels punts inicial i final de cada recta, es fa una regressió lineal, i d'aquesta manera, els landmarks finalment obtinguts seran molt més precisos.

La coneguda formulació de la regressió lineal, és la següent:

$$y = \mathbf{b_1} x + \mathbf{b_0} \tag{4.1}$$

$$\mathbf{b_1} = \frac{N \sum_{i=1}^{N} (x_i y_i) + \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{N \sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i \right)^2} \qquad \mathbf{b_0} = \frac{\sum_{i=1}^{N} y_i - \mathbf{b_1} \sum_{i=1}^{N} x_i}{N} \tag{4.2}$$

La representació final dels landmarks detectats, que serà la que s'utilitzarà en la resta de l'algoritme, és la del punt inicial i final de la recta després de la regressió. Aquests dos punts s'obtenen a partir de la projecció dels punts inicial i final abans de la regressió, sobre la recta resultant de l'estimació lineal. Una mostra d'aquest procés es pot observar a la figura 4.7

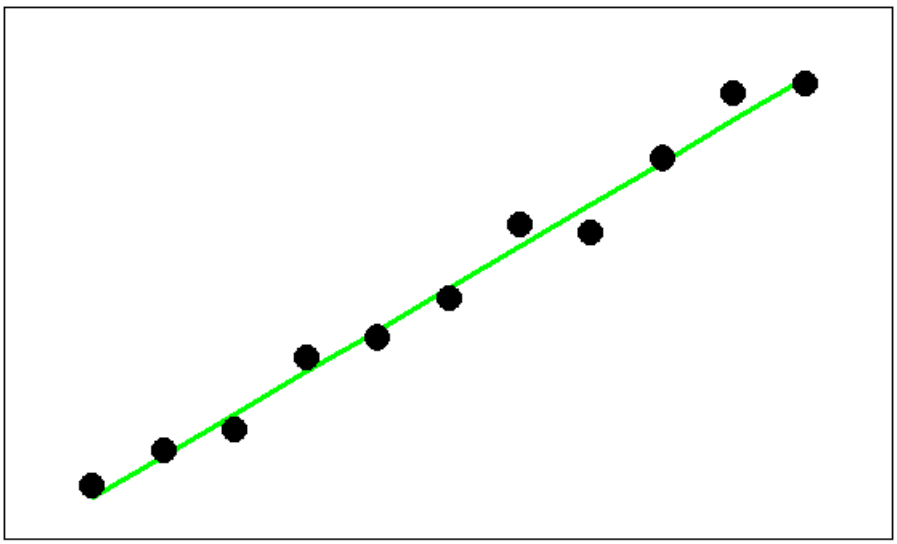

Figura 4.7: Regressió lineal

Capítol 5

Construcció del mapa i localització del robot

En aquest capítol s'explica i es descriu pas a pas el mètode utilitzat per a la construcció d'un mapa de l'entorn i la localització del robot, a partir de les dades que rep del sensor, és a dir, els landamrks extrets mitjançant els algoritmes descrits en el capítol 4. D'alguna manera, podríem considerar el sensor làser i els algoritmes de detecció de landmarks com a un sensor únic de landmarks, i que aporta les dades d'entrada per al mètode que s'explica en aquest capítol. És per això, que d'ara endavant ens referirem com a sensor, a la unitat conjunta formada pel sensor làser i els algoritmes d'extracció de landmarks.

El procés global és iteratiu, i a cada iteració el sensor làser realitza un escombrat, detecta una sèrie de landmarks i, amb aquestes dades, comença una iteració del CML. Un cop s'ha actualitzat el mapa i s'ha corregit la posició del robot, el sensor làser pot tornar a fer un escombrat i detectar nous landmarks, i d'aquesta manera continuaria indefinidament.

Els tres passos principals que composen aquest mètode són els següents:

1. **Associació de landmarks**. Aquest punt, que s'explica amb detall en l'apartat 5.1, és el més crític i alhora, el més complicat de resoldre. Consisteix en comparar els

landmarks detectats en un escombrat del sensor, amb els que ja hi ha en el mapa de l'entorn creat fins a aquell moment. L'objectiu és identificar els landmarks del mapa entre els landmarks detectats per, d'aquesta manera, poder-los utilitzar com a referència a l'hora d'intentar localitzar el robot.

2. **Localització del robot**. L'objectiu d'aquesta part és trobar la posició del robot, o una zona d'incertesa en la que sabem que el robot es troba amb un cert grau de confiança. Aquesta tasca es porta a terme mitjançant l'ajut d'una adaptació al cas concret del conegut filtre de Kalman, que s'exposa a l'apartat 5.2. Alhora, aquest també corregeix la posició dels landmarks del mapa i també els assigna una zona d'incertesa.

3. **Actualització del mapa**. Aquest procés es basa en la modificació dels landmarks ja existents al mapa segons els landmarks detectats en un escombrat. Habitualment, aquest procés el que fa és allargar landmarks, ja que degut a oclusions, el sensor pot no haver vist el landmark sencer, i posteriorment, quan es desplaça el robot, el landmark és detectat en la seva totalitat. L'altra funció, és la d'afegir els landmarks detectats en un escombrat, que no s'han associat amb cap dels existents en el mapa, cosa que habitualment significa que és un landmark que encara no s'havia detectat anteriorment, i per tant, un landmark nou. Una explicació més detallada es pot trobar a l'apartat 5.3.

Però abans de tot això cal definir l'estructura de dades amb la que es representarà el mapa. Aquest, simplement és un conjunt dels landmarks que s'han anat detectant. Alhora, cal també decidir quina serà la representació d'aquests landmarks. Com que són segments de recta, la millor representació possible és una estructura amb dos punts, l'inicial i el final, ja que a partir d'aquests es poden calcular tots els altres paràmetres de la recta fàcilment, i com ja es veurà quan s'expliqui el filtre de Kalman, és la representació més adient per a la seva implementació.

5.1 Associació de landmarks

Fins al moment, el robot disposa de mètodes per extreure landmarks de l'entorn a partir de les dades proporcionades pel sensor làser que disposa. Però aquesta capacitat no és suficient per a poder localitzar el robot dins d'un mapa, ja que cal que també sigui capaç d'identificar aquests landmarks que detecta entre els ja apresos anteriorment.

Aquest pas és de vital importància en el procés, ja que serveix per determinar les referències que després serviran per localitzar el robot. Les conseqüències d'un error en aquest pas, com hom pot imaginar, poden originar errors de molta més magnitud a l'hora de corregir la posició del robot, ja que associar un landmark amb un altre de diferent, pot desviar l'estimació de la posició del robot en molta distància. Per aquest motiu cal evitar, per sobre de tot, que es produeixin errors d'associació. Cal observar també que el fet de no identificar un landmark detectat amb un de ja existent, només implica la pèrdua d'una referència per la posterior localització del robot, però no afegeix error en el procés. Per tant, cal tenir molt en compte quan es dissenya l'algoritme que és molt millor un error de no identificació d'un landmark, que no pas un error d'associació, i per això quan s'associen landmarks s'ha de tenir una seguretat molt elevada de que s'està fent correctament.

Hi ha dues vies d'associació de landmarks. Una és l'associació per carcterístiques, que tracta de trobar invariants per cada tipus de landmark, és a dir, característiques úniques que permetrien identificar ràpidament al landmark com el color, la forma, etc.

L'altra és l'associació espacial, que associa dos landmarks quan aquests estan situats a la mateixa posició o suficientment propers com per ser considerat el mateix landmark, ja que físicament és impossible que en una mateixa posició hi hagi dos obstacles. Aquesta segona associació té la gran limitació que els landmarks han de ser estàtics, ja que si són mòbils, la seva associació espacial no és possible.

Degut a les limitacions del sensor que s'utilitza, és a dir, un sensor làser, l'associació per característiques és molt difícil de dur a terme. És la típica associació que fan servir

els mètodes d'associació que treballen amb imatges, on habitualment les característiques que es poden treure dels landmarks són colors, formes o àrees. En el cas del sensor de profunditat no es poden extreure massa característiques invariants dels landmarks. A més, el tipus de landmark escollit, també degut a les dades que dóna el làser, no permet l'extracció de característiques. L'única seria la llargada, però aquesta és molt poc fiable degut a la gran quantita d'oclusions que es produeixen.

D'altra banda, les restriccions fixades en l'apartat 4.2 per tal que una recta pugui ser considerada com a landmark, anaven enfocades a que la gran majoria dels landmarks acceptats fossin parets o part del contorn d'una habitació, i per tant, habitualment fixes, evitant així la principal limitació de l'associació espacial.

Per totes aquestes raons, el tipus d'associació que realitza el mètode presentat en aquest projecte serà del segon tipus, és a dir, associació espacial.

El principal enemic de qualsevol mètode d'associació són les oclusions. Sempre que s'intenta qualsevol mètode basat en el reconeixement, com és el cas, s'ha d'intentar evitar aquest fenòmen en la mesura del possible. En el nostre cas, malauradament, aquest fenòmen és inevitable per la pròpia naturalesa del robot mòbil. Aquest ha de navegar autònomament, i per tant, és pràcticament segur que hi haurà landmarks, o part d'ells que quedaran darrera d'altres, i que impediran la seva detecció total o parcial. Per tant, cal tenir molt en compte aquest fet, ja que de no produir-se, el mètode d'associació a dissenyar seria molt diferent, ja que havent-hi oclusions, dos landmarks associats no han de ser forçosament iguals, sino que un dels dos pot ser una petita part de l'altre.

Un cop fet aquest anàlisi, ja podem explicar el mètode emprat.

5.1.1 Funció *Similar*

Aquesta és una funció booleana que retorna cert quan considera que els dos landmarks que analitza corresponen a un mateix obstacle real, i per tant, s'han d'associar.

En primer lloc, és important especificar la nomenclatura que s'utilitzarà a partir d'a-

quest moment. En un mètode de construcció de mapes, hi ha dos tipus de landmarks: uns són els que s'extreuen de cada escombrat del làser, que denominarem *landmarks detectats*, i els altres són els landmarks ja introduits en el mapa i refinats cada iteració, que denominarem *landmarks del mapa*.

Com que l'objectiu és identificar els landmarks detectats d'entre els del mapa, les associacions només s'han de buscar entre landmarks de diferent grup. En altres paraules, no és necessari buscar associacions entre dos landmarks detectats, o entre dos landmarks del mapa, ja que no seria de cap utilitat.

Segons el nostre punt de vista, dos landmarks seran considerats *similars*, i per tant associats, quan compleixin les següents condicions:

1. **Condició d'angle**. Per passar aquesta condició, l'angle entre les dues rectes comparades ha de ser menor que un cert valor, que no pot ser massa gran. Aquest valor ha de ser major que l'error més gran que els sensors de moviment del robot poden arribar a cometre en una sola iteració. De no ser així, la posició del robot no podria ser corregida ja que mai es trobarien referències. Conseqüentment, aquest valor és el màxim angle que podrà corregir l'algoritme de localització, en el que es refereix a l'orientació del robot.

   Aquesta condició és bastant potent per dos motius: un és que utilitza una informació molt fiable i amb un error molt petit, ja que la orientació de la recta prové d'un ajust per regressió, i per tant, l'error de mesura que podia cometre el sensor queda reduit dràsticament. L'altre, és que amb una sola comparació es poden descartar la gran majoria de les associacions, ja que la probabilitat de que dos landmarks que corresponen a diferents obstacles reals tinguin la mateixa orientació és molt baixa. Això fa que el procés d'associació s'agilitzi en gran mesura, perquè si dos landmarks no compleixen aquesta condició, l'algoritme descarta directament la possible associació. A la figura 5.1 es poden observar una sèrie d'exemples sobre aquesta condició.

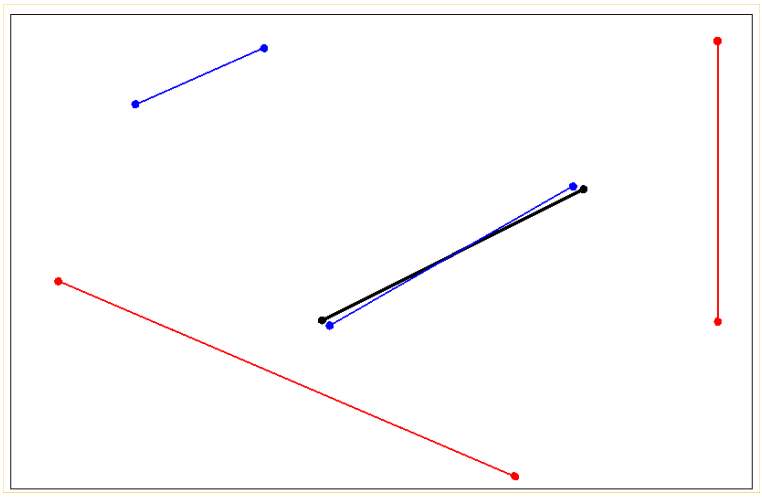

Figura 5.1: A la figura s'observa en negre un landmark del mapa, i una sèrie de landmarks detectats, de color blau si compleixen la condició d'angle, o vermell en cas contrari

Com es pot comprovar, aquesta condició no és ni molt menys suficient, per això hi ha també la condició de posició.

2. **Condició de posició**. Per a complir aquesta condició, s'estableix una zona com la de la figura 5.2 al voltant d'un dels dos landmarks analitzats. Aquesta zona està formada per un rectangle de dimensions $2 \cdot d_{err}$ i la longitud de la recta, i dues semiesferes de radi d_{err} a cadascun dels costats.

   La distància d_{err} que determina la grandària de la zona d'associació, ha de tenir un valor major que l'error més gran que pot cometre el sistema de posicionament del robot en una iteració. De la mateixa manera, serà la màxima correcció que el filtre de Kalman podrà fer sobre la posició del robot.

   Per tal que es compleixi la condició de posició, l'altre landmark ha de tenir com a mínim un dels seus dos extrems a l'interior d'aquesta zona. Si supera aquesta condició, els dos landmarks es consideren similars, i per tant, que pertanyen al mateix obstacle real. La raó d'aquesta condició és que si un landmark ha superat

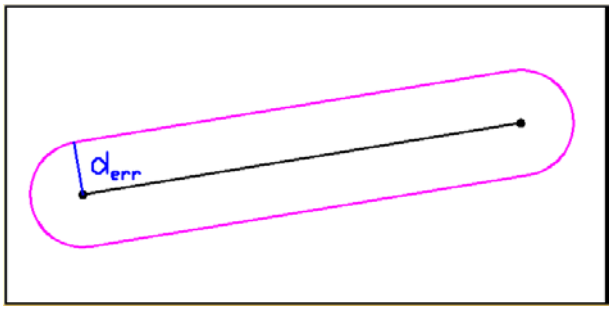

Figura 5.2: Zona on pot estar el landmark a associar, per tal de ser considerat similar

la condició d'angle, i per tant té una orientació igual, si té un punt dins de la zona d'associació segur que ambdós landmarks pertanyen al mateix obstacle, tot i que és molt possible que degut a les oclusions, l'altre punt no caigui dins d'aquesta zona.

Una sèrie d'exemples d'aquesta segona condició es poden veure a la figura 5.3

Cal evitar però, casos com el de la figura 5.4, en que segons les condicions d'angle i de posicio, els dos landmarks haurien de ser associats, però que realment no són el mateix. Aquests casos es dónen quan el landmark a associar només té un punt dins de la zona d'associació molt proper a un dels extrems del landmark comparat, i l'altre extrem està molt lluny de la recta. En aquests casos concrets, si es projecta una recta sobre l'altra, s'observa que la intersecció és molt petita, i en alguns casos nul·la. Per tant, com que és de vital importància evitar associacions errònies, les parelles de landmarks que es trobin en aquesta situació, hauran de complir una condició més, i aquesta serà que la intersecció de la projecció d'una recta sobre l'altra, amb aquesta segona, tingui una longitud mínima d_{inter}, que pot tenir perfectament el mateix valor que d_{err}.

Si s'analitzen les propietats de la condició de posició, es pot deduir que aquesta no és commutativa, però la funció *Similar* ha de ser-ho, ja que és lògic que si un landmark es considera que és el mateix que un altre, l'altre sigui també el mateix que el primer. Per tal d'aconseguir la commutativitat, l'únic que cal fer en el cas

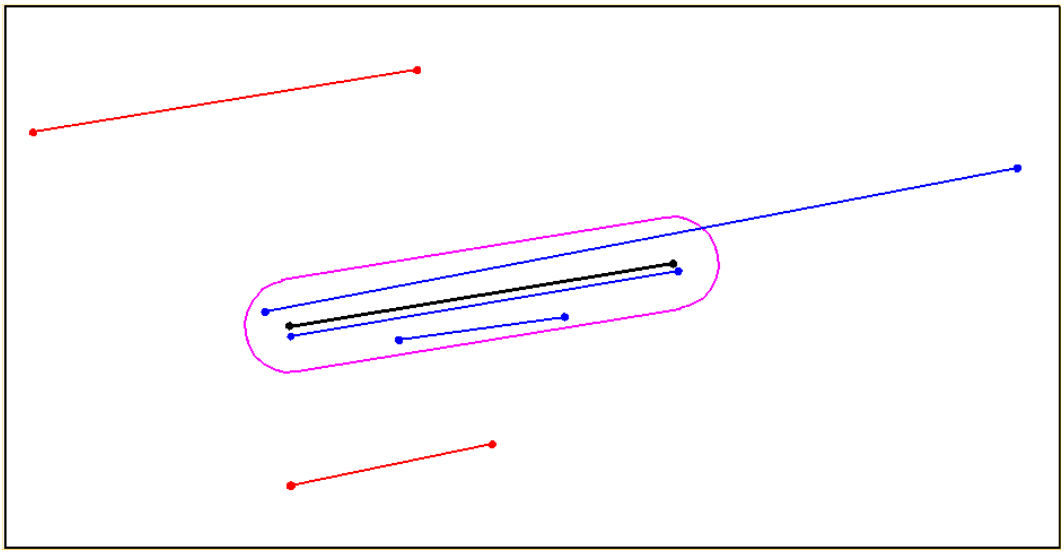

Figura 5.3: Exemples de landmarks que s'intenten associar amb el landmark del mapa, de color negre. Els de color blau, compleixen la condició de posició mentre que els vermells no. El contorn lila delimita la zona d'associació

que dos landmarks compleixin la condició d'angle però no la de posició, és tornar a provar aquesta segona condició invertint la funció dels dos landmarks. El que abans servia per establir la zona de posició, passa a ser el que serveix per comprovar si algun dels seus extrems és dins de la zona, i viceversa.

D'aquesta manera, la funció *Similar* descrita en aquest apartat es pot implementar com s'observa a continuació:

Funció Similar

si Diferencia_Angle(*landmark1,landmark2*) $> \alpha_{max}$

retorna fals

fsi

si Compleix_Condicio_Distancia(*landmark1,landmark2,*d_{err})

retorna cert

fsi

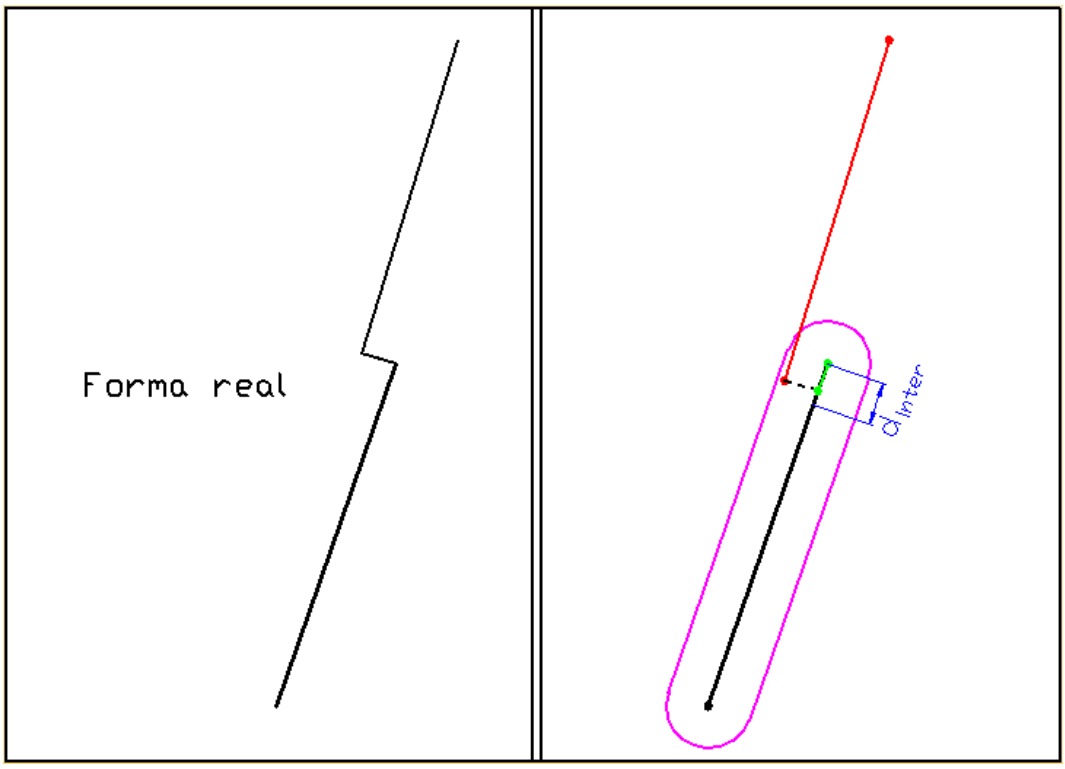

Figura 5.4: Cas en què dos landmarks no s'han d'associar tot i complir la condició de posició i d'angle

si Compleix_Condicio_Distancia(*landmark2,landmark1,*d_{err})

retorna cert

fsi

retorna fals

Funció Compleix_Condicio_Distancia

punts_dins=0

si Dins_Zona_Associacio(*landmark1.extrem1,landmark2,*d_{err})

punts_dins=*punts_dins*+1

si Interseccio(Projeccio(*landmark1,landmark2*),landmark2) > d_{inter}

retorna cert

fsi

fsi

si Dins_Zona_Associacio(*landmark1.extrem2,landmark2,*d_{err})

punts_dins=*punts_dins*+1

si *punts_dins*=2

retorna cert

fsi

si Interseccio(Projeccio(*landmark1,landmark2*),landmark2) > d_{inter}

retorna cert

fsi

fsi

retorna fals

5.2 Localització del robot

A la figura 5.5 es pot veure més clarament en què consisteix la localització. El robot, parteix des d'una posició qualsevol i va muntant un mapa de landmarks. En un moment donat, segons els seus encòders arriba a la posició de color negre, i des d'aquest punt, captura dades amb el làser, extreu landmarks i obté les rectes 1' i 2' de la figura. Fins aquell moment, en el mapa s'hi havien inclòs dos landmarks, que són 1 i 2. Clarament es pot veure com el landmark 1' correspon al 1 del mapa, així com el 2' amb el 2, però els landmarks detectats 1' i 2' estan desplaçats cap a la dreta i cap amunt, i a més presenten una petita desviació respecte els landmarks 1 i 2 del mapa. Això significa que la mesura dels encòders i la odometria del robot han comès error en la mesura de la posició, i que per tant, el robot realment es troba en la posició marcada en vermell. La missió

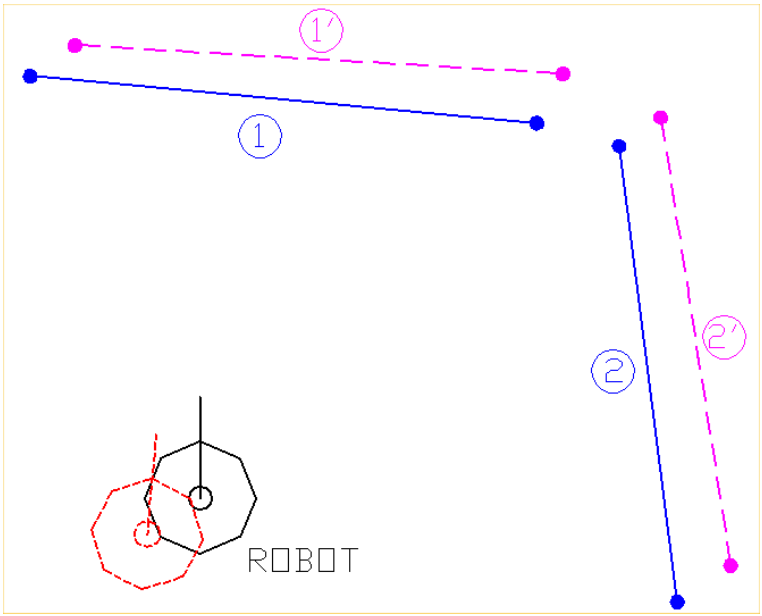

Figura 5.5: Localització d'un robot. El robot calcula la seva posició (en negre), que després és corregida per l'algoritme (en vermell), comparant els landmarks detectats (en lila) amb els ja coneguts (en blau)

dels algoritmes de localització és fer una estimació d'aquesta posició real del robot. En el nostre cas, l'algoritme que s'utilitzarà serà una adaptació del filtre de Kalman, anomenada Filtre de Kalman Extès (*Extended Kalman Filter* - EKF).

