Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Implemetació de tècniques de percepció NDT
i ajust d'aquestes al domini submarí

Document: Memòria

Alumne: Miguel Malagón Pedrosa

Tutors: Pau Vial Serrat i Narcís Palomeras Rovira
Departament: Arquitectura i Tecnologia de Computadors
Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): Juny 2021

Universitat de Girona
Escola Politècnica Superior

Projecte Fi de Grau

Implemetació de tècniques de percepció NDT i ajust d'aquestes al domini submarí

Autors:
Miguel Malagón Pedrosa
Ricard Segura Duran

Juny 2021

Grau en Enginyeria Informàtica

Tutors:
Pau Vial Serrat
Narcís Palomeras Rovira

Resum

En aquest treball es dissenya i implementa una llibreria en C++ encarregada de solucionar el problema del registre de dades de sensors de rang per generar mapes d'ocupació submarins. Un punt important del disseny ha estat l'eficiència del codi ja que es vol una execució de la llibreria en temps real per millorar la qualitat de l'exploració submarina amb Vehicles Submarins Autònoms. Ja existeixen tècniques que permeten fer el registre amb sensors laser, però les nostres dades es recullen amb sensors acústics degut a que les ones electromagnètiques no són gaire bones per treballar sota l'aigua. Aquestes dades són molt més sorolloses i lentes, per tant s'ha adaptat la tècnica de registre a aquest tipus de dades. Com a punt de partida s'ha pres la tècnica Normal Distribution Transform basada en representar núvols de punts amb Gaussian Mixture Models.

Agraïments

Per començar vull agrair molt especialment al nostre tutor, Pau Vial per la gran ajuda per guiar-nos a fer el projecte i a tot l'entorn del CIRS per acollir-nos.

Índex

Capítol 1

Introducció

1.1 Motivacions

En aquest treball es vol aprofundir en la percepció i la construcció de mapes submarins, duta a terme per robots robots submarins. Aquests robots, coneguts com a Autonomous Underwater Vehicles (AUV), utilitzen sensors per poder percebre el seu entorn. Això és degut a que per les característiques de l'aigua no es poden utilitzar els sensors basats en fenòmens electromagnètics que s'utilitzen a nivell terrestre o aeri, com ara làsers o càmeres òptiques, i s'han d'utilitzar sonars, basats en fenòmens acústics.

Hi ha diversos problemes al treballar amb sensors acústics en l'entorn submarí. Els sonars, a diferència de la seva contrapart electromagnètica, són molt més lents i generen més soroll. Per tant, la utilització de les tècniques actuals de l'estat de l'art orientades principalment a sensors làser no donen resultats satisfactoris en l'entorn submarí.

1.2 Objectius

En aquest treball es vol implementar una tècnica específica per fer el registre de núvols de punts obtinguts amb un sonar en un entorn submarí. També es fixen els següents subobjectius per ampliar el principal:

- Lectura dels articles científics on es presenten les tècniques de registre que es prenen de referència.

- Proposar una estructura de classes per a la implementació de la llibreria ben organitzada i fàcil d'entendre, ja que posteriorment es publicara en un repositori per a que es pugui utilitzar per part de qualsevol altre investigador

- Aconseguir una implementació que s'executi en temps real sobre el vehicle Sparus II, per aquest motiu s'ha escollit C++ com a llenguatge per la llibreria.

- Per facilitar la seva utilització a tercers, fer la documentació del codi utilitzant el programa doxygen, el qual pot generar tant html com pdf per organitzar la documentació a partir del codi comentat d'una manera concreta.
- Testejar totes les implementacions amb dades reals adquirides amb els AUVs del laboratori.
- Per processar les dades de proba provinents d'una aplicació robòtica, utilitzar el middleware ROS que permet la comunicació dels diferents sistemes del robot.
- Aprofundir en l'eficiència computacional del codi implementant solucions particulars per mètodes específics.
- Ampliar la tècnica implementada pel processament de núvols de punts en 3 dimensions. Aquest tractament encara no s'ha efectuat en el laboratori on es desenvolupa aquest projecte.

1.3 Entorn de desenvolupment

El projecte es realitza en l'entorn de l'Institut VICOROB, concretament en el Centre d'Investigació en Robòtica Submarina, ubicat al Parc Tecnologic i Cientific de la UDG. El complex, un dels més avançats en la robotica submarina a nivell europeu, esta copost per dos edificis. Per un costat, el primer edifici allotja oficines i laboratoris. Per l'altra banda, el segon edifici disposa d'una serie de facilitats ideals per al testeig dels robots: un tanc de proves de 6x12 metres i 5 metres de profunditat, un pont grua per a la inmersio i recuperacio, tallers i sala de control situada per sota del nivell de l'aigua, que permet, una visió directa de l'interior del tanc a través d'una gran finestra subaquatica.

A més de tot això, el laboratori compta amb AUVs desenvolupats integrament pel grup de recerca. Aquests robots disposen d'una amplia gamma de sensors i equipaments d'ultima generacio, tals com unitats per a la navegacio (DVL, sensor presió, INS, GPS, USBL), camares d'alta definicio, modems acustics i també sonars d'imatges que poden fer escombrats laterals o de rang. Per a la realització d'experiments al mar, el grup disposa d'una embarcació de 7 metres d'eslora equipat amb un receptor GPS-RTK, AHRS i USBL. A part d'això, el grup d'investigacio compta amb dos tècnics altament capacitats per donar suport a doctorats, investigadors i estudiants, per a la realitzar la preparacio dels robots i per ajudar en la realitzacio d'experiments al mar.

Capítol 2

Viabilitat

2.1 Paràmetres necessaris per al desenvolupament del projecte

Per al desenvolupament d'aquest projecte és necessari disposar, per una banda, d'un ordinador amb sistema operatiu Ubuntu que sigui força potent per als diversos càlculs que es realitzaran. Per altra banda, cal tenir tot el sistema referent al submarí i enregistrament de les dades.

A part d'això, cal un sistema de connexió remot entre ordinador i submarí. Per això, és necessari disposar d'un encaminador per a donar cobertura en una àrea força amplia, com més amplia millor.

2.2 Costos dels paràmetres necessaris

Segons els paràmetres explicats anteriorment el cost serà més o menys elevats segons les especificacions i les diferents opcions per a cada paràmetre. Per a l'encaminador cal que tingui un bon senyal i capacitat de transmetre i rebre com més dades millor, que tingui banda 5 GHz és opcional, però molt recomanable, la banda necessària per a l'encaminador és la de 2,4 GHz. En el mercat actual un bon router que compleixi amb les nostres necessitats i que sigui ben valorat, pot valdre entre 30 a 150 euros.

L'ordinador haurà de complir diverses especificacions. Per a la memòria RAM es necessita que tingui entre 8 i 16 GB. En el nostre cas, com es farà en el sistema operatiu Ubuntu i les dades a mostrar no són molt pesades, no cal que incorporem cap targeta gràfica. Com a processador, un Intel i7 o un AMD Ryzen 7 serien el millor. En cas que per disponibilitat no fos possible trobar cap dels dos, un Intel i7 o AMD Ryzen 5 serien els substituts. Processadors que tinguin menor capacitat potser serien justos en algun moment de l'execució. Si s'opta per comprar un ordinador que ja vingui muntat amb aquests processadors, la memòria RAM i tots els altres components necessaris per a l'ordinador, el preu podria variar entre uns 600-700€ fins a uns 1400-1500€.

Malauradament, no es disposa d'un preu o un rang final per al conjunt de components que formen el submarí que es fa servir al CIRS (figura 2.1). No

obstant, si s'hagués de pagar el seu valor sense cap ajut, seria impossible de dur a terme aquest projecte per a un estudiant de grau normal.

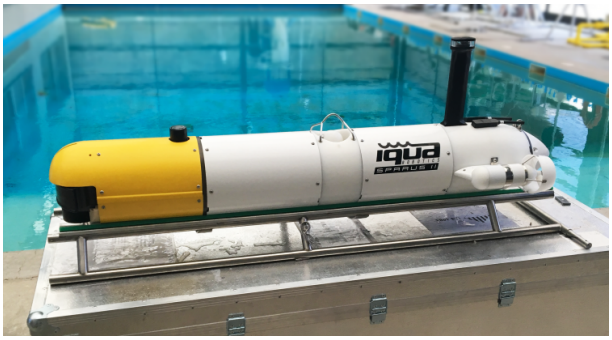

Figura 2.1: Exemple d'un AUV SPARUS II.

2.3 Costos de desenvolupament

Un cop definits els paràmetres necessaris per a poder iniciar el projecte, es pot tenir en compte el cost que suposaria per a una empresa, el fet de contractar un treballador per a fer aquesta feina.

Serà necessari un enginyer informàtic que porti a terme totes les tasques que s'han fet en aquest projecte. Aquesta persona ha d'estar cinc mesos treballant dedicant com a mínim quatre hores cada dia, excepte dels dies festius que hi hagi. Si el treballador cobra uns 20 € l'hora, el salari brut total per a aquesta persona seria d'uns 8400 €, tenint en compte que l'empresa hauria de pagar una mica més per contractar a aquesta persona a l'empresa.

Capítol 3

Metodologia

Per a aquest projecte s'ha seguit una metodologia de desenvolupament de tipus àgil, on per cada una o dues setmanes es defineix com un sprint per a cada funcionalitat. S'ha realitzat d'una forma iterativa i incremental, on cada versió nova que es genera millora la versió anterior del projecte, refinant i millorant les diferents parts i on alhora es van generant noves característiques.

Com que el projecte es va començar al gener, només es disposaven de quatre mesos per a la realització del projecte, i hi havia moltes tasques a fer: estudi previ de la tècnica, desenvolupament de les interfícies, documentació, implementació i ús d'altres tècniques en l'estat de l'art, etc., es va fer un desglossament en etapes i una posterior separació de la feina de cada etapa a fer fent servir aquest tipus de metodologia.

El desenvolupament d'aquest projecte és particular: hi ha una part de desenvolupament conjunt amb un altre estudiant i una part de desenvolupament individual. La part comuna s'ha desenvolupat amb l'estudiant Ricard Segura, que realitza un projecte amb la mateixa particularitat. Ambdós projectes es basen en la implementació de la mateixa llibreria per després fer les evolucions pertinents per a la realització de cada projecte. S'ha decidit de fer la part base de la llibreria conjuntament per trencar el gel en la temàtica amb companyia i alleugerir la càrrega de feina evitant haver de fer dues vegades la mateixa implementació. Alhora, fer el disseny de la llibreria conjuntament, ha permès acordar-lo i desenvolupar els dos projectes sobre la mateixa idea. Per aquest motiu els projectes s'han desglossat en dues parts, una part és la part comuna de familiarització, disseny i implementació de la part base de la llibreria i, una segona part, per al treball individual d'experimentació i ampliació corresponents.

Per a la part comuna s'ha fet una separació en diferents etapes. De cada etapa es mira al principi que és el que hauria d'haver-hi i que caldria fer. A partir d'aquí es comenta amb el professorat per veure si és correcte el plantejament i, posteriorment, es reparteix la feina a fer. En la part individual també s'ha realitzat una divisió de la feina en etapes.S'ha definit en etapes incrementals, començant per l'optimització general dels mètodes de registre, passant a una optimització més concreta. Per últim, l'adaptació de la llibreria a núvols 3D en dues parts, Front-End i Back-End.

Capítol 4

Planificació

4.1 Estratègia per al desenvolupament

Tenint en compte la metodologia explicada al capítol 3, es va fer una primera reunió d'inici del projecte. En aquesta reunió es va fer la planificació de la feina per al projecte. Es van desglossar les tasques de desenvolupament principals per a la part conjunta per realitzar els tres tipus d'objectes en diferents terminis. Posteriorment, es duria a terme una documentació parcial de la part conjunta i es començaria a fer la part única de cada estudiant (figura 4.1 sobre la planificació).

4.2 Planificació tasques

A la figura 4.1 es mostra el diagrama de Gantt de la planificació del projecte on s'hi mostra la planificació organitzada en setmanes i tasques. De les tasques que es veuen les que componen la part comuna del projecte són les etapes de la 1 fins a la 7, començant per a l'aprenentatge de la tècnica i una primera implementació, passant per al desenvolupament de les diferents components de la llibreria, fins a arribar a la documentació de tot el que es va fer en la part conjunta del projecte.

Les tasques que van a continuació d'aquestes, són les etapes en què s'ha dividit la part individual. Aquí s'ha realitzat l'optimització del codi donant una setmana per aplicar la paral·lelització i dos per trobar i implementar altres maneres de retallar temps d'execució. I finalment abans de dedicar-se a escriure la memòria, s'han donat 2 setmanes per ampliar el Front-End a casos 3D i 2 més per el Back-End.

4.3 Comentaris respecte a la planificació

Ja des d'un principi, va haver-hi canvis respecte a la planificació original, l'etapa inicial d'aprenentatge amb una primera implementació es va allargar una setmana més. Com que era una tècnica nova amb la qual no s'havia treballat anteriorment, va ser més costos a l'hora d'entendre com funcionava

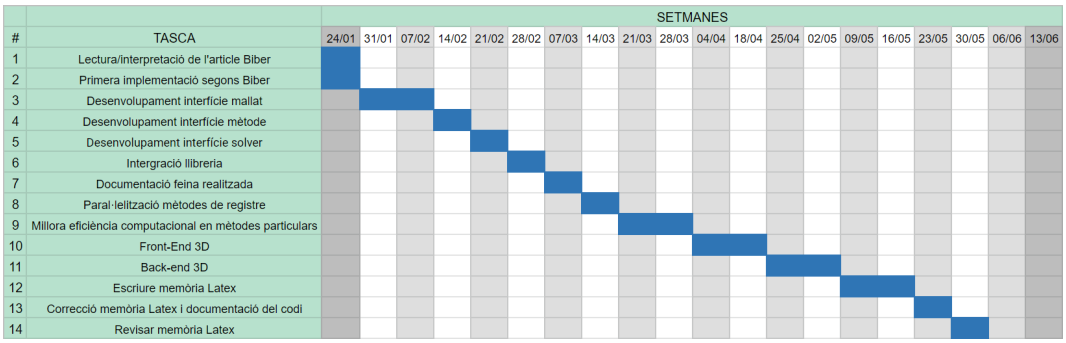

Figura 4.1: Planificació inicial per al projecte.

i es va decidir allargar aquesta etapa. I encara hi va haver més retrassos deguts a una problemàtica en l'ampliació a tres dimensions del Back-End.

El desenvolupament d'aquest projecte no ha estat diferent que d'altres projectes, hi ha hagut diversos problemes al llarg de tot el projecte que han afectat la planificació. Per dificultats amb diverses d'aquestes tasques, diferents etapes van portar més temps de l'esperat, per la qual cosa, totes les etapes següents es van veure afectades, arribant així a què hi hagués múltiples etapes en una mateixa setmana.

Capítol 5

Marc de treball i conceptes previ

L'entorn submarí és un domini complex i desfavorable per a les aplicacions robòtiques, ja que la majoria de sistemes de percepció i comunicació utilitzats en l'àmbit terrestre no hi funcionen. L'aigua actua com un gran atenuador de les ones electromagnètiques i, en conseqüència, els sistemes de comunicació Wi-Fi o de localització GPS no hi funcionen. Alhora, el camp de visió de les càmeres òptiques no arriba més enllà dels pocs metres si no hi ha partícules en suspensió a l'aigua, cosa també difícil de garantir. A més, si es volen fer servir sensors làsers subaquàtics - que encara es troben en fase de desenvolupament - s'han d'emprar feixos molt potents que són molt perillosos per a qualsevol humà que hi estigui a prop. Per tant, sota l'aigua cal recórrer a fenòmens acústics per construir sistemes de comunicació i percepció, ja que les ones mecàniques sí que s'hi propaguen bé. No obstant això, les ones mecàniques són ordres de magnitud més lentes que les ones electromagnètiques i, en conseqüència, proporcionen menys dades i molt més sorolloses. En conseqüència, els algoritmes que les processen han de ser més avançats, amb una robustesa i complexitat superiors.

Els Vehicles Autònoms Submarins (AUV) són plataformes robòtiques dotades de sistemes de percepció i control que operen sota l'aigua i que permeten efectuar tasques de localització, mapatge o planificació de trajectòria de manera totalment autònoma, és a dir, sense interacció amb cap altre sistema. Per tant, un cop definida la missió a realitzar, el robot se submergeix i la intervenció humana es limita només a la supervisió de què el sistema del robot es mantingui en funcionament i, si no, anar-lo a recollir quan per flotabilitat arribi a la superfície. Actualment, els AUVs són una tecnologia a l'alça que ha de permetre avançar en l'exploració del fons marí i en el manteniment autònom de nombroses infraestructures submarines. Tots els estudis mostren un interès creixent per aquest tipus de tecnologia i la necessitat d'aquests robots per moltes aplicacions marines: des de l'exploració científica del fons marí per interessos biològics, geològics o arqueològics; fins a la inspecció de plataformes petrolieres, parcs eòlics, gasoductes o línies elèctriques.