Per a introduir la notació i el procés que realitza el filtre de Kaman, posarem el següent exemple. Suposem que el robot porta ja una estona construint un mapa i en un moment donat es troba en una posició, que després d'haver estat corregida, s'estima que és el vector de tres coordenades $x_{r,k|k} = [x, y, \theta]^T$, amb una certa incertesa representada per la matriu de covariança 3×3, $P_{r,k|k}$. Les variables x, y representen la posició del centre del robot, mentre que θ representa l'orientació. A partir d'aquí el robot continua movent-se i llegeix un nou contorn del làser. En aquest precís instant els encòders mesuren la distància u_k recorreguda pel robot des de l'últim punt on s'han capturat dades del làser, és a dir, $x_{r,k}$. Amb aquestes dades, es pot calcular l'estimació de la nova posició del robot

a priori $x_{r,k+1|k}$, amb una incertesa $P_{r,k+1|k}$ bastant gran. La nomenclatura *a priori*, es refereix a dades obtingudes abans que es comparin els nous landmarks detectats amb els ja existents en el mapa. Un cop es realitza la comparació, s'obté l'estimació de la posició del robot *a posteriori*, $x_{r,k+1|k+1}$, amb una incertesa $P_{r,k+1|k+1}$ molt més petita que la *a priori*. Les posicions reals del robot $x_{r,k}$ i $x_{r,k+1}$, no es poden determinar de cap manera, sino que només poden ser estimades. Tot i això, el filtre de Kalman garanteix que l'error quadràtic entre els valors reals i els estimats sigui mínim.

5.2.1 Actualització de l'estat

Per començar amb la formulació matemàtica, ho podem fer amb el model del moviment del robot, que serveix per calcular la posició actual del robot, a partir de la seva posició anterior, de les lectures locals dels encòders i d'una variable aleatòria gaussiana v_k, que defineix l'error comès en la mesura. Queda definit com

$$x_{r,k+1} = f_r(x_{r,k}, u_k, v_k) \tag{5.1}$$

on la funció f varia segons cada robot, i en el cas del robot Marco

$$f_r = \begin{pmatrix} x_k + \sqrt{u_x^2 + u_y^2}\cos\left(\theta_k + \arctan\left(\frac{u_y}{u_x}\right)\right) + \sqrt{v_x^2 + v_y^2}\cos\left(\theta_k + \arctan\left(\frac{v_y}{v_x}\right)\right) \\ y_k + \sqrt{u_x^2 + u_y^2}\sin\left(\theta_k + \arctan\left(\frac{u_y}{u_x}\right)\right) + \sqrt{v_x^2 + v_y^2}\sin\left(\theta_k + \arctan\left(\frac{v_y}{v_x}\right)\right) \\ \theta_k + u_\theta + v_\theta \end{pmatrix} \tag{5.2}$$

El filtre de Kalman treballa amb un vector d'estat x_k, on es guarda tota la informació de la posició del robot $x_{r,k}$ i dels landmarks $x_{l,k}$ i és de la següent forma

$$x_k = \begin{pmatrix} x_{r,k} \\ x_{l,k} \end{pmatrix} = \begin{pmatrix} x_{r,k} \\ x_{l,k}^{(1)} \\ \vdots \\ x_{l,k}^{(n)} \end{pmatrix} \tag{5.3}$$

on $x_l^i = [x_1^i, y_1^i, x_2^i, y_2^i]^T$ és el vector que conté les coordenades dels punts extrems dels landmarks del mapa. El filtre actualitza aquest vector d'estat a cada iteració.

Com podem observar, l'equació 5.1 fa referència a les posicions reals del robot $x_{r,k}$, que són sempre desconegudes pel robot. Per tant, cal treballar amb les estimacions d'aquests, i d'aquesta manera l'equació 5.1 queda

$$x_{r,k+1|k} = f_r(x_{r,k|k}, u_k, 0) \tag{5.4}$$

S'observa que el tercer paràmetre de la funció és 0, degut a que en aquest cas no es pretén calcular la posició exacta del robot, sino només una estimació que, òbviament, ja inclourà un cert error v_k que no es pot conèixer.

L'equació 5.4 és només l'equació d'actualització del vector posició del robot, però l'equació completa d'actualització del vector d'estat és

$$x_{k+1|k} = f(x_{r,k}, u_k, 0) \tag{5.5}$$

on en aquest cas, la funció completa f és la següent:

$$f = \begin{pmatrix} f_r \\ f_l \end{pmatrix} = \begin{pmatrix} f_r \\ x^{(1)}_{l,k|k} \\ \vdots \\ x^{(n)}_{l,k|k} \end{pmatrix} \tag{5.6}$$

Es pot comprovar que la funció només canvia la part de l'estat que correspon a la posició del robot, ja que es contempla que els landarks romanen estàtics.

Un cop tenim l'estimació *a priori* de l'estat, s'ha de calcular la seva corresponent matriu de covariança per tal de saber la incertesa d'aquesta estimació. La matriu completa de covariança és de la forma

$$P_k = \begin{pmatrix} P_{r,k} & P_{rl,k} \\ P_{lr,k} & P_{l,k} \end{pmatrix} \tag{5.7}$$

on $P_{r,k}$ i $P_{l,k}$ són les matrius de covariança del robot i dels landmarks respectivament, mentre que les altres dues submatrius, $P_{rl,k}$ i $P_{lr,k}$, indiquen la correlació existent entre la incertesa de les posicions dels landmark i del robot. Per calcular l'actualització de la

matriu de covariances, és a dir, la incertesa de l'estat *a priori*, s'utilitza la fórmula

$$P_{k+1|k} = \begin{pmatrix} F_r P_{r,k|k} F_r^T + V_r & F_r P_{rl,k} \\ P_{lr,k} F_r^T & P_{l,k} \end{pmatrix} \tag{5.8}$$

on

$$F_r = \left.\frac{\partial f}{\partial x}\right|_{(x_{k|k},u_k,0)} = \begin{pmatrix} 1 & 0 & -\sqrt{u_x^2+u_y^2}\sin(\theta_k + \arctan(\frac{u_y}{u_x})) \\ 0 & 1 & \sqrt{u_x^2+u_y^2}\cos(\theta_k + \arctan(\frac{u_y}{u_x})) \\ 0 & 0 & 1 \end{pmatrix} \tag{5.9}$$

i V_r es pot considerar la matriu de covariança de l'error comès pels encòders a l'hora de mesurar el desplaçament. Aquesta matriu es calcula a partir de l'observació del comportament del robot, del que hem modelat l'error dividint-lo en tres components. La primera està realacionada amb el canvi de direcció del moviment, que pot venir determinat per l'angle ψ_u. La segona està relacionada amb la distància recorreguda d_u, mentre que la última component va lligada a l'ajust final de la direcció, representat per l'angle u_θ. D'aquesta manera, la desviació estàndard en coordenades polars de l'error de mesura de desplaçament, la calculem proporcional a cadascuna de les components citades.

$$\sigma_\psi = \alpha_\psi \psi_u \tag{5.10}$$

$$\sigma_d = \alpha_d d_u \tag{5.11}$$

$$\sigma_\theta = \alpha_\theta u_\theta \tag{5.12}$$

I a partir d'aquí la matriu V_r es pot obtenir com

$$V_r = R_u \begin{pmatrix} \sigma_d^2 & 0 & 0 \\ 0 & (d_u \tan\sigma_\psi)^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} R_u^T \tag{5.13}$$

on la matriu R_u repreenta la matriu de pas de coordenades polars a cartesianes i és

$$R_u = \begin{pmatrix} \cos(\theta_k + \psi_u) & -\sin(\theta_k + \psi_u) & 0 \\ \sin(\theta_k + \psi_u) & cos(\theta_k + \psi_u) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.14}$$

5.2.2 Correcció de l'estat

Fins aquest punt de l'algoritme, hem pogut estimar la posició del robot a partir de les dades dels encòders, per tant, encara disposem de la informació proporcionada pel sensor làser per a refinar el vector d'estat i obtenir d'aquesta manera, l'estimació *a posteriori* del vector d'estat, $x_{k+1|k+1}$.

En primer lloc s'ha de predir la posició dels landmarks que veurà el sensor làser en aquesta iteració. Per fer-ho, s'ha de passar la part del vector d'estat corresponent als landmarks, que està en coordenades globals, a coordenades locals. El vector que representa aquesta predicció serà com una estimació *a priori* del que veurà el sensor de profunditat, i li donarem la nomenclatura $z_{k+1|k}$. Es calcula com

$$z_{k+1|k} = h(x_{k+1|k}) \tag{5.15}$$

I en el cas del robot Marco, la funció h es pot expressar de la següent forma:

$$h = \begin{pmatrix} R^T(x^{(1)}_{l,k+1|k} - t_r) \\ R^T(x^{(2)}_{l,k+1|k} - t_r) \\ \vdots \\ R^T(x^{(n)}_{l,k+1|k} - t_r) \end{pmatrix} \tag{5.16}$$

on

$$R = \begin{pmatrix} \cos(\theta_{k+1|k}) & -\sin(\theta_{k+1|k}) & 0 & \\ \sin(\theta_{k+1|k}) & \cos(\theta_{k+1|k}) & & \\ 0 & & \cos(\theta_{k+1|k}) & -\sin(\theta_{k+1|k}) \\ & & \sin(\theta_{k+1|k}) & \cos(\theta_{k+1|k}) \end{pmatrix} \tag{5.17}$$

$$t = [x_{k+1|k}, y_{k+1|k}, x_{k+1|k}, y_{k+1|k}]^T \tag{5.18}$$

Per altra banda també tenim el vector z_{k+1} que conté la informació dels landmarks detectats pel làser en la iteració actual, també en coordenades locals. La manera com omplir aquest vector no és trivial i s'explica amb detall a l'apartat 5.2.5.

Un cop fet això, ja es poden comparar les prediccions amb les observacions del làser, i calcular la diferència entre l'una i l'altra:

$$\tilde{z}_{k+1|k} = z_{k+1} - z_{k+1|k} \tag{5.19}$$

A partir d'aquí ja estem en condicions de calcular l'estimació *a posteriori* del vector d'estat, $x_{k+1|k+1}$ que es pot fer amb la fórmula de la correcció d'estat, que és la que segueix:

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1}\tilde{z}_{k+1|k} \tag{5.20}$$

De la fórmula 5.20 només falta saber com es calcula la matriu K_{k+1}. Aquesta rep el nom de *matriu de guany de Kalman*, i es calcula mitjançant la fórmula

$$K_{k+1} = P_{k+1|k}H_x^T S^{-1} \tag{5.21}$$

on la matriu H_x és la *matriu jacobiana de l'observació* i la matriu S s'nomena *matriu d'innovació*. La matriu H_x es calcula amb l'expressió

$$H_x = \left.\frac{\partial h}{\partial x}\right|_{(x_{k+1|k})} \tag{5.22}$$

Aplicant l'anterior equació, la matriu H_x té una forma com la que segueix:

$$H_x = \begin{pmatrix} H_{x_r}^{(1)} & H_{x_l}^{(1)} & 0 & \cdots & \cdots & 0 \\ H_{x_r}^{(2)} & 0 & H_{x_l}^{(2)} & 0 & \cdots & 0 \\ \vdots & \vdots & & & \ddots & \vdots \\ H_{x_r}^{(n)} & 0 & \cdots & \cdots & 0 & H_{x_l}^{(n)} \end{pmatrix} \tag{5.23}$$

on les matrius

$$H_{x_l}^{(i)} = R^T \tag{5.24}$$

$$H_{x_r}^{(i)} = \begin{pmatrix} -\cos(\theta_{k+1|k}) & -\sin(\theta_{k+1|k}) & \\ \sin(\theta_{k+1|k}) & -\cos(\theta_{k+1|k}) & \dot{R}^T(x_{l,k+1|k}^{(i)} - t) \\ -\cos(\theta_{k+1|k}) & -\sin(\theta_{k+1|k}) & \\ \sin(\theta_{k+1|k}) & -\cos(\theta_{k+1|k}) & \end{pmatrix} \tag{5.25}$$

Per altra banda, la matriu d'innovació S s'obté amb la següent fórmula:

$$S = H_x P_{k+1|k} H_x^T + W_{k+1} \tag{5.26}$$

La matriu W_{k+1} és una matriu diagonal per blocs 4×4, i determina la qualitat de la mesura del làser per a cada landark. Aquesta matriu és de la forma

$$W_{k+1} = \begin{pmatrix} W_{k+1}^{(1)} & & & 0 \\ & W_{k+1}^{(2)} & & \\ & & \ddots & \\ 0 & & & W_{k+1}^{(n)} \end{pmatrix} \tag{5.27}$$

Cada matriu $W_{k+1}^{(i)}$ es calcula analitzant el sistema de mesura del làser. D'aquesta manera considerem que l'error de mesura del làser és la unió de l'error dependent de la distància mesurada d_z, i el que depen de l'angle ψ_z de cada mesura. D'aquesta manera, les desviacions estàndar de cada component són proporcionals a aquests valors:

$$\sigma_d = \beta_d d_z \tag{5.28}$$

$$\sigma_\psi = \beta_\psi \psi_z \tag{5.29}$$

i les submatrius $W_{k+1}^{(i)}$ es calculen segons

$$W_{k+1}^{(i)} = R_\psi \begin{pmatrix} (\sigma_d^{(1)})^2 & & & 0 \\ & (d_z^{(1)} \tan \sigma_\psi^{(1)})^2 & & \\ & & (\sigma_d^{(2)})^2 & \\ & & & (d_z^{(2)} \tan \sigma_\psi^{(2)})^2 \end{pmatrix} R_\psi^T \tag{5.30}$$

on R_ψ és

$$R_\psi = \begin{pmatrix} \cos\psi_z^{(1)} & -\sin\psi_z^{(1)} & & \\ \sin\psi_z^{(1)} & \cos\psi_z^{(1)} & 0 & \\ & & \cos\psi_z^{(2)} & -\sin\psi_z^{(2)} \\ 0 & & \sin\psi_z^{(2)} & \cos\psi_z^{(2)} \end{pmatrix} \tag{5.31}$$

Les variables amb superíndex 1 o 2 indica a quin dels dos extrems de la recta corresponen els valors.

Els factors de proporcionalitat β_ψ i β_d juntament amb els factors α_ψ, α_d i α_θ de les equacions 5.10-5.12, són els que marquen el grau de confiança que donem al làser o als encòders, respectivament. Si els primers prenen valors majors que els segons, significa que a l'hora de trobar l'estimació *a posteriori* de la posició del robot, el filtre de Kalman donarà més valor a la informació aportada pel làser que a la informació aportada pels sensors de moviment.

L'única cosa que queda per fer és calcular la incertesa del vector d'estat *a posteriori*, representada per la matriu de covariança $P_{k+1|k+1}$. Aquesta matriu es calcula amb l'expressió

$$P_{k+1|k+1} = (I - K_{k+1}H_x)P_{k+1|k}(I - K_{k+1}H_x)^T + K_{k+1}W_{k+1}K_{k+1} \tag{5.32}$$

Com es podrà comprovar a la secció de resultats, els valors de la matriu de covariança *a posteriori* són molt inferiors que els de la matriu de covariança *a priori*. Això és degut a que la matriu *a posteriori* mesura la incertesa de l'estat un cop es té en compte la informació del làser, i per tant, com que s'utilitza informació de diverses fonts, el resultat és molt més fiable. Precisament aquest és un dels objectius del filtre de Kalman. A part de corregir la posició, disminueix en gran mesura la zona d'incertesa de la posició del robot.

5.2.3 Innovació seqüencial

Fins ara hem vist la formulació matemàtica teòrica del filtre de Kalman, però a l'hora de portar-la a la pràctica apareix un problema important. Si s'observa l'equació 5.21, hom pot comprovar que per calcular la matriu K_{k+1} és necessari el càlcul de la inversa de la matriu d'innovació S. Aquest procés té un alt cost computacional i encara més si tenim en compte les dimensions que pot arribar a tenir la matriu S. Suposant que n és el nombre

de landmarks, les dimensions d'S són:

$$\dim(\mathrm{S}):\ 4n \times 4n \tag{5.33}$$

Per tant, en el cas que s'hagin detectat, per exemple, dotze landmarks, les dimensions de la matriu S serien 48×48, cosa que suposaria un temps excessiu en el càlcul de la inversa. Per tant, és necessari buscar un mètode alternatiu per a aquest pas, que eviti el càlcul d'aquestes inverses tant grans. El nom que rep el mètode utilitzat s'anomena *Innovació seqüencial*.

Com el propi nom diu, el càlcul de l'estimació *a posteriori* del vector d'estat es realitza de forma seqüencial, substituint l'equació 5.20 per

$$x_{k+1|k+1} = x_{k+1|k} + \sum_{i=1}^{n} \left(K_{k+1}^{(i)} \tilde{z}_{k+1|k}^{(i)} \right) \tag{5.34}$$

Aquesta nova fórmula calcula una *matriu de guany de Kalman* individual per a cada landmark, que es calcula com

$$K_{k+1}^{(i)} = P_{k+1|k} (H_x^{(i)})^T (S^{(i)})^{-1} \tag{5.35}$$

on ara la *matriu d'innovació* individual de cada landmark es calcula mitjançant l'equació

$$S^{(i)} = H_x^{(i)} P_{k+1|k} (H_x^{(i)})^T + W_{k+1}^{(i)} \tag{5.36}$$

Si ara es calculen les dimensions de la matriu $S^{(i)}$, es pot comprovar que sempre són 4×4, i per tant, a més de reduir el tamany de la inversa a calcular, aquest mètode sempre assegura que no es faran inversions majors d'aquest tamany.

A més, està demostrat matemàticament a [8, 15] que l'ús de la innovació seqüencial no empitjora el comportament del filtre de Kalman sempre i quan el soroll dels sensors pugui ser considerat com a blanc i descorrelacionat. Per tot això, l'ús de la innovació seqüencial és pràcticament obligatori en cas que es vulgui utilitzar un filtre de Kalman per aplicacions en temps real.

5.2.4 Inicialització de les variables

Tota la formulació mostrada fins al moment, parteix d'una informació acumulada guardada en el vector d'estat anterior $x_{k|k}$, que conté l'estimació *a posteriori* de la posició del robot i dels landmarks corresponent a la iteració anterior, juntament amb la corresponent incertesa, representada per la matriu de covariança $P_{k|k}$. Però quan l'algoritme comença, no es disposa d'informació anterior i, per tant, cal trobar una manera de començar.

El primer que cal fer, és fixar la posició inicial del robot. Aquesta es pot fixar en qualsevol valor, però el més llògic és fixar-lo en l'origen del sistema de coordenades globals $x_{r,0} = [0, 0, 0]^T$, és a dir, la posició (0,0) i una orientació de 0 graus. Posteriorment, cal fer un escombrat amb el làser abans d'iniciar el moviment, per a detectar algun landmark, ja que fins que no se'n detecti cap, no té sentit aplicar el filtre de Kalman. Els landmarks detectats en aquest escombrat, poden ser introduïts directament dins del vector d'estat inicial x_0, ja que els eixos de coordenades locals i globals coincideixen. Per tant,

$$x_{l,0} = z_0 \tag{5.37}$$

i

$$x_0 = \begin{pmatrix} x_{r,0} \\ x_{l,0} \end{pmatrix} \tag{5.38}$$

Finalment, només queda inicialitzar la matriu de covariança P_0 que com ja s'ha vist a l'equació 5.7 és de la forma

$$P_0 = \begin{pmatrix} P_{r,0} & 0 \\ 0 & P_{l,0} \end{pmatrix} \tag{5.39}$$

Les submatrius $P_{rl,0}$ i $P_{lr,0}$ són nul·les ja que la correlació entre la posició dels landmarks i del robot és inexistent.

La submatriu $P_{r,0}$ representa la incertesa de la posició inicial. Intuitivament hauria de ser també nul·la, però per raons matemàtiques de convergència del filtre no és aconsellable, per tant la inicialitzarem com una matriu diagonal on cada valor de la posició té la seva

variança, però no hi ha establerta una correlació entre ells.

$$P_{r,0} = \begin{pmatrix} \gamma_{r,x} & 0 & 0 \\ 0 & \gamma_{r,y} & 0 \\ 0 & 0 & \gamma_{r,\theta} \end{pmatrix} \tag{5.40}$$

on habitualment $\gamma_x = \gamma_y \neq \gamma_\theta$, i prenen tots ells valors molt petits.

Per altra banda, $P_{l,0}$ representa la incertesa inicial sobre la posició dels landmarks, i més concretament, sobre la posició de cadascun dels dos extrems de la recta. Aquesta submatriu està formada alhora, per diverses submatrius, una en particular per cada landmark detectat en el primer escombrat.

$$P_{l,0} = \begin{pmatrix} P_{l,0}^{(1)} & & & 0 \\ & P_{l,0}^{(2)} & & \\ & & \ddots & \\ 0 & & & P_{l,0}^{(n)} \end{pmatrix} \tag{5.41}$$

on cada submatriu és

$$P_{l,0}^{(i)} = \begin{pmatrix} \gamma_{x,1}^{(i)} & 0 & 0 & 0 \\ 0 & \gamma_{y,1}^{(i)} & 0 & 0 \\ 0 & 0 & \gamma_{x,2}^{(i)} & 0 \\ 0 & 0 & 0 & \gamma_{y,2}^{(i)} \end{pmatrix} \tag{5.42}$$

Els valors γ indiquen la variança de cada coordenada de cadascun dels dos extrems del landmark i poden prendre tots el mateix valor.

Com es pot observar, la matriu P_0 és completament diagonal i per tant, és com si a l'inici no existís cap correlació entre magnituds que intervenen en el filtre. Aquestes s'aniran creant i actualitzant a mesura que es vagin fent iteracions, així com la resta de valors de la matriu.

5.2.5 Valors del vector z_{k+1}

Com s'ha comentat a l'apartat 5.2.2, el procés d'omplir els valors del vector d'observacions z_{k+1} no és trivial. Això és degut bàsicament a les oclusions, que fan que alguns dels landmarks, en algunes iteracions, no siguin vistos o només se'n detecti una part.

El vector z_{k+1} va estretament lligat al vector de predicció de les observacions $z_{k+1|k}$. Aquest vector inclou els valors ordenats de tots els extrems de les rectes detectades fins al moment, i això significa que per a que tingui sentit l'equació 5.19, que calcula la diferència entre predicció i observacions, el vector z_{k+1} ha d'incloure tots els extrems dels landmarks i en el mateix ordre que el vector $z_{k+1|k}$. Per tant, el problema que es planteja és quins valors escollir per a aquells landmarks que en aquella iteració en concret queden parcial o totalment ocluïts.

L'altre problema que se'ns planteja és el de discenir en quins casos els extrems de dos landmarks *similars* són considerats com el mateix punt i en quins casos considerem que es produeixen oclusions. A la figura 5.6 es pot veure un cas on es fa difícil decidir si s'ha produït una petita oclusió o la petita diferència entre els dos punts és deguda als errors dels sensors.

En primer lloc, considerarem que dos extrems de dos landmarks similars, com els de la figura 5.6, corresponen a un mateix punt real si disten menys d'una distància llindar d_l, que no pot ser gaire gran. En aquest cas, com probablement succeeix amb l'extrem e_1^1 de la figura, omplirem l'espai corresponent a la observació del citat extrem dins del vector z_{k+1}, amb els valors de l'extrem $e_1^{1'}$. En cas contrari, com és el cas de l'extrem e_2^1, l'espai correponent l'omplirem amb la projecció de l'extrem e_2^1 sobre la recta 1', que és on suposadament hauríem vist l'extrem en cas que no s'hagués produit cap oclusió.

Finalment, si un landmark no ha estat vist en una iteració, omplirem directament l'espai corresponent en el vector $\tilde{z}_{k+1|k}$ amb valors nuls, és a dir $\tilde{z}^{(i)}_{k+1|k} = [0,0,0,0]^T$. D'aquesta manera, en realitzar el càlcul de l'estimació *a posteriori* de l'estat, ja sigui amb la fórmula 5.20 o la 5.34 els landmarks no vistos no modifiquen el càlcul.

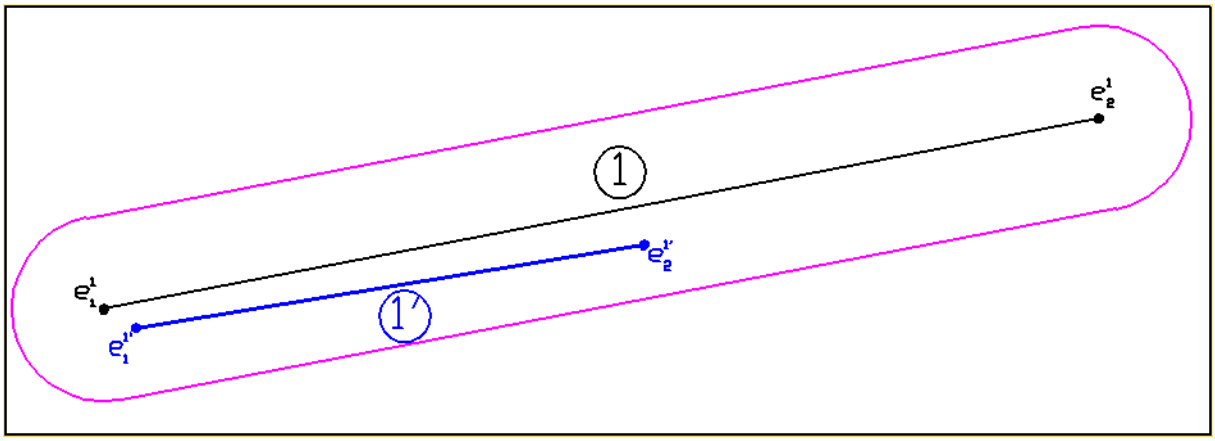

Figura 5.6: Dos landmarks similars. El landmark 1 (negre) és un landmark del mapa i el landmark 1' (blau) és un landmark detectat en l'última iteració. Es fa difícil saber si els extrems e_1^1 i $e_1^{1'}$ són en en realitat el mateix punt real o hi ha una petita oclusió, mentre que aquesta sí que es produeix en el cas de l'extrem e_2^1, ja que l'extrem $e_2^{1'}$ no correspon al mateix punt físic.

5.2.6 Convergència del filtre de Kalman

A continuació es descriuen tres propietats teòriques importants sobre la convergència del filtre de Kalman, i el seu comportament quan el nombre d'iteracions és molt gran i s'apropa a l'infinit. Les propietas són ls següents:

1. La primera propietat consisteix en que la matriu de covariances de l'error d'estimació de l'estat dels landmarks, $P_{l,k|k}$,decreix assimptòticament a mesura que augmenta el nombre d'iteracions. Matemàticament es pot expressar com

$$\det P_{l,k+1|k+1} \leq \det P_{l,k|k} \tag{5.43}$$

2. La segona propietat ens indica que el mapa, a mesura que el nombre d'iteracions s'acosta a infinit, es va tornant *completament correlacionat*. Aquest terme significa que havent observat un sol landmark, es pot deduir exactament la posició de tots els

altres. En forma de límit, aquesta propietat es pot escriure amb la següent expressió:

$$\lim_{k\to\infty} \det P_{l,k|k} = 0 \tag{5.44}$$

3. La tercera propietat tracta sobre la part de la matriu de covariança que mesura la incertesa de la posició del robot, $P_{r,k|k}$. Ens diu que la precisió màxima amb la que es pot determinar la posició absoluta del robot, està limitada per la incertesa inicial d'aquesta. En el cas d'un robot sense soroll

$$\lim_{k\to\infty} P_{r,k|k} = P_{r,0} \tag{5.45}$$

5.3 Actualització del mapa

Fins ara, l'algoritme global de construcció de mapes només corregeix la posició del robot, i en menor mesura la dels landmarks del mapa. Però als segons, a part de corregir lleugerament la seva posició, de moment no se'ls aplica cap més modificació. Es podria donar el cas que fins a la iteració actual k, un determinat landmark no hagi estat vist en la seva totalitat degut bàsicament a les oclusions, o a algun altre fenòmen, i que en la iteració k+1, la part que falta és detectada. Seria convenient que l'algoritme de construcció del mapa, disposés d'algun mecanisme per afegir aquest tros detectat al landmark incomplet que hi havia en el mapa. En aquest apartat s'explicarà el mètode utilitzat per a realitzar aquesta tasca.

Posem per exemple, el cas mostrat a la figura 5.7, on es suposa que el robot té una capacitat de visió de 360º. Quan aquest es troba a la posició 1, només pot detectar els landmarks de color blau, que són el landmark 1 i 4, i només una part del landmark 2. Però quan posteriorment el robot avança i arriba a la posició 2, ja són dins del seu angle de visió els landmarks marcats en blau i verd, que són els landmarks 1,2 i 4, i parcialment el 3. El que hauria de fer el mètode d'actualització del mapa seria afegir el tros de landmark 3 detectat, i actualitzar el landmark 2, allargant-lo amb el nou tros que el robot ha detectat.

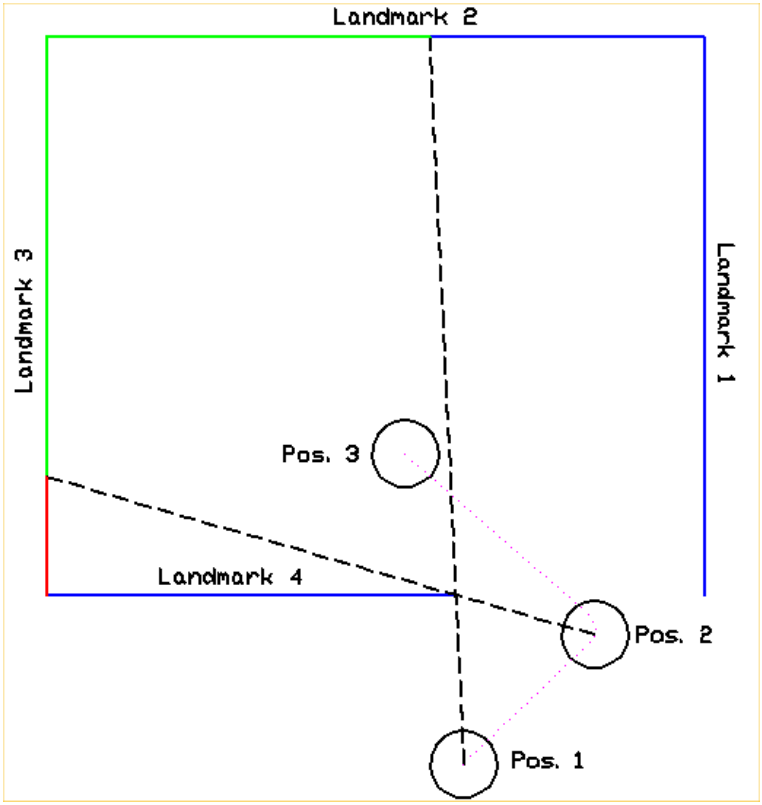

Figura 5.7: Exemple d'actualització del mapa

El mateix succeiria amb el landmark 3 quan el robot arribés a la posicó 3 de la figura. Aquesta doncs, és la missió del mètode a dissenyar, i que s'explica a continuació.

L'algoritme a dissenyar, ha de ser capaç d'actualitzar el mapa relitzant bàsicament tres accions, que són les següents:

- **Allargar landmarks del mapa.** Aquesta acció s'ha de realitzar quan hi ha un landmark detectat que és considerat *similar* amb un dels landmarks del mapa, i el primer dels dos, és més llarg per algun dels dos extrems, com succeeix a la figura 5.8, que seria el cas del landmark 2, per exemple, quan el robot passa de la posició 1 a la 2 de la figura 5.7. En aquest cas, el mapa haurà de ser actualitzat canviant el landmark existent per la fusió dels dos landmarks, el del mapa i el detectat. Més

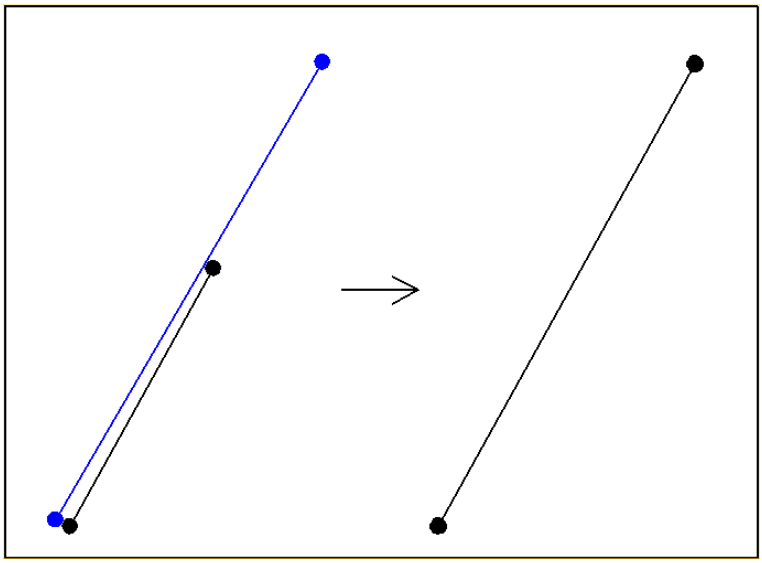

Figura 5.8: A l'esquerra tenim un landmark del mapa (en negre) i un detectat (en blau), que són *similars*. A la dreta apareix el landmark resultant de l'allargament.

endavant s'explicarà més clarament com es realitza la unió dels landmarks.

- **Unir landmarks del mapa.** Quan un dels landmarks detectats és *similar* a més d'un dels landmarks del mapa simultànemaent, com en el cas de la figura 5.9, aquests landmarks del mapa s'han de fusionar i passar a ser un de sol, que serà el resultant de la unió de tots landmarks, els del mapa i el detectat.

- **Afegir landmarks al mapa.** Els landmarks detectats en una iteració que no són associats amb cap landmark del mapa, es considera que no han estat detectats anteriorment i, per tant, cal afegir-los al mapa.

En alguns casos, per un mateix grup de landmarks pot haver-se de realitzar operacions d'allargament i d'unió simultànemaent.

Hi ha diverses maneres d'implementar un algoritme que realitzi les funcions descrites anteriorment. El que s'ha escollit per aquest projecte, es mostra a continuació.

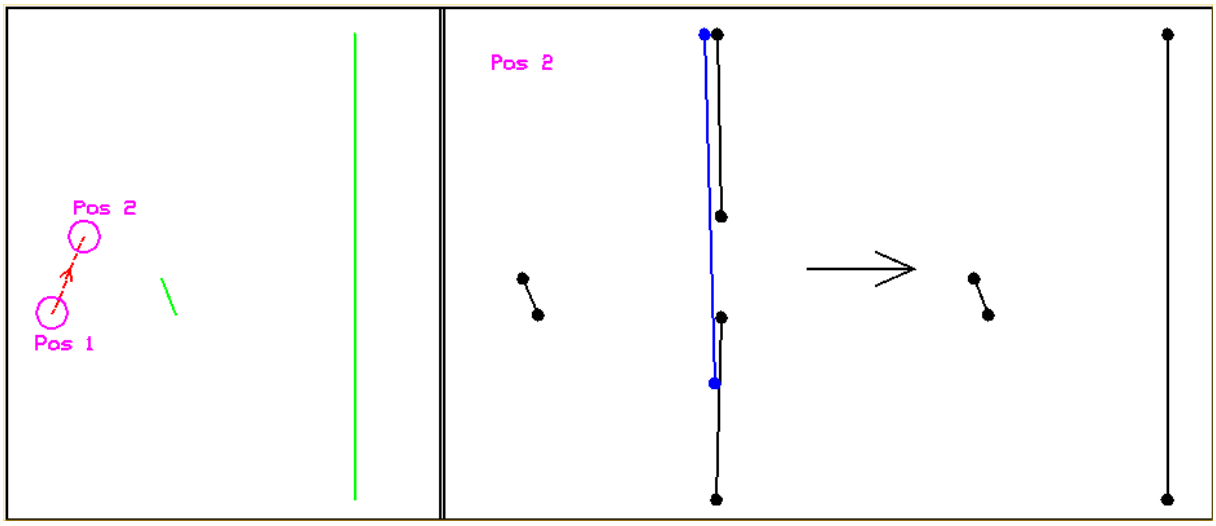

Figura 5.9: A l'esquerra s'observa un entorn per on es mou el robot, on els obstacles apareixen de color verd. Al centre , es veuen els landmarks del mapa (en negre) i els detectats (en blau) quan el robot arriba a la posició 2, mentre que a la dreta, apareixen els landmarks del mapa definitius després d'haver aplicat l'algoritme d'actualització

Actualització del mapa

per i **entre** 0 **i** *num_landmarks_detectats* **fer**

landmark_detectat_es_nou=**cert**

landmark_unio=*landmark_detectat*[i]

num_landmarks_associats=0

per j **entre** 0 **i** *num_landmarks_mapa* **fer**

si(Similar(*landmark_detectat*[i],*landmark_mapa*[j]))

landmark_unio=Unio(*landmark_unio*,*landmark_mapa*[j])

landmark_mapa_associats[*num_landmarks_mapa_associats*]=j

num_landmarks_mapa_associats=*num_landmarks_mapa_associats*+1

landmark_detectat_es_nou=**fals**

fsi

```
        fper
        si(landmark_detectat_es_nou)
            Inserir_landmark_mapa(landmark_detectat[i])
        sino
            per k entre 0 i num_landmarks_associats fer
                Eliminar_landmark_mapa(landmark_mapa[landmark_mapa_associats[k]])
            fper
            Inserir_landmark_mapa(landmark_unio,landmark_mapa)
        fsi
    fper
```

És important assenyalar que un cop acaba l'algoritme d'actualització dels landmarks del mapa, cal modificar el vector d'estat $x_{k+1|k+1}$, que en la següent iteració passarà a ser $x_{k|k}$.

A continuació s'explicaran les funcions "Unio" i "Inserir_landmark_mapa", que apareixen en l'algoritme utilitzat.

5.3.1 Unió de dos landmarks

Matemàticament, el resultat de la unió de dos conjunts és un altre conjunt amb els elements de tots dos conjunts, sense repetir els que pertanyen alhora a ambdós. Si s'expressa mitjançant una fórmula, tenim que

$$A \cup B = A + B - A \cap B \tag{5.46}$$

En la nostra funció Unio ens basarem en aquest concepte matemàtic, però modificant-lo i adaptant-lo al cas concret, ja que la unió real entre els conjunts de punts de dos landmarks com els de la figura 5.8, donaria com a resultat un conjunt de punts format pels de les dues rectes, que no és el resultat desitjat.

En la majoria dels casos, els landmarks que la funció ha d'unir són un landmark detectat i un landmark del mapa, ja corregit pel filtre de Kalman, i creat a partir de totes les iteracions acumulades pel mètode de construcció de mapes. Per aquest motiu, considerem que és molt més vàlid el landmark del mapa que el detectat, i per tant, la unió es farà entre el landmark del mapa i la projecció del landmark detectat sobre l'anterior. D'aquesta manera sí que es pot aplicar la fórmula d'unió 5.46. Això només succeeix en el cas que algun dels extrems, es consideri que no correspon al mateix punt físic segons els criteris establerts en l'apartat 5.2.5. En cas contrari , no es fa cap projecció i es pren l'extrem del landmark del mapa com a extrem del landmark projectat, d'aquesta manera el landmark resultant no serà modificat, com a mínim per aquell costat. En l'exemple de la figura 5.8, l'extrem de la dreta del landmark detectat, no s'hauria de projectar, mentre que el de la dreta sí.

En cas que el landmark detectat sigui *similar* a dos landmarks del mapa, com és el cas de la figura 5.9, l'algoritme que utilitzem uneix primer el landmark detectat amb un dels dos landmarks del mapa, i posteriorment, la unió d'aquests dos amb l'altre landmark del mapa, quedant finalment la unió entre tots els landmarks.

5.3.2 Inserció de landmarks dins el mapa

La inserció d'un landmark, vol dir que el landmark en qüestió passa a formar part dels landmarks del mapa. Això significa alhora que passa a formar part del vector d'estat i que, per tant, ha de tenir un espai dins de la matriu de covariança que caldrà inicialitzar.

Aquesta inicialització no pot ser igual que l'explicada a l'apartat 5.2.4, ja que allà s'explica com inicialitzar la matriu de covariances quan encara no hi ha errors acumulats, ni canvis de posició del robot. D'alguna manera, el que es pretén amb el mètode d'inicialització que es presenta, és que es tinguin en compte aquests factors. Aquest pas també és crucial pel fet que si no es fes aquesta inicialització, el filtre de Kalman perdria les propietats descrites a l'apartat 5.2.6.

A l'hora d'afegir un landmark a l'estat, es pot afegir de la següent forma:

$$\begin{pmatrix} x_{r,k+1|k+1} \\ x^{(1)}_{l,k+1|k+1} \\ \vdots \\ x^{(n)}_{l,k+1|k+1} \\ x^{(n+1)}_{l,k+1|k+1} \end{pmatrix} = G \begin{pmatrix} x_{r,k+1|k+1} \\ x^{(1)}_{l,k+1|k+1} \\ \vdots \\ x^{(n)}_{l,k+1|k+1} \\ z^{(n+1)}_{k+1} \end{pmatrix} \tag{5.47}$$

on la matriu G, segons les equacions 5.5 i 5.15 es pot escriure com

$$G = \begin{pmatrix} I_{3+4\times n} & 0 \\ -(H^{(n+1)}_{x_l})^{-1} H^{(n+1)}_{x_r} & (H^{(n+1)}_{x_l})^{-1} \end{pmatrix} \tag{5.48}$$

on les matrius $H^{(n+1)}_{x_r}$ i $H^{(n+1)}_{x_l}$ es calculen mitjançant les equacions 5.25 i 5.24.

Un cop trobada la matriu G, la nova matriu de covariança $P_{k+1|k+1}$ es calcula com

$$P_{k+1|k+1} = G \begin{pmatrix} P^{(ant)}_{k+1|k+1} & 0 \\ 0 & W^{(n+1)} \end{pmatrix} G^T \tag{5.49}$$

Si desenvolupem aquestes equacions, i considerem que tenim una matriu de covariança abans d'afegir el landmark n+1, que serà de l'estil

$$P_{k+1|k+1} = \begin{pmatrix} P_{r,k+1|k+1} & P^{(1)}_{rl,k+1|k+1} & \cdots & P^{(n)}_{rl,k+1|k+1} \\ (P^{(1)}_{rl,k+1|k+1})^T & P^{(1)}_{l,k+1|k+1} & \cdots & P^{(1,n)}_{ll,k+1|k+1} \\ \vdots & \vdots & \ddots & \vdots \\ (P^{(n)}_{rl,k+1|k+1})^T & (P^{(n,1)}_{ll,k+1|k+1})^T & \cdots & P^{(n)}_{l,k+1|k+1} \end{pmatrix} \tag{5.50}$$

La notació d'aquesta matriu és una mica complexa. Per entendre-la millor direm que els subíndex r indica que la matriu es refereix a la posició del robot; el subíndex rl indica que la matriu conté les covariances entre el robot i el landmark indicat al superíndex; el subíndex ll vol dir que la submatriu és lamatriu de covariances entre els dos landmarks mostrats al superíndex; i per últim, les submatrius amb subíndex l, són les matrius de covariança del landmark que marca el superíndex. Pel que fa al subínex *k+1—k+1*, com ja se sap, es refereix a una variable *a posteriori* de l'instant de temps k+1.

Finalment, per afegir un nou landmark, cal calcular les submatrius corresponents a aquest. I la manera de calcular-les és la següent:

$$P^{(n+1)}_{rl,k+1|k+1} = -P_{r,k+1|k+1}((H^{(n+1)}_{x_l})^{-1}H^{(n+1)}_{x_r})^T \tag{5.51}$$

$$P^{(i,n+1)}_{ll,k+1|k+1} = (H^{(i)}_{x_l})^{-1}H^{(i)}_{x_r}P_{r,k+1|k+1}((H^{(n+1)}_{x_l})^{-1}H^{(n+1)}_{x_r})^T \tag{5.52}$$

$$P^{(n+1)}_{l,k+1|k+1} = (H^{(n+1)}_{x_l})^{-1}H^{(n+1)}_{x_r}P_{r,k+1|k+1}((H^{(n+1)}_{x_l})^{-1}H^{(n+1)}_{x_r})^T + (H^{(n+1)}_{x_l})^{-1}W^{(i)}H^{(n+1)}_{x_l} \tag{5.53}$$

D'aquesta manera s'acaba la descripció del mètode utilitzat per la construcció de mapes i localització del robot. En els apartats següents es mostraran imatges de mapes obtinguts experimentalment amb el robot Marco i la interfície de navegació RoboGUI, comentats al capítol 3. També s'analitzaran aquests resultats obtinguts i s'explicaran les conclusions a les que s'ha arribat després de l'estudi fet en aquest projecte.

Capítol 6

Resultats

En aquest capítol es presenten els resultats experimentals obtinguts amb els diferents algoritmes explicats al llarg del document.

6.1 Algoritme de detecció de landmarks

A continuació es mostren els resultats experimentals de l'algoritme presentat al capítol 4 per a detectar landmarks. Aquest és la unió de l'IEPF, un filtre que determina quines rectes donades per l'IEPF són considerades landmarks, i un posterior procés d'acondicionament. El conjunt, té com a entrada un contorn de punts mesurats pel sensor làser en un entorn real, i com a sortida els landmarks detctats en aquella iteració.

Els valors utilitzats en l'experiment són els següents:

- El llindar d_s s'ha fixat en un valor de 5 cm.

- Pel que fa als criteris d'acceptació de rectes com a landmarks, s'ha considerat que una recta ha de tenir un mínim de 5 punts, una distància mínima de 40 cm, una distància màxima entre punts consecutius de 35 cm, i una distància màxima respecte del robot de 8 metres.

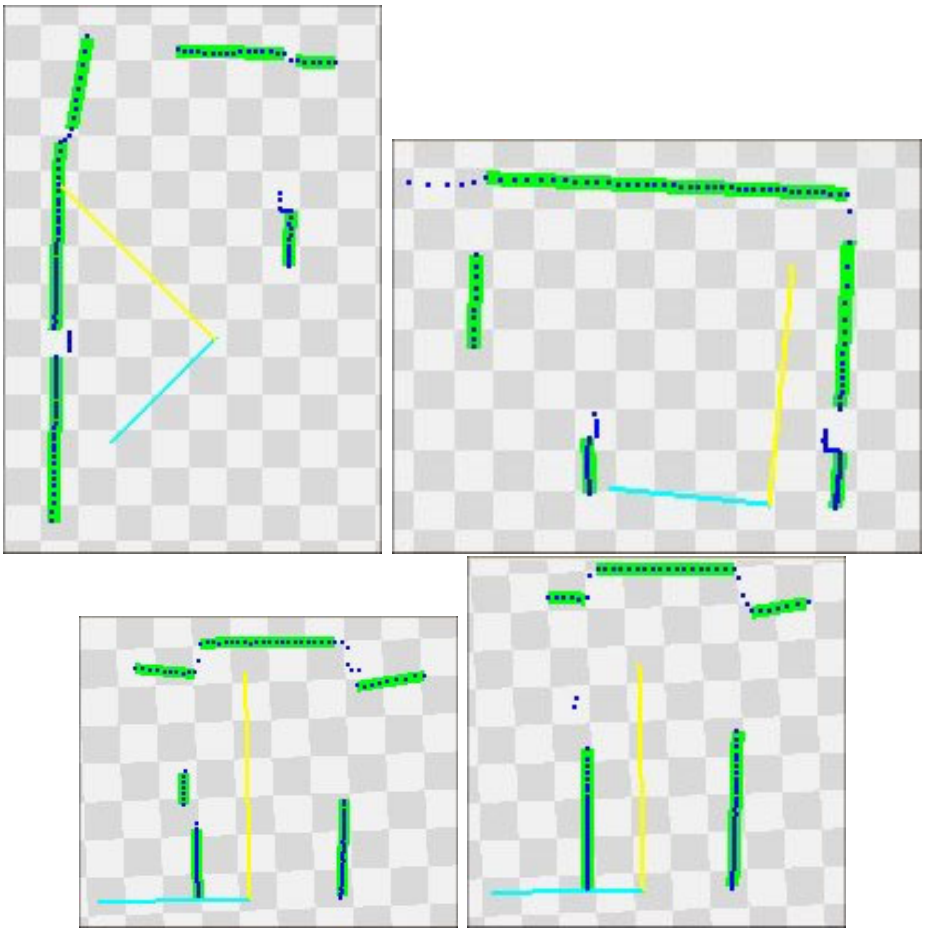

Figura 6.1: Aplicació del mètode de detecció de landmarks

Alguns exemples on s'aplica el mètode de detecció de landmarks, es poden veure a la figura 6.1

Els punts blaus representen les lectures del làser, i les línies de color verd són els landmarks extrets. Es pot observar que el mètode fuciona correctament i que detecta els landmarks principals. També es pot veure que en zones confoses no es detecta res, i en alguns casos on es veu alguna recta, no es considera landmark perquè és massa curta, i com ja s'ha explicat anteriorment donaria molts problemes en el tractament posterior.

En altres exemples, com els de la figura 6.2, es pot veure més clarament com l'algoritme

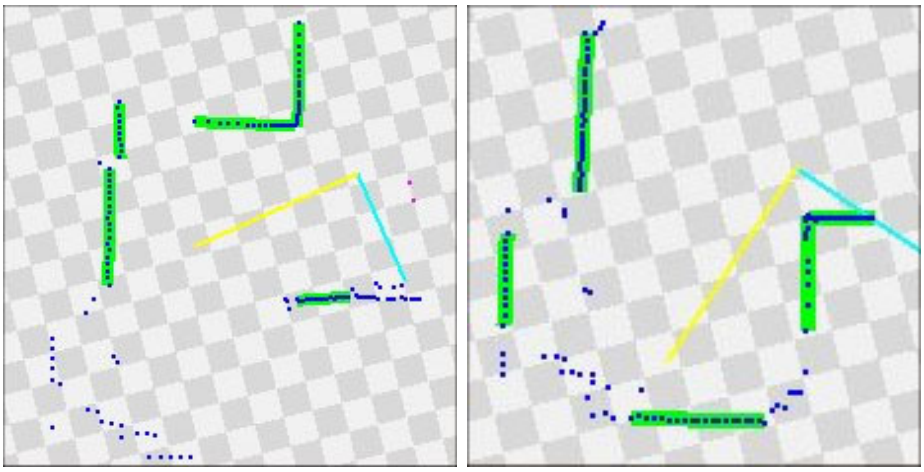

Figura 6.2: Aplicació del mètode de detecció de landmarks

no és capaç d'extreure cap landmark de zones on el contorn és molt difús. Això és bastant lògic, ja que l'algoritme busca línies rectes i en aquestes seria realment difícil trobar-ne. A més, sempre és molt més convenient pel global del procés, que no es detecti un landmark en una iteració, que no pas que se n'extregui un d'inexistent.

A més, a la imatge de l'esquerra de la figura 6.2 també s'observa un dels punts febles d'aquest mètode, que és la poca robustesa als punts anomenats *outliers*. A la zona de la dreta de la citada imatge, s'intueix una recta bastant clara, on hi ha uns 4 o 5 punts que se'n separen. Aquests punts són els *outliers*, que impedeixen que la recta sencera sigui detectada.

La realització de tots els càlculs necessaris per extreure els landmarks d'un contorn, empra uns 10ms de mitjana, que representa un temps molt baix envers altres algoritmes que realitzen la mateixa tasca.

Per tant, comprovem que aquest mètode de detecció de landmarks és bastant bo en la majoria dels casos, i tot i tenir algunes mancances, veurem que no són d'excessiva importància per a la construcció final del mapa de l'entorn del robot.