En l'actualitat els AUVs comercials són àmpliament utilitzats per a l'exploració a mar obert. En aquest mode de funcionament, els robots segueixen trajectòries predefinides adaptant només la profunditat per mantenir una alçada constant respecte el fons marí, la qual cosa es detecta amb sensors

de rang acústic. Tanmateix, és necessari dotar aquests vehicles de capacitats avançades que els permetin desenvolupar tasques més intel·ligents, com l'exploració o la inspecció de determinades zones que tenen una estructura tridimensional complexa i que no puguin ser resoltes amb trajectòries predefinides. Aquest tipus d'entorns tenen un gran interès científic i industrial, encara que actualment només són accessibles per submarinistes professionals o per Vehicles Operats Remotament (ROV) que fan servir un cable umbilical.

El mapeig de zones submarines emprant sensors de rang acústic muntats en AUVs actualment està limitat per l'acumulació de deriva durant la navegació del robot amb sensors inercials. La deriva provoca que quan es revisita un espai ja explorat les vistes no concordin i el mapa resultant sigui incoherent. Per evitar-ho, cal dotar el robot d'un sistema d'associació de vistes que permeti determinar el desplaçament del robot que registra les dues vistes. El resultat del registre es pot utilitzar en una aplicació de Mapeig i Localització Simultanis (SLAM) ([Durrant-Whyte 2006] i [Bailey 2006]) que, combinat amb la navegació a la deriva, permeti determinar la localització del robot en un mapa coherent del seu entorn construït a partir de les dades del sensor de rang. El problema de SLAM és un problema fonamental dins la comunitat de la robòtica mòbil. En aquest treball es vol avançar en aquest problema implementant una eina de registre de núvols de punts especialitzada per núvols de punts adquirits amb sensors de rang acústics. Aquests núvols de punts tenen la propietat de ser poc densos - és a dir, d'estar formats per un nombre de punts reduït - i d'estar contaminats per molt de soroll procedent del sensor.

5.1 El problema de registre

Donat un robot en moviment equipat amb un sensor de rang que a l'instant t_0 observa un obstacle i a l'instant t_1 observa el mateix obstacle des d'una altra posició a l'espai; el problema de registre vol determinar quin és el desplaçament del robot entre t_0 i t_1 a partir de les observacions de l'entorn. A la figura 5.1 esquerra es pot veure que els núvols de punts captats per un sensor de rang no quadren si s'associen segons el resultat de la navegació a la deriva del robot, és a dir, a partir de la integració de les mesures dels sensors inercials del robot. En canvi, a la figura 5.1 dreta es pot veure com aplicant un algorisme de registre les vistes quadren i es pot generar un mapa coherent de l'entorn del robot. Per tant, el problema de registre determina la transformada ${}^{k}T_{k+1}$ entre les posicions consecutives k i $k+1$ del robot.

Els algorismes de registre es poden classificar en dues categories. Per una banda, la categoria dels mètodes punt a punt inclou tots aquells mètodes basats en l'algorisme del Punt més Proper Iteratiu [ICP 022]. Es tracta de

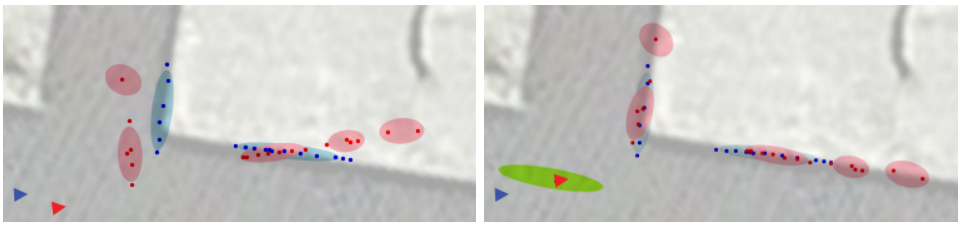

Figura 5.1: Exemple del problema de registre amb dades d'un Mechanical Scanning Profiling Sonar muntat en un AUV captades a l'escullera del Port de Sant Feliu de Guíxols. Esquerra: Associació de vistes a partir del sistema de navegació a la deriva. En blau el núvol de punts de referència i en vermell el núvol de punts mòbil. Dreta: Vistes registrades.

mètodes iteratius on en cada iteració s'associa cada punt de l'escaneig mòbil al punt més proper de l'escaneig de referència i es busca la transformació entre els escanejos que minimitza la distància euclidiana. Quan les associacions no canvien o la distància no millora, l'algorisme ha convergit.

Per altra banda, la categoria dels mètodes basats en camps inclou tots aquells mètodes on almenys algun dels dos núvols de punts s'expressa mitjançant un Model de Mescles de Gaussianes (GMM). Un GMM és un model molt emprat per la comunitat de l'aprenentatge automàtic per classificar dades de manera no supervisada que té la següent forma:

$$p(x|\theta) = \sum^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \qquad \text{amb} \qquad \sum^{K} \pi_k = 1, \tag{5.1}$$

on $\mathcal{N}(x|\mu_k, \Sigma_k)$ és un model Gaussià amb mitjana μ_k i covariància Σ_k i π_k és el pes de la component gaussiana k per assegurar que la suma de totes les distribucions integra a u. Existeixen diverses tècniques per fitar un GMM a un núvol de punts, a partir d'ara anomenades Front-End. Un dels Front-Ends més bàsics consisteix a generar una component gaussiana per cada punt de l'escaneig i donar a totes les components la mateixa covariància. Aquesta tècnica l'utilitza l'algorisme ICP Generalitzat [Generalized-ICP 022].

Dins de la categoria de mètodes basats en camps podem distingir entre els mètodes Punt a Distribució (P2D) i Distribució a Distribució (D2D). Els mètodes P2D busquen la solució de Maximum Likelihood de l'escaneig mòbil $\mathcal{M}$ expressat en forma de núvol de punts i l'escaneig fixe $\mathcal{F}$ modelat per un GMM:

$$p(\mathcal{D}|\theta) = \prod^{\mathcal{D}} p_{\mathcal{F}}(x = R(\phi)q_d + t|\pi, \mu, \Sigma) = \prod^{\mathcal{D}} \sum^{K} \pi_k \mathcal{N}(x = R(\phi)q_d + t|\mu_k, \Sigma_k),$$

on $p_{\mathcal{F}}(x|\pi,\mu,\Sigma)$ modela l'escaneig fixe, $\mathcal{D} = \{q_1,\ldots,q_n\}$ és el núvol de punts mòbil i $\theta = (t,\phi)$ és la rotació i la translació que es volen determinar. Es tracta d'un mètode costós computacionalment, ja que cal avaluar el model de $\mathcal{M}$ per cada punt de $\mathcal{F}$.

Els mètodes D2D minimitzen la distància $\mathcal{L}^2$ entre dos núvols de punts modelats amb un GMM. La distància $\mathcal{L}^2$ entre dos GMM té la forma analítica derivada a [Jiang 2011] que permet expressar el problema com:

$$\theta^* = \arg\max_{\theta} \mathcal{L}^2(\theta) = \arg\max_{\theta} \sum_{i=1}^{n_{\mathcal{F}}} \sum_{j=1}^{n_{\mathcal{F}}} \pi_{\mathcal{F}i}\pi_{\mathcal{M}j}\mathcal{N}(0|\mu_{\mathcal{F}i} - R(\phi)\mu_{\mathcal{M}i} - t, \Sigma_{\mathcal{F}i} + R(\phi)\Sigma_{\mathcal{M}i}R(\phi)^T),$$

on $p_{\mathcal{F}}(x|\pi,\mu,\Sigma)$ modela l'escaneig fixe, $p_{\mathcal{M}}(x|\pi,\mu,\Sigma)$ modela l'escaneig mòbil i $\theta = (t,\phi)$ és la rotació i la translació que es volen determinar. Es tracta d'un mètode menys costós computacionalement, ja que modelar els dos escanejos per mitjà de GMMs permet comprimir les dades.

5.2 La Transformada de Distribucions Normals (NDT)

La tècnica de registre coneguda com a Transformada de Distribucions Normals (NDT) va ser proposada per primer cap a [Biber 2003]. Es tracta d'una tècnica de registre de núvols de punts basada en camps que primer es va plantejar com a P2D, donant lloc a la metodologia NDT-P2D. Més endavant, a [Stoyanov 2012] es va formular el problema D2D, donant lloc a la metodologia D2D.

El tret distintiu i característic de totes les metodologies NDT és el seu Front-End. A diferència de la tècnica GICP anteriorment presentada, la tècnica NDT té per objectiu comprimir el núvol de punts complint $N << K$, on N és la dimensió del núvol de punts i K és el nombre de components del GMM. Per fer-ho, es genera una graella cartesiana uniforme sobre el núvol de punts i s'assigna cada punt a una casella. Per cada casella amb un nombre suficient de punts p_{min} es genera una component k:

$$\begin{aligned}
\mu_k &= \frac{1}{N_k}\sum^{N} x_n, \\
\Sigma_k &= \frac{1}{N_k}\sum^{N} (x_n - \mu_k)(x_n - \mu_k)^T, \\
\pi_k &= \frac{N_k}{N},
\end{aligned}$$

on $X = (x_0, \ldots, x_{N_k})$ són els punts de l'escaneig associats a la cel·la k.

Com que les funcions objectives dels problemes d'optimització que defineixen les metodologies P2D i D2D són contínues i diferenciables, la tècnica NDT suggereix resoldre'ls aplicant el mètode de Newton (algorisme 10.1 de [Bierlaire 2015]). Es tracta d'un mètode iteratiu definit a partir de l'aproximació d'ordre 2 de la Sèrie de Taylor que requereix el càlcul de la primera i la segona derivada de la funció objectiu. Encara que l'avaluació de la segona derivada de la funció incrementi la complexitat del càlcul, la seva convergència és major que pels algorismes d'una sola derivada.

Capítol 6

Requisits del sistema

6.1 Requisits funcionals

Aquesta secció conté tots els requisits funcionals que ha de tenir la nostra llibreria de cares a fer-la publica per a altres investigadors:

- Programació en C++
- Recepció de dades
- Extracció dels núvols de punts (2D o 3D)
- Extracció de l'odometria
- Creació d'objectes de Gaussian Mixtures Model
- Creació de Scan Matching Methods amb distribució P2D o D2D
- Calcul score, primera i segona derivades amb Scan Matchin Method
- Creació de Solvers amb Newton Method o LineaSearch Newton Method
- Calcul de l'optim amb el Solver

6.2 Requisits no funcionals

Aquesta secció conté tots els requisits no funcionals de la nostra llibreria:

- Llibreria del Front-End
- Plots de Gaussian Mixtures Models
- Plots d'objectes Scan Matching Mehods
- Plots d'objectes Solver
- Paràmetres com a shared pointers o constants

Capítol 7

Estudi i decisions

Al llarg del desenvolupament del projecte s'han fet servir diversos softwares i programes. En alguns casos s'han utilitzat de principi a fi, mentre que en altres, han estat descartades posteriorment per complexitat o per incompatibilitat amb altres eines o amb el mateix sistema.

7.1 Descripció del maquinari

Com s'ha especificat anteriorment en el capítol 2, la màquina en la qual s'han de fer les operacions cal que sigui força potent per a la part de processadors i memòria. Per a la part de la memòria RAM s'ha optat per un ordinador amb una capacitat de 16 GB, de processador s'ha fet servir un Intel-i7, tot i que es podria haver fet servir un Intel-i5 o el corresponent processador d'AMD. Com que en el nostre cas s'ha utilitzat el sistema operatiu Ubuntu i les dades a mostrar no han estat molt pesades, no s'ha incorporat cap targeta gràfica.

7.2 Programari per al desenvolupament

Per escriure codi i gestionar les versions amb més comoditat han calgut dos softwares bàsics. Un és el Visual Studio Code, com a editor de codi, i l'altre és la plataforma Bitbucket, com a gestió de versions Git.

El Visual Studio Code [VisualSC 022] s'ha fet servir tant per l'edició de codi com per pujar al repositori les noves modificacions. Es tracta d'un editor de codi font que ha estat construit sobre el framework anomenat Electron [Electron 022]. Aquest editor disposa de diferents opcions i terminals per a la realització de comandes Git del projecte.

Bitbucket [Bitbucket] és una eina d'allotjament de codi i col·laboració per a equips basada en Git [Git 022]. Amb aquesta eina s'ha generat un repositori on s'ha guardat tant el codi de la branca principal com totes les altres branques que s'han generat per a la realització de les tasques.

7.3 Llibreries i Frameworks emprats

7.3.1 Llibreries

Pel desenvolupament d'aquest projecte s'han emprat tres llibreries diferents: la llibreria Eigen [Eigen], la lliberia Matplotlib per a C++ [Matplotlib 022] i la Point Cloud Library [PCL 022]. La llibreria Eigen [Eigen] s'ha fet servir des de el principi del projecte, ja que permet definir objectes matricials i realitzar operacions matricials en C++. És l'equivalent en C++ de la llibreria NumPy de Python [Python 022]. Eigen és una llibreria en C++ que es troba templaritzada per a l'àlgebra lineal. Permet definir matrius i vectors i disposa de solvers numèrics i molts algoritmes de l'àlgebra lineal.

La llibreria de Matplotlib [Matplotlib 022] per a ús en C++ és una llibreria creada per poder fer servir la llibreria Matplotlib nativa del llenguatge Python en C++, amb l'objectiu de poder visualitzar dades. El problema d'aquesta llibreria és que els objectes escrits en C++ són limitats i per moltes tasques encara cal recòrrer a la versió de Python.

7.3.2 Frameworks

Pel desenvolupament d'aquest projecte s'han fet servir dos frameworks: un és ROS i l'altre és Boost. El Robotic Operation System (ROS) [ROS] és un framework, on s'agrupen un conjunt d'eines, llibreries i convencions informàtiques que tenen com a objectiu fer que sigui més simple la creació i programació amb robots.

El Boost [Boost 022] és un framework creat per a estendre les capacitats del llenguatge de programació C++. Aquest conjunt de llibreries aporta ajuda a l'hora de dur a terme tasques i estructures tals com l'àlgebra lineal, generació de nombres aleatoris, multiprocessament, etc.

7.4 Programari descartat

En aquesta secció es nombren els diferents softwares que han estat descartats durant el desenvolupament del projecte. Com a editor de codi també hi havien les opcions de Gedit [Gedit 022], Atom [Atom 022] i Sublime Text [SublimeText 022]. No obstant, com que el Visual Studio Code [VisualSC 022] també permet l'ús del terminal i la realització de comandes Git, les altres van ser descartades.

Per a mostrar les dades en plots es varen provar varis softwares, entre ells openGL [OpenGL 022], Gnuplot [Gnuplot 022] i Matlab [Matlab 022]. No

obstant, per problemes de compatibilitat, ja sigui amb el mateix repositori amb softwares que ja es feien servir, van ser rebutjats.

Capítol 8

Anàlisi i disseny del sistema

8.1 Anàlisi del problema

Després de fer una primera implementació senzilla de la tècnica sense la utilització de classes, es va observar que la tècnica es pot dividir en tres parts relacionades entre elles. Per una banda, s'han de fitar un Gaussian Mixtures Models (GMM) a un núvol de punts. Per una altra, s'han d'establir les funcions que defineixen el problema de registre com un problema d'optimització i, finalment, solucionar el problema d'optimització.

La primera part és el que anomenem Front-End, el qual donat un núvol de punts fita un GMM. Un GMM és un model probabilístic format per un sumatori de distribucions gaussianes tal com s'observa a l'equació 5.1. Aquestes distribucions, anomenades components k, estan definides per una mitjana μ_k, una matriu de covariància Σ_k i un pes π_k; on la suma dels pesos de totes les components ha de ser 1. Per tant, el Front-End permet passar d'una representació discreta de la superfície detectada a un model continu, a més de fer una compressió de les dades.

La segona part és el mètode de registre. Aquest defineix el problema de registre com un problema d'optimització, definint variables de decisió - increment de pose entre els dos scans a registrar - i funció de cost - que poden definir els dos problemes plantejats a la secció 5.1: Punt a Distribució (P2D) i Distribució a Distribució (D2D) -. Per una banda, el mètode P2D registra un scan parametritzat com a núvol de punts contra un scan parametritzat amb un GMM. Per tant, la funció de cost maximitza el likelihood del núvol de punts sobre el model avaluant tots els punts del núvol a totes les components del GMM. Per altra banda, el mètode D2D registra dos scans parametritzats amb GMM. Per tant, la funció de cost defineix la divergència entre les dues distribucions per minimitzar-la.