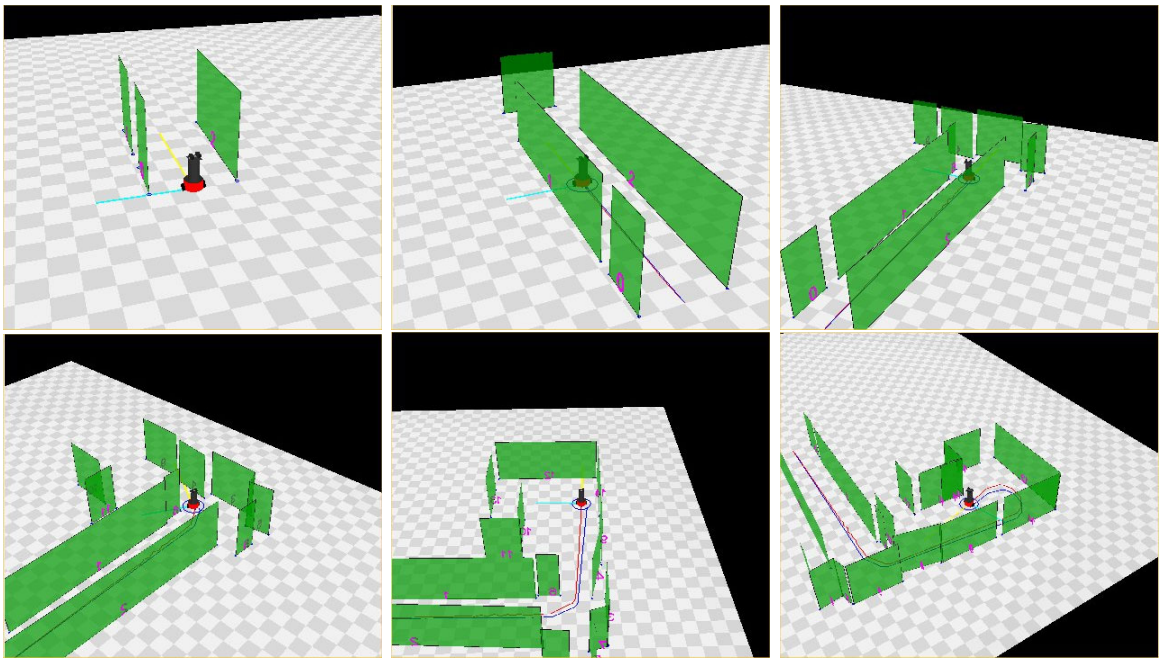

Figura 6.3: Exemple de mapa creat amb el mètode presentat en aquest projecte. En verd es veuen els landmarks del mapa; la línia vermella indica l'estimació de la trajectòria que ha seguit el robot, mentre que la línia blava mostra la trajectòria mesurada pels encòders i la odometria del robot.

6.2 Algoritme de localització del robot i construcció de mapes

En aquest apartat es mostren els resultats experimentals obtinguts amb l'algoritme de CML descrit a l'apartat 5. Això inclou l'algoritme d'associació de landmarks, la posterior localització del robot i l'actualització dels landmarks del mapa. Els resultats es presenten conjuntament, mostrant imatges de mapes obtinguts.

Un primer exemple de mapa obtingut amb la solució del CML proposada es pot observar a la figura 6.3. Es pot veure com el robot, a mesura que es va desplaçant va construïnt el mapa, afegint noves parets i retocant algunes de les ja presents a mesura que

s'apropa a elles. Cal recordar però, que tot i que es mostra un mapa tridimensional, el sensor làser només detecta obstacles en un sol pla, però per fer el mapa visualment més entenedor, s'ha optat per aquesta representació.

El mapa resultant obtingut en aquest cas és força satisfactori, i més que suficient per a moltes aplicacions que requereixin un mapa. Però el mètode, a més de construir un mapa, corregeix la posició del robot. Això es pot observar clarament a la figura 6.3, on la diferència que hi ha entre la trajectòria estimada pel mètode i la mesurada pel robot es va incrementant a mesura que el robot es mou. Aquest és l'error de *dead reckoning* i que provoca la deriva del robot. Es pot comprovar també que, com ja s'havia comentat, aquest error augmenta sobretot quan el robot gira, per tant el mètode de localització ha de ser especialment eficaç en aquests casos. Com s'observa al mapa de l'exemple, els objectius requerits es compleixen. Cal dir, però, que l'entorn utilitzat per fer la prova del cas anterior, tenia landmarks fixos (requisit indispensable per poder utilitzar aquest mètode), i el mobiliari era pràcticament inexistent.

Un altre exemple, també amb landmarks fixos però amb mobiliari d'oficina, es pot observar a la figura 6.4. En aquest segon cas, on l'entorn s'acosta més a un de real, es comprova que hi ha zones on no apareixen landmarks. Això és degut a que a les citades zones hi ha cadires i cables, que fan impossible la detecció de línies rectes, i com que aquestes són l'únic tipus de landmark detectable, el mètode no aconsegueix mapar aquella zona. No obstant, la part de localització del robot es desenvolupa correctament i es comprova un cop més la diferència entre la trajectòria estimada pel mètode i la mesurada pels sensors de moviment del robot. Això vol dir que el mètode de CML evita un cop més la deriva del robot.

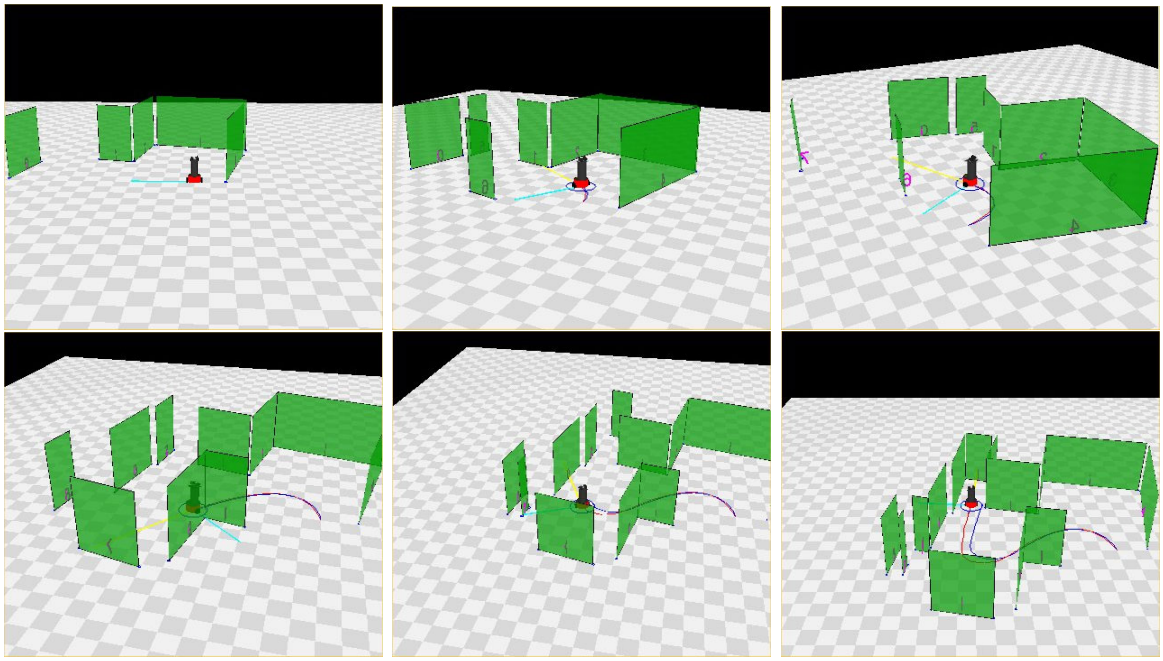

Figura 6.4: Un altre exemple de mapa creat amb el mètode de CML.

Capítol 7

Conclusions

7.1 Valoració dels resultats

Un cop vistos els resultats del mètode proposat en aquest projecte, es pot dir que s'han complert en bona mesura els objectius marcats a l'apartat 1.2. Tot seguit passem a comentar-los:

- El mètode genera un mapa de l'entorn que, en algunes ocasions és incomplet, però pot ser una representació suficient per a moltes aplicacions. Això és degut a que els únics landmarks que es detecten són els de tipus línia recta, que com s'explica al capítol 4, és degut bàsicament al tipus de sensor utilitzat.

- És molt robust a l'hora de realitzar el procés de localització i correcció de la posició del robot.

- Demostra una gran robustesa pel treball amb dades imprecises, ja que com es pot comprovar en els exemples del capítol anterior, la trajectòria mesurada pels sensors de moviment del robot difereix molt de la real, i alhora el làser, tot i tenir major precisió, també comet un cert error de mesura.

- Integra perfectament les dues fonts d'informació de què disposa, és a dir, els encòders

i el sensor de profunditat, basant-se en les dades aportades per ambdós per a la realització dels càlculs i estimacions, i alhora per reduir la incertesa i l'error de cadascun d'ells. En aquest sentit cal destacar a més, que en cap moment es produeixen ambigüitats amb la informació rebuda.

- El robot crea el mapa de forma totalment autònoma, sense necessitat d'ajuda externa. L'únic que es fa al principi és assignar-li la trajectòria desitjada, i a partir d'aquí el robot crea el mapa de l'entorn.

- L'últim objectiu és el de robustesa davant canvis de l'entorn, i aquest objectiu és l'únic que no s'ha pogut complir.

Per tant, es pot afirmar que tot i que aquest mètode no arriba a ser la solució *ideal* del problema del CML, sí que es pot considerar una bona aproximació.

A més, cal afegir que la programació en llenguatge C++ ha estat ralitzada modularment sempre que ha estat possible, de tal manera que, en cas de variacions del hardware, el codi es podria adaptar amb molta facilitat.

7.2 Recomanacions per a un treball futur

Per acabar aquest document, passem a citar les recomanacions per a futurs treballs. Aquestes corresponen bàsicament a la millora dels dos objectius que no compleix plenament el mètode proposat, i són les següents:

- Per a millorar el mapa construït, una recomanació seria la d'intentar treballar amb altres tipus de landmarks, com podrien ser arcs de circumferència o altres formes geomètriques, tot i que segurament seria difícil trobar algoritmes d'extracció eficients i ràpids per aquest tipus de landmarks.

- Seria també convenient equipar el sensor làser amb algun sistema que permetés obtenir informació de més d'un pla, per tal de poder fer mapes tridimensionals més

verídics, ja que els mostrats en el capítol anterior, en realitat són bidimensionals.

- Una altra manera de detectar altres tipus de landmarks seria la integració d'informació provinent de diferents tipus de sensors com podrien ser les càmeres. A més, es podrien fer mapes més realistes, ja que els landmarks detectats pel làser es podrien representar amb les imatges captades per aquestes.

- L'altra millora proposada seria la d'introduir paràmetres temporals als landmarks de tal manera que el mètode pogués actuar convenientment davant de canvis que es produeixin a l'entorn.

- Per tal de fer el robot encara més autònom, es podria integrar algun sistema de planificació de trajectòries, de tal manera que el robot sabés per on moure's per tal d'anar creant el mapa.

Bibliografia

[1] J. Andrade-Cetto, *The Kalman Filter.* Doc. Tèc. IRI-DT-02-01 Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Març 2002.

[2] J. Andrade-Cetto, A. Sanfeliu, *Concurrent map building and localization on indoor dynamic environments.* International Journal of Pattern Recognition and Artificial Intelligence, vol. 16, núm. 3, pàgs. 361-374, Maig 2002.

[3] J. Andrade-Cetto, A. Sanfeliu, *Concurrent map building and localization with landmark validation.* In Proceedings of the 16th IAPR International Conference on Pattern Recognition, vol. 2, pàgs. 693-696, IEEE Computer Society, Quebec, Agost 2002.

[4] J. Andrade-Cetto, A. Sanfeliu, *Environment learning for indoor mobile robots.* PhD Thesis, Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Gener 2003.

[5] N. Ayache, O. Faugeras, *Mantaining representations of the environment of a mobile robot.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 5, núm. 6, pàgs 804-819, Desembre 1989.

[6] M. Bayouth, P. Koopman, *Functional evolution of an automated highway system for incremental deployment.* Transportation Research Record, vol. 1651, pàgs 80-88, Juny 1998.

[7] J.A. Castellanos, J.M.M. Montiel, J. Neira, J.D. Tardós, *The SPMap: A probabilistic framework for simultaneous localization and map building.* IEEE Transactions on Robotics and Automation, vol. 15, núm. 5, pàgs. 948-952, Octubre 1999.

[8] E.D. Dickmanns, *Recursive state estimation.* Lecture Notes, California Institute of Technology, Primavera 1996.

[9] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, M. Csorba, *A solution to the simultaneous localisation and map building.* IEEE Transactions on Robotics and Automation, vol. 17, núm. 3, pàgs. 229-241, Juny 2001.

[10] D.H. Douglas, T.K. Peucker, *Algorithm for the reduction of the number of points required to represent a digitalized line or its caricature.* Canadian Cartographer, vol. 10, núm. 2, pàgs. 112-122, Desembre 1973.

[11] D. Fox, W. Burgard, S. Thrun, *Markov localization for mobile robotsin dynamic environments.* Journal of Artificial Intelligence Research, vol. 30, pàgs 391-427, Novembre 1999.

[12] J. Hershberger, J. Smoeyink, *Speeding up the Douglass-Peucker line simplification algorithm.* Proceedings of the 5th International Symposium on Spatial Data Handling, pàgs. 134-143, Charleston, Agost 1992.

[13] F. Moreno, J. Andrade-Cetto, A. Sanfeliu, *Desenvolupament d'un sistema d'estereovisió per un robot mòbil.* Doc. Tèc. IRI-DT-01-01, Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Març 2001.

[14] S. Thrun, W. Burgard, D. Fox, *A probailistic approach to concurrent mapping and localization.* Machine Learning and Robot, vol. 31, núm. 1, pàgs 29-53, 1998.

[15] G. Welch, G. Bishop, *SCAAT: incremental tracking with incomplete information.* In T. Whitted, ed. Computer Graphics, Proceedings of the ACM SIGGRAPH Conference, pàgs 333-344, ACM Press, Los Angeles, Agost 1997.

Apèndix A

Pressupost

A.1 Dades de partida

En aquest primer apartat s'especifiquen els costos per hora establerts per a la confecció del pressupost; d'una banda, el cost per hora del personal i, de d'altra, el cost per hora d'utilització dels equips.

A.1.1 Cost per hora del personal

S'estima que el cost per hora del personal dedicat al desenvolupament del projecte està resumit en la següent taula:

Analista	45 euros /hora
Programador	35 euros /hora
Operador	20 euros /hora

A.1.2 Cost per any d'utlilització dels equips

Amb la finalitat de calcular el cost per any d'utilització dels equips s'ha de tenir present: el cost del personal del Centre de Càlcul (CCIRI) de l'IRI (Institut de Robòtica i Infromàtica Industrial), els contractes de manteniment i, finalment, l'amortització dels equips.

Cost de l'equip de visió

L'equip de visió conjuntament amb el robot està format per:

- Robot Pioneer 2-DX, amb les llibreries Saphira. El seu valor està estimat en 10.460 euros.
- Ordinador PC Pentium II 300 MHz, 128MB de RAM, 10GB de disc dur (HD) i monitor de 17". El seu preu està estimat en 1.000 euros.
- Sensor làser de profunditat Leuze Lumiflex RS4 Rotoscan, amb un preu estimat de 3.000 euros.
- 2 transmissors de ràdio Ethernet OTC Telecom Air Ezy 2400, amb una valoració total de 150 euros.
- Cablejat i connectors en general. El seu preu s'estima en 60 euros.

Resumint, el cost total de l'equipament és:

Robot	10.460
Ordinador	1.000
Sensor làser	3.000
Transmissors ràdio Ethernet	150
Cablejat i connectors	60
Cost total de l'equip	**14.670 euros**

Considerant l'amortització de l'equip en 5 anys, amb uns interessos del 5%, l'anualitat de l'amortització és:

$$COST(Equip) = 14.670 \times \frac{0,05 \times (1,05)^5}{(1,05)^5 - 1} = 3.388 euros/any$$

Cost de manteniment

El cost de manteniment anual de l'equip s'estima en un 10% del valor total de compra:

$$COST(Manteniment) = 14.670 \times 0,1 = 1.467 euros/any$$

Despeses diverses

En aquestes despeses s'inclouen el consum elèctric i possibles reparacions:

$$COST(Divers) = 240 euros/any$$

Cost per any d'utilització dels equips

Tenint en compte els costos anteriors, es calcula un cost anual d'utilització dels equips de:

Cost de l'equip de visió	3.388 euros /any
Cost de manteniment	1.467 euros /any
Despeses diverses	240 euros /any
Cost total anual d'utilització dels equips	**5.095 euros /any**

Si s'estima que treballen 45 setmanes a l'any, un total de 40 hores setmanals i amb un factor d'utilització de l'equip d'un 80% obtenim, finalment:

Cost total per hora d'utilització dels equips = 3.54 euros /hora

A.2 Cost de desenvolupament

Tenint presents els preus de l'apartat anterior, es calcula el cost de desenvolupament del projecte, equip i despeses diverses. No s'inclou el cost del inmobilitzat de l'IRI (Institut de Robòtica i Informàtica Industrial), sinó només els que fan referència als recursos humans i equips informàtics.

A.2.1 Cost del personal

Aproximadament, s'estima que el temps emprat per al desenvolupament d'aquest projecte ha estat de 8 mesos, és a dir, 30 setmanes tenint present que un any té 45 setmanes laborables. La dedicació mitjana ha estat de 40 hores setmanals. El temps utilitzat es pot classificar en: 30% del temps com analista, 60% com a programador i un 10% com a operador.

Cost Analista	360 h x 45 euros /hora =	16.200 euros
Cost Programador	720 h x 35 euros /hora =	25.200 euros
Cost Operador	120 h x 20 euros /hora =	2.400 euros
Cost Personal		**43.800 euros**

A.2.2 Cost d'utilització dels equips

Del total del temps dedicat, un 70% ha estat a la implementació de programes i utilització de l'equip de visió. Això resulta:

$$Temps\ d'utilització\ de\ l'equip = 30\ setmanes \times 40\ hores/setmana \times 0,7 = 840h$$

Cost d'utilització dels equips = 840h x 3.54 euro /hora = 2.974 euros

A.2.3 Despeses diverses

Es consideran les despeses associades als següents conceptes:

Documentació	180 euros
Paper i impressió	120 euros
Despeses diverses	**300 euros**

A.2.4 Cost total de desenvolupament

El cost total de desenvolupament del projecte és el resultat de la suma dels tres darrers costos:

Cost Personal	43.800 euros
Cost Utilització equips	2.974 euros
Despeses diverses	300 euros
Cost total desenvolupament	**47.074 euros**

Apèndix B

Codi

B.1 Construcció de mapes i localització

B.1.1 VSWalls.cpp

```
// VSWalls.cpp: implementació de la classe CVSWalls
//////////////////////////////////////////////////////////////////////

/*************************************************************************
*                                                                        *
* NOM DE L'ARXIU:     VSWalls.cpp                                        *
* DATA:               Març 2002                                          *
* VERSIÓ:             0.01                                               *
* OBJECTIU:           Construeix un mapa de l'entorn a partir de les dades *
*                     proporcionades per un sensor làser                 *
* CODI ORIGINAL DE:   Albert Checa Puimedon i Juan Andrade-Cetto          *
*                     (C) Institut de Robòtica Industrial                *
*                     Universitat Politècnica de Catalunya               *
* ALGORITME BASAT EN: Duda & Hart 1973, i la nostra intuició             *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                 *
*                                                                        *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                     *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                        *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                        *
*************************************************************************/

#include "stdafx.h"
```

```
#include "RoboGUI.h"
#include "matrices.h"
#include "VSWalls.h"
#include "iepf.h"
#include "rs4.h"
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

#define SUPERPEQUENO 1e-10

double expuntos[133][2];
double expuntos2[133][2];
double posicion [3];
double ** Prr;
double ** Prrold;

FILE *dumpfile;
int iter;

////////////////////////////////////////////////////////////////////
// Algoritme de construcció de mapes i localització
////////////////////////////////////////////////////////////////////

CVSWalls::CVSWalls()
{
type = "Walls";
llista=fopen("scan_acceso_iri.dat","r");

//Inicialització de les variables del filtre de Kalman
Prr=alloc_mat_d(3,3);
matriz0(Prr,3,3);
Prrold=alloc_mat_d(3,3);
matriz0(Prrold,3,3);
xk1k.vec=alloc_vec_d(803);
xk1k.n=3;
xk1k1.vec=alloc_vec_d(803);
xk1k1.n=3;
zk1k.vec=alloc_vec_d(803);
zk1k.n=0;
ezk1k.vec=alloc_vec_d(803);
ezk1k.n=0;
pkk.mat=alloc_mat_d(803,803);
pkk.nf=3;
pkk.nc=3;
identity(pkk.mat,pkk.nf);
posicion[0]=0;
posicion[1]=0;
```

```
posicion[2]=0;
ax=0;ay=0;ath=0;
u.vec=alloc_vec_d(3);u.n=3;
eposr.vec=alloc_vec_d(3);eposr.n=3;
for(int i=0;i<3;i++) eposr.vec[i]=0;
iter=0;

double a,b;

a= .5;  // sigma inicial del robot x,y
b= .1;  // sigma inicial del robot theta

ar_xy = 20; // sigma x,y robot = 3cm
ar_alpha=5*PI/180; // sigma theta robot = 3 grados

pkk.mat[0][0]=a*a;
pkk.mat[1][1]=a*a;
pkk.mat[2][2]=b*b*(PI/180*PI/180);

az=2; //0.5; // sigma paret = 5cm, desv. estàndard de l'error del sensor
winic = 2; //0.5; // sigma inicial pels landmarks

dumpfile=fopen("matrices_kalman.dat","w");
}

CVSWalls::~CVSWalls()
{
fclose(llista);
fclose(dumpfile);
release_mat_d(pkk.mat);
release_vec_d(xk1k.vec);
release_vec_d(xk1k1.vec);
release_vec_d(ezk1k.vec);
release_vec_d(zk1k.vec);
release_vec_d(u.vec);
release_vec_d(eposr.vec);
}

int CVSWalls::Init()
{
return 0;
}

int *CVSWalls::FindLandmarks(void *args)
{
double p1[2],p2[2],dist, alpha,d1;
int s,e,i,m,n;
pared p;
punto punto0;
int * pp;
```

```
int tt;

Crs4 *ptrRS4; //objetcte làser
Ciepf *ptrIEPF; // objetcte IEPF
ptrRS4 = (Crs4 *)args;

ptrIEPF = new Ciepf(ptrRS4->contour);

//Lectura de les dades del sensor làser
if(!ReadLaserData(ptrIEPF))
{
tt=0;
pp=&tt;
return pp;
}

//Vector u de desplaçament del robot

u.vec[0]=x-ax;
u.vec[1]=y-ay;
u.vec[2]=th-ath;

//Vector posició del robot xr

posicion[0]=x;
posicion[1]=y;
posicion[2]=th;

//Omplim les coordenades del vector d'estat de Kalman corresponents a la posicio del robot

for(m=0;m<3;m++)
xk1k.vec[m]=posicion[m];


//Aplicació algoritme IEPF
ptrIEPF->rectas(ptrIEPF);

//Borrar llista de landmarks dibuixats a l'entorn Open GL
m_pListDetLmks->RemoveAll();

punto0[0]=0;
punto0[1]=0;

//Conversió coordenades làser locales a coordenades OpenGL globals

for(m=0;m<133;m++)
{
//Pas de m a cm y canvi de coord x per y
expuntos[m][0]=ptrIEPF->puntos[m][1]*100;
expuntos[m][1]=ptrIEPF->puntos[m][0]*100;
```

```
//Variable per a omplir l'arxiu de dades escanejades pel làser
expuntos2[m][0]=ptrIEPF->puntos[m][1]*100;
expuntos2[m][1]=ptrIEPF->puntos[m][0]*100;

//angle entre eixos làser i OpenGL
alpha = atan2(expuntos[m][1],expuntos[m][0])+th;

//distància del punt actual al centre de coord làser
d1 = Distancia(expuntos[m],punto0);

//Conversió de coordenades locals a globals
expuntos[m][0]=d1*cos(alpha)+x;
expuntos[m][1]=d1*sin(alpha)+y;
}

//Borrado de la lista de landmarks detectados en la iteración actual
detLmk.Reset();

//Comptador de parets detectades a la iteración actual
Wallcounter=0;

for(Wallindex=0;Wallindex<ptrIEPF->ct;Wallindex++)
{
s=ptrIEPF->lineas[Wallindex]; //s=índex del punt inicial de la recta
e=ptrIEPF->lineas[Wallindex+1]; //e=índex del punt final de la recta

if((e-s)>1)
{
//Partir les parets que tinguin punts consecutius separats més d'una certa distància
e=RefinarPared(s,e,ptrIEPF);
}

//Obtenir totes les dades de la recta (constants b1 i b0,distància, r2...)
p=TratarPared(s,e,ptrIEPF->puntos);

//Les rectes que cumpleixen unes característiques les considerems parets i les afegim
//a la llista de parets detectades a la iteració actual
if(p.accept)
{
newLmk= new CDetectedLandmark(Wallcounter,
CrearPunt3D(p.p1[0],p.p1[1],0),
CrearPunt3D(p.p2[0],p.p2[1],0),
CrearPunt2D(0,0));

newLmk->p=p;
if(p.p1[0]<p.p2[0])
{
newLmk->min=p.p1[0]; //min=menor valor de coord x de un punto de la recta
newLmk->max=p.p2[0]; //max=mayor valor de coord x de un punto de la recta
```

```
}
else
{
newLmk->min=p.p2[0];
newLmk->max=p.p1[0];
}

newLmk->type = WALL_LDK;
detLmk.Insert(*newLmk);
Wallcounter++;
}

}

fprintf(dumpfile,"\n\n*************ITERACION %d*************\n\n\n",iter);
fprintf(dumpfile,"Número de landmarks al empezar: %d\n\n",oldLmk.numofWalls);

if(oldLmk.numofWalls>0)
{
LlenarEstado();
//Filtre de Kalman
Kalman();
ax=xk1k1.vec[0];
ay=xk1k1.vec[1];
ath=xk1k1.vec[2];
}

//Actualització del mapa
ActualizarLmks();




fprintf(dumpfile,"Número de landmarks al acabar: %d\n",oldLmk.numofWalls);

iter++;

//Dibuixar parets a la representació en 3D de l'aplicació
Dibujar();

// Destrucció de l'objecte IEPF
ptrIEPF->~Ciepf();

tt=1;
pp=&tt;
return pp;
}


pared CVSWalls::TratarPared(int ini, int fin, xycontorno puntos)
```

```
{
pared p;
int k,n;
double xi_x,yi_y,mah=0,a=0,b=0,c=0,avgx,avgy,/*b0,b1,*/b2;
double sumx=0,sumy=0,sumxy=0,sumx2=0,sumy2=0,alpha1, alpha2, d1, d2,dmax;
double SCr,SCd,dactual=0,measi,est_dp2p,alfa,beta,gamma,dif,aux;
punto centro,next_pt,minsq_pt,punto0;

centro[0]=0;
centro[1]=0;

n=fin-ini+1;
p.maxdp2p=0;

//Càcul dels paràmetres de la recta segons el mètodo dels mínims quadrats
for(k=ini;k<=fin;k++)
{
sumx = sumx + puntos[k][0];
sumy = sumy + puntos[k][1];
sumxy = sumxy + puntos[k][0] * puntos[k][1];
sumx2 = sumx2 + puntos[k][0] * puntos[k][0];
sumy2 = sumy2 + puntos[k][1] * puntos[k][1];
if(k!=ini)
dactual=Distancia(puntos[k-1],puntos[k])*100;

if(dactual>p.maxdp2p)
p.maxdp2p=dactual;
}

avgx=sumx/n;
avgy=sumy/n;

//Càlcul de les constants de la regressió lineal
if((n * sumx2 - sumx * sumx) >= (n * sumy2 - sumy * sumy)) {
p.b1 = (n * sumxy - sumx * sumy) / (n * sumx2 - sumx * sumx);
p.b0 = (sumy - p.b1 * sumx) / n;
b2 = puntos[ini][1]+(1/p.b1)*puntos[ini][0];

//Punts extrems de la recta en coordenades locals del làser
p.z1[0]=(b2-p.b0)/(p.b1+1/p.b1);
p.z1[1]=p.b0+p.b1*p.z1[0];

b2 = puntos[fin][1]+(1/p.b1)*puntos[fin][0];

p.z2[0]=(b2-p.b0)/(p.b1+1/p.b1);
p.z2[1]=p.b0+p.b1*p.z2[0];

alfa=atan(p.b1);
if((p.z2[0]-p.z1[0]) < 0 )
alfa+PI;
```

```
}
// Quan la línea es quasi vertical, el denominador del càlcul de la regressió
//és molt proper a 0, per la qual cosa cal fer un tractament especial, fent
//un canvi de coordenades
else
{
p.b1 = (n * sumxy - sumx * sumy) / (n * sumy2 - sumy * sumy);
p.b0 = (sumx - p.b1 * sumy) / n;
b2 = puntos[ini][0]+(1/p.b1)*puntos[ini][1];

//Punts extrems de la recta en coordenades locals del làser
p.z1[1]=(b2-p.b0)/(p.b1+1/p.b1);
p.z1[0]=p.b0+p.b1*p.z1[1];

b2 = puntos[fin][0]+(1/p.b1)*puntos[fin][1];

p.z2[1]=(b2-p.b0)/(p.b1+1/p.b1);
p.z2[0]=p.b0+p.b1*p.z2[1];

// cal recalcular els valors de b0 i b1 per tal que la recta quedi vertical
if((p.b1*p.b1)<SUPERPEQUENO) {
p.b0 = -p.b0/p.b1;
p.b1 = 1/p.b1;
}
else {
p.b0 = -p.b0/SUPERPEQUENO;
p.b1 = 1/SUPERPEQUENO;
}

// ara si que es pot calcular alfa
alfa=atan(p.b1);
if((p.z2[0]-p.z1[0]) < 0 )
alfa+PI;
}



//Càlcul dels extrems del landmark z1 y z2, en coordenades locals
aux=p.z1[0];
p.z1[0]=p.z1[1]*100;
p.z1[1]=aux*100;
aux=p.z2[0];
p.z2[0]=p.z2[1]*100;
p.z2[1]=aux*100;

//Càlcul de la distància de la recta
p.dist=Distancia(p.z1,p.z2);

alpha1 = atan2(p.z1[1],p.z1[0])+th;
alpha2 = atan2(p.z2[1],p.z2[0])+th;
```

```
d1 = Distancia(p.z1,centro);
d2 = Distancia(p.z2,centro);

local2global(p.z1[0],p.z1[1],&(p.p1[0]),&(p.p1[1]));
local2global(p.z2[0],p.z2[1],&(p.p2[0]),&(p.p2[1]));

p.b1=(p.p2[1]-p.p1[1])/(p.p2[0]-p.p1[0]);
p.b0=p.p1[1]-p.b1*p.p1[0];


if(d1>d2)
dmax=d1;
else
dmax=d2;

//Condicions que posem a una recta donada per l'IEPF per a que sigui considerada lmk:
//1- Tingui más de 40 cm de longitud
//2- La distància entre punts consecutius sigui menor que 35cm
//3- Que no estigui més lluny de 6m del robot
//4- Que tingui com a mínim 5 punts

if(p.dist>40 && (p.maxdp2p<35) && (dmax<600) && ((fin-ini)>4))
p.accept=true;
else
p.accept=false;


return p;


}

//Funció per calcular la distància entre dos punts
double CVSWalls::Distancia(punto p1,punto p2)
{
double d;
d=sqrt((p2[0]-p1[0])*(p2[0]-p1[0])+(p2[1]-p1[1])*(p2[1]-p1[1]));
return(d);
}

//Funció per partir rectes que tinguin dos punts consecutius que
//distin més de 35cm
int CVSWalls::RefinarPared(int ini,int fin,Ciepf * ptrIEPF)
{
int brkpt;
double dp2p;
bool salir;

salir=false;
```

```
for(int k=ini;(k<fin && !salir);k++)
{
dp2p=Distancia(ptrIEPF->puntos[k],ptrIEPF->puntos[k+1]);
if(dp2p>0.35)  //en metres
{
brkpt=k;
for(int n=ptrIEPF->ct;n>Wallindex;n--)
{
ptrIEPF->lineas[n+1]=ptrIEPF->lineas[n];
}
if(k==ini)
{
ptrIEPF->lineas[Wallindex+1]=brkpt+1;
ptrIEPF->ct++;
return (brkpt+1);
}else
{
ptrIEPF->lineas[Wallindex+1]=brkpt;
ptrIEPF->ct++;
return (brkpt);
}

}
}
return(fin);
}

//Inicialització de les variables que falten del filtre de Kalman
void CVSWalls::InicializarDatos()
{
bool stop=false;

detLmk.index=0;
stop=detLmk.Empty();
if(!stop)
{
while(!stop)
{
detLmk.Data[detLmk.index].marca=false;
stop=detLmk.isLast();
detLmk.index++;
}
detLmk.index=0;
}

oldLmk.index=0;

}

void CVSWalls::Dibujar()
```

```
{
bool stop;

oldLmk.index=0;
stop=oldLmk.Empty();
if(!stop)
{
while(!stop)
{
oldLmk.Data[oldLmk.index].m_id=oldLmk.index;
m_pListDetLmks->AddTail(&oldLmk.Data[oldLmk.index]);
if(!oldLmk.isLast())
oldLmk.index++;
else
stop=true;
}
oldLmk.index=0;
}
}

bool CVSWalls::Similar(pared det,pared old,float umbral)
{
double alfa1,alfa2,other,y11,y12,y21,y22,b02,aux,disterr,angerr,maxd;
double d [4];
bool ok,p1dentro;

p1dentro=false;

//Considerem una paret similar a una altra quan tinguin una diferència en l'angle
//menor que "angerr" i a més algun dels seus punts extrems estigui dins d'un rectangle
//al voltant de la recta deixant "disterr" de marge per cada costat. Addicionalment
//es considera com a zona d'acceptació dos semicircumferencies a cada extrem del
//rectángle de diàmetre disterr.Cal aclarir que aquesta funció pot no donar el mateix
//invirtint les rectes d'ordre ja que no és commutativa.Només mira si "det" está dins
//de la zona al voltant de "old".Per això al fer la comparació a les funcions d'associar
//s'ha de posar Similar(1,2) || Similar(2,1)

//Defineix el rectangle que té per base la distància de la recta i altura 2*disterr
disterr=umbral; //en cm
disterr=20.0;
//Diferència admesa entre l'angle de las dues rectes
angerr=5*PI/180; //en rad

//Obtenim l'angle de les rectes
alfa1=atan(det.b1);
alfa2=atan(old.b1);

//Corregim per si en algun cas al sumar o restar angerr, donem una vuelta de més
//i l'altre angle és el mateix però amb una volta menys
if(alfa1>0)
```

```
other=alfa1-PI;
else
other=alfa1+PI;

//Comprovem que compleix la condició d'angle, sino sortim
if(!(((alfa1>alfa2-angerr) && (alfa1<alfa2+angerr)) ||
((other>alfa2-angerr) && (other<alfa2+angerr))))
return false;

//Comprovem que algun dels extrems està dins del rectangle + semicircumferències

//Calculem el valor que tindria un punt amb x=p1[0] sobre cadascuna de les rectes
//del rectangle i posteriorment comprovem que p1[1] està dins dels rangs d'aquests
//valors.y11,y12 són les coordenades i de tall amb els costats paral·lels a la recta i
//y21,y22 són com els costats perpendiculars
y11=old.b1*det.p1[0]+(old.b0+disterr/cos(alfa2));
y12=old.b1*det.p1[0]+(old.b0-disterr/cos(alfa2));
b02=old.p1[1]-1/old.b1*old.p1[0];
y21=1/old.b1*det.p1[0]+b02;
b02=old.p2[1]-1/old.b1*old.p2[0];
y22=1/old.b1*det.p1[0]+b02;

//Posem sempre el valor menor a y11 i y21
if(y11>y12)
{
aux=y11;
y11=y12;
y12=aux;
}

if(y21>y22)
{
aux=y21;
y21=y22;
y22=aux;
}

//Calculem la distància dels extrems d'una recta amb els de l'altra que ens servirà
//per determinar si els punts es troben dins alguna de las semiesferes, en cas
//que no es trobessin al rectangle
d[0]=Distancia(det.p1,old.p1);
d[1]=Distancia(det.p1,old.p2);
d[2]=Distancia(det.p2,old.p1);
d[3]=Distancia(det.p2,old.p2);

maxd=disterr/2+1;

//Posem a d[0] la distància més petita
for(int in=0;in<2;in++)
{
```

```
if(d[in]<maxd)
maxd=d[in];
}

//Realitzem la comprovació.Si la compleix ja no fa falta seguir
if(((det.p1[1]>y11) && (det.p1[1]<y12) && (det.p1[1]>y21) &&
(det.p1[1]<y22)) || (maxd<(disterr/2)))
{
if((d[0]>20) && (d[1]>20))
{
return true;
}
else
p1dentro=true;
}

//Si no compleix per un dels dos extrems mirem l'altre
y11=old.b1*det.p2[0]+(old.b0+disterr/cos(alfa2));
y12=old.b1*det.p2[0]+(old.b0-disterr/cos(alfa2));
b02=old.p1[1]-1/old.b1*old.p1[0];
y21=1/old.b1*det.p2[0]+b02;
b02=old.p2[1]-1/old.b1*old.p2[0];
y22=1/old.b1*det.p2[0]+b02;

if(y11>y12)
{
aux=y11;
y11=y12;
y12=aux;
}

if(y21>y22)
{
aux=y21;
y21=y22;
y22=aux;
}

for(in=2;in<4;in++)
{
if(d[in]<maxd)
maxd=d[in];
}

if(((det.p2[1]>y11) && (det.p2[1]<y12) && (det.p2[1]>y21) &&
(det.p2[1]<y22)) || (maxd<(disterr/2)))
{
if(((d[2]>20) && (d[3]>20)) || p1dentro)
{
return true;
```

```
}
}


return false;
}

int CVSWalls::ReadLaserData(Ciepf * ci)
{
double num;
double num2;

for(int t=0;t<133;t++)
{
if(feof(llista))
return 0;
fscanf(llista,"%lf",&num);
ci->puntos[t][0]=num;
fscanf(llista,"%lf",&num);
ci->puntos[t][1]=num;
}
fscanf(llista,"%lf",&num2);
x=num2+eposr.vec[0];
fscanf(llista,"%lf",&num2);
y=num2+eposr.vec[1];
fscanf(llista,"%lf",&num2);
th=num2+eposr.vec[2];

return 1;
}

//Aplica el filtre de Kalman
void CVSWalls::Kalman()
{
matriz pk1k,K,vr,Fr,Fv;
vector sr,v0,sr2;

int i;

v0.vec=alloc_vec_d(3);
v0.n=3;
sr.vec=alloc_vec_d(3);
sr.n=3;
sr2.vec=alloc_vec_d(3);
sr2.n=3;

v0.vec[0]=0.0;
v0.vec[1]=0.0;
v0.vec[2]=0.0;
```

```
sr2.n=3;
sr2.vec[0]=ar_xy*ar_xy;
sr2.vec[1]=ar_xy*ar_xy;
sr2.vec[2]=ar_alpha*ar_alpha;

vr.mat=alloc_mat_d(3,3);
vr.nc=3;
vr.nf=3;
diagonal(sr2.vec,vr.mat,sr2.n);
ImprimirMatriz(dumpfile,vr.mat,0,0,vr.nf,vr.nc,"Vr");

Fr.mat=alloc_mat_d(3,3);
Fv.mat=alloc_mat_d(3,3);
LlenarFvFr(posicion,sr.vec,Fr.mat,Fv.mat);
Fr.nc=3;
Fr.nf=3;
Fv.nc=3;
Fv.nf=3;

pk1k=Calcularpk1k(Fv,Fr,vr);

Copiarmatriz(pk1k.mat,Prr,0,0,3,3,0,0);

ImprimirMatriz(dumpfile,pk1k.mat,0,0,pk1k.nf,pkk.nc,"Pk1k");

K=CalcularK(pk1k,az);

ImprimirVector(dumpfile,xk1k1.vec,0,xk1k1.n,"xk1k1");

release_vec_d(v0.vec);
release_vec_d(sr.vec);
release_vec_d(sr2.vec);
release_mat_d(vr.mat);
release_mat_d(Fr.mat);
release_mat_d(Fv.mat);
release_mat_d(pk1k.mat);
release_mat_d(K.mat);
}

void CVSWalls::LlenarFvFr(double *v, double * sr, double ** Fr, double ** Fv)
{
double du,dv,fiu,fiv;

du=sqrt(u.vec[0]*u.vec[0]+u.vec[1]*u.vec[1]);
dv=0;
fiu=atan2(v[1],v[0]);
fiv=0;

Fr[0][0]=1;
Fr[0][1]=0;
```

```
Fr[0][2]=-du*sin(v[2]+fiu)-dv*sin(v[2]+fiv);
Fr[1][0]=0;
Fr[1][1]=1;
Fr[1][2]=du*sin(v[2]+fiu)+dv*cos(v[2]+fiv);
Fr[2][0]=0;
Fr[2][1]=0;
Fr[2][2]=1;

Fv[0][0]=sr[0]*cos(v[2]+fiv)/dv;
Fv[0][1]=sr[1]*cos(v[2]+fiv)/dv;
Fv[0][2]=0;
Fv[1][0]=sr[0]*sin(v[2]+fiv)/dv;
Fv[1][1]=sr[1]*sin(v[2]+fiv)/dv;
Fv[1][2]=0;
Fv[2][0]=0;
Fv[2][1]=0;
Fv[2][2]=1;
}

matriz CVSWalls::Calcularpk1k(matriz Fv,matriz Fr,matriz vr)
{
int i,j;
matriz aux1,aux2,aux3,aux4,aux5,aux6,p,pr,pzr;

pr.mat=alloc_mat_d(3,3);
pr.nc=3;
pr.nf=3;
Copiarmatriz(pkk.mat,pr.mat,0,0,3,3,0,0);

p.mat=alloc_mat_d(pkk.nf,pkk.nc);
p.nf=pkk.nf;
p.nc=pkk.nc;

aux1.mat=alloc_mat_d(3,3);
aux2.mat=alloc_mat_d(3,3);
aux3.mat=alloc_mat_d(3,3);
aux4.mat=alloc_mat_d(3,3);

transpose(Fr.mat,aux4.mat,Fr.nf,Fr.nc);
aux4.nc=3;
aux4.nf=3;
mult_mat(pr.mat,aux4.mat,aux2.mat,pr.nf,aux4.nf,aux4.nc);
aux2.nc=3;
aux2.nf=3;
mult_mat(Fr.mat,aux2.mat,aux1.mat,Fr.nf,aux2.nf,aux2.nc);
aux1.nc=3;
aux1.nf=3;
aux3.nc=3;
aux3.nf=3;
sum_mat(aux1.mat,vr.mat,aux3.mat,aux1.nf,aux1.nc);
```

```
Copiarmatriz(aux3.mat,p.mat,0,0,aux3.nf,aux3.nc,0,0);

if(pkk.nc>3)
{
Copiarmatriz(pkk.mat,p.mat,3,3,pkk.nf,pkk.nc,3,3);
pzr.mat=alloc_mat_d(pkk.nf-3,3);
Copiarmatriz(pkk.mat,pzr.mat,3,0,pkk.nf,3,0,0);
pzr.nf=pkk.nf-3;
pzr.nc=3;
aux5.mat=alloc_mat_d(pzr.nf,3);
mult_mat(pzr.mat,aux4.mat,aux5.mat,pzr.nf,aux4.nf,aux4.nc);
aux5.nf=pzr.nf;
aux5.nc=aux4.nc;
Copiarmatriz(aux5.mat,p.mat,0,0,aux5.nf,aux5.nc,3,0);
aux6.mat=alloc_mat_d(3,aux5.nf);
transpose(aux5.mat,aux6.mat,aux5.nf,aux5.nc);
aux6.nf=aux5.nc;
aux6.nc=aux5.nf;
Copiarmatriz(aux6.mat,p.mat,0,0,aux6.nf,aux6.nc,0,3);

release_mat_d(pzr.mat);
release_mat_d(aux5.mat);
release_mat_d(aux6.mat);
}

release_mat_d(pr.mat);
release_mat_d(aux1.mat);
release_mat_d(aux2.mat);
release_mat_d(aux3.mat);
release_mat_d(aux4.mat);
return p;
}

matriz CVSWalls::CalcularK(matriz pk1k, float az)
{
matriz K,Ki,Hxri,Hxzi,Hxi,Hxit,Hwi,Wi,Si,invSi,Rt,dRt,Hx;
matriz ident,mat0,aux1,aux2,aux3,aux4,aux5,W;
vector xzi,t,sz2;
double *truco [4], num;
int i,j,l;

mat0.mat=alloc_mat_d(4,4);
matriz0(mat0.mat,4,4);
mat0.nf=4;
mat0.nc=4;
K.mat=alloc_mat_d(xk1k.n,zk1k.n);
W.mat=alloc_mat_d(zk1k.n,zk1k.n);
W.nf=zk1k.n;
W.nc=zk1k.n;
identity(W.mat,zk1k.n);
```

```
Hx.mat=alloc_mat_d(4*oldLmk.numofWalls,3+4*oldLmk.numofWalls);
Hx.nf=4*oldLmk.numofWalls;
Hx.nc=3+4*oldLmk.numofWalls;

Hxri.mat=alloc_mat_d(4,3);
Hxri.nc=3;
Hxri.nf=4;
Hxri.mat[0][0]=-cos(posicion[2]);
Hxri.mat[1][0]=sin(posicion[2]);
Hxri.mat[2][0]=-cos(posicion[2]);
Hxri.mat[3][0]=sin(posicion[2]);
Hxri.mat[0][1]=-sin(posicion[2]);
Hxri.mat[1][1]=-cos(posicion[2]);
Hxri.mat[2][1]=-sin(posicion[2]);
Hxri.mat[3][1]=-cos(posicion[2]);

Hxzi.mat=alloc_mat_d(4,4);
Hxzi.nf=4;
Hxzi.nc=4;
Hxzi.mat[0][0]=cos(posicion[2]);
Hxzi.mat[1][0]=-sin(posicion[2]);
Hxzi.mat[0][1]=sin(posicion[2]);
Hxzi.mat[1][1]=cos(posicion[2]);
Hxzi.mat[2][0]=0;
Hxzi.mat[3][0]=0;
Hxzi.mat[2][1]=0;
Hxzi.mat[3][1]=0;
Hxzi.mat[0][2]=0;
Hxzi.mat[1][2]=0;
Hxzi.mat[0][3]=0;
Hxzi.mat[1][3]=0;
Hxzi.mat[2][2]=cos(posicion[2]);
Hxzi.mat[3][2]=-sin(posicion[2]);
Hxzi.mat[2][3]=sin(posicion[2]);
Hxzi.mat[3][3]=cos(posicion[2]);

sz2.vec=alloc_vec_d(4);
sz2.n=4;
for(l=0;l<4;l++)
{
sz2.vec[l]= az*az;
}

Wi.mat=alloc_mat_d(sz2.n,sz2.n);
diagonal(sz2.vec,Wi.mat,sz2.n);
Wi.nf=sz2.n;
Wi.nc=sz2.n;

Hxi.mat=alloc_mat_d(4,oldLmk.numofWalls*4+3);
Hxi.nf=4;
```

```
Hxi.nc=oldLmk.numofWalls*4+3;

Hxit.mat=alloc_mat_d(Hxi.nc,Hxi.nf);
Hxit.nf=Hxi.nc;
Hxit.nc=Hxi.nf;

aux1.nf=pk1k.nf;
aux1.nc=Hxit.nc;

aux2.nf=Hxi.nf;
aux2.nc=aux1.nc;

Si.mat=alloc_mat_d(aux2.nf,aux2.nc);
Si.nf=aux2.nf;
Si.nc=aux2.nc;

invSi.mat=alloc_mat_d(Si.nf,Si.nc);
invSi.nf=Si.nf;
invSi.nc=Si.nc;

Ki.mat=alloc_mat_d(aux1.nf,invSi.nc);
Ki.nf=aux1.nf;
Ki.nc=invSi.nc;

aux3.nf=Ki.nf;
aux3.nc=Hxi.nc;

ident.mat=alloc_mat_d(aux3.nf,aux3.nc);
ident.nf=aux3.nf;
ident.nc=aux3.nc;

aux4.nf=ident.nf;
aux4.nc=ident.nc;

aux5.nf=aux4.nc;
aux5.nc=aux4.nf;

for(i=0;i<oldLmk.numofWalls;i++)
{
Hxri.mat[0][2]=-sin(posicion[2])*(xk1k.vec[3+i*4]-posicion[0])+
cos(posicion[2])*(xk1k.vec[3+i*4+1]-posicion[1]);
Hxri.mat[1][2]=-cos(posicion[2])*(xk1k.vec[3+i*4]-posicion[0])-
sin(posicion[2])*(xk1k.vec[3+i*4+1]-posicion[1]);
Hxri.mat[2][2]=-sin(posicion[2])*(xk1k.vec[3+i*4+2]-posicion[0])+
cos(posicion[2])*(xk1k.vec[3+i*4+3]-posicion[1]);
Hxri.mat[3][2]=-cos(posicion[2])*(xk1k.vec[3+i*4+2]-posicion[0])-
sin(posicion[2])*(xk1k.vec[3+i*4+3]-posicion[1]);

Copiarmatriz(Hxri.mat,Hxi.mat,0,0,Hxri.nf,Hxri.nc,0,0);
```

```
for(j=0;j<oldLmk.numofWalls;j++)
{
if(j==i)
{
Copiarmatriz(Hxzi.mat,Hxi.mat,0,0,Hxzi.nf,Hxzi.nc,0,j*4+3);
}
else
{
Copiarmatriz(mat0.mat,Hxi.mat,0,0,mat0.nf,mat0.nc,0,j*4+3);
}
}

transpose(Hxi.mat,Hxit.mat,Hxi.nf,Hxi.nc);

Copiarmatriz(Hxi.mat,Hx.mat,0,0,Hxi.nf,Hxi.nc,4*i,0);

aux1.mat=alloc_mat_d(pk1k.nf,Hxit.nc);
aux1.nf=pk1k.nf;
aux1.nc=Hxit.nc;
mult_mat(pk1k.mat,Hxit.mat,aux1.mat,pk1k.nf,Hxit.nf,Hxit.nc);
aux2.mat=alloc_mat_d(Hxi.nf,aux1.nc);
aux2.nf=Hxi.nf;
aux2.nc=aux1.nc;
mult_mat(Hxi.mat,aux1.mat,aux2.mat,Hxi.nf,aux1.nf,aux1.nc);

Copiarmatriz(Wi.mat,W.mat,0,0,Wi.nf,Wi.nc,i*Wi.nf,i*Wi.nc);

sum_mat(aux2.mat,Wi.mat,Si.mat,aux2.nf,aux2.nc);

inverse4x4(Si.mat,invSi.mat);

mult_mat(aux1.mat,invSi.mat,Ki.mat,aux1.nf,invSi.nf,invSi.nc);
Copiarmatriz(Ki.mat,K.mat,0,0,Ki.nf,Ki.nc,0,Ki.nc*i);

for(j=0;j<xk1k.n;j++)
{
num=0;
for(int k=0;k<4;k++)
num=num+Ki.mat[j][k]*ezk1k.vec[4*i+k];

xk1k1.vec[j]=xk1k.vec[j]+num;
xk1k.vec[j]=xk1k1.vec[j];
}

aux3.mat=alloc_mat_d(Ki.nf,Hxi.nc);
aux3.nf=Ki.nf;
aux3.nc=Hxi.nc;
mult_mat(Ki.mat,Hxi.mat,aux3.mat,Ki.nf,Hxi.nf,Hxi.nc);

identity(ident.mat,aux3.nf);
```

```
aux4.mat=alloc_mat_d(ident.nf,ident.nc);
aux4.nf=ident.nf;
aux4.nc=ident.nc;
dif_mat(ident.mat,aux3.mat,aux4.mat,ident.nf,ident.nc);
aux5.mat=alloc_mat_d(aux4.nc,aux4.nf);
aux5.nf=aux4.nc;
aux5.nc=aux4.nf;
transpose(aux4.mat,aux5.mat,aux4.nf,aux5.nc);
release_mat_d(aux1.mat);
aux1.mat=alloc_mat_d(aux4.nf,pk1k.nc);
aux1.nf=aux4.nf;
aux1.nc=pk1k.nc;
mult_mat(aux4.mat,pk1k.mat,aux1.mat,aux4.nf,pk1k.nf,pk1k.nc);
release_mat_d(aux2.mat);
aux2.mat=alloc_mat_d(aux1.nf,aux5.nc);
aux2.nf=aux1.nf;
aux2.nc=aux5.nc;
mult_mat(aux1.mat,aux5.mat,aux2.mat,aux1.nf,aux5.nf,aux5.nc);
release_mat_d(aux3.mat);
aux3.mat=alloc_mat_d(Ki.nc,Ki.nf);
aux3.nf=Ki.nc;
aux3.nc=Ki.nf;
transpose(Ki.mat,aux3.mat,Ki.nf,Ki.nc);
release_mat_d(aux1.mat);
aux1.mat=alloc_mat_d(Ki.nf,Wi.nc);
aux1.nf=Ki.nf;
aux1.nc=Wi.nc;
mult_mat(Ki.mat,Wi.mat,aux1.mat,Ki.nf,Wi.nf,Wi.nc);
release_mat_d(aux4.mat);
aux4.mat=alloc_mat_d(aux1.nf,aux3.nc);
aux4.nf=aux1.nf;
aux4.nc=aux3.nc;
mult_mat(aux1.mat,aux3.mat,aux4.mat,aux1.nf,aux3.nf,aux3.nc);
sum_mat(aux2.mat,aux4.mat,pk1k.mat,aux2.nf,aux2.nc);

release_mat_d(aux1.mat);
release_mat_d(aux2.mat);
release_mat_d(aux3.mat);
release_mat_d(aux4.mat);
release_mat_d(aux5.mat);
}
xk1k1.n=xk1k.n;
K.nf=Ki.nf;
K.nc=Ki.nc*oldLmk.numofWalls;

for(int m=0;m<3;m++)
eposr.vec[m]=eposr.vec[m]+xk1k1.vec[m]-posicion[m];

ImprimirMatriz(dumpfile,Hx.mat,0,0,Hx.nf,Hx.nc,"Hx");
ImprimirMatriz(dumpfile,K.mat,0,0,K.nf,K.nc,"K");
```

```
ImprimirMatriz(dumpfile,W.mat,0,0,W.nf,W.nc,"W");
Copiarmatriz(pk1k.mat,pkk.mat,0,0,pk1k.nf,pk1k.nc,0,0);
pkk.nf=pk1k.nf;
pkk.nc=pk1k.nc;

ImprimirMatriz(dumpfile,pkk.mat,0,0,pkk.nf,pkk.nc,"Pk1k1");

release_mat_d(mat0.mat);
release_mat_d(W.mat);
release_mat_d(Hx.mat);
release_mat_d(Hxri.mat);
release_mat_d(Hxzi.mat);
release_mat_d(Wi.mat);
release_mat_d(Hxi.mat);
release_mat_d(Hxit.mat);
release_mat_d(Si.mat);
release_mat_d(invSi.mat);
release_mat_d(Ki.mat);
release_vec_d(sz2.vec);


return K;
}

//Omple els vectors d'estat xk1k i zk1k i zk1k1
void CVSWalls::LlenarEstado()
{
int i,j;
bool ok1=false,ok2=false;
vector zk1k1;

double *puntotemp;

zk1k.n=4*oldLmk.numofWalls;
zk1k1.vec=alloc_vec_d(zk1k.n);
xk1k.n=3+zk1k.n;
zk1k1.n=zk1k.n;
ezk1k.n=zk1k.n;

for(i=0;i<oldLmk.numofWalls;i++)
{
xk1k.vec[3+i*4]=oldLmk.Data[i].p.p1[0];
xk1k.vec[3+i*4+1]=oldLmk.Data[i].p.p1[1];
xk1k.vec[3+i*4+2]=oldLmk.Data[i].p.p2[0];
xk1k.vec[3+i*4+3]=oldLmk.Data[i].p.p2[1];
global2local(oldLmk.Data[i].p.p1[0],oldLmk.Data[i].p.p1[1],
&(zk1k.vec[i*4]),&(zk1k.vec[i*4+1]));
global2local(oldLmk.Data[i].p.p2[0],oldLmk.Data[i].p.p2[1],
&(zk1k.vec[i*4+2]),&(zk1k.vec[i*4+3]));
```

```
for(j=0;j<detLmk.numofWalls;j++)
{
if(Similar(detLmk.Data[j].p,oldLmk.Data[i].p,10) ||
Similar(oldLmk.Data[i].p,detLmk.Data[j].p,10))
{
if((Distancia(detLmk.Data[j].p.p1,oldLmk.Data[i].p.p1) < 10) && (!ok1))
{
global2local(detLmk.Data[j].p.p1[0],detLmk.Data[j].p.p1[1],
&(zk1k1.vec[i*4]),&(zk1k1.vec[i*4+1]));
ezk1k.vec[i*4]=zk1k1.vec[i*4]-zk1k.vec[i*4];
ezk1k.vec[i*4+1]=zk1k1.vec[i*4+1]-zk1k.vec[i*4+1];
ok1=true;
}
if((Distancia(detLmk.Data[j].p.p2,oldLmk.Data[i].p.p1) < 10)  && (!ok1))
{
global2local(detLmk.Data[j].p.p2[0],detLmk.Data[j].p.p2[1],
&(zk1k1.vec[i*4]),&(zk1k1.vec[i*4+1]));
ezk1k.vec[i*4]=zk1k1.vec[i*4]-zk1k.vec[i*4];
ezk1k.vec[i*4+1]=zk1k1.vec[i*4+1]-zk1k.vec[i*4+1];
ok1=true;
}
if((Distancia(detLmk.Data[j].p.p1,oldLmk.Data[i].p.p2) < 10)  && (!ok2))
{
global2local(detLmk.Data[j].p.p1[0],detLmk.Data[j].p.p1[1],
&(zk1k1.vec[i*4+2]),&(zk1k1.vec[i*4+3]));
ezk1k.vec[i*4+2]=zk1k1.vec[i*4+2]-zk1k.vec[i*4+2];
ezk1k.vec[i*4+3]=zk1k1.vec[i*4+3]-zk1k.vec[i*4+3];
ok2=true;
}
if((Distancia(detLmk.Data[j].p.p2,oldLmk.Data[i].p.p2) < 10)  && !ok2)
{
global2local(detLmk.Data[j].p.p2[0],detLmk.Data[j].p.p2[1],
&(zk1k1.vec[i*4+2]),&(zk1k1.vec[i*4+3]));
ezk1k.vec[i*4+2]=zk1k1.vec[i*4+2]-zk1k.vec[i*4+2];
ezk1k.vec[i*4+3]=zk1k1.vec[i*4+3]-zk1k.vec[i*4+3];
ok2=true;
}
if(!ok1) // buscar punt a la observació més propera a l'extrem de la paret
{
puntotemp = Proyeccion(oldLmk.Data[i].p.p1,detLmk.Data[j].p);
global2local(puntotemp[0], puntotemp[1], &(zk1k1.vec[i*4]),
&(zk1k1.vec[i*4+1]));
ezk1k.vec[i*4]=zk1k1.vec[i*4]-zk1k.vec[i*4];
ezk1k.vec[i*4+1]=zk1k1.vec[i*4+1]-zk1k.vec[i*4+1];
ok1 = true;
}
if(!ok2) // buscar punt a la observació més propera a l'extrem de la paret
{
puntotemp = Proyeccion(oldLmk.Data[i].p.p2,detLmk.Data[j].p);
global2local(puntotemp[0], puntotemp[1], &(zk1k1.vec[i*4+2]),
```

```
&(zk1k1.vec[i*4+3]));
ezk1k.vec[i*4+2]=zk1k1.vec[i*4+2]-zk1k.vec[i*4+2];
ezk1k.vec[i*4+3]=zk1k1.vec[i*4+3]-zk1k.vec[i*4+3];
ok2 = true;
}

}
}
if(!ok1)
{
ezk1k.vec[i*4]=0;
ezk1k.vec[i*4+1]=0;
zk1k1.vec[i*4]=-1;
zk1k1.vec[i*4+1]=-1;
}
if(!ok2)
{
ezk1k.vec[i*4+2]=0;
ezk1k.vec[i*4+3]=0;
zk1k1.vec[i*4+2]=-1;
zk1k1.vec[i*4+3]=-1;
}

ok1=false;
ok2=false;
}
ImprimirVector(dumpfile,xk1k.vec,0,xk1k.n,"xk1k");
ImprimirVector(dumpfile,zk1k.vec,0,zk1k.n,"zk1k");
ImprimirVector(dumpfile,zk1k1.vec,0,zk1k1.n,"zk1k1");
ImprimirVector(dumpfile,ezk1k.vec,0,ezk1k.n,"ezk1k");

release_vec_d(zk1k1.vec);
}

void CVSWalls::global2local(double xglob,double yglob, double *xloc, double *yloc)
{
double b[2];
b[0]=xglob-posicion[0];
b[1]=yglob-posicion[1];

*xloc=cos(posicion[2])*b[0]+sin(posicion[2])*b[1];
*yloc=-sin(posicion[2])*b[0]+cos(posicion[2])*b[1];
}

void CVSWalls::local2global(double xloc,double yloc, double *xglob, double *yglob)
{
*xglob=cos(posicion[2])*xloc-sin(posicion[2])*yloc+posicion[0];
*yglob=sin(posicion[2])*xloc+cos(posicion[2])*yloc+posicion[1];
}
```

```
void CVSWalls::ActualizarLmks()
{
int i,j,k;
int auxindex;
bool begin=false,stop=false,done=false,end=false;
CDetectedLandmark auxLmk,unionLmk;
int * modif;

modif=(int*)malloc(2*sizeof(int));

//Fixa la posició del robot a xk1k1.vec[0..2]
if(oldLmk.numofWalls>0)
{
posicion[0]=xk1k1.vec[0];
posicion[1]=xk1k1.vec[1];
posicion[2]=xk1k1.vec[2];
}

for(i=0;i<oldLmk.numofWalls;i++)
{
oldLmk.Data[i].p.p1[0]=xk1k1.vec[3+i*4];
oldLmk.Data[i].p.p1[1]=xk1k1.vec[3+i*4+1];
oldLmk.Data[i].p.p2[0]=xk1k1.vec[3+i*4+2];
oldLmk.Data[i].p.p2[1]=xk1k1.vec[3+i*4+3];
oldLmk.Data[i].p.b1=(oldLmk.Data[i].p.p2[1]-oldLmk.Data[i].p.p1[1])/
(oldLmk.Data[i].p.p2[0]-oldLmk.Data[i].p.p1[0]);
oldLmk.Data[i].p.b0=oldLmk.Data[i].p.p1[1]-
oldLmk.Data[i].p.b1*oldLmk.Data[i].p.p1[0];
oldLmk.Data[i].p.dist=Distancia(oldLmk.Data[i].p.p1,oldLmk.Data[i].p.p2);
}

for(i=0;i<detLmk.numofWalls;i++)
{
local2global(detLmk.Data[i].p.z1[0],detLmk.Data[i].p.z1[1],
&(detLmk.Data[i].p.p1[0]),&(detLmk.Data[i].p.p1[1]));
local2global(detLmk.Data[i].p.z2[0],detLmk.Data[i].p.z2[1],
&(detLmk.Data[i].p.p2[0]),&(detLmk.Data[i].p.p2[1]));

detLmk.Data[i].p.b1=(detLmk.Data[i].p.p2[1]-detLmk.Data[i].p.p1[1])/
(detLmk.Data[i].p.p2[0]-detLmk.Data[i].p.p1[0]);
detLmk.Data[i].p.b0=detLmk.Data[i].p.p1[1]-detLmk.Data[i].p.b1*detLmk.Data[i].p.p1[0];
}

InicializarDatos();

//Comença la comparació
if(!detLmk.Empty() && !oldLmk.Empty())
{
end=false;
while(!end)
```

```
{
oldLmk.index=0;

//unionLmk al final acaba sent la unió del detLmk actual amb tots els
//oldLmk que es considerin similars
unionLmk = detLmk.Data[detLmk.index];
stop=false;
auxindex=0;

while(!stop)
{
if(Similar(detLmk.Data[detLmk.index].p,oldLmk.Data[oldLmk.index].p,10) ||
Similar(oldLmk.Data[oldLmk.index].p,detLmk.Data[detLmk.index].p,10))
{
//Unims unionLmk amb el oldLmk que s'ha considerat similar
unionLmk=Union(unionLmk,oldLmk.Data[oldLmk.index],modif);
//Incluim el oldLmk a la lista d'associts a aquest detLmk
asoc[auxindex]=oldLmk.index;
auxindex++;
//Marquem aquest detLmk com associat
detLmk.Data[detLmk.index].marca=true;
}

if(!oldLmk.isLast())
oldLmk.index++; //Passem al següent oldLmk
else
stop=true; //Si es l'últim oldLmk s'arriba al final
}

//Inserim els oldLmk que considerem nous, o sigui, que no s'hagin associat
//amb cap oldLmk i substituim els associats pels nous lmks actualitzats
if(detLmk.Data[detLmk.index].marca==false)
{
oldLmk.Insert(detLmk.Data[detLmk.index]);
InsertarElementoP(oldLmk.numofWalls-1);
}
else
{
for(int k=0;k<auxindex;k++)
{
if(k==0 && (modif[0] || modif[1]))
oldLmk.Data[asoc[k]]=unionLmk;
if(k>0)
{
oldLmk.Delete(asoc[k]-k+1);
QuitarElementosP(asoc[k]);
}
}
}
//Continuem iterant sempre que hi hagi detLmks
```

```
if(!detLmk.isLast())
detLmk.index++;
else
end=true;
}
}
//Si oldLmk està buida l'omplim amb detLmk
else
{
stop=detLmk.isLast();
while(!stop)
{
oldLmk.Insert(detLmk.Data[detLmk.index]);
InsertarElementoP(oldLmk.numofWalls-1);
if(!detLmk.isLast())
detLmk.index++;
else
stop=true;
}
}
ImprimirMatriz(dumpfile,pkk.mat,0,0,pkk.nf,pkk.nc,"Pk1k1 después de Actualizar Lmks");

for(i=0;i<oldLmk.numofWalls;i++)
{
for(j=0;j<2;j++)
{
for(k=0;k<2;k++)
{
oldLmk.Data[i].covar1[j][k]=pkk.mat[3+i*4+j][3+i*4+k];
oldLmk.Data[i].covar2[j][k]=pkk.mat[5+i*4+j][5+i*4+k];
}
}
}

Copiarmatriz(pkk.mat,Prrold,0,0,3,3,0,0);

free(modif);

}

void CVSWalls::QuitarElementosP(int k)
{
Copiarmatriz(pkk.mat,pkk.mat,0,3+(k+1)*4,3+(k*4),pkk.nc,0,3+k*4);
Copiarmatriz(pkk.mat,pkk.mat,3+(k+1)*4,0,pkk.nf,3+(k*4),3+k*4,0);
Copiarmatriz(pkk.mat,pkk.mat,3+(k+1)*4,3+(k+1)*4,pkk.nf,pkk.nc,3+k*4,3+k*4);
pkk.nf=pkk.nf-4;
pkk.nc=pkk.nc-4;
}

//Calcula la unió de dos landmarks
```

```
CDetectedLandmark CVSWalls::Union(CDetectedLandmark lmk1,CDetectedLandmark lmk2,int * modif)
{
double d [6];
int comb=0,in,aux1,aux2;
CDetectedLandmark lmk0;
int sim [2];
double * pr1, * pr2;
punto prjct1, prjct2;
double xmin,xmax,ymin,ymax;
double x [2][2];

modif[0]=0;
modif[1]=0;

//Calculem distàncies entre punts extrems de la recta
d[0]=Distancia(lmk1.p.p1,lmk2.p.p1);
d[1]=Distancia(lmk1.p.p1,lmk2.p.p2);
d[2]=Distancia(lmk1.p.p2,lmk2.p.p1);
d[3]=Distancia(lmk1.p.p2,lmk2.p.p2);

if(d[0]<10)
{
pr1=(double *)malloc(2*sizeof(double));
pr1[0]=lmk2.p.p1[0];
pr1[1]=lmk2.p.p1[1];
sim[0]=1;
}
if(d[1]<10)
{
pr1=(double *)malloc(2*sizeof(double));
pr1[0]=lmk2.p.p2[0];
pr1[1]=lmk2.p.p2[1];
sim[0]=1;
}
if(d[2]<10)
{
pr2=(double *)malloc(2*sizeof(double));
pr2[0]=lmk2.p.p1[0];
pr2[1]=lmk2.p.p1[1];
sim[1]=1;
}
if(d[3]<10)
{
pr2=(double *)malloc(2*sizeof(double));
pr2[0]=lmk2.p.p2[0];
pr2[1]=lmk2.p.p2[1];
sim[1]=1;
}

if(sim[0]!=1)
```

```
pr1=Proyeccion(lmk1.p.p1,lmk2.p);
if(sim[1]!=1)
pr2=Proyeccion(lmk1.p.p2,lmk2.p);
if(lmk2.p.p1[0]<lmk2.p.p2[0])
{
xmin=lmk2.p.p1[0];
ymin=lmk2.p.p1[1];
xmax=lmk2.p.p2[0];
ymax=lmk2.p.p2[1];
aux1=0;
aux2=1;
}
else
{
xmin=lmk2.p.p2[0];
ymin=lmk2.p.p2[1];
xmax=lmk2.p.p1[0];
ymax=lmk2.p.p1[1];
aux1=1;
aux2=0;
}
x[0][0]=pr1[0];
x[0][1]=pr1[1];
x[1][0]=pr2[0];
x[1][1]=pr2[1];

for(in=0;in<2;in++)
{
if(x[in][0]<xmin)
{
xmin=x[in][0];
ymin=x[in][1];
modif[aux1]=1;
}
if(x[in][0]>xmax)
{
xmax=x[in][0];
ymax=x[in][1];
modif[aux2]=1;
}
}

lmk0=lmk2;

lmk0.p.p1[0]=xmin;
lmk0.p.p1[1]=ymin;
lmk0.p.p2[0]=xmax;
lmk0.p.p2[1]=ymax;

//Omplim la resta de dades
```

```
lmk0.m_egoPosition=CrearPunt3D(lmk0.p.p1[0],lmk0.p.p1[1],0);
lmk0.m_absPosition=CrearPunt3D(lmk0.p.p2[0],lmk0.p.p2[1],0);
lmk0.m_imgPosition=CrearPunt2D(0,0);

if(lmk0.p.p1[0]>lmk0.p.p2[0])
{
lmk0.max=lmk0.p.p1[0];
lmk0.min=lmk0.p.p2[0];
}
else
{
lmk0.max=lmk0.p.p2[0];
lmk0.min=lmk0.p.p1[0];
}

//Recalculem els paràmetres de la regressió de la recta
lmk0.p.b1=(lmk0.p.p2[1]-lmk0.p.p1[1])/(lmk0.p.p2[0]-lmk0.p.p1[0]);
lmk0.p.b0=lmk0.p.p1[1]-lmk0.p.b1*lmk0.p.p1[0];
lmk0.p.dist=d[0];

return lmk0;

}

//Calcula la projecció d'un punt p1 sobre una recta r
double * CVSWalls::Proyeccion(punto p1,pared r)
{
double * result;
double b02;

result=(double *)malloc(2*sizeof(double));

b02=p1[1]-p1[0]/r.b1;

result[0]=(b02-r.b0)/(r.b1-1/r.b1);
result[1]=r.b1*result[0]+r.b0;

return result;
}

//Reinicialitza els valors de la matriu de covariança
void CVSWalls::ResetElementoP(int elem,int opt)
{
int l,m;
matriz O1,O1t,O2,O2t,prin;

if(opt==1)
{
l=0;
m=2;
```

```
}
if(opt==2)
{
l=2;
m=0;
}

O1.mat=alloc_mat_d(2,3+4*elem+l);
O1.nf=2;
O1.nc=3+4*elem+l;
matriz0(O1.mat,O1.nf,O1.nc);

O1t.mat=alloc_mat_d(3+4*elem+l,2);
O1t.nf=3+4*elem+l;
O1t.nc=2;
transpose(O1.mat,O1t.mat,O1.nf,O1.nc);

O2.mat=alloc_mat_d(2,4*(oldLmk.numofWalls-elem-1)+m);
O2.nf=2;
O2.nc=4*(oldLmk.numofWalls-elem-1)+m;
matriz0(O2.mat,O2.nf,O2.nc);

O2t.mat=alloc_mat_d(4*(oldLmk.numofWalls-elem-1)+m,2);
O2t.nf=4*(oldLmk.numofWalls-elem-1)+m;
O2t.nc=2;
transpose(O2.mat,O2t.mat,O2.nf,O2.nc);

prin.mat=alloc_mat_d(2,2);
prin.nf=2;
prin.nc=2;
identity(prin.mat,prin.nf);
for(int i=0;i<2;i++)
prin.mat[i][i]=winic*winic;


Copiarmatriz(O1.mat,pkk.mat,0,0,O1.nf,O1.nc,3+4*elem+l,0);
Copiarmatriz(O1t.mat,pkk.mat,0,0,O1t.nf,O1t.nc,0,3+4*elem+l);
Copiarmatriz(O2.mat,pkk.mat,0,0,O2.nf,O2.nc,3+4*elem+l,3+4*(elem+1)-m);
Copiarmatriz(O2t.mat,pkk.mat,0,0,O2t.nf,O2t.nc,3+4*(elem+1)-m,3+4*elem+l);
Copiarmatriz(prin.mat,pkk.mat,0,0,prin.nf,prin.nc,3+4*elem+l,3+4*elem+l);
}

//Inicialització de la matriu de covariança pels landmarks nous
void CVSWalls::InsertarElementoP(int elem)
{
matriz N,Nt,prin,Hxz,Hxzt,Hxr,Hxz1Hxr,Hxz1Hxrt,aux1,pi,Hpr,W,aux2,aux3,prit;
int i;

oldLmk.Data[elem].covar1=alloc_mat_d(2,2);
oldLmk.Data[elem].covar2=alloc_mat_d(2,2);
```

```
N.mat=alloc_mat_d(3+4*elem,4);
N.nf=3+4*elem;
N.nc=4;

Hxz.mat=alloc_mat_d(4,4);
Hxz.nf=4;
Hxz.nc=4;
identity(Hxz.mat,Hxz.nf);
Hxz.mat[0][0]=cos(posicion[2]);
Hxz.mat[1][0]=-sin(posicion[2]);
Hxz.mat[0][1]=sin(posicion[2]);
Hxz.mat[1][1]=cos(posicion[2]);
Copiarmatriz(Hxz.mat,Hxz.mat,0,0,2,2,2,2);

Hxr.mat=alloc_mat_d(4,3);
Hxr.nf=4;
Hxr.nc=3;
Hxr.mat[0][0]=-cos(posicion[2]);
Hxr.mat[1][0]=sin(posicion[2]);
Hxr.mat[0][1]=-sin(posicion[2]);
Hxr.mat[1][1]=-cos(posicion[2]);
Copiarmatriz(Hxr.mat,Hxr.mat,0,0,2,2,2,0);

i=elem;
Hxr.mat[0][2]=-sin(posicion[2])*(oldLmk.Data[i].p.p1[0]-posicion[0])+
cos(posicion[2])*(oldLmk.Data[i].p.p1[1]-posicion[1]);
Hxr.mat[1][2]=-cos(posicion[2])*(oldLmk.Data[i].p.p1[0]-posicion[0])-
sin(posicion[2])*(oldLmk.Data[i].p.p1[1]-posicion[1]);
Hxr.mat[2][2]=-sin(posicion[2])*(oldLmk.Data[i].p.p2[0]-posicion[0])+
cos(posicion[2])*(oldLmk.Data[i].p.p2[1]-posicion[1]);
Hxr.mat[3][2]=-cos(posicion[2])*(oldLmk.Data[i].p.p2[0]-posicion[0])-
sin(posicion[2])*(oldLmk.Data[i].p.p2[1]-posicion[1]);

Hxzt.mat=alloc_mat_d(4,4);
Hxzt.nf=Hxz.nc;
Hxzt.nc=Hxz.nf;
transpose(Hxz.mat,Hxzt.mat,Hxz.nf,Hxz.nc);

Hxz1Hxr.mat=alloc_mat_d(4,3);
Hxz1Hxr.nf=Hxzt.nf;
Hxz1Hxr.nc=Hxr.nc;
mult_mat(Hxzt.mat,Hxr.mat,Hxz1Hxr.mat,Hxzt.nf,Hxr.nf,Hxr.nc);

Hxz1Hxrt.mat=alloc_mat_d(4,4);
Hxz1Hxrt.nf=Hxz1Hxr.nc;
Hxz1Hxrt.nc=Hxz1Hxr.nf;
transpose(Hxz1Hxr.mat,Hxz1Hxrt.mat,Hxz1Hxr.nf,Hxz1Hxr.nc);

aux1.mat=alloc_mat_d(4,4);
```

```
aux1.nf=3;
aux1.nc=3;
mult_scmat(-1,pkk.mat,aux1.mat,3,3);

pi.mat=alloc_mat_d(aux1.nf,Hxz1Hxrt.nc);
pi.nf=aux1.nf;
pi.nc=Hxz1Hxrt.nc;
mult_mat(aux1.mat,Hxz1Hxrt.mat,pi.mat,aux1.nf,Hxz1Hxrt.nf,Hxz1Hxrt.nc);
Copiarmatriz(pi.mat,N.mat,0,0,pi.nf,pi.nc,0,0);
release_mat_d(pi.mat);

prit.mat=alloc_mat_d(4,3);
prit.nf=4;
prit.nc=3;

pi.mat=alloc_mat_d(prit.nf,Hxz1Hxrt.nc);
pi.nf=prit.nf;
pi.nc=Hxz1Hxrt.nc;

Hpr.mat=alloc_mat_d(Hxz1Hxr.nf,3);
Hpr.nf=Hxz1Hxr.nf;
Hpr.nc=3;
mult_mat(Hxz1Hxr.mat,pkk.mat,Hpr.mat,Hxz1Hxr.nf,3,3);

aux1.nf=prit.nf;
aux1.nc=prit.nc;

for(i=0;i<elem;i++)
{
Copiarmatriz(pkk.mat,prit.mat,3+i*4,0,3+(i+1)*4,3,0,0);
mult_scmat(-1,prit.mat,aux1.mat,prit.nf,prit.nc);
mult_mat(aux1.mat,Hxz1Hxrt.mat,pi.mat,aux1.nf,Hxz1Hxrt.nf,Hxz1Hxrt.nc);
Copiarmatriz(pi.mat,N.mat,0,0,pi.nf,pi.nc,3+i*pi.nf,0);
}

Nt.mat=alloc_mat_d(N.nc,N.nf);
Nt.nf=N.nc;
Nt.nc=N.nf;
transpose(N.mat,Nt.mat,N.nf,N.nc);

Copiarmatriz(N.mat,pkk.mat,0,0,N.nf,N.nc,0,3+elem*4);
Copiarmatriz(Nt.mat,pkk.mat,0,0,Nt.nf,Nt.nc,3+elem*4,0);

prin.mat=alloc_mat_d(4,4);
prin.nf=4;
prin.nc=4;

aux1.nf=Hpr.nf;
aux1.nc=Hxz1Hxrt.nc;
mult_mat(Hpr.mat,Hxz1Hxrt.mat,aux1.mat,Hpr.nf,Hxz1Hxrt.nf,Hxz1Hxrt.nc);
```

```
W.mat=alloc_mat_d(4,4);
W.nf=4;
W.nc=4;
identity(W.mat,W.nf);
for(i=0;i<4;i++)
W.mat[i][i]=winic*winic;

aux2.mat=alloc_mat_d(Hxzt.nf,W.nc);
aux2.nf=Hxzt.nf;
aux2.nc=W.nc;
mult_mat(Hxzt.mat,W.mat,aux2.mat,Hxzt.nf,W.nf,W.nc);

aux3.mat=alloc_mat_d(aux2.nf,Hxz.nc);
aux3.nf=aux2.nf;
aux3.nc=Hxz.nc;
mult_mat(aux2.mat,Hxz.mat,aux3.mat,aux2.nf,Hxz.nf,Hxz.nc);

sum_mat(aux1.mat,aux3.mat,prin.mat,aux1.nf,aux1.nc);

Copiarmatriz(prin.mat,pkk.mat,0,0,prin.nf,prin.nc,3+elem*4,3+elem*4);
pkk.nf=pkk.nf+4;
pkk.nc=pkk.nc+4;

release_mat_d(N.mat);
release_mat_d(Nt.mat);
release_mat_d(prin.mat);
release_mat_d(Hxz.mat);
release_mat_d(Hxzt.mat);
release_mat_d(Hxr.mat);
release_mat_d(Hxz1Hxr.mat);
release_mat_d(Hxz1Hxrt.mat);
release_mat_d(aux1.mat);
release_mat_d(aux2.mat);
release_mat_d(aux3.mat);
release_mat_d(pi.mat);
release_mat_d(Hpr.mat);
release_mat_d(W.mat);
release_mat_d(prit.mat);
}
```