Per últim, el solucionador utilitza mètodes numèrics per resoldre el problema d'optimització definit pel mètode de registre. En aquest treball ens centrem en mètodes del gradient, és a dir, mètodes basats en les derivades de la funció de cost.

8.2 Disseny del sistema

L'element base de la llibreria és la classe GaussianMixtureModel que permet emmagatzemar un GMM i inclou mètodes de visualització. El Front-End s'ha implementat com una llibreria de funcions, les quals actuen com constructors de la classe GaussianMixturesModel, donant un núvol i certs paràmetres d'entrada per obtenir un GMM de sortida. Al no tenir una única tècnica per fitar un GMM a un núvol de punts, s'ha decantat per crear aquesta llibreria. Altrament, els constructors que tinguin els mateixos tipus d'argument, s'haurien de diferenciar amb un paràmetre extra que especifiqui quina tècnica es vol utilitzar. Generant un constructor que crida altres constructors segons el valor d'aquest paràmetre, complicaria innecessàriament la creació d'un GMM. En canvi, utilitzant la llibreria podem donar diferents noms als constructors sense complicar-los.

Els mètodes de registre s'han implementat amb una interfície i un seguit de subclasses. Això ens permet utilitzar les funcions definides per la interfície que estan implementades en cada subclasse sense haver de saber quina és la subclasse, el que ens permet donar al solucionador un punter a un objecte del tipus de la interfície i que aquest utilitzi les funcions implementades a les subclasses. Les funcions que utilitza el solucionador són compute_score(), compute_score_and_gradient() i compute_score_gradient_and_hessian(). Gràcies a la interfície, el solucionador funciona amb qualsevol de les subclasses sense la necessitat d'implementar un solucionador per cada ScanMatchingMethod. La interfície està templaritzada per poder definir subclasses tant en casos bidimensionals com tridimensionals, ja que les diferents funcions depenent fortament de les dimensions de les variables de decisió. Alhora, s'ha decidit que els scans a registrar siguin atributs de la interfície definits al construir la classe. Per tant, per cada registre particular cal construir un mètode particular. D'aquesta manera s'evita haver de passar els scans complerts cada vegada que es vol avaluar el cost o les seves derivades, fent transparent el solucionador dels scans a registrar.

El solucionador, igual que els mètodes de registre, s'ha implementat amb una interfície, ja que es volen fer diferents variacions de solucionadors que es puguin utilitzar de la mateixa manera. Es defineixen unes funcions a implementar en totes les subclasses, com la funció compute_optimum(), que és l'encarregada de fer l'optimització. Aquesta interfície també es troba templaritzada segons la dimensió, però a diferència de l'anterior, les subclasses també ho estan, ja que el mètode de Newton i les modificacions implementades segueixen la mateixa estructura independentment de la dimensió.

L'estructura bàsica del codi per utilitzar la llibreria és la següent. En primer lloc, cal fitar un GMM als scans pertinents que es volen registrar:

```
shared_ptr<GaussianMixturesModel<2>> gmm = ndt_constructor(scan1, 3,
    5);
```

A continuació, es defineix el mètode de registre a utilitzar passant els scans a registrar en la seva forma correcta segons el mètode triat:

```
PointsToDistribution2D p2d = PointsToDistribution2D(gmm,scan2);
shared_ptr<ScanMatchingMethods<3>> method =
    make_shared<PointsToDistribution2D>(p2d);
```

Tot seguit, es defineix el solucionador que es vol aplicar passant el mètode ja construït:

```
LineSearchNewtonMethod<3> lsnm (meth, 0.0001, 0.0001, 100,
    pow(10,-4), 0.99, 2.0, 100);
shared_ptr<Solver<3>> solv =
    make_shared<LineSearchNewtonMethod<3>>(lsnm);
```

Finalment, es resol el problema d'optimització cridant el mètode compute_optimum() del solucionador, passant-li una inicialització per la solució que normalment prové del sistema de navegació a la deriva del robot:

```
solution = solv->compute_optimum(make_tuple(t,h))
```

El solucionador retorna el valor de la transformada que registra els dos scans i, alhora, una matriu de covariança que mesura la incertesa de la solució donada.

El diagrama amb l'estructura final de les classes de la llibreria es mostra a la figura 8.1. S'hi mostren les interfícies i les herències implementades. Pel que fa a la implementació de la llibreria de Front-Ends, en aquest projecte només s'ha implementat una funció basada en la tècnica NDT, anomenada ndt_constructor(). La tècnica NDT clusteritza un núvol de punts projectant una graella sobre el núvol. Per cada cel·la amb un mínim nombre de punts, genera una component al GMM fitant una distribució gaussiana.

De la interfície ScanMatchingMethods s'han implementat 3 subclasses. Les classes Points2Distrtibution2D i Points2Distrtibution3D, defineixen un registre P2D en dues i tres dimensions respectivament: mentre que la classe Distrtibution2Distrtibution2D, defineix un registre D2D bidimensional. Les implementacions s'han separat completament ja que, tot i que la funció de cost només varia entre P2D i D2D, els càlculs de les derivades són completament diferents en el cas bidimensional i en el cas tridimensional. El codi és impossible de templaritzar. Destacar que la classe Distribution2Distribution3D encara no està implementada i forma part del treball futur del projecte.

Finalment, de la interfície Solver s'han implementat 4 subclasses. Totes

tenen de base la implementació de la classe NewtonMethod, la qual aplica el mètode de Newton per resoldre un problema de minimització sobre la funció de cost del registre i obtenir la transformació òptima entre els scans. En aquest algorisme, segons les derivades primera i segona de la funció objectius definides a les implementacions de ScanMatchingMethod, a cada iteració es decideix una direcció i un pas per a l'optimització. La classe LineSearchNewtonMethod afegeix al mètode de Newton un algorisme basat en les condicions de Wolfe que determina la mida òptima del pas que defineix el mètode de Newton a cada iteració. La classe CholeskyNewtonMethod aplica la factorització de Cholesky modificada per factoritzar la matriu hessiana i pertorbar-la si és indefinida per fer-la definida positiva i assegurar una direcció de minimització. Per acabar, la classe CholeskyLineSearchNewtonMethod combina les funcionalitats de LineSearchNewtonMethod i CholeskyLineSearchNewtonMethod per tenir un solucionador complet.

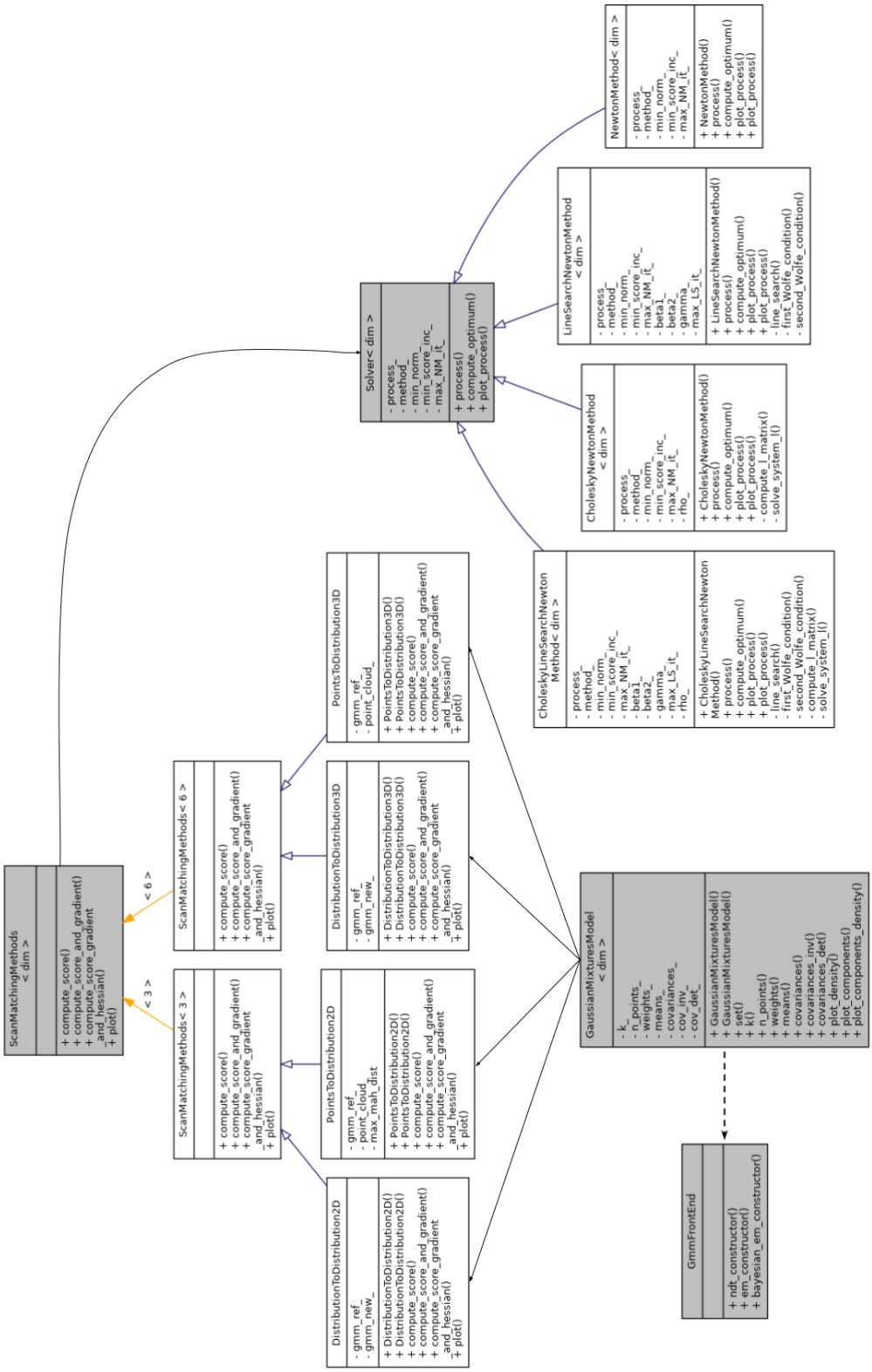

Figura 8.1: Diagrama de classes

Capítol 9

Implementació i proves

9.1 Objecte Gaussian Mixtures Model

Tota la llibreria es basa en la utilització de Gaussian Mixtures Models, els quals són models probabilístics definits per múltiples distribucions gaussianes. Presentats a la secció 5.1.

Per tant, es va decidir implementar la classe GaussianMixturesModel per emmagatzemar les dades necessaries que conformen un GMM. Un GMM es pot definir amb els paràmetres següents: K número de components del model, $\phi_{i=1...K}$ pes de cada component on la suma de totes ha de ser 1, $\mu_{i=1...K}$ la mitjana de la component i i $\sigma_{i=1...K}$ covariància de la component i. Aquests paràmetres es guarden com atributs de l'objecte de la següent manera:

```
class GaussianMixturesModel{
    private:
        int k_;

        shared_ptr<vector<int>> n_points_;
        shared_ptr<vector<double>> weights_;
        shared_ptr<vector<Eigen::Matrix<double,2,1>>> means_;
        shared_ptr<vector<Eigen::Matrix<double,2,2>>> covariances_;
```

Els mètodes que utilitzen l'objecte GaussianMixturesModel necessiten de manera recurrent el càlcul del determinant i de la inversa de la matriu de covariància. Per estalviar temps de càlcul, es va decidir precalcular-los en el constructor de la classe i emmagatzemar-los en llistes com la resta d'atributs.

```
        shared_ptr<vector<Eigen::Matrix<double,2,2>>> cov_inv_;
        shared_ptr<vector<double>> cov_det_;
            ...
}
```

Aquests mètodes estan tots implementats per dues dimensions, la implementació en tres dimensions s'explica més endevant a la secció 9.7.

Per poder visualitzar els GMM que es generen, s'implementen tres mètodes de plot per representar-los en gràfics. Aquest mètodes són plot_density(), plot_components() i plot_components_density().

```
void plot_density(int index, const int& resolution = 200);
void plot_components(int index, const double& scalar = 4.0);
void plot_components_density(const std::vector<Eigen::Vector2d>&
    scan,int index, const int& resolution = 200, const double& scalar
    = 4.0);
```

El primer, plot_density(), genera una imatge de mida resolution x resolution, on es discretitza per cada pixel el valor de la funció de cost del GMM avaluat en aquell punt seguint l'equació 5.1 , tal i com es veu en el següent tall de codi.

```
for(int i=0; i<resolution; i++){
    for(int j=0; j<resolution; j++){
            ...
        for(int k=0;k<k_;k++){
            img[(resolution-1-j)*resolution + i] += weights_->at(k) *
                ( 1 / ( 2 * M_PI * sqrt(cov_det_->at(k)) ) ) * ( exp(
                -(0.5) * ((punt_actual-means_->at(k)).transpose() *
                cov_inv_->at(k) * (punt_actual-means_->at(k)))(0)) );
        }
    }
}
```

El segon, plot_components(), mostra les mitjanes de cada component i dibuixa una el·lipse al voltant d'aquestes per representar la covariància. Aquesta el·lipse es calcula amb els vectors i valors propis de la covariància i es discretitzen 10 punts que pertanyen a aquesta per fer una línia que els recorri tots formant així una forma amb el perfil de l'el·lipse.

El tercer, plot_components_density(), genera els gràfics dels dos mètodes anteriors junts en una mateixa figura. D'aquesta manera, es poden comparar més fàcilment les dues imatges.

El la figura 9.1, es veu un exemple dels gràfics generats per les funcions, a l'esquerra es veu la densitat del GMM i a la dreta les components tal com s'ha explicat anteriorment, també s'ha afegit una guia visual a la dreta per veure la graella utilitzada per generar el GMM.

9.2 Frot-End NDT

A la llibreria de Front-Ends hem definit la funció ndt_constructor() que es basa en la tècnica de les Normal Distribution Transform presentada a la secció 5.2 . És un mètode que agrupa els punts d'un núvol de punts en components projectant una graella cartesiana sobre el núvol. A cada casella amb

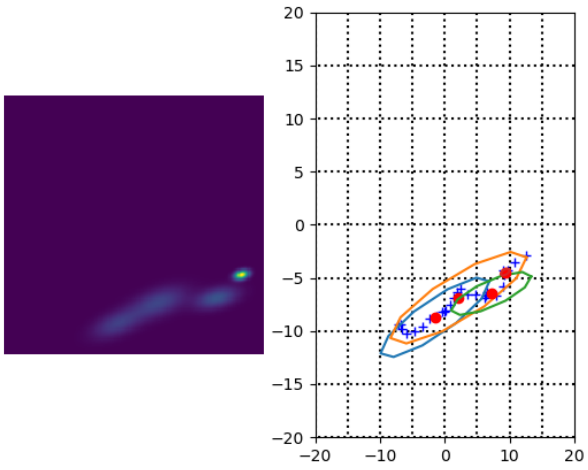

Figura 9.1: Diagrama de classes

un mínim nombre de punts li ajusta una distribució gaussiana, generant així múltiples distribuccions que defineixen un GMM.

```
shared_ptr<GaussianMixturesModel<2>> ndt_constructor(const
    std::vector<Eigen::Matrix<double,2,1>>& scan, const int&
    points_threshold, const double& cell_size, const double&
    vaps_min_ratio=-1){
```

La funció ndt_constructor() necessita com a paràmetres: scan que és un núvol de punts que el donem en forma de vector de matrius de 2x1 d'Eigen, points_threshold és el mínim de punts que ha d'haver en cada component, cell_size el tamany de les caselles quadrades en les que es dividirà la graella en les mateixes unitats que scan, i per últim vaps_min_ratio indica en cas de ser > 0, si s'ha de corregir la covariància en cas de que sigui massa deformada, fent que els valors propis segueixin el ratio donat.

Per fer el control d'aquesta graella s'ha fet ús d'una matriu d'enters que guarda l'index dels vectors on s'emmagatzema la informació de cada component del GMM. D'aquesta manera s'aconsegueix una lookup table que ens indica quina posició dels vectors s'ha d'actualitzar a partir de les coordenades del punt.

Per calcular la covariància amb la fòrmula convencional de l'equació es necessita tenir el valor de la mitjana ja calculada. Per evitar fer una estructura de dades molt gran en memòria, no es guarda cadascun dels punts que corresponen a cada casella de la graella. Aquest fet ens obliga a calcular tant la mitjana com la covariància de manera incremental, les equacions utilitzades

són les explicades a la secció 5.2.

Aquest càlcul es fa al codi de la següent manera:

```
for(int i=0;i<scan.size();i++){
        ...
    // actual is the index of the component the point is in
    covariances[actual] += scan[i] * scan[i].transpose();
}

for(int i=0; i<covariances.size();i++){
    covariances[i] = (1.0 / (n_points[i]-1)) * (covariances[i] -
       n_points[i] * ( means[i] * means[i].transpose() ));
           ...
}
```

Un cop obtingudes totes les components del GMM es comprova que totes tinguin el mínim de punts establert i s'eliminen les que no. Al haver eliminat les sobrants es fan la resta de càlculs necessaris, com ara donar el pes de cada component, el qual és el número de punts que hi ha a cada component dividit per el total per així tenir els pesos normalitzats. En aquest punt és on es fa la segona part del càlcul de la covariància tal com s'ha explicat.