B.1.2 VSWalls.h

```
// VSWalls.h: interface per la classe CVSWalls.
//////////////////////////////////////////////////////////////////////

/*****************************************************************************
```

```
*                                                                            *
* NOM DE L'ARXIU:     VSWalls.h                                              *
* DATA:               Març 2002                                              *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Construeix un mapa de l'entorn a partir de les dades   *
*                     proporcionades per un sensor làser                     *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* ALGORITME BASAT EN: Duda & Hart 1973, i la nostra intuició                 *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a  *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis, *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.  *
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this     *
* disclaimer is included in the modified version.                            *
******************************************************************************/

#if !defined(AFX_VSWALLS_H)
#define AFX_VSWALLS

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "VisionWrapper.h"
#include "RobotWrapper.h"
#include <afxtempl.h>
#include "iepf.h"
#include "ListofLmk.h"

typedef punto xycontorno [133];

#define ANG_ERR 3.0;
#define DIST_ERR 0.1;

typedef struct
{
double ** mat;
int nf,nc;
}matriz;

typedef struct
{
double * vec;
```

```
int n;
}vector;



class CVSWalls : public CVisionWrapper
{
public:
CVSWalls();
virtual ~CVSWalls();

virtual int Init();

// Llista de landmarks detectats a la iteració actual
CListofLmks detLmk;
//Llista de landmarks acumulats actualizada a cada iteració
CListofLmks oldLmk;
//Variable auxiliar per treballar amb els detLmk
CDetectedLandmark *newLmk;

//Llista d'índexos de oldLmk asociats amb un determinat detLmk en una iteració
int asoc [20];

virtual void SetRobot(CRobotWrapper *pRobot) {m_pRobot = pRobot;}

int ReadLaserData(Ciepf*);

private:

//Variables de control del robot
double x, ax, y, ay, th, ath, speed, rspeed;

// Variables globals pel filtre de Kalman
float ar_xy; // sigma x,y robot = 3cm
float ar_alpha; // sigma theta robot = 3 grados
float az; // sigma pared = 5cm, desv. estàndar de l'error del sensor
double winic; // sigma^2 inicial pels landmarks

//Comptador de landmarks detectats en una iteració
int Wallcounter;

FILE *llista;

int Wallindex;

//Vector d'estat a priori
vector xk1k;
//Vector d'estat a posteriori
vector xk1k1;
//Vector de lectures del làser
```

```
vector ezk1k;
//Vector d'estimación de la posició dels landmarks
vector zk1k;
//Matriu de covariança
matriz pkk;
//Vector comanda de moviment u
vector u;
//Vector suma dels errors de posició del robot
vector eposr;

// Trobar landmarks i actualizar els que ja teníem
virtual int *FindLandmarks(void *args);
// Obtenir dades d'una recta
pared TratarPared(int ini, int fin, xycontorno puntos);
// Distancia entre 2 punts
double Distancia(punto p1,punto p2);
//Partir recta en cas que dos puntos consecutius estiguin molt separats
int RefinarPared(int,int,Ciepf*);
//Inicialitza índexos i marques de detLmk i oldLmk
void InicializarDatos();
//Introdueix la lista oldLmk ja actualizada dins la lista de lmk que han de dibuixar-se
void Dibujar();
//Busca la unió entre 2 parets
CDetectedLandmark Union(CDetectedLandmark,CDetectedLandmark,int*);
//Ens diu si una pared de detLmk és una que ja teníem a oldLmk (o part d'ella)
bool Similar(pared det,pared old,float unmral);
//Actualitza la posició del robot i la dels landmarks mitjançant el filtre de Kalman
void Kalman();
//Omple les matrius Fv i Fr
void LlenarFvFr(double *u, double * sr, double ** Fr, double ** Fv);
//Calcula la matriu de covariança a priori pk1k
matriz Calcularpk1k(matriz Fv,matriz Fr, matriz vr);
//Calcula la matriu de guany de Kalman K
matriz CalcularK(matriz pk1k, float az);
//Omple els vectors d'estat
void LlenarEstado();
//Passa un punt de coordenades globals a locals
void global2local(double xglob,double yglob, double *xloc, double *yloc);
//Passa un punt de coordenades locals a globals
void local2global(double xloc,double yloc, double *xglob, double *yglob);
//Actualitza el mapa
void ActualizarLmks();
//Treu un landmark de la llista oldLmk
void QuitarElementosP(int k);
//Reinicialitza la matriu de covariança d'un landmark actualitzat
void ResetElementoP(int elem,int opt);
//Inicialitza la matriu de covariança d'un landmark nou
void InsertarElementoP(int elem);
//Calcula la projecció del punt p1 sobre la recta r
double * Proyeccion(punto p1,pared r);
```

```
};

#endif // !defined(AFX_VSWALLS)
```