A la figura 9.2 veiem la relació entre entre número de punts del núvol original, el numero de components generades i el temps d'execució. Com era d'esperar tots aquests són proporcionals entre si.

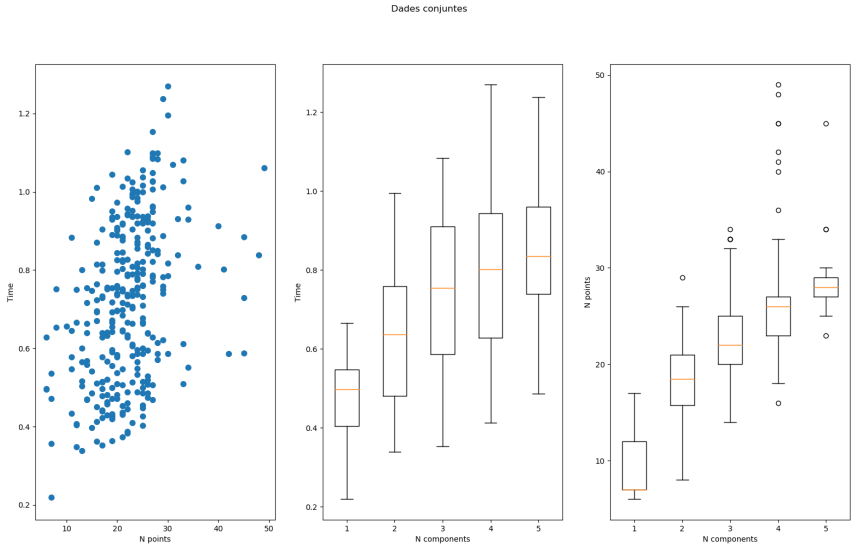

Figura 9.2: Comparació entre número de punts del núvol original, el numero de components generades i el temps d'execució al generar un GMM amb la tècnica NDT

9.3 Mètodes de registre

Els mètodes de registre defineixen la funció de cost d'un registre donada una transformació entre els dos núvols de punts. Per solucionar el problema de registre, el mètode també ha de definir la primera derivada per obtenir el gradient que s'utilitzara posteriorment al solucionador i la segona derivada que dona la matriu hessiana utilitzada també en el soluccionador.

A la interfície ScanMatchingMethods es defineixen aquestes funcions que han d'implementar tots els mètodes de registre, compute_score(), compute_score_and_gradient(), compute_score_gradient_and_hessian() i plot().

```
template<int dim>
class ScanMatchingMethods{
    public:
        virtual double compute_score(Eigen::Vector<double,dim> t) = 0;

        virtual tuple<double,Eigen::Vector<double,dim>>
            compute_score_and_gradient (Eigen::Vector<double,dim> t) =
            0;

        virtual tuple<double,Eigen::Vector<double,dim>,
            Eigen::Matrix<double,dim,dim>>
```

```
            compute_score_gradient_and_hessian(Eigen::Vector<double,dim>
            t) = 0;

        virtual void plot(int index, vector<float> process,
            Eigen::Matrix<double,dim,1> t, Eigen::Matrix<double,dim,1>
            optimal)=0;
};
```

Els mètodes per calcular el cost, gradient i hessiana només necessiten la transformació que es vol aplicar al segon registre, ja sigui 2D amb forma $[x, y, rz]$, com 3D amb forma $[x, y, z, rx, ry, rz]$.

9.3.1 P2D

Degut a les diferències entre els casos 2D i 3D s'ha decidit no templaritzar la classe i fer objectes concrets pels casos bidimensional i tridimensional simplificant així el codi d'aquests, dels quals només s'ha implementat el 2D de moment.

La primera subclasse d'aquesta interfície és PointsToDistribution2D. En aquest objecte els dos escanejos a registrar són un GMM i un núvol de punts, passats al constructor i que es guarden com atributs de l'objecte.

```
class PointsToDistribution2D : public ScanMatchingMethods<3>{
    private:
        shared_ptr<GaussianMixturesModel<2>> gmm_ref_;
        shared_ptr<vector<Eigen::Vector<double,2>>> point_cloud_;
    public:
        PointsToDistribution2D(const shared_ptr
            <GaussianMixturesModel<2>>& gmm_ref, const shared_ptr
            <vector <Eigen::Vector<double,2>>>& point_cloud);
            ...
}
```

Com ja s'ha explicat, les subclasses de ScanMatchingMethod han d'implementar la funció de cost i les seves derivades.

```
double compute_score(Eigen::Vector<double,3> t);
tuple<double,Eigen::Vector<double,3>>
    compute_score_and_gradient(Eigen::Vector<double,3> t);
tuple<double,Eigen::Vector<double,3>,Eigen::Matrix<double,3,3>>
    compute_score_gradient_and_hessian(Eigen::Vector<double,3> t);
```

En la funció compute_score() es calcula el cost després d'aplicar la transformació t al núvol de punts, tal com es veu en l'equació 9.1, on N es el

numero de punts, K el numero de components,w_j el pes de la component, Σ_j la covariància d'una component, μ_j la mitjana d'una component i p_i un punt x_i del núvol després d'aplicar la transformació t.

$$Score = -\sum_{i=0}^{N}\sum_{j=0}^{K} w_j \frac{1}{2\pi\sqrt{det(\Sigma_j)}} exp(-\frac{1}{2}((p_i-\mu_j)^T\Sigma_j^{-1}(p_i-\mu_j))) \quad (9.1)$$

La funció compute_score_and_gradient() a més de fer el mateix càlcul del cost que l'anterior també calcula la primera derivada de la funció de cost al aplicar la transformació t seguint l'equació:

$$\begin{aligned} Gradient &= (g_0, g_1, g_2)^T, \\ g_0 &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} A(1,0)^T, \\ g_1 &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} A(0,1)^T, \\ g_2 &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} A(-sin(t_2), -cos(t_2); cos(t), -sin(t_2))p_i, \\ A &= Score_{ij}(\mu_j - p_i)^T\Sigma_j^{-1}. \end{aligned} \quad (9.2)$$

La funció compute_score_gradient_and_hessian() caclcula tot l'anterior més la matriu hessiana seguint l'equació: 9.3

$$\begin{aligned} Hessiana &= (h_{00}, h_{01}, h_{02}; h_{01}, h_{11}, h_{12}; h_{02}, h_{12}, h_{22}), \\ h_{00} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}(((\lambda^T\Sigma_j^{-1}(1,0)^T)^2) - (A(1,0)^T)), \\ h_{01} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}(\lambda^T\Sigma_j^{-1}(0,1)^T * \lambda^T\Sigma_j^{-1}(1,0)^T - A(0,1)^T), \\ h_{02} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}(\lambda^T\Sigma_j^{-1}\delta_{rot}x_i\lambda^T\Sigma_j^{-1}(1,0)^T - A\delta_{rot}x_i), \\ h_{11} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}((\lambda^T\Sigma_j^{-1}(0,1)^T)^2 - B(0,1)^T), \\ h_{12} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}(\lambda^T\Sigma_j^{-1}\delta_{rot}x_i\lambda^T\Sigma_j^{-1}(0,1)^T - B\delta_{rot}x_i), \\ h_{22} &= -\textstyle\sum_{i=0}^{N}\sum_{j=0}^{K} Score_{ij}((\lambda^T\Sigma_j^{-1}\delta_{rot}x_i)^2 + (-(\delta_{rot}x_i)^T\Sigma_j^{-1}\delta_{rot}x_i + \lambda^T\Sigma_j^{-1}\delta\delta_{rot}x_i))), \\ \lambda &= \mu_j - p_i, \\ \delta_{rot} &= (-sin(t_2), -cos(t_2); cos(t_2), -sin(t_2)), \\ \delta\delta_{rot} &= (-cos(t_2), sin(t_2); -sin(t_2), -cos(t_2)), \\ A &= (\Sigma_j^{-1}(1,0)^T)^T, \\ B &= (\Sigma_j^{-1}(0,1)^T)^T. \end{aligned} \quad (9.3)$$

9.3.2 D2D

L'implementacío de la classe s'ha fet sense templaritzar, separant així els casos bidimensional i tridimensional en objectes diferents per simplicitat del

codi. Dels quals, de moment només s'ha implementat el cas 2D.

La segona subclasse de la interfície ScanMatchingMethods és DistributionDistribution2D. En aquest objecte els dos escanejos a registrar són dos GMM, passats al constructor i que es guarden com atributs de l'objecte.

```
class DistributionToDistribution2D : public ScanMatchingMethods<3>{
    private:
        shared_ptr<GaussianMixturesModel<2>> gmm_ref_;
        shared_ptr<GaussianMixturesModel<2>> gmm_new_;

    public:
        DistributionToDistribution2D(const
            shared_ptr<GaussianMixturesModel<2>>& gmm_ref, const
            shared_ptr<GaussianMixturesModel<2>>& gmm_new);
            ...
};
```

Igual que amb l'anterior i a totes les subclasses de ScanMatchingMethod s'han d'implementar la funció de cost i les seves derivades.

```
double compute_score(Eigen::Vector<double,3> t);
tuple<double,Eigen::Vector<double,3>>
    compute_score_and_gradient(Eigen::Vector<double,3> t);
tuple<double,Eigen::Vector<double,3>,Eigen::Matrix<double,3,3>>
    compute_score_gradient_and_hessian(Eigen::Vector<double,3> t);
```

On la funció de cost està definida per l'equació 9.4, on K_1 és el número de components del primer GMM, K_2 el número de components del segon GMM, w_i el pes de la component, Σ_i la covariància d'una component, μ_i la mitjana d'una component i t el vector de 3 components que dona la transformació en $[x, y, rz]$.

$$\begin{aligned} Score &= -\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} w_i w_j exp(-\tfrac{1}{2}(\lambda^T(\Sigma_i + rot(t_2)\Sigma_j rot(t_2)^T)^{-1}\lambda)), \\ \lambda &= \mu_i - rot(t_2)\mu_j - (t_0, t_1)^T, \\ rot(x) &= (cos(x), -sin(x); sin(x), cos(x)). \end{aligned} \tag{9.4}$$

La funció compute_score_and_gradient() calcula tant el cost, com la primera derivada que és el gradient, aquest càlcul es fa després d'aplicar la

transformació t, seguint l'equació següent:

$$\begin{aligned}
Gradient &= (g_0, g_1, g_2)^T, \\
g_0 &= -\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} Score_{ij}\lambda^T(\Sigma_i + rot(t_2) * \Sigma_j * rot(t_2)^T)^{-1}(1,0)^T, \\
g_1 &= -\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} Score_{ij}\lambda^T(\Sigma_i + rot(t_2) * \Sigma_j * rot(t_2)^T)^{-1}(0,1)^T, \\
g_2 &= -\sum_{i=0}^{K_1}\sum_{j=0}^{K_2} Score_{ij}B, \\
\lambda &= \mu_i - rot(t_2)\mu_j - (t_0, t_1)^T, \\
A &= \lambda^T(\Sigma_i + rot(t_2)\Sigma_j rot(t_2)^T)^{-1}C(\Sigma_i + rot(t_2)\Sigma_j rot(t_2)^T)^{-1}; \\
B &= (\lambda^T(\Sigma_i + rot(t_2)\Sigma_j rot(t_2)^T)^{-1}\delta_{rot}(t_2)\mu_j) + \tfrac{1}{2}(A\lambda) \\
C &= \delta_{rot}(t_2)\Sigma_j rot(t_2)^T + rot(t_2)\Sigma_j \delta rot(t_2)^T; \\
rot(x) &= (cos(x), -sin(x); sin(x), cos(x)),, \\
\delta_{rot}(x) &= (-sin(x), -cos(x); cos(x), -sin(x))
\end{aligned} \tag{9.5}$$

Per últim la compute_score_gradient_and_hessian() fa tots els càlculs anteriors i també obté la matriu hessiana, ja que l'equació per aquesta és molt més complexa s'ha deixat en forma de codi per que sigui més llegible.

```
for(int i=0; i<gmm_ref_->k(); i++){
    for(int j=0; j<gmm_new_->k(); j++){
        lamb = mu->at(i) - rotacio * mu_new->at(j) - translacio;
        eta = (sigma->at(i) + rotacio * sigma_new->at(j) *
            rotacio.transpose()).inverse();
        lamb_T_eta = lamb.transpose() * eta;
        hmk = w->at(i) * w_new->at(j) * 1.0 * exp( -0.5 * (lamb_T_eta
            * lamb));

        dR_mu_M = d_rot * mu_new->at(j);
        d_eta = d_rot * sigma_new->at(j) * rotacio.transpose() +
            rotacio * sigma_new->at(j) * d_rot.transpose();
        A = lamb_T_eta * d_eta * eta;
        B = (lamb_T_eta * dR_mu_M)(0) + 0.5 * (A * lamb);
            ...
        lamb_T_eta_d_tras0 = lamb_T_eta * d_tras.col(0);
        lamb_T_eta_d_tras1 = lamb_T_eta * d_tras.col(1);

        h_00 -= hmk * pow( lamb_T_eta_d_tras0,2) -
            float(d_tras.col(0).transpose() * eta * d_tras.col(0));
        h_01 -= hmk * (lamb_T_eta_d_tras1 * lamb_T_eta_d_tras0 -
            d_tras.col(0).transpose() * eta * d_tras.col(1));
        h_02 -= hmk * ( B * (lamb_T_eta_d_tras0) -
            (d_tras.col(0).transpose() * eta * dR_mu_M + A *
            d_tras.col(0)));
        h_11 -= hmk * ( pow(lamb_T_eta_d_tras1,2) -
```

```
            d_tras.col(1).transpose() * eta * d_tras.col(1) );
        h_12 -= hmk * ( B * (lamb_T_eta_d_tras1) -
            (d_tras.col(1).transpose() * eta * dR_mu_M + A *
            d_tras.col(1)));

        G2 = - dR_mu_M.transpose() * eta * dR_mu_M ;
        G3 = - 2.0 * (A * dR_mu_M)(0);
        G4 = float( lamb_T_eta * d_d_rot * mu_new->at(j) );
        G5_aux = d_d_rot * sigma_new->at(j) * rotacio + 2.0 * ( d_rot
            * sigma_new->at(j) * d_rot ) + rotacio * sigma_new->at(j)
            * d_d_rot - 2.0 *( d_eta * eta * d_eta );
        G5 = 0.5 * float( lamb_T_eta * G5_aux *
            lamb_T_eta.transpose() );

        h_22 -= hmk * ( pow(B,2) + G2 + G3 + G4 + G5 );
    }
}

hessian = Eigen::Matrix<double,3, 3>{{h_00,h_01,h_02},
                                     {h_01,h_11,h_12},
                                     {h_02,h_12,h_22}};
```

9.4 Solver

El solucionador es l'encarregat de resoldre el problema de registre expressat com a problema d'optimització. La variable objectiu és la transformació entre els dos escanejos. Els solucionadors que s'implementen estan basats en mètodes del gradient, concretament el mètode de Newton i les seves variacions. Per això, a la definició dels mètodes ha calgut definir també les derivades primera i segona de la funció objectiu.

```
template <int dim>
class Solver{
    private:
        vector<double> process_;
        shared_ptr<ScanMatchingMethods<dim>> method_;
        double min_norm_;
        double min_score_inc_;
        int max_NM_it_;

    public:
        virtual vector<float> process() = 0;
```

```
        virtual tuple<int,Eigen::Matrix<double,dim,1>,
            Eigen::Matrix<double,dim,dim>> compute_optimum(const
            tuple<Eigen::Vector<double, dim>,
            Eigen::Matrix<double,dim,dim>>& t_init) = 0;
        virtual void plot_process(int index,
            Eigen::Matrix<double,dim,1> t, Eigen::Matrix<double,dim,1>
            optimal) = 0;
};
```

Per a la implementació del solucionador, s'ha definit una interfície anomenada Solver, amb els mètodes compute_optimum(), process() i plot_process(). També els atributs min_norm_, min_score_inc_ i max_NM_it_ que defineixen els criteris de parada. L'atribut method_ de tipus ScanMatchingMethods permet definir les funcions de cost i les seves derivades per l'optimització. Finalment, a l'atribut de tipus vector process_ es guarda el valor del score de cada iteració del solucionador.

compute_optimum() és el mètode encarregat de resoldre l'optimització, retornant la transformació òptima juntament amb la inversa de la matriu hessiana, que proporciona una mesura de la incertesa del registre, i un enter, que permet saber quin dels criteris ha causat la parada de l'algoritme. El mètode process() és un getter de l'atribut i plot_process() crida a la funció de plot del method_ per fer els gràfics dels resultats de l'optimització i el seu procés.

9.4.1 Newton Method

La classe NewtonMethod implementa el mètode de newton per resoldre un problema de minimització sobre la funció de cost per obtenir la millor transformació per fer el scan matching. Com que aquest és el mètode bàsic no trobem cap diferència amb els atributs i mètodes definits per la interfície.

```
template <int dim>
class NewtonMethod: public Solver<dim>{
    private:
        vector<float> process_;
        shared_ptr<ScanMatchingMethods<dim>> method_;
        double min_norm_;
        double min_score_inc_;
        int max_NM_it_;

    public:
        NewtonMethod(const shared_ptr<ScanMatchingMethods<dim>>&
            method, const double& min_norm, const double&
```

```
            min_score_inc, const int& max_NM_it);

        vector<float> process();

        tuple<int,Eigen::Matrix<double,dim,1>,
            Eigen::Matrix<double,dim,dim>> compute_optimum(const
            tuple<Eigen::Vector<double, dim>,
            Eigen::Matrix<double,dim,dim>>& t_init);

        void plot_process(int index, Eigen::Matrix<double,dim,1> t,
            Eigen::Matrix<double,dim,1> optimal);
};
```

En el mètode compute_optimum() hi ha implementat l'algoritme del mètode de Newton. En aquest s'itera modificant el valor d'una transformació llavor a partir de les seves derivades aplicant

$$x_{n+1} = x_n - \alpha \frac{f'(x_n)}{f''(x_n)}, \tag{9.6}$$

obtenint cada cop un valor més proper al desitjat fins que algun dels criteris de parada pari l'execució del bucle. Aquests són els ja definits a l'interfície, que paren el mètode quan s'arriba a un numero d'iteracions, quan la millora del cost es molt petita o quan el pas es molt petit i per tant ja es molt proper a la solució real.

A continuació, es mostra un tros del codi del mètode compute_optimum(). Com es veu, el càlcul del nou òptim no es suma directament, ja que no es pot composar la part de la rotació amb la suma. Per tant, es crida una funció que multiplica les matrius de transformació generades per el vector t i les torna a deixar en forma de vector.

```
//initialize variables
while(!stop_criteria){
    d=-(hessian.inverse() * gradient);

    optimal = composite_o_plus(optimal, d*0.01);

    score_and_derivates =
        method_->compute_score_gradient_and_hessian(optimal);

    score = get<0>(score_and_derivates);
    gradient = get<1>(score_and_derivates);
    hessian = get<2>(score_and_derivates);
```

```
    process_.push_back(score);

        //update stop_criteria

}
// return the result
```

El resultat a retornar pot tenir diferents implicacions depenent del criteri que ha parat l'optimització. Si l'algorisme convergeix, es retorna l'òptim i un 0 que indica convergència. Si s'ha arribat al numero màxim d'iteracions establert, es compara el cost de l'estat inicial amb el final, i es retorna el millor juntament amb un 1 per representar que no s'ha convergit. Si durant el procés iteratiu el problema queda mal condicionat amb una matriu hessiana no invertible, es trenca l'optimització i es retorna la llavor amb un 2 que indica que l'optimització s'ha trencat.

```
tuple<int,Eigen::Matrix<double,dim,1>,Eigen::Matrix<double,dim,dim>>
    result;
if(bad_conditioning){
    result = make_tuple(2,initial, hess);
}
else if(i>=max_NM_it_){
    if(process_.front()<process_.back()){
        result = make_tuple(1,initial, hess);
    }
    else{
        result = make_tuple(1,optimal,hessian);
    }
}
else if(found){
    result = make_tuple(0,optimal,hessian);
}

return result;
```

9.4.2 Line search

La classe LineSearchNewtonMethod també implementa el mètode de Newton per resoldre un problema de minimització, però afegeix el mètode de line_search i el càlcul de les condicions de Wolfe en les funcions first_Wolfe_condition i second_Wolfe_condition per poder determinar la mida òptima del pas en la

direcció que defineix el mètode de Newton. Aquestes noves funcions també tenen una sèrie de constants que es defineixen al constructor i es guarden com atributs de la classe. Els nous atributs són: max_LS_it_ que posa un màxim d'iteracions al line_search, beta1_ i beta2_ són constants que s'utilitzen en l'avaluació de la primera i segona condició de Wolfe, i finalment gamma_ que s'utilitza pel càlcul del factor α pel pas del mètode de Newton.

```
template <int dim>
class LineSearchNewtonMethod: public Solver<dim>{
    private:
            ...
        double beta1_;
        double beta2_;
        double gamma_;
        int max_LS_it_;

        double line_search(const
            tuple<double,Eigen::Vector<double,dim>,
            Eigen::Matrix<double,dim,dim>>& score_and_derivates, const
            Eigen::Matrix<double,dim,1>& d, const
            Eigen::Matrix<double,dim,1>& xk, const int& iter);
        bool first_Wolfe_condition(const double& alpha, const
            Eigen::Matrix<double,dim,1>& d, const
            Eigen::Matrix<double,dim,1>& gradient, const double&
            score, const double& score_plus);
        bool second_Wolfe_condition(const
            Eigen::Matrix<double,dim,1>& d, const
            Eigen::Matrix<double,dim,1>& gradient, const
            Eigen::Matrix<double,dim,1>& gradient_plus);

    public:
        LineSearchNewtonMethod(const
            shared_ptr<ScanMatchingMethods<dim>>& method, const
            double& min_norm, const double& min_score_inc, const int&
            max_NM_it, const double& beta1, const double& beta2, const
            double& gamma, const int& max_LS_it);
                ...
};
```

En la mètode compute_optimum(), igual que a l'anterior, s'hi troba implementat l'algoritme del mètode de Newton. Aquest algorisme itera modificant el valor d'una transformació llavor a partir de les seves derivades, tal com es veu en l'equació 9.6, fins que es troba la solució òptima, quan els criteris de

parada aturen l'optimització. No obstant, en aquest cas α no és una constant fixada, sinó que a cada iteració se'n busca un valor òptim gràcies al mètode line_search. En conseqüència, ara el mètode compute_optimum() segueix la següent estructura:

```
    ...
while(!stop_criteria){
    d=-(hessian.inverse() * gradient);

    alpha = line_search(score_and_derivates, d, optimal);

    optimal = composite_o_plus(optimal, d*alpha);
        ...
}
```

Al mètode line_search se li ha de passar el mètode de registre, la direcció que es vol seguir i l'òptim actual.

```
double line_search(const tuple<double,Eigen::Vector<double,dim>,
    Eigen::Matrix<double,dim,dim>>& score_and_derivates, const
    Eigen::Matrix<double,dim,1>& d, const
    Eigen::Matrix<double,dim,1>& xk)
```

Amb aquesta informació es pot implementar l'algorisme explicat a la pàgina 274 del llibre [Bierlaire 2015]. Aquest consisteix en modificar α segons si es compleixen les les condicions de Wolfe. Per això es defineixen dues variables reals que estableixen el limit esquerra i dreta d'α. A cada iteració es comprova si es compleix la primera condició de Wolfe. En cas que no sigui així, s'actualitza el limit dret d'α al valor actual d'aquesta i dóna com a nou valor d'α el put mig entre els limits dret i esquerra. En cas de que es compleixi la primera condició però no la segona, s'actualitza el límit esquerra i es torna a donar el punt mig com a nova α. No obstant, si el límit dret encara val infinit per l'inicialització el nou valor d'α és l'anterior multiplicat per gamma_. Aquest algoritme s'executa fins que es compleixen les dues condicions i, en conseqüència, es troba la solució o s'arriba al màxim d'iteracions.

La funció first_Wolfe_condition avalua la primera condició de Wolfe

$$score\_plus \leq score + \alpha\beta_1 gradient^T d$$

que, quan és falsa, indica que el pas és massa llarg. La funció demana els paràmetres: alpha, que és la α actual; d, que és la direcció que estableix el Mètode de Newton; gradient, que és el gradient actual; score, que és el cost actual; i score_plus, que és el cost fent el pas actual multiplicat per α. En

aquest mètode s'avalua l'igualtat

La funció second_Wolfe_condition avalua la segona condició de Wolfe

$$(gradient\_plus^T d) \geq (\beta_2 gradient^T d)$$

que, quan és falsa, indica que el pas és massa curt. La funció demana els paràmetres: d, que és la direcció que estableix el Mètode de Newton; gradient, que és el gradient actual; i gradient_plus, que és el gradient fent el pas actual multiplicat per α.

9.5 Vectorització dels mètodes de registre

Per calcular les derivades del mètode P2D cal avaluar tot un núvol de punts en un GMM. El mètode D2D avalua dos GMM un contra l'altre. Aquests càlculs repetitius es poden vectoritzar i fer-los més eficients computacionalment repartint-los en diferents fils del processador aplicant threading. Concretament, en aquesta secció, es volen paral·lelitzar els càlculs dels mètodes `compute_ score()`, `compute_score_and_gradient()` i `compute_score_gradient_ and_hessian()` de les classes `PointsToDistribution2D` i `DistributionTo Distribution2D`.

El threading està implementat amb l'API d'OpenMP [OpenMP 022] la qual permet multiprocessament en diferents fils. Es basa en paral·lelitzar a partir d'un fil principal i generar un numero de sub-threads a partir de forks els quals es reparteixen la feina i s'executen concurrentment en diferents cores del processador.

Per paral·lelitzar el codi dels mètodes `compute_score()`, `compute_score _and_gradient()` i `compute_score_gradient_ and_hessian()` s'aplica la comanda `#pragma omp parallel for` la qual permet repartir les iteracions del bucle entre tots el fils d'execució. No obstant, per poder obtenir resultats amb sentit, s'ha d'indicar com es reparteixen les variables. Per aquest motiu utilitzem `default(shared)` perquè les variables de dins el bucle siguin totes compartides entre els diferents fils i `reduction(-: ... )` que indica com s'han d'unir les variables un cop s'ha finalitzat la paral·lelització, en aquest cas particular, restant.

```
#pragma omp parallel for default(shared)
    reduction(-:score,j_0,j_1,j_2,h_00,h_01,h_02,h_11,h_12,h_22)
    num_threads(2)
for(int i=0; i<point_cloud_->size(); i++){
    ...
}
```

Les variables del resultat són enters en comptes de matrius i vectors Eigen, ja que no es pot utilitzar el `reduction` amb variables de tipus Eigen. Per tant, es declara una variable entera per cada posició del vector i de la matriu només els elements que formen la matriu triangular ja que aquesta es simètrica. Un cop acabada l'execució s'utilitzen aquestes variables per emplenar el vector jacobià i la matriu hessiana.

```
Eigen::Vector<double,6> jacobian =
    Eigen::Vector<double,6>{{j_0},{j_1},{j_2},{j_3},{j_4},{j_5}};
Eigen::Matrix<double,6, 6> hessian = Eigen::Matrix<double,6, 6>
    {{h_00,h_01,h_02,h_03,h_04,h_05},
     {h_01,h_11,h_12,h_13,h_14,h_15},
     {h_02,h_12,h_22,h_23,h_24,h_25},
     {h_03,h_13,h_23,h_33,h_34,h_35},
     {h_04,h_14,h_24,h_34,h_44,h_45},
     {h_05,h_15,h_25,h_35,h_45,h_55}};
```

9.6 Millora de l'eficiència computacional de la tècnica P2D

Quan es vol resoldre un registre segons la tècnica P2D, en cada iteració de l'optimització cal avaluar tot un núvol de punts de dimensió N en un model gaussià de dimensió K, on $N >> K$. En canvi, la tècnica D2D és molt més eficient computacionalment ja que usa els GMM com a eina de compressió dels núvols de punts i només requereix l'avaluació d'un GMM contra un altre. En aquesta secció volem presentar una millora a la tècnica P2D per poder detectar quines són les avaluacions rellevants de punts en components i evitar els càlculs d'aquelles irrellevants.

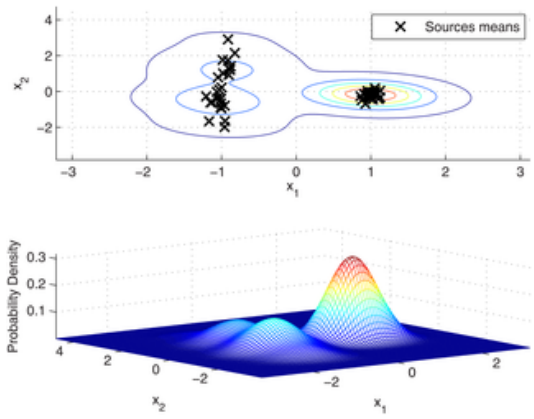

Figura 9.3: Relleu generat per un GMM

Com es veu a la figura 9.3 un GMM el podem visualitzar com una plana amb turons de diferent alçada. Quan avaluem un punt contra el model, estem mirant l'alçada d'aquell punt per cada turó. Si els turons estan separats, que és l'habitual en els GMM que utilitzem, un punt tindrà com a màxim alçada en un turó i per la resta de turons l'alçada serà negligible. El que volem fer és detectar de manera automàtica quin és el turó rellevant i evitar els càlculs del punt per les components irrellevants. A l'hora de calcular el likelihood del punt en el model aquest filtre serà poc rellevant, ja que els càlculs són pocs. En canvi, a l'hora d'avaluar derivades l'afectació ha de ser notòria, ja que les expressions són molt més complexes.

Per efectuar aquest filtratge automàtic proposem un test de Chi quadrat per cada punt amb cada component del GMM. El test de Chi quadrat es basa en determinar la distància de Mahalanobis entre un punt i una distribució gaussiana:

$$d_M = (p - \mu_k)^T \Sigma_k^{-1} (p - \mu_k), \tag{9.7}$$

on p és un punt, μ_k és la mitjana que defineix la distribució k i Σ_k és la matriu de covariància de la distribució. Com es veu, a diferència de la distància euclídia - que considera $\Sigma_k = I$ - la distància del punt amb el centre de la component està ponderada per la incertesa d'aquella component. Per tant, donat un punt a una distància euclídia igual amb dues distribucions de diferent incertesa, la distància de Mahalanobis serà major amb la distribució més certa, ja que la campana serà més tancada.

El test de Chi quadrat estableix un llindar a aquesta distància segons la dimensió DF de l'espai i el nivell de confiança que vulguem considerar. Aquests llindars estan tabulats a la taula de la figura 9.4. Per un problema al pla (DF=2) i establint una confiança del 0.05 , el llindar és 5.991 . Aquest valor ens està dient que només acceptem aquells punts situats a menys de 6 vega-

des la incertesa de la component mesurada des del centre de la component.

	Chi-Square Right-Tail Probability ($\geq \chi^2$)									
DF	0.995	0.99	0.975	0.95	0.9	0.1	0.05	0.025	0.01	0.005
1	---	---	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566	39.997
21	8.034	8.897	10.283	11.591	13.240	29.615	32.671	35.479	38.932	41.401
22	8.643	9.542	10.982	12.338	14.041	30.813	33.924	36.781	40.289	42.796
23	9.260	10.196	11.689	13.091	14.848	32.007	35.172	38.076	41.638	44.181
24	9.886	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980	45.559
25	10.520	11.524	13.120	14.611	16.473	34.382	37.652	40.646	44.314	46.928
26	11.160	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642	48.290
27	11.808	12.879	14.573	16.151	18.114	36.741	40.113	43.195	46.963	49.645
28	12.461	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278	50.993
29	13.121	14.256	16.047	17.708	19.768	39.087	42.557	45.722	49.588	52.336
30	13.787	14.953	16.791	18.493	20.599	40.256	43.773	46.979	50.892	53.672
40	20.707	22.164	24.433	26.509	29.051	51.805	55.758	59.342	63.691	66.766
50	27.991	29.707	32.357	34.764	37.689	63.167	67.505	71.420	76.154	79.490
60	35.534	37.485	40.482	43.188	46.459	74.397	79.082	83.298	88.379	91.952
70	43.275	45.442	48.758	51.739	55.329	85.527	90.531	95.023	100.425	104.215
80	51.172	53.540	57.153	60.391	64.278	96.578	101.879	106.629	112.329	116.321
90	59.196	61.754	65.647	69.126	73.291	107.565	113.145	118.136	124.116	128.299
100	67.328	70.065	74.222	77.929	82.358	118.498	124.342	129.561	135.807	140.169

Figura 9.4: Taula amb els llindars del test de Chi quadrat segons la dimensió del problema i la confiança que vulguem admetre.