B.1.3 iepf.cpp

```
/*************************************************************************
*                                                                        *
* NOM DE L'ARXIU:     iepf.cpp                                             *
* DATA:               Març 2002                                          *
* VERSIÓ:             0.01                                               *
* OBJECTIU:           Donada una llista de punts, buscar les linies rectes *
*                     utilitzant el mètode Iterative End Point Fit       *
* CODI ORIGINAL DE:   Albert Checa Puimedon                              *
*                     (C) Institut de Robòtica Industrial                *
*                     Universitat Politècnica de Catalunya               *
* ALGORITME BASAT EN: Duda & Hart 1973, i la nostra intuició             *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                 *
*                                                                        *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                     *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                        *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                        *
*************************************************************************/

#include "iepf.h"


Ciepf::Ciepf(float *buffer) {

ct = 1;
for(int i=0;i<133;i++)
{
contour[i] = buffer[i];

//Com que contour només conté distàncies y nosaltres el que necessitem són els
//punts hem de fer la transformació.Ho fem sabent que el làser escaneja
//190°, començant desde -5 i acabant a 185, i fa 133 lectures.
puntos[i][0]=contour[i]*cos(i*190.0*4*PI/(528.0*180)-5*PI/180);
puntos[i][1]=contour[i]*sin(i*190.0*4*PI/(528.0*180)-5*PI/180)+0.102;
// aquest valor 0.102 és la distància que hi ha entre el làser i el centre
// de gir del robot. La mesura no ha estat presa amb massa precisió
```

```
}
}

Ciepf::~Ciepf() {

}

//Algoritme IEPF de transformació d'un contorn de punts a un contorn poligonal
int Ciepf::iepf(int inicio, int fin) {
int breakpt, r1, r2,j;

// Comprobar si la recta es pot partir més (conté més de 2 punts)
if((fin-inicio)<=1)
{
return(-1);
}

//Busquem el punt (breakpt es només l'índex) per on partir, si és que s'ha de partir
breakpt = split(inicio,fin,UMBRAL_DISTANCIA);

//Comprovem que s'ha de partir (quan ja no s'ha de partir breakpt=-1)
if (breakpt != -1)
{
//Apliquem el mètode IEPF a la primera part de la recta partida.Això omplirà
//la lista lineas amb les particions resultants d'aquesta recta
r1=iepf(inicio,breakpt);
//Ara introduïm l'índex del breakpt a la lista
lineas[ct] = breakpt;
j=ct;
ct++;
//Partim la segona part que ens acabarà d'omplir la lista "lineas"
r2=iepf(breakpt,fin);

//Comencem a intentar unir, sempre las duess rectes al voltant del punt que acabem
//de partir
if(r2==1)
{
merge(j,0,UMBRAL_DISTANCIA);
}
else
{
if(r1==1)
{
merge(j,fin,UMBRAL_DISTANCIA);
}
}
}
else
{
return(-1);
```

```
}

return(1);
}

int Ciepf::split(int inicio, int fin, float umbral) {
float pke[2],pks[2],nk[2],tki[2],pkes[2],dki,dkj;
int breakpt=-1;
float maxd=0.0, maxu=0.0;
int i;
float longitud;

// coordenades del punt d'inici de la cadena
pks[0]=puntos[inicio][0];
pks[1]=puntos[inicio][1];

// coordenades del punto final de la cadena
pke[0]=puntos[fin][0];
pke[1]=puntos[fin][1];

// calcula la distància entre pks i pke
longitud = sqrt((pke[0]-pks[0])*(pke[0]-pks[0])+(pke[1]-pks[1])*(pke[1]-pks[1]));


// calcula la normal a la recta definida per ps i pe
nk[0]=(pke[1]-pks[1])/longitud;
nk[1]=(-pke[0]+pks[0])/longitud;

//Calcula el punt mig entre pke i pks
pkes[0]=(pke[0]-pks[0])/2 + pks[0];
pkes[1]=(pke[1]-pks[1])/2 + pks[1];

for (i=inicio+1;i<fin;i++)
{
// coordenades del punt a comparar
tki[0]=puntos[i][0];
tki[1]=puntos[i][1];

// distància de tki a la recta definida per pks y pke
dki=fabs((tki[0]-pks[0])*nk[0]+(tki[1]-pks[1])*nk[1]);

if(dki>maxd)
{
maxd=dki;
breakpt=i;
}
}

//  Si el punt més llunyà a la recta està més aprop que un llindar, no partim
if(maxd<umbral)
```

```
{
return(-1);
}

return breakpt;
}


int Ciepf::merge(int j, int cond, float umbral)
{
int breakpt,l,r,p2, g=1;

//Quan unim dues rectes contigües,el que farem es substituir l'extrem comú
//a ambdúes per un 0 a la llista "lineas" que després haurem de treure
p2=cond;
//Si p2 és 0 vol dir que hem partit la segona recta, si no és 0 vol dir
//que la segona part no ha fet ninguna partició

//Mirem si l'anterior valor a j és un 0 i no es el primer punto.Si és 0, retrocedim
//una posició per a obtenir el verdader extrem de la recta que té per punt final
//el punt j
for(l=j-1;(lineas[l]==0) && (l!=0);l--);

//Si hem fet partició a la segona part de la recta busquem el punt extrem de
//la recta que té per inici el punto j
if(cond == 0)
{
for(r=j+1;(lineas[r]==0) && r<ct+1;r++);
p2=lineas[r];
}

//Ara ja tenim les dues rectes que intentarem unir

//Busquem el punt més llunyà a la recta. En cas que breakpt sigui -1 vol dir
//que podem unir les rectes
breakpt = split(lineas[l],p2,umbral);

if(breakpt==-1)
{
//Si no partim sustituim el punto j per un 0
lineas[j]=0;
}
else
return(0);

return(1);
}


int Ciepf::rectas(Ciepf *ptrIEPF) {
```

```
int i,s,e,u,v,k;

//Inserim el primer vèrtex
ptrIEPF->lineas[0]=0;

//Omplim "lineas" mitjançant l'algoritme IEPF
ptrIEPF->iepf(0,132);

//Introduim l'últim vèrtex
ptrIEPF->lineas[ptrIEPF->ct]=132;

i=1;

//Eliminem els 0 introduits a "lineas" per la funció merge començant des del
//segon punt perquè el primer sempre és 0 i no ha estat introduit pel merge
while(i<ptrIEPF->ct)
{
if(ptrIEPF->lineas[i]==0)
{
for(k=i;k<ptrIEPF->ct;k++)
{
ptrIEPF->lineas[k]=ptrIEPF->lineas[k+1];
}
ptrIEPF->ct=ptrIEPF->ct-1;
}
else
i++;
}

return 0;
}
```

B.1.4 iepf.h

```
// iepf.h : Encapçalament de l'arxiu iepf.cpp

#ifndef DEF_IEPF
#define DEF_IEPF

/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     iepf.h                                                 *
* DATA:               Març 2002                                              *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Donada una llista de punts, buscar les linies rectes   *
*                     utilitzant el mètode Iterative End Point Fit           *
```

```
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* ALGORITME BASAT EN: Duda & Hart 1973, i la nostra intuició                 *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a  *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis, *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.  *
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this     *
* disclaimer is included in the modified version.                            *
******************************************************************************/

#include "math.h"
#include "stdio.h"
#include "rs4.h"

#ifndef PI
  #define PI 3.14159265
#endif

//llindar de distancia al dividir linies (5 cm)
#define UMBRAL_DISTANCIA 0.05
//Tamany de la cadena de punts que retorna el làser
#define TAMANO 133

typedef double punto [2];


class Ciepf {
public:
Ciepf(float*);
~Ciepf();

//S'encarrega dels ajustos necessaris per a que "lineas" quedi correctament
int rectas(Ciepf *ptrIEPF);
//Realitza l'agrupació dels punts en rectes mitjançant el mètode IEPF.Aquest és un
//mètode recursiu
int iepf(int, int);
float contour[133]; //llista de distàcies enviades pel làser en un escombrat
punto puntos[133]; //llista de punts obtinguts en un escombrat en coordenades del làser
int lineas[133]; //llista d'indexos de punts que són extrems de recta
int ct; //comptador global de rectes obtingudes

private:
// donats dos índexos del contorn, retorna el punt per on s'ha de partir
```

```
int split(int, int, float);
//Uneix les rectes partides que es poden ajuntar
int merge(int,int,float);
};


#endif
```

B.1.5 rs4.cpp (del client del socket)

```
// rs4.cpp : Implementació de la classe Crs4
//
/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     rs4.cpp                                                *
* DATA:               Desembre 2001                                          *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Rebre un contorn de punts del làser a través d'un socket *
*                     Concretament, aquesta és la part del client del socket   *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                            *
*****************************************************************************/
#include "rs4.h"

Crs4::Crs4() {
mutxEndThread = new CMutex();
WSAStartup(0x0101,&wsd);
}

Crs4::~Crs4() {
go = false;
delete mutxEndThread;
}

int Crs4::Init()
```

```
{
// simular nous frames si no tenim la capturadora instalada
go = true;

AfxBeginThread(pulserFG,NULL);

return 0;
}


UINT Crs4::pulserFG(void *args)
{
void *args1;

((Crs4 *) me)->mutxEndThread->Lock();
while(((Crs4 *) me)->go)
{
Sleep(400);
args1 = ((Crs4 *) me);
NewFrameCallback(args1);
}

((Crs4 *) me)->mutxEndThread->Unlock();

return 0;
}

void Crs4::Connect()
{

int i;
char texto[100];

// Establir connexió mitjançant protocol TCP amb el robot MARCO
try {
if(!clientsocket.Create()) { // creació del socket
sprintf(texto, "Error creating TCP/IP client: error code %d",
clientsocket.GetLastError());
MessageBox(NULL,texto,"ERROR",MB_OK);
//return false;
exit(0);
}
//Adreça IP del servidor
for(i=0;i<NUMTRIES;i++) {
// if(clientsocket.Connect("147.83.76.189",8000)) { // Padilla
// if(clientsocket.Connect("147.83.76.194",8000)) { // Guridi
if(clientsocket.Connect("147.83.76.240",8000)) { // MARCO
// if(clientsocket.Connect("147.83.76.100",8000)) { // Samponi
// if(clientsocket.Connect("147.83.76.102",8000)) { // Zandonai
```

```
break;
}
else {
sprintf(texto, "Error connecting to TCP/IP server after %d tries\nerror
code %d. %d tries left", i+1, clientsocket.GetLastError(),NUMTRIES-i-1);
MessageBox(NULL,texto,"Communication ERROR",MB_ICONERROR);
}

}

if(i==NUMTRIES) {
sprintf(texto, "Can't connect to server. Program will be closed");
MessageBox(NULL,texto,"Communication ERROR",MB_ICONERROR);
exit(0);
}

clientsocket.fileout = new CSocketFile(&clientsocket);
clientsocket.arOut = new CArchive(clientsocket.fileout, CArchive::store);
clientsocket.filein = new CSocketFile(&clientsocket);
clientsocket.arIn = new CArchive(clientsocket.filein, CArchive::load);
}
catch(CException *e){
clientsocket.Close();
e->Delete();
}
}

void Crs4::Capture()
{
int a;

//connectem amb l'ordinador on el làser està connectat
Connect();

try
{
//Enviem un 1 al servidor que farà que aquest llegeixi dades del làser
*clientsocket.arOut << 1;
clientsocket.arOut->Flush();
}
catch (CException* pEx)
{
pEx->Delete();
}

// Aquí comença l'enviament de dades

int yalei = 0;
float infromsocket;
char texto[100];
```

```
for(;;) {
do {
if(!clientsocket.arIn->IsStoring())
{
*clientsocket.arIn >> contour[yalei];
// 1.077 és el factor d'escala de la calibració del làser
// Juliol 19 2002. Albert Checa i Juan Andrade
// amb això ara les mesures del làser estan en metres
contour[yalei] = contour[yalei]*1.077;
yalei++;
} else;
}

while (!clientsocket.arIn->IsBufferEmpty());

if(yalei>132) break;
}

clientsocket.Close();
}
```

B.1.6 rs4.h (del client del socket)

```
// rs4.h : Encapçalament de l'arxiu rs4.cpp

#ifndef DEF_CRS4_V_001
#define DEF_CRS4_V_001

/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     rs4.h                                                  *
* DATA:               Desembre 2001                                          *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Rebre un contorn de punts del làser a través d'un socket *
*                                                                            *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a  *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis, *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.  *
```

```
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this     *
* disclaimer is included in the modified version.                            *
*****************************************************************************/

//#include <windows.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <afxmt.h>
#include <afxsock.h>


#include "stdafx.h"
#include "RoboGUI.h"
#include "FrameGrabberWrapper.h"

typedef HANDLE portstream_fd;
#define TIMEOUT_CHAR_READ -1
#define NO_PARITY 0x00
#define EVEN_PARITY 0x18
#define ODD_PARITY 0x08
#define NUMTRIES 5


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#undef AFX_API
#define AFX_API AFX_EXT_CLASS
class CMyInt : public CObject {
DECLARE_DYNAMIC( CMyInt )
public:
int myint;
};
class CMyDouble : public CObject {
DECLARE_DYNAMIC( CMyDouble )
public:
double mydouble;
};
#undef AFX_API
#define AFX_API


// Definició de la classe socket de comunicació amb el robot.Aquí l'utilizarem com a
//client. L'ordinador del robot haurà de tenir un altre socket actuant com a servidor
class CMySocket : public CSocket {
public:
CSocketFile* filein;
```

```
CSocketFile* fileout;
CArchive* arIn;
CArchive* arOut;
};


// Classe Leuze Rotoscan 4
class Crs4 : public CFrameGrabberWrapper {
public:
Crs4();
virtual ~Crs4();
//Inicialització de les dades del làser
virtual int Init();
//Funció encarregada d'executar i control·lar el thread de lectura de dades del làser
static UINT pulserFG(void *args);

unsigned char mensaje[300]; // tamany del paquet del protocol del RS4
float contour[133]; //
bool ready; // variable que em diu si estic capturant amb el laser o no

//Variables per al socket
WSADATA wsd;
CSocket serversocket;
CMySocket clientsocket;
//Funció que connecta el socket amb l'ordinador al que està conectat el làser
void Connect();
//Obté les dades enviades pel làser a través del socket
void Capture();

private:
//Variables per control·lar l'estat del thread
bool go;
CMutex *mutxEndThread;
};

#endif
```

B.1.7 DetectedLandmark.h

```
// DetectedLandmark.h: definició de la classe CDetectedLandmark.
//////////////////////////////////////////////////////////////////////

/*************************************************************************
*                                                                        *
* NOM DE L'ARXIU:     DetectedLandmark.h                                 *
* DATA:               Desembre 2001                                      *
* VERSIÓ:             0.01                                               *
```

```
* OBJECTIU:           Definir totes les variables necessàries per un landmark  *
*                     de tipus paret                                           *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                    *
*                     (C) Institut de Robòtica Industrial                      *
*                     Universitat Politècnica de Catalunya                     *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                       *
*                                                                              *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                           *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                              *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                              *
*******************************************************************************/

#if !defined(AFX_DETECTEDLANDMARK_H__8E6F20A1_E31A_11D5_824D_00C04F034069__INCLUDED_)
#define AFX_DETECTEDLANDMARK_H__8E6F20A1_E31A_11D5_824D_00C04F034069__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "LlibreriaFuncions.h"

typedef double punto [2];

typedef struct
{
punto p1,p2,z1,z2;
float b1,b0,dist,maxdp2p,r2;
bool accept;
}pared;

class CDetectedLandmark
{
public:
CDetectedLandmark();
CDetectedLandmark(int id, classePUNT3D ego,
classePUNT3D absl, classePUNT2D img);
virtual ~CDetectedLandmark();

//Indica el tipus de landmark
int type;
int GetId() {return m_id;}
classePUNT3D GetEgoPosition() {return m_egoPosition;}
classePUNT3D GetAbsPosition() {return m_absPosition;}
classePUNT2D GetImgPosition() {return m_imgPosition;}
```

```
void SetEgoPosition(classePUNT3D new_pos)
{m_egoPosition = new_pos;}
void SetImgPosition(classePUNT2D new_pos)
{m_imgPosition = new_pos;}


//Variable que conté totes les dades de la recta en cas que el LMK sigui paret
pared p;

//Booleà utilizat per saber si el landmark ha estat associat amb un altre detectat abans
bool marca;

//Serveixen per ordenar els landmarks si es volen ordenar a una llista
float min,max;

double  ** covar1, ** covar2;


//Indica el número que portarà el landmark a la seva etiqueta si es vol utilizar
int m_id;
classePUNT3D m_egoPosition;
classePUNT3D m_absPosition;
classePUNT2D m_imgPosition;


private:

};

#endif // !defined(AFX_DETECTEDLANDMARK_H__8E6F20A1_E31A_11D5_824D_00C04F034069__INCLUDED_)
```