Pel que fa a la implementació, s'implementa el test dins els mètodes "compute_score()", "compute_score_and_gradient()" i "compute_score_gradient_and_hessian()" de la classe "Point2Distribution2d". Tots aquests mètodes consten d'un bucle on es recorren tots els punts del núvol de punts enllaçat amb un altre bucle on es recorren totes les components del GMM:

```
for(int i=0; i<point_cloud_->size(); i++){
    Eigen::Vector<double,2> transformed_point = rotacio *
        point_cloud_->at(i) + translacio;
    for(int j=0; j<gmm_ref_->k(); j++){
        ...
    }
}
```

És a l'inici d'aquest segon bucle on s'avalua la distància de Mahalanobis i s'efectua el test:

```
Eigen::Vector<double,2> lamb = mu->at(j)-transformed_point;
Eigen::Matrix<double,1,2>lamb_T_inf = lamb.transpose() *
    sigma_inv->at(j);
```

```
double mahalanobis_dist = lamb_T_inf * lamb;

if(mahalanobis_dist <= max_mah_dist){
    ...
```

Si es passa, es fan les avaluacions necessàries i, si no, es segueix amb la següent parella punt-component. Com s'intueix, aquest test no ha de tenir gaire efecte al mètode "compute_score()", ja que com es veu a l'equació del likelihood

$$Score = -\sum_{n=1}^{N}\sum_{k=1}^{K} w_k \frac{1}{2\pi \det \Sigma_k} \mathrm{e}^{-\frac{1}{2}(p_n-\mu_k)^T \Sigma_k^{-1}(p_n-\mu_k)} \tag{9.8}$$

el score no és més que l'exponencial de la distància de Mahalanobis afectada per una constant. En canvi, per les derivades l'afectació es major ja que la dimensió dels objectes és major - el jacobià és un vector de 3 components i la hessiana és una matriu simètrica de $3x3$ - i els càlculs necessaris són més complexes.

9.7 Front-End 3D

Dins la comunitat del CIRS, fins a la realització d'aquest treball encara no s'-havien processat núvols de punts 3D utilitzant la tècnica de GMM-Registration. Només s'havien fet registres 3D aplicant l'algorisme ICP sense aconseguir resultats rellevants ni molt menys determinar de manera automàtica la incertesa del registre. En aquesta secció volem presentar la templarització del mètode"ndt_constructor()" per processar tant núvols de punts bidimensionals com tridimensionals.

Per fer-ho, el mètode es templaritza segons la variable entera "dim":

```
template<int dim>
shared_ptr<GaussianMixturesModel<dim>> ndt_constructor(const
    std::vector<Eigen::Matrix<double,dim,1>>& scan,const int&
    points_threshold, const double& cell_size, const double&
    vaps_min_ratio=-1);
```

Aquesta variable fa referència a la dimensió del vector punts i, per tant, pren els valors de 2 o 3. Utilitzant la llibreria Eigen [cita], es poden templaritzar variables punts i matrius de covariança segons aquesta variable:

```
Eigen::Matrix<double,dim,1> point;
```

```
Eigen::Matrix<double,dim,dim> covariance_matrix;
```

Per tant, ha calgut tornar a declarar tots els vectors i matrius del codi utilitzant aquest objectes templaritzats.

Fet aquest canvi en els tipus, s'ha hagut de modificar la graella cartesiana utilitzada per establir les components del GMM. Fins ara la graella era un vector de vectors d'enters que permetia saber a quina posició del vector de components pertany cada casella. En canvi, pel cas tridimensional es necessita un vector de vectors de vectors. No obstant, no interessa ocupar una dimensió més de memòria quan es treballa en dues dimensions. Per solucionar-ho, s'ha implementat una array de 3 dimensions on pel cas bidimensional la primera dimensió (corresponent a la z) sempre és 1. Per tant, quan ens adrecem a un cas bidimensional ho hem de fer amb [1,x,y] i quan ens adrecem a un cas tridimensional ho hem de fer amb [x,y,z].

```
//Inicialitzacio de la graella
//Abans:
vector<vector<vector<int>>> grid(vector<vector<int>>(cell_number,
    vector<int>(cell_number)));
// Ara:
vector<vector<vector<int>>> grid;
if(dim==2) grid = vector<vector<vector<int>>>
    (1,vector<vector<int>>(cell_number, vector<int>(cell_number)));
else if(dim ==3) grid = vector<vector<vector<int>>>
    (cell_number,vector<vector<int>>(cell_number,
    vector<int>(cell_number)));
```

```
//Acces a la graella
int n;
Eigen::Matrix<int,dim,1> grid_pos;
    ...
//Abans:
grid[grid_pos(0)][grid_pos(1)] = n;
// Ara:
if(dim==2){
    grid[0][grid_pos(0)][grid_pos(1)] = n;
}
else if(dim==3){
    grid[grid_pos(0)][grid_pos(1)][grid_pos(2)] = n;
}
```

Un cop templaritzada la funció "ndt_constructor()", cal templaritzar també l'estructura de dades "GaussianMixtureModel<dim>" segons la variable

entera "dim"per poder emmagatzemar un GMM fitat en un núvol de punts tridimensional. Alhora, cal reimplementar els mètodes “plot_density()” i “plot_components()” per representar models tridimensionals. El mètode “plot_density()” per un GMM bidimensional genera una imatge del model avaluant-lo segons una matriu de la resolució desitjada aplicant l’equació 9.8. En un GMM tridimensional la matriu té tres dimensions i és impossible representar-la en una imatge, ja que és un cub solid del qual només es veuen les parets que molt probablement tenen un score de zero. Per fer aquesta representació es decideix fer seccions al model segons un eix escollit i per cada capa generar-ne una imatge. Per generar cada imatge s’utilitza la llibreria [Matplotlib 022] per a C++.

Per cridar aquesta funció es poden especificar diferents paràmetres: `index` indica el numero de l’scan que es vol mostrar i s’utilitza per donar nom a l’arxiu de l’imatge, `resolution` és un paràmetre opcional que indica la mida en píxels de l’amplada i alçada de l’imatge on es discretitza el resultat de la funció score 9.8, `n_talls` indica la quantitat de capes a fer al llarg de l’eix `eix` que pot tenir els valors 0, 1 o 2 representant els eixos x, y i z respectivament.

```
void plot_density(int index, const int& resolution = 200, int
    n_talls=-1, int eix = 2);
```

El mètode “plot_components()” mostra les components d’un GMM a partir dels el·lipsoïdes que defineixen les seves mitjanes i matrius de covariança. Aquesta informació per un GMM tridimensional es pot seguir representant en un gràfic tridimensional. No obstant, la llibreria Matplotlib per a C++ no permet dibuixar superfícies tridimensionals. Per tant, cal fer el gràfic utilitzant la llibreria Matplotlib per a Python. Com que es tracta d’un mètode només utilitzat per debugar la parametrització del mètode de Front-End i mai s’executa quan s’efectua un registre, s’ha decidit generar el GMM en C++, guardar les mitjanes i covariances en un fitxer de text i finalment, llegir-lo en un script de Python on es genera el gràfic.

Per cridar el mètode s’utilitza la mateixa funcio que en 2D la diferència es que el parametre scalar no s’utilitza, ja que només es guarden en un fitxer les mitges i les covàriancies, per poder fer el gràfic s’ha dexecutar després d’aquesta funció el script de Python `plot3dScan.py`

```
void plot_components(int index, const double& scalar = 4.0);
```

Al script de Python es dibuixen els el·lipsoides a partir de la descomposició en vectors i valors propis de la matriu de covariància i les mitjanes per col·locar el centre. Els valors propis defineixen els radis de l’el·lipsoïde en cada eix i els vectors propis determinen l’orientació de l’el·lipsoïde segons el

sistema de coordenades global de la figura.

El següent codi s'executa per cada component a dibuixar en el gràfic. Primer, es fa la factorització per obtenir la matriu de rotació i els 3 radis de l'el·lipsoide. Posteriorment, es calculen les coordenades que descriuen la superfície d'aquest el·lipsoide i s'aplica la transformació de rotació i translació per obtenir la posició i orientació final. Finalment, un cop es tenen les coordenades transformades es crida la funció `plot_surface` per dibuixar-lo.

```
center = means[indx]

radii, rotation = linalg.eigh(covariances[indx])
rotation=np.transpose(rotation)

# calculate cartesian coordinates for the ellipsoid surface
u = np.linspace(0.0, 2.0 * np.pi, 60)
v = np.linspace(0.0, np.pi, 60)
x = radii[0] * np.outer(np.cos(u), np.sin(v))
y = radii[1] * np.outer(np.sin(u), np.sin(v))
z = radii[2] * np.outer(np.ones_like(u), np.cos(v))

for i in range(len(x)):
    for j in range(len(x)):
        [x[i,j],y[i,j],z[i,j]] = np.dot([x[i,j],y[i,j],z[i,j]],
           rotation) + center

ax.plot_surface(x, y, z, rstride=3, cstride=3,
    color=m.to_rgba(indx), linewidth=0.1, alpha=1, shade=True)
```

9.8 Solver basat en la factorització de Cholesky modificada

En el mètode de Newton implementat a la secció 9.4.1 ens podem trobar problemes en el cas de que algun dels valors propis de la matriu Hessiana sigui negatiu. Això provocaria una matriu Hessiana indefinida que no ens asseguraria una direcció de minimització per resoldre l'optimització. Per evitar matrius indefinides, proposem utilitzar la factorització de Cholesky modificada [cita !!!!!!] per manipular numèricament la matriu imposant una direcció de minimització.

La factorització de Cholesky modificada s'assegura que la matriu Hessiana sigui definida positiva amb la mínima pertorbació possible i, alhora, que

estigui ben condicionada. Aquesta factorització consisteix en pertorbar un a un els valors de la diagonal de la matriu fins que tots els valors propis de la matriu siguin positius, forçant un forma triangular a la matriu factoritzada:

$$H + \tau I = LL^T.$$

Aquesta forma és convenient alhora de resoldre sistemes d'equacions lineals ja que les equacions es poden resoldre en cascada. El sistema

$$d = -H^{-1}g$$

es transforma a

$$Lz = g$$
$$L^T d = -z$$

on $H = LL^T - \tau I$. D'aquesta manera s'assegura una direcció de minimització pel mètode de Newton.

Aquesta factorització s'ha implementat a la classe "CholeskyNewtonMethod", la qual necessita que s'especifiqui quin mmètode de scan matchin s'utilitzarrà en l'optimització junt amb els criteris de parada min_norm, min_score_inc, max_NM_it, que també necessita el NewtonMethod convencional, junt amb el paràmetre rho utilitzat en la funció per computar la matriu L.

```
CholeskyNewtonMethod (const shared_ptr <ScanMatchingMethods <dim>>&
    method, const double& min_norm, const double& min_score_inc,
    const int& max_NM_it,const double& rho);
```

La funció compute_l_matrix s'encarrega de generar la matriu triangular L, i retorna una tupla amb la matriu L, una matriu de permutació que indica la permutació que s'ha fet a la matriu original per obtenir aquesta, i tau, un vector que indica la pertorbació causada als elements de la diagonal de la matriu original per generar la matriu L.

```
tuple<Eigen::Matrix<double,dim,dim>, Eigen::Matrix<double,dim,dim>,
    Eigen::Matrix<double,dim,1>> compute_l_matrix
    (Eigen::Matrix<double,dim,dim> A0);
```

Per fer trobar L s'utilitza un algorisme explicat en la pàgina 278 del llibre [Bierlaire 2015], on es busca el valor propi més petit, es permuta la matriu per colocar aquest com el primer i es modifica el valor a la diagonal per fer que aquest sigui positiu, això es repeteix cada cop amb la submatriu quadrada que queda per sota de l'element modificat fins a que la matriu obtinguda sigui definida positiva.

I la funció solve_system_l s'encarrega de resoldre x del sistema d'equa-

cions $v = Lx$ on L és una matriu triangular inferior o superior indicat al booleà lower. Per aquest càlcul s'ha fet una funció apart, que resol el sistema en forma de cascada i així evita el càlcul de la inversa de la matriu.

```
Eigen::Matrix<double,dim,1>
    CholeskyNewtonMethod<dim>::solve_system_l
    (Eigen::Matrix<double,dim,dim> l, Eigen::Matrix<double,dim,1> v,
    const bool& lower)
```

Alhora, la factorització es pot combinar amb l'algorisme de Line Search presentat a la secció 9.4.2 per assegurar una bona direcció de minimització i una bona mida per l'esglaó d'optimització. Per això, també s'ha implementat la classe CholeskyLineSearchNewtonMethod. On s'utilitzen els mètodes line_search, compute_l_matrix i solve_system_l ja descrits.

```
CholeskyLineSearchNewtonMethod(const
    shared_ptr<ScanMatchingMethods<dim>>& method, const double&
    min_norm, const double& min_score_inc, const int& max_NM_it,
    const double& beta1, const double& beta2, const double& gamma,
    const int& max_LS_it,const double& rho);
```

9.9 Back-end 3D

En aquesta secció es mostra com s'ha intentat fer un registre de dos núvols de punts tridimensionals. A l'entorn del CIRS aquesta acció encara no s'havia intentat. Per efectuar el registre es decideix començar per la tècnica P2D, ja que la funció de cost és més senzilla analíticament i, en conseqüència, les derivades també ho són. Per això s'implementa la classe PointsToDistribution3D

```
PointsToDistribution3D(const shared_ptr<GaussianMixturesModel<3>>&
    gmm_ref,const shared_ptr<vector<Eigen::Vector<double,3>>>&
    point_cloud);
```

que, hereda de ScanMatchingMethods templaritzat en $dim = 6$.

El motiu de no haver fet la templarització en el PointsToDistribution2D, és que els mètodes a implementar quedaven com dos mètodes independents en funció de dim i es va decidir que fer dos classes diferents era més correcte conceptualment.

Feta la implementació, es prova el registre d'un mateix núvol de punts pertorbat segons una translació i una rotació petites agafat del data set presentat a la secció 10.1.2 Un escaneig es representa en forma de GMM aplicant

el Front-End presentat a la secció 9.7 i l'altre es manté com a núvol de punts. No obstant, el núvol de punts està format per desenes de milers de punts i calcular el registre P2D en un sol fil de la CPU tarda de l'ordre dels 2 minuts. Per fer testos de manera més efectiva, es decideix retallar el núvol de punts quedant-nos només 1 punt de cada 10 punts rebuts. A la figura 9.5 es pot veure com, tot i la disminució de dades, el núvol de punts conserva la forma de l'escaneig inicial.

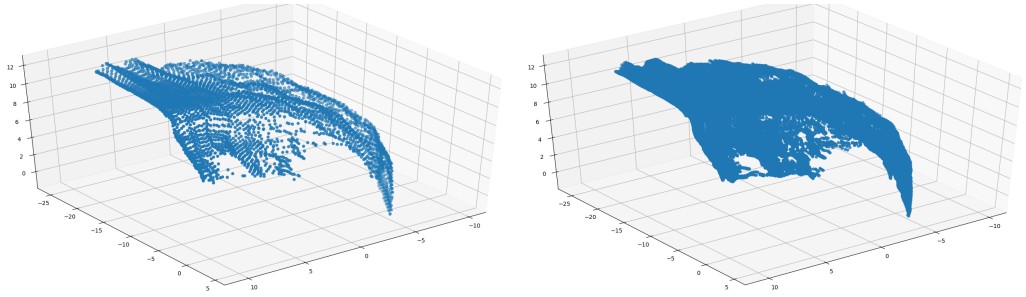

Figura 9.5: Núvol de punts original a la dreta i el mateix reduït a l'esquerra.

Al intentar el registre amb el núvol de punts lleuger, els resultats no donen l'esperat. Encara que el problema està plantejat per maximitzar el likelihood, el solucionador el minimitza tot separant els núvols de punts. Pensem que potser la llavor que donem a l'optimitzador no és prou bona, provocant una hessiana indefinida que causa el funcionament invers. Per assegurar una bona direcció de maximització utilitzem el solucionador CholeskyNewtonMethod que implementa la factorització de Cholesky modificada descrita a la secció 9.8 . Aquest mètode pertorba la matriu hessiana en cas que aquesta sigui indefinida per tal d'assegurar que sigui definida positiva. No obstant, la factorització provoca unes pertorbacions molt agressives que acaben generant passos en orientació corresponents a rotacions pures en un sol eix.

Veient que el problema no està causat per una mala inicialització, sinó que la causa està en el càlcul de les derivades, decidim solucionar el problema només utilitzant la primera derivada de la funció de cost, tot utilitzant el mètode del gradient. Veiem que el problema tampoc es soluciona i, al acabar-se el temps per al treball, decidim deixar aquí l'intent de registre. Pel que sembla falla l'aplicació de la formula de Gallego [Gallego 2015], corresponent a la derivada de l'acció del grup SO(3) quan aquest està parametritzat amb matrius de rotació. Per determinar la falla cal analitzar aquesta derivada i si pot ser més convenient parametritzar la rotació amb quaternions.