B.1.8 Pioneer2DX.cpp

```
// Pioneer2DX.cpp: implementation of the CPioneer2DX class.
//
//////////////////////////////////////////////////////////////////////

/*************************************************************************
*                                                                        *
* NOM DE L'ARXIU:     Pioneer2DX.cpp                                     *
* DATA:               Maig 2002                                          *
* VERSIÓ:             0.01                                               *
* OBJECTIU:           Defineix la classe CPioneer2DX i les seves funcions per  *
*                     obtenir la posició del robot o dibuixar-lo         *
* CODI ORIGINAL DE:   Albert Checa Puimedon                              *
*                     (C) Institut de Robòtica Industrial                *
```

```
*                     Universitat Politècnica de Catalunya                    *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                      *
*                                                                             *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                          *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                             *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                             *
*******************************************************************************/

#include "stdafx.h"
#include "RoboGUI.h"
#include "Pioneer2DX.h"

#include "saphira.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPioneer2DX::CPioneer2DX()
{
type = "Pioneer 2 DX";
m_bIsConnected = false;

//address = "pioneer.upc.es";
address = "147.83.76.240";

pan = 0;
tilt = 0;

disc_1=gluNewQuadric();
disc_2=gluNewQuadric();
disc_3=gluNewQuadric();
disc_4=gluNewQuadric();
disc_5=gluNewQuadric();
disc_6=gluNewQuadric();
disc_7=gluNewQuadric();
disc_8=gluNewQuadric();
cilindre_1=gluNewQuadric();
```

```
cilindre_2=gluNewQuadric();
cilindre_3=gluNewQuadric();
cilindre_4=gluNewQuadric();
cilindre_5=gluNewQuadric();
cilindre_frw=gluNewQuadric();
cilindre_flw=gluNewQuadric();
cilindre_brw=gluNewQuadric();
cilindre_blw=gluNewQuadric();
disc_frw = gluNewQuadric();
disc_flw = gluNewQuadric();
disc_brw = gluNewQuadric();
disc_blw = gluNewQuadric();
gluQuadricDrawStyle(cilindre_1,GLU_FILL);
gluQuadricDrawStyle(disc_1,GLU_FILL);
gluQuadricDrawStyle(cilindre_2,GLU_FILL);
gluQuadricDrawStyle(disc_2,GLU_FILL);
}

CPioneer2DX::~CPioneer2DX()
{

}

int CPioneer2DX::Connect(char *location, bool sim)
{
int port;

if (sfIsConnected) return true;

sfOnConnectFn(myConnectFn); /* register a connection function */
sfStartup(NULL, 0, 1);

sfBehaviorControl=1;

port = (sim)? sfLOCALPORT:TCP;

/*
m_x=0;
m_y=0;
m_th=0;
m_speed = 0;
m_frontBumpers = m_rearBumpers = 0;
*/
sfSerialBaud = 9600;
sfComServer=location;
sfComServerPort=8101;

bReady = false;

sfConnectToRobot(port,location);
```

```
if (sfIsConnected){

m_bIsConnected = true;

while (!bReady) ;

// sfRobotComInt(sfCOMBUMPSTOP,3); // Enable BUMPSTALL

// sfAddEvalFn("bumpers", bumpers, sfINT, 0);
// sfInitProcess(bumpers, "bumpers");

// sfAddEvalConst("sfGoToPos",sfBEHAVIOR,sfGoToPos);
// sfInitControlProcs();

sfSetMaxVelocity(50); // 50 mm por segundo
sfSetMaxRVelocity(20); // 5 grados por segundo

return 1;
}

return 0;
}

int CPioneer2DX::Disconnect()
{
sfDisconnectFromRobot();
m_bIsConnected = false;

return 1;
}

void CPioneer2DX::myConnectFn(void)
{
sfInitBasicProcs(); /* start up comm processes */
bReady = true;
}

int CPioneer2DX::SetMotors(bool on)
{
sfRobotComInt(sfCOMENABLE,on);
SetSpeed(0);
SetRSpeed(0);

return 1;
}

int CPioneer2DX::SetSpeed(double speed)
{
sfSetVelocity((int) speed);
```

```
return 1;
}

int CPioneer2DX::SetRSpeed(double speed)
{
sfSetRVelocity(speed);

return 1;
}

int CPioneer2DX::IncrSpeed(double incr)
{
sfSetVelocity(sfRobot.tv + (int)incr);

return 1;
}

int CPioneer2DX::IncrRSpeed(double incr)
{
sfSetRVelocity(sfRobot.rv + (int) incr);

return 1;
}

int CPioneer2DX::Turn(double angle)
{
sfSetDHeading(angle);

return 1;
}

int CPioneer2DX::Move(float xcm,float ycm)
{
sfprocess prc;

sfSetMaxVelocity(50);
sfSetPosition(xcm*10);

/* point *goal;
goal=sfCreateLocalPoint(xcm*10,ycm*10,0);
sfAddPoint(goal);
prc=*sfStartBehavior(sfGoToPos,"sfGoToPos",0,1,0,50,goal,1);
sfBehaviorOn("sfGoToPos");
// sfStartBehavior(sfKeepOff,"Keep off",0,1,0,100.0,0.4);*/

return 1;
}

int CPioneer2DX::Stop()
```

```
{
SetSpeed(0);
SetRSpeed(0);

return 1;
}

double CPioneer2DX::GetSpeed()
{
return (double) sfRobot.tv;
}

double CPioneer2DX::GetRSpeed()
{
return (double) sfRobot.rv;
}

void CPioneer2DX::GetPosition(double &x, double &y, double &th, double &speed,
  double &rspeed)
{
x = sfRobot.ax/10;
y = sfRobot.ay/10;
th = sfRobot.ath;
speed = sfRobot.tv;
// rspeed= sfRobot.rv;
rspeed= -sfRobot.rv;
}


void CPioneer2DX::GetState(unsigned char &fb, unsigned char &rb,
   int &lm, int &rm, int &batt)
{
fb = (sfRobot.bumpers & 0xFE00) >> 8;
rb = sfRobot.bumpers & 0xFE;
lm = sfStalledMotor(sfLEFT);
rm = sfStalledMotor(sfRIGHT);
batt = sfRobot.battery;
}


void CPioneer2DX::DrawRobot() {

glPushMatrix();


glRotated(90,0,0,1);
glTranslated(0,0,4);
glColor3f(1.0f,0.0f,0.0f);
gluCylinder(cilindre_1,20,20,20,40,10);  // base
gluDisk(disc_2,0,20,40,10);              // base por debajo
```

```
glTranslated(0,0,20);
glPushMatrix();
glColor3f(0.1f,0.1f,0.1f);
gluDisk(disc_1,0,20,40,10);              // placa por encima de la base

glColor3f(0.0f,0.0f,0.0f);
glTranslated(20,0,-16);
glRotated(90,0,1,0);
gluCylinder(cilindre_2,8,8,4,40,10);
gluDisk(disc_3,0,8,40,10);
glTranslated(0,0,4);
gluDisk(disc_4,0,8,40,10);

glTranslated(0,0,-44);
glRotated(180,0,1,0);
gluCylinder(cilindre_3,8,8,4,40,10);
gluDisk(disc_5,0,8,40,10);
glTranslated(0,0,4);
gluDisk(disc_6,0,8,40,10);

glPopMatrix();
glTranslated(8,-12,0);
glColor3f(1.0f,0.0f,0.0f);
glPrisma(2,2,1);
glTranslated(0,0,0.2);
glColor3f(0.0f,0.5f,0.0f);
glRectd(-3,3,-4,-3);

glTranslated(-8,12,-0.2);
glTranslated(0,5,0);
glColor3f(0.1f,0.1f,0.1f);
gluCylinder(cilindre_4,12,12,50,40,10);
glTranslated(0,0,50);
glRectd(-13,-13,13,13);

gluCylinder(cilindre_5,3,3,3,40,40);;
glColor3f(0.35f,0.35f,0.35f);
glTranslated(0,0,3);

pan=15;
tilt=-15;

glRotated(pan,0,0,1);
glRotated(tilt,1,0,0);
glTranslated(0,0,3*tan(tilt*3.141529/180));

glPrisma(24,4,2);
glTranslated(8,2,2);
glColor3f(0.08f,0.08f,0.08f);
glPrisma(7,12,6);
```

```
glTranslated(-16,0,0);
glPrisma(7,12,6);
glTranslated(0,-6,3);
glRotated(90,1,0,0);
glColor3f(0.2f,0.2f,0.2f);
glTranslated(0,0,0.1);
gluDisk(disc_7,0,2,10,10);
glTranslated(16,0,0);
gluDisk(disc_8,0,2,10,10);

glPopMatrix();
}

void CPioneer2DX::glPrisma(double b1, double b2, double h) {
glPushMatrix();
glTranslated(-b1/2,-b2/2,0);
glRectf(0,0,b1,b2);
glTranslated(0,0,h);
glRectf(0,0,b1,b2);
glRotated(90,0,1,0);
glRectf(0,0,h,b2);
glTranslated(0,0,b1);
glRectf(0,0,h,b2);
glRotated(-90,1,0,0);
glRectf(0,0,h,b1);
glTranslated(0,0,b2);
glRectf(0,0,h,b1);
glPopMatrix();
}

\end{verabtim}
\normalsize

\subsection{Pioneer2DX.h}

\footnotesize
\begin{verbatim}

// Pioneer2DX.h: interface per la classe CPioneer2DX.
//
//////////////////////////////////////////////////////////////////////

/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     Pioneer2DX.cpp                                         *
* DATA:               Maig 2002                                              *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Defineix la classe CPioneer2DX i les seves funcions per  *
*                     obtenir la posició del robot o dibuixar-lo             *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
```

```
*                    (C) Institut de Robòtica Industrial                      *
*                    Universitat Politècnica de Catalunya                     *
* CONTRIBUCIONS:     Juan Andrade-Cetto                                       *
*                                                                             *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                          *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                             *
* You may modify this code (or portions thereof) provided a copy of this      *
* disclaimer is included in the modified version.                             *
*******************************************************************************/

#if !defined(AFX_PIONEER2DX_H_INCLUDED_)
#define AFX_PIONEER2DX_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "RobotWrapper.h"

#ifndef bReady_defined
#define bReady_defined
static bool bReady;
#endif

#include "gl\gl.h"
#include "gl\glu.h"
#include <gl\glut.h>

//! ActivMedia Pioneer 2 DX class
/*! This class implements the robot Pioneer 2 DX from ActivMedia. */
class CPioneer2DX : public CRobotWrapper
{
public:
CPioneer2DX();
virtual ~CPioneer2DX();

virtual int Connect(char *location, bool sim);
virtual int Disconnect();

static void myConnectFn(void);

int SetMotors(bool on);
int Turn(double angle);
int Stop();
int SetSpeed(double speed);
```

```
int SetRSpeed(double speed);
int IncrSpeed(double incr);
int IncrRSpeed(double incr);
double GetSpeed();
double GetRSpeed();
void GetPosition(double &x, double &y, double &th, double &speed, double &rspeed);
void GetState(unsigned char &fb, unsigned char &rb, int &lm, int &rm, int &batt);
void DrawRobot();
int Move(float xcm,float ycm);

private:
double pan, tilt;
GLUquadricObj *disc_1;
GLUquadricObj *disc_2;
GLUquadricObj *disc_3;
GLUquadricObj *disc_4;
GLUquadricObj *disc_5;
GLUquadricObj *disc_6;
GLUquadricObj *disc_7;
GLUquadricObj *disc_8;
GLUquadricObj *cilindre_1;
GLUquadricObj *cilindre_2;
GLUquadricObj *cilindre_3;
GLUquadricObj *cilindre_4;
GLUquadricObj *cilindre_5;
GLUquadricObj *cilindre_frw,*cilindre_flw,*cilindre_brw,*cilindre_blw;
GLUquadricObj *disc_frw,*disc_flw,*disc_brw,*disc_blw;
void glPrisma(double, double, double);
};

#endif // !defined(AFX_PIONEER2DX_H_INCLUDED_)
```

B.1.9 Listoflmk.cpp

```
/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     Listoflmk .cpp                                         *
* DATA:               Agost 2002                                             *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Defineix la classe CListoflmk, que corresponen a les   *
*                     llistes de landmarks del mapa i landmarks detectats    *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
```

```
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a    *
* copy of the License at http://www.mozilla.org/MPL/                           *
* Software distributed under the License is distributed on an "AS IS" basis,   *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.    *
*                                                                              *
* You may modify this code (or portions thereof) provided a copy of this       *
* disclaimer is included in the modified version.                              *
*******************************************************************************/

#include "ListofLmk.h"


CListofLmks::CListofLmks()
{
numofWalls=0;

//La variable type del landmark de valor -1 ens indica que el landmark no és vàlid.
//S'utiliza per exemple per saber quants landmarks té la lista ja que el primer que
//té type -1 és el que va després de l'últim landmark de la lista. Al principi els
//posem tots a -1 perquè cap és vàlid
for(index=0;index<200;index++)
Data[index].type=-1;

index=0;
}

CListofLmks::~CListofLmks()
{
delete Data;
}

bool CListofLmks::isLast()
{
//Si un landmark és l'últim,el següent ha de tenir el type=-1
if(Data[index+1].type==-1 || index==199)
return 1;
else
return 0;
}

void CListofLmks::Reset()
{
numofWalls=0;

for(index=0;index<numofWalls;index++)
Data[index].type=-1;

index=0;
```

```
}

void CListofLmks::Insert(CDetectedLandmark lmk)
{
//Fiquem el landmark a la posició indicada
// Data[pos]=lmk;
Data[numofWalls]=lmk;
//Augmentem el comptador d'elements a la lista
numofWalls++;
//Tornem l'índex a que apunti al primer landmark
index=0;
}

void CListofLmks::Delete(int item)
{
//Movem els landmarks que van després del que es vol esborrar una posició
//cap a l'esquerra
for(index=item;index<numofWalls;index++)
Data[index]=Data[index+1];

//Invalidem l'últim que ha quedat repetit
Data[numofWalls].type=-1;
//Restem el nombre de landmarks que conté la lista
numofWalls--;
}

int CListofLmks::GetInsertPosition(float min,int ini,int fin)
{
int centro;

if(fin==ini)
{
if(min>Data[ini].min)
return (ini+1);
else
return ini;
}

centro=(fin+ini)/2;

if(min>Data[centro].min)
return (GetInsertPosition(min,centro+1,fin));
else
return (GetInsertPosition(min,ini,centro));
}

bool CListofLmks::Empty()
{
if(numofWalls==0)
return 1;
```

```
else
return 0;
}
```

B.1.10 Listoflmk.h

```
/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     Listoflmk .h                                           *
* DATA:               Agost 2002                                             *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Defineix la classe CListoflmk, que corresponen a les   *
*                     llistes de landmarks del mapa i landmarks detectats    *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a  *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis, *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.  *
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this     *
* disclaimer is included in the modified version.                            *
*****************************************************************************/


#if !defined(AFX_LISTOFLMK_H)
#define AFX_LISTOFLMK_H

#include "DetectedLandmark.h"

class CListofLmks
{
public:
CListofLmks();
~CListofLmks();

//Inserta el landmark a la seva posició corresponent
void Insert(CDetectedLandmark);
//Ens diu si el landmark és l'últim de la llista
bool isLast();
//Borra de la llista el landmark que està a la posició que li entrem
void Delete(int);
```

```
//Borra tots els landmarks de la llista
void Reset();
//Ens diu si la llista està buida
bool Empty();

//LLista que conté tots els landmarks
CDetectedLandmark Data [200];

//Ens diu el número d'elements que tenim dins de la llista
int numofWalls;

//És l'índex que apunta al landmark que es llegeixi actualment
int index;


private:
//Ens dóna la posició on s'ha d'inserir un landmark.
int GetInsertPosition(float,int,int);

};

#endif
```

B.1.11 matrices.h

```
/*****************************************************************************
*                                                                            *
* NOM DE L'ARXIU:     matrices.h                                             *
* DATA:               Agost 2002                                             *
* VERSIÓ:             0.01                                                   *
* OBJECTIU:           Defineix les operacions necessàries amb matrius        *
*                                                                            *
* CODI ORIGINAL DE:   Albert Checa Puimedon                                  *
*                     (C) Institut de Robòtica Industrial                    *
*                     Universitat Politècnica de Catalunya                   *
* CONTRIBUCIONS:      Juan Andrade-Cetto                                     *
*                                                                            *
* The contents of this file are subject to the Mozilla Public License; you may *
* not use this file except in compliance with the License. You may obtain a  *
* copy of the License at http://www.mozilla.org/MPL/                         *
* Software distributed under the License is distributed on an "AS IS" basis, *
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for *
* the specific language governing rights and limitations under the License.  *
*                                                                            *
* You may modify this code (or portions thereof) provided a copy of this     *
* disclaimer is included in the modified version.                            *
*****************************************************************************/
```

```
#if !defined(AFX_MATRICES_H)
#define AFX_MATRICES_H

#include <math.h>
#include <stdlib.h>
#include <malloc.h>

//Reservar espai de memòria per un vector de doubles
double *alloc_vec_d(int n) {
  return(double*)malloc(n*sizeof(double));
}

//Reservar espai de memòria per una matriu de doubles
double **alloc_mat_d(int r, int c) {
  int i;
  double *p_M;
  double **M;
  p_M = alloc_vec_d(c*r);
  M   = (double**)malloc(r*sizeof(double*));
  for(i=0; i<r;i++)
    M[i] = p_M + (i*c);
  return(M);
}

//Alliberar memòria d'un vector de doubles
void release_vec_d(double* v) {
  free(v);
}

//Alliberar memòria d'una matriu de doubles
void release_mat_d(double** M) {
  free(M[0]);
  free(M);
}

//Reservar espai de memòria per un vector de integers
int *alloc_vec_i(int n) {
  return(int*)malloc(n*sizeof(int));
}

//Alliberar memòria d'un vector de integers
void release_vec_i(int* v) {
  free(v);
}

//Producte escalar entre dos vectors
double dot(double *a, double *b, int n) {
  int i;
  double z;
```

```
  z = 0.0;
  for (i=0; i<n; i++)
    z += a[i] * b[i];
  return (z);
}

/*-------------------------------------------------------------------

        Producte de d'una matriu per un vector
                a:        input matrix  m x n
                b:        input vector  n x 1
                x:        output matrix m x 1

-------------------------------------------------------------------*/

void mult_mv(double **a, double *b, double *x, int m, int n)
{
int i,j;
for (i=0; i<m; i++)
{
x[i] = 0;
for (j=0; j<n; j++)
{
x[i] = x[i] + a[i][j] * b[j];
}
}
}

/*-------------------------------------------------------------------

        Producte de dues matrius
                a:        input matrix  m x n
                b:        input matrix  n x o
                x:        output matrix m x o

-------------------------------------------------------------------*/

void mult_mat(double **a, double **b, double **x, int m, int n, int o)
{
int i,j,k;
for (i=0; i<m; i++)
{
for (j=0; j<o; j++)
{
x[i][j] = 0;
for (k=0; k<n; k++)
{
x[i][j] += a[i][k] * b[k][j];
}
```

```
}
}
}

//Producte de un vector per un escalar
void mult_scvec(double a, double *b, double *x, int m)
{
int i;
for (i=0; i<m; i++)
{
x[i] = a*b[i];
}
}

//Producte d'una matriu per un escalar
void mult_scmat(double a, double **b, double **x, int m, int n)
{
int i,j;
for (i=0; i<m; i++)
{
for(j=0;j<n;j++)
{
x[i][j] = a*b[i][j];
}
}
}

//Suma de dos vectors
void sum_vec(double *a, double *b, double *x, int m)
{
int i;
for (i=0; i<m; i++)
{
x[i] = a[i]+b[i];
}
}

//Suma de dues matrius
void sum_mat(double **a, double **b, double **x, int m, int n)
{
int i,j;
for (i=0; i<m; i++)
{
for(j=0;j<n;j++)
{
x[i][j] = a[i][j]+b[i][j];
}
}
}
```

```
//Resta de dos vectors
void dif_vec(double *a, double *b, double *x, int m)
{
int i;
for (i=0; i<m; i++)
{
x[i] = a[i]-b[i];
}
}

//Resta de dues matrius
void dif_mat(double **a, double **b, double **x, int m, int n)
{
int i,j;
for (i=0; i<m; i++)
{
for(j=0;j<n;j++)
{
x[i][j] = a[i][j]-b[i][j];
}
}
}

//Creació d'una matriu identitat de les dimensions desitjades
void identity(double ** x, int m)
{
int i,j;
for (i=0; i<m; i++)
{
for (j=0; j<m; j++)
{
if(i==j)
x[i][j]=1;
else
x[i][j]=0;
}
}
}

//Creació d'una matriu diagonal amb els valors d'un vector a la diagonal
void diagonal(double * v, double ** x, int m)
{
int i,j;
for (i=0; i<m; i++)
{
for (j=0; j<m; j++)
{
if(i==j)
x[i][j]=v[i];
else
```

```
x[i][j]=0;
}
}
}

//Transposa una matriu
void transpose(double ** a, double ** x, int m, int n)
{
int i,j;
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
x[j][i]=a[i][j];
}
}
}

//Copia els valors d'una matriu o part d'ella a una altra
void Copiarmatriz(double ** a,double ** x,int nfi,int nci,int nff,int ncf,int inif,int inic)
{
int i,j,k,l;
k=inif;
for(i=nfi;i<nff;i++)
{
l=inic;
for(j=nci;j<ncf;j++)
{
x[k][l]=a[i][j];
l++;
}
k++;
}
}

//Copia un vector o part d'ell a un altre
void Copiarvector(double * a, double * x, int ni, int nf, int inin)
{
int i,j;
j=inin;
for(i=ni;i<nf;i++)
{
x[j]=a[i];
j++;
}
}

//Calcula la inversa d'una matriu 4x4
void inverse4x4(double ** m, double ** x)
{
```

```
double a,b,c,d,e,f,g,h,k,l,det;

a=m[0][0];
b=m[0][1];
c=m[0][2];
d=m[0][3];
e=m[1][1];
f=m[1][2];
g=m[1][3];
h=m[2][2];
k=m[2][3];
l=m[3][3];

det=(a*e*h*l-a*e*k*k-a*f*f*l+2*a*f*g*k-a*g*g*h-b*b*h*l+
b*b*k*k+2*b*f*c*l-2*b*f*d*k-2*b*g*c*k+2*b*g*d*h-e*c*c*l+
2*c*e*d*k+c*c*g*g-2*c*g*d*f-e*d*d*h+d*d*f*f);

x[0][0]=(e*h*l-e*k*k-f*f*l+2*f*g*k-g*g*h)/det;
x[0][1]=-(b*h*l-b*k*k-f*c*l+f*d*k+g*c*k-g*d*h)/det;
x[0][2]=(b*f*l-b*g*k-e*c*l+e*d*k+c*g*g-g*d*f)/det;
x[0][3]=-(b*f*k-b*g*h-e*c*k+e*d*h+f*c*g-d*f*f)/det;
x[1][0]=x[0][1];
x[1][1]=(a*h*l-a*k*k-c*c*l+2*c*d*k-d*d*h)/det;
x[1][2]=-(a*f*l-a*g*k-c*b*l+c*d*g+d*b*k-d*d*f)/det;
x[1][3]=(a*f*k-a*g*h-c*b*k+c*c*g+d*b*h-d*c*f)/det;
x[2][0]=x[0][2];
x[2][1]=x[1][2];
x[2][2]=(a*e*l-a*g*g-b*b*l+2*b*d*g-d*d*e)/det;
x[2][3]=-(a*e*k-a*g*f-b*b*k+b*c*g+d*b*f-d*c*e)/det;
x[3][0]=x[0][3];
x[3][1]=x[1][3];
x[3][2]=x[2][3];
x[3][3]=(a*e*h-a*f*f-b*b*h+2*b*c*f-c*c*e)/det;
}

//Creació d'una matriu nul·la de les dimensions desitjades
void matriz0(double **x,int nf,int nc)
{
int i,j;
for(i=0;i<nf;i++)
{
for(j=0;j<nc;j++)
{
x[i][j]=0;
}
}
}

//Escriure una matriu a un fitxer
void ImprimirMatriz(FILE *fp, double **m, int f1, int c1, int f2, int c2, char *nombre)
```

```
{
int i,j;

fprintf(fp,"Matriz: %s\n",nombre);
for(i=f1;i<f2;i++) {
for(j=c1;j<c2;j++) {
fprintf(fp, "%f   ", m[i][j]);
}
fprintf(fp,"\n");
}
fprintf(fp,"\n");
}

//Escriure un vector a un arxiu
void ImprimirVector(FILE *fp, double *m, int f1, int f2, char *nombre)
{
int i,j;

fprintf(fp,"Vector: %s\n",nombre);
for(i=f1;i<f2;i++)
{
fprintf(fp, "%f   ", m[i]);
}
fprintf(fp,"\n\n");
}

#endif
```