Encara que es solucionés el problema en les derivades de la funció objectiu, veiem que el temps de registre d'un núvol de punts aplicant la tècnica P2D és excessivament alt per resoldre'l en temps real. Com que els núvols de

punts acústics tridimensionals tenen una mida considerable, considerem que la paral·lelització dels càlculs cal efectuar-la en una targeta gràfica.

Capítol 10

Implantació i resultats

10.1 Data sets utilitzats

Per als diferents experiments de validació proposats en aquest projecte s'utilitzen diferents data sets amb dades reals adquirides en un entorn submarí.

10.1.1 Data sets bidimensionals

Els data sets de dades bidimensionals s'han adquirit amb un Mechanical Scaning Profiling Sonar muntant en un AUV. Aquest tipus de sensor està format per un sonar perfilador - un únic feix acústic molt prim - que rota al pla accionat per un actuador mecànic. El sensor es munta paral·lel al pla de moviment de l'AUV i, per tant, s'obtenen núvols de punts 2D de l'entorn del robot. El sensor està configurat amb un camp de visió de 270º simètrics respecte de l'eix de moviment del robot.

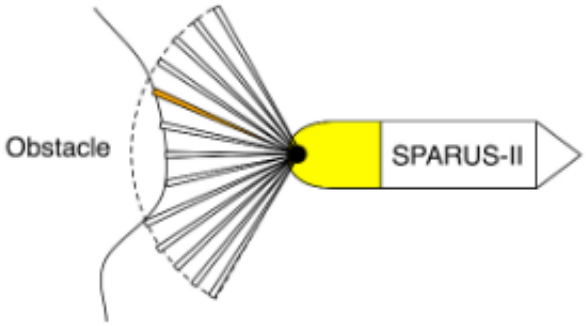

Figura 10.1: Exemple funcionament Sonar.

Com a AUV s'utilitza el vehicle desenvolupat pel Centre d'Investigació en Robòtica Submarina (CIRS) Sparus II. Es tracta d'un vehicle holonòmic en forma de torpede equipat amb sensors inercials (DVL i IMU). Disposa d'una paylod frontal on es munta el Mechanical Scaning Profiling Sonar. Per adquirir el dataset el robot s'opera des d'una barca a través d'una boia en mode debug.

Un dels datasets està adquirit en un entorn estructurat amb formes cartesianes i parets verticals, fent irrellevants els canvis de profunditat del robot. El dataset és correspon a una ruta en bucle entre uns blocs de formigó situats a l'espigó principal del port de Sant Feliu de Guíxols (Girona) mostrats a la figura 10.2 esquerra. L'altre dataset està adquirit en un entorn natural amb formes molt irregulars. El dataset es correspon a una volta i mitja a l'illot anomenat la Galera situat al Golfet del Cap de Creus (Girona) mostrat a la figura 10.2 dreta.

Figura 10.2: Entorn on s'han adquirit els datasets. Esquerra: Espigó principal del port de Sant Feliu de Guíxols. Dreta: Illot la Galera situat al Golfet del Cap de Creus.

Com que els datasets estan adquirits amb l'AUV submergit, no es disposa de senyal de GPS - ja que les ones electromagnètiques només penetren pocs centímetres dins de l'aigua - i, per tant, no es disposa del ground truth de la trajectòria seguida pel robot. En conseqüència, no es disposa d'una mesura exacta de la trajectòria del robot i no es pot mesurar l'error de cada tècnica de registre.

10.1.2 Data sets tridimensionals

Els data sets de dades tridimensionals s'han adquirit amb un Multibeam Profiling Sonar instal·lat en una plataforma de Pan&Tilt tot muntat en el Girona 500 AUV (Fig 10.3). Un Multibeam Profiling Sonar és un sensor format per un vector de Profiling Sonars individuals que permeten obtenir simultàniament per cada temps de mostreix una línia de punts. Muntant aquest sensor sobre una plataforma Pan&Tilt que rota en un eix de manera sincronitzada amb el temps de mostreig del Multibeam Profiling Sonar es poden aconseguir núvols de punts tridimensionals.

El Girona 500 és un AUV desenvolupat pel CIRS. Es tracta d'un vehicle no holonòmic, format per tres torpedes, que permet rotació en yaw i equipat

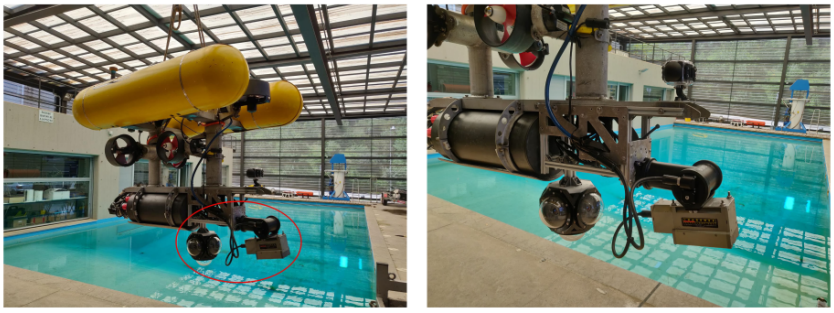

Figura 10.3: Girona 500 AUV amb el Multibeam Profiling Sonar muntat en la plataforma Pan&Tilt.

amb sensors inercials (DVL i IMU). Disposa d'una paylod frontal on es munta el Multibeam Profiling Sonar. Per adquirir el dataset el robot s'opera des d'una barca a través d'una boia en mode debug.

El data set està adquirit en una zona de blocs situats a l'espigó principal del port de Sant Feliu de Guíxols (Girona) mostrats a la figura 10.2 esquerra. El data set està format per alguns núvols de punts on s'escaneja la base d'aquests blocs. Un escaneig d'exemple es pot veure a la figura 10.4.

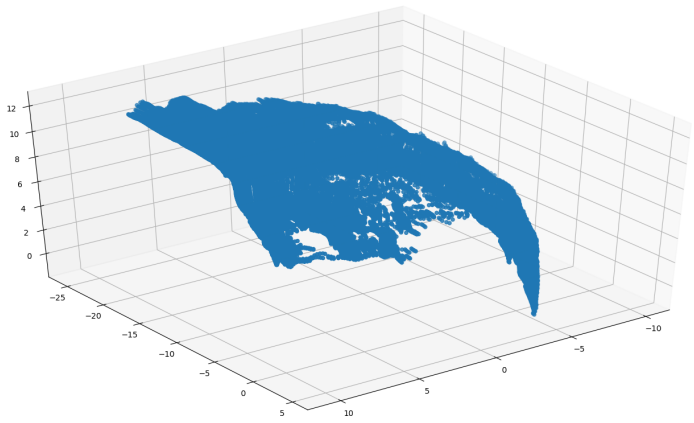

Figura 10.4: Desc.

10.2 Vectorització dels mètodes de registre

En aquesta secció s'experimenta la millora introduïda a la secció 9.5 sobre les classes `PointsToDistribution2D` i `DistributionToDistribution2D`. El

que volem obtenir amb aquest experiment es veure la quantitat de fils que s'han d'utilitzar al paral·lelitzar per aconseguir el millor temps possible.

Per fer aquesta prova s'ha mesurat el temps d'execució de les funcions `compute_score()`, `compute_score_and_gradient()` i `compute_score_gradient_and_hessian()` de les classes `PointsToDistribution2D` i `DistributionToDistribution2D`. Per cadascuna de les classes s'ha fet un boxplot separat on es mostra el temps d'execució de cada mètode utilitzant la paral·lelització i sense utilitzar-la. A cadascún dels boxplots hi ha 6 columnes on estan els diferents temps del mètodes per parelles sense paral·lelitzar i paral·lelitzant respectivament.

S'han utilitzat 2 datasets diferents per aquesta prova, tots dos són núvols de punts en 2D però un es troba en un entorn estructurat i l'altre un entorn desesctructurat descrits en la secció 10.1.1.

La figura 10.5 correspon a l'entorn estructurat i s'han utilitzat el nombre màxim de fils de l'ordinador que en aquest cas eren 8. En aquest es veuen una gran quantitat d'outliers, però podem observar que la mitja si que es troba reduïda en gran mesura, per exemple, on es més fàcilment visible es en les columnes 5 i 6 de l'esquerra on la mitja és 6 vegades més petita al paral·lelitzar.

La figura 10.6 correspon a l'entorn estructurat i s'han utilitzat 4 fils, aquí si que es troba molt més reduida la quantitat d'outliers però la mitja no es troba reduïda tan dràsticament. En el cas del P2D trobem en les columnes 5 i 6 una reducció de més de la meitat passant de 9 ms sense paral·lelitzar a 4 ms.

La figura 10.7 correspon a l'entorn estructurat i s'han utilitzat 2 fils, aquí els outliers es poden considerar desapareguts i encara s'observa una reducció notable del temps de mitja de 5ms a 2ms en la cinquena i sisena columna del cas P2D.

La figura 10.8 correspon a l'entorn desestructurat i s'han utilitzat el nombre màxim de fils de l'ordinador que en aquest cas eren 8, comparant-lo amb la figura 10.5 on s'utilitza la mateixa cuantitat de fils en aquest entorn trobem encara més outliers. Tot i així això pot estar provocat pel fet de que aquest segon dataset conté una major quantitat de scans (+200 vs 60). Pel que fa a les mitjes es veuen les dos figures semblants treient el cas D2D on a la columna 6 segurament per la gran quantitat d'outliers la mitja s'ha quedat practicament igual que a la columna 5, fins i tot ha augmentat lleugerament.

La figura 10.9 correspon a l'entorn desestructurat i s'han utilitzat 4 fils, observem el mateix comportament que a la figura 10.6 però a l'igual que s'ha comentat anteriorment com que aquest dataset és mes gran es poden observat una major quantitat d'outliers.

La figura 10.10 correspon a l'entorn desestructurat i s'han utilitzat 2 fils,

seguint la tendència amb els anteriors, no podem concloure que hi hagi una gran diferencia amb els resultats de l'entorn estructurat representats a la figura 10.7.

En general a les figures podem veure que el fet d'utilitzar una major quantitat de fils millora la mitjana dels temps, però alhora apareixen més outliers (especialment visibles en la figura 10.8). Aquests outliers són deguts a que el sistema operatiu ocupa els cores necessaris del processador per altres tasques i retrasa el programa. Per poder reduir aquestes col·lisions s'han utilitzat només 2 fils, ja que amb 2 ja s'aconsegueix una millora considerable evitant outliers.

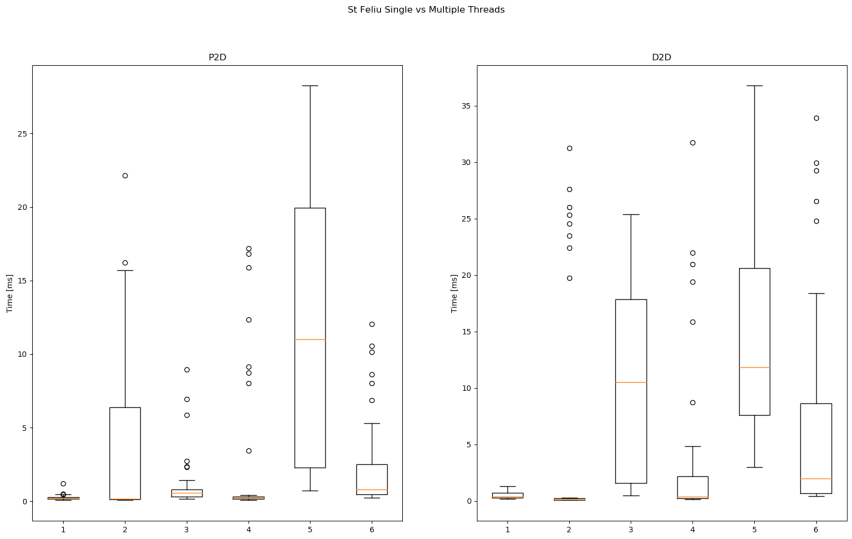

Figura 10.5: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 8 fils al paral·lelitzar, en un entorn estructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana paral·lelitzant

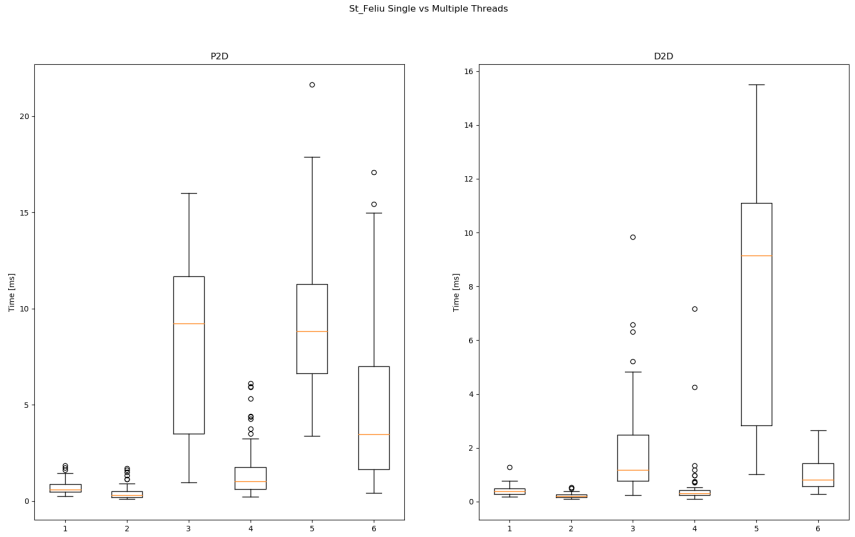

Figura 10.6: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 4 fils al paral·lelitzar, en un entorn estructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana paral·lelitzant

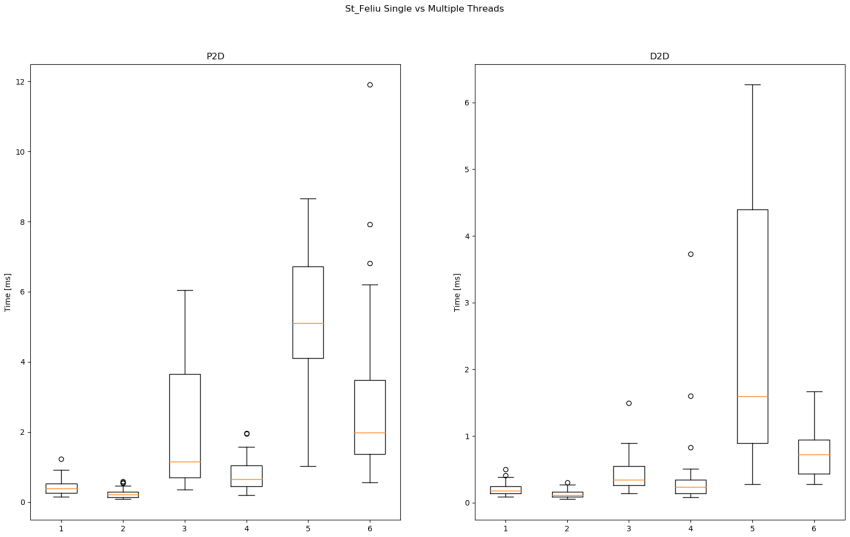

Figura 10.7: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 2 fils al paral·lelitzar, en un entorn estructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana paral·lelitzant

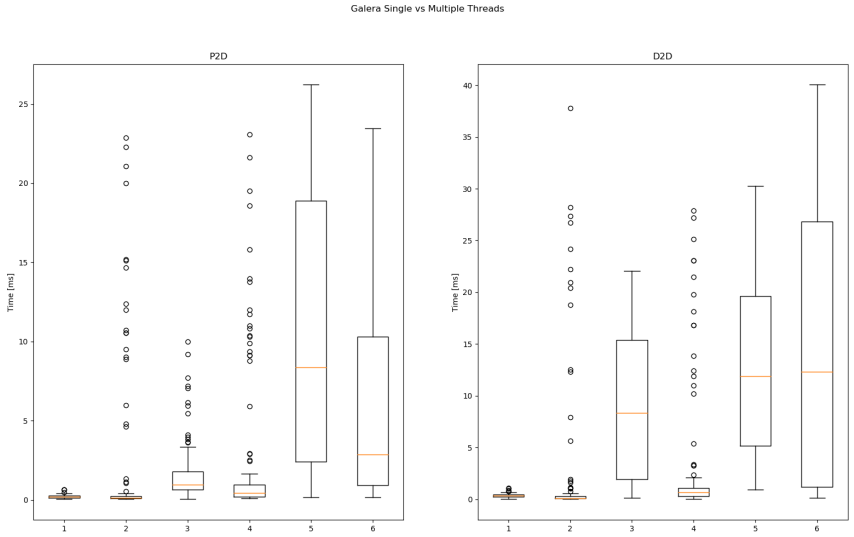

Figura 10.8: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 8 fils al paral·lelitzar, en un entorn desestructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana para·lelitzant

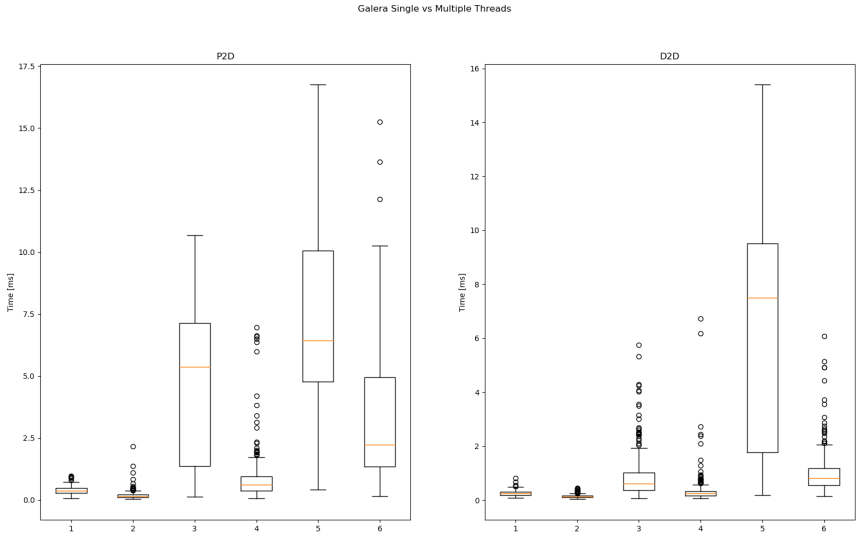

Figura 10.9: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 4 fils al paral·lelitzar, en un entorn desestructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana paral·lelitzant

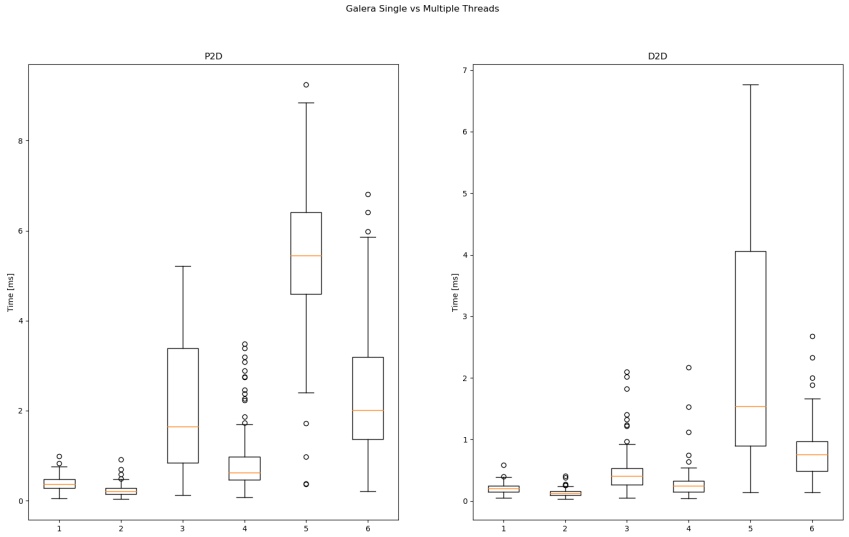

Figura 10.10: Temps emprat per fer els càlculs de score, vector jacobià i matriu hessiana, utilitzant 2 fils al paral·lelitzar, en un entorn desestructurat, a l'esquerra amb el mètode P2D i a la dreta amb D2D, cada columna del boxplot representa un càlcul diferent, 1:score, 2:score paral·lelitzant, 3:score i jacobià, 4:score i jacobià paral·lelitzant, 5:score, jacobià i hessiana, 6:score, jacobià i hessiana paral·lelitzant

10.3 Millora de l'eficiència computacional de la tècnica P2D

En aquest experiment volem mesurar la incidència sobre el registre de la millora sobre la tècnica P2D introduïda a la secció 9.6 i quin estalvi de temps de computació suposa. Per fer-ho, comparem resultats i temps de computació amb la mateixa tècnica de registre sense implementar el filtre. Volem demostrar que el temps de càlcul es redueix sense afectar el resultat del registre.

En primer lloc, mesurem l'estalvi de temps de càlcul que suposa filtrar les parelles punt-component. A les figures 10.11 i 10.12 es mostren dos boxplots on al de l'esquerra es representa el temps de càlcul sense aplicar el test de Chi quadrat i al de la dreta el temps aplicant-lo. A cada columna del box plot es mostra el temps de càlcul per una avaluació sencera d'un núvol de punts en un GMM per cada un dels mètodes de la classe "Point2Distribution2d".

Aquests càlculs normalment es fan paral·lelitzant en 2 threads, però per evitar possibles outliers deguts a l'ocupació dels threads pel sistema operatiu, es va desactivar per fer aquestes proves i recollir les dades.

Tal com és d'esperar, a les figures 10.11 i 10.12 es veu com l'estalvi de temps en l'score és inexistent. En canvi, sí que s'observa millora en els altres mètodes. Pel gradient, la mitjana del temps de càlcul es redueix a la meitat, passant de 2.5ms de mitja a 1.2ms en el cas estructurat i de 2ms a 1ms en el entorn desestructurat, ja que s'estalvien 8 productes matricials cada vegada que es rebutja una parella. Per la hessiana, la millora encara és més significativa i el temps de càlcul es redueix a un 40% de l'original, passant de 5.2ms a 2.2ms i de 4.5ms a 1.8ms en els entorns estructurat i desestructurat respectivament, ja que s'estalvien 13 productes matricials per cada parella rebutjada.

També podem observar que la mitja del temps de càlcul en l'entorn desestructurat és menor a la del estructurat. Això es degut a que els scans del data set de la Galera contenen menys punts que els del data set del port de Sant Feliu, ja que en l'exploració al voltant de la Galera només s'arriba a detectar una paret a un dels costats de AUV. En canvi, al port es poden detectar més parets en el camp de visió del sensor al passar per exemple entre dos del blocs que formen un passadís.

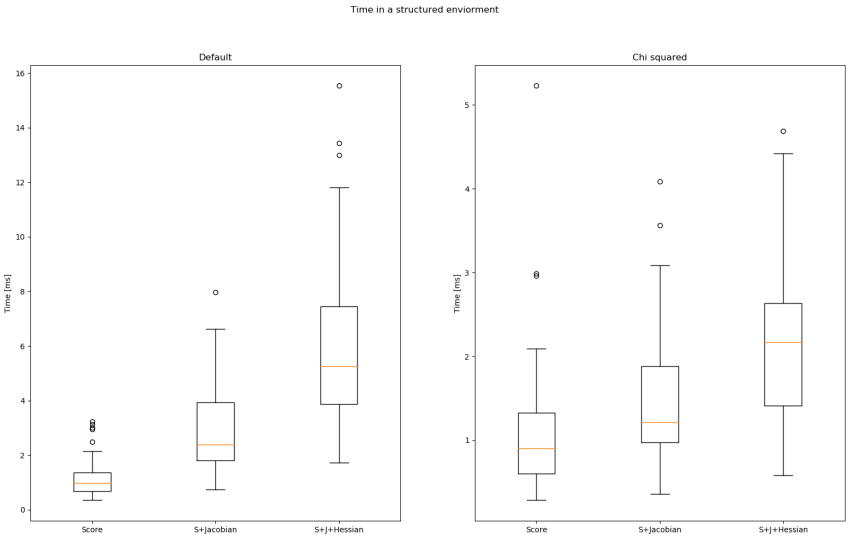

Figura 10.11: Temps per computar score, jacobiana i hessiana en un entorn estructurat.

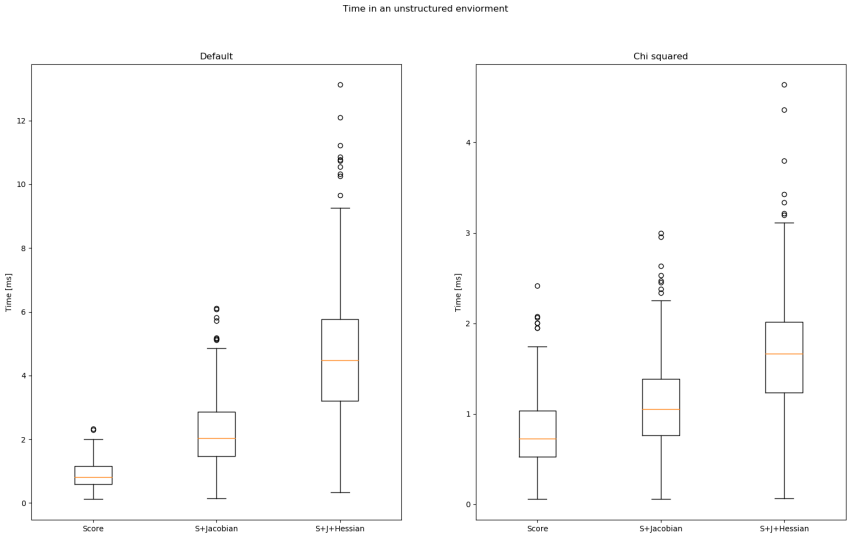

Figura 10.12: Temps per computar score, jacobiana i hessiana en un entorn desestructurat.

En segon lloc, executem dos mètodes de registre en paral·lel - un aplicant el mètode P2D original i l'altre aplicant el mètode que incorpora el filtre - i avaluem l'error entre els registres. A les figures 10.13 i 10.14 es mostra la distància euclídia entre la translació obtinguda per cada registre i la distància euclídia entre la rotació obtinguda per cada registre. La mitjana de l'error en la translació es de l'ordre dels centímetres i la mitjana de l'error en la rotació es de l'ordre de les dècimes de radians pels dos data sets. Per tant, es conclou que el test de Chi quadrat no afecta al registre ja que es rebutgen aparellaments punt-component que no aporten cap informació significativa.

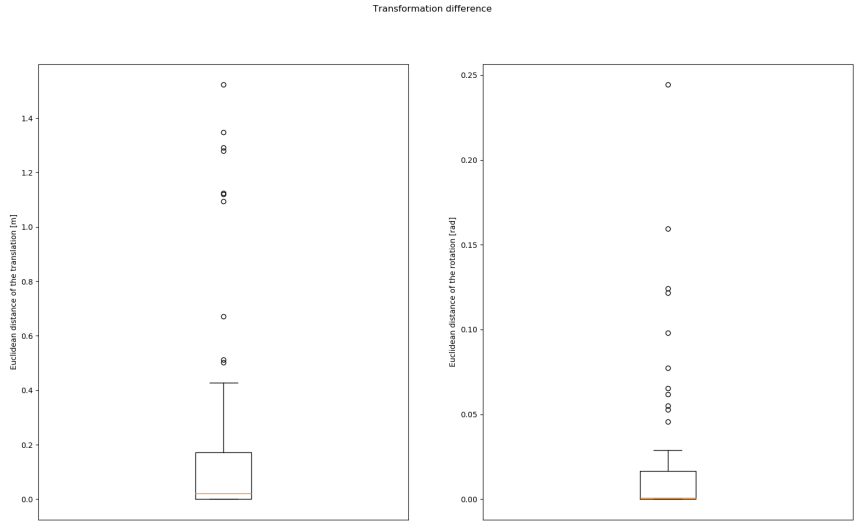

Figura 10.13: Error en el registre en un entorn estructurat.

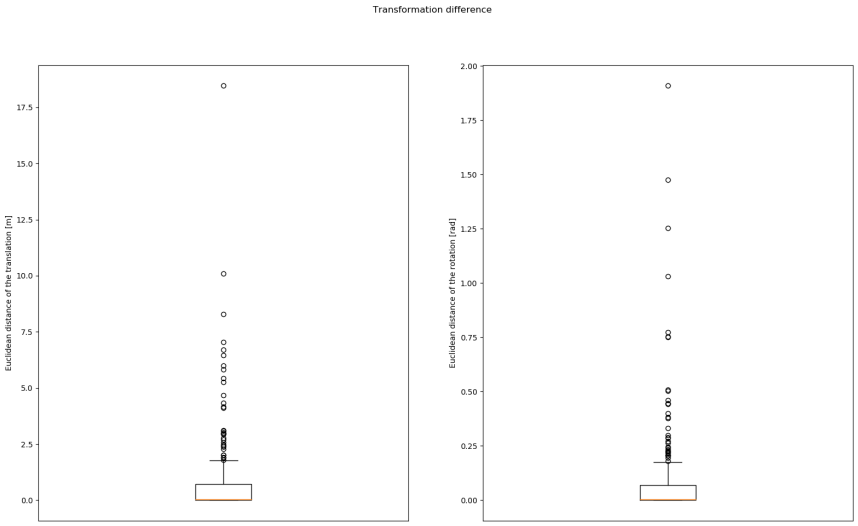

Figura 10.14: Error en el registre en un entorn desestructurat.

A continuació es mostren exemples concrets dels registres efectuats. A les següents figures (10.15, 10.16, 10.17 i 10.18) es troben exemples de diferències entre els dos mètodes. Cal tenir en compte que l'score amb chi squared en les mateixes condicions serà menor, ja que l'original té en compte tots els punts i implementant el test no.

En la figura 10.15 es veu un exemple de matching amb un bon resultat per part dels dos mètodes on s'ha corregit correctament la posició del scan.

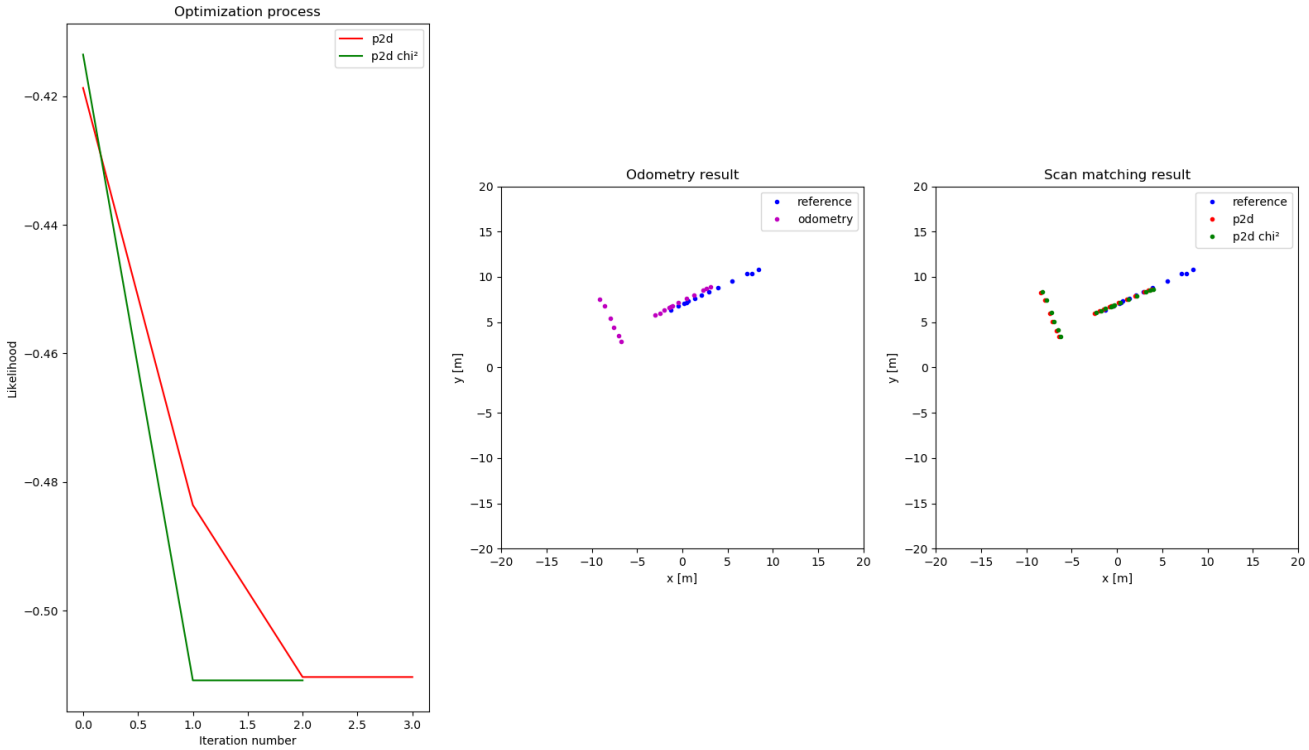

Figura 10.15: Exemple d'un registre particular pel dataset estructurat. Esquerra: Evolució del score durant el procés d'optimització. Centre: Associació d'escanejos provinent del sistema de navegació a la deriva. Dreta: Registre amb els mètodes P2D òptim i P2D no òptim.

En la figura 10.16, al contrari que en la 10.15, tots dos mètodes han necessitat més iteracions, 7 en total, per acabar convergint i obtenir la solució.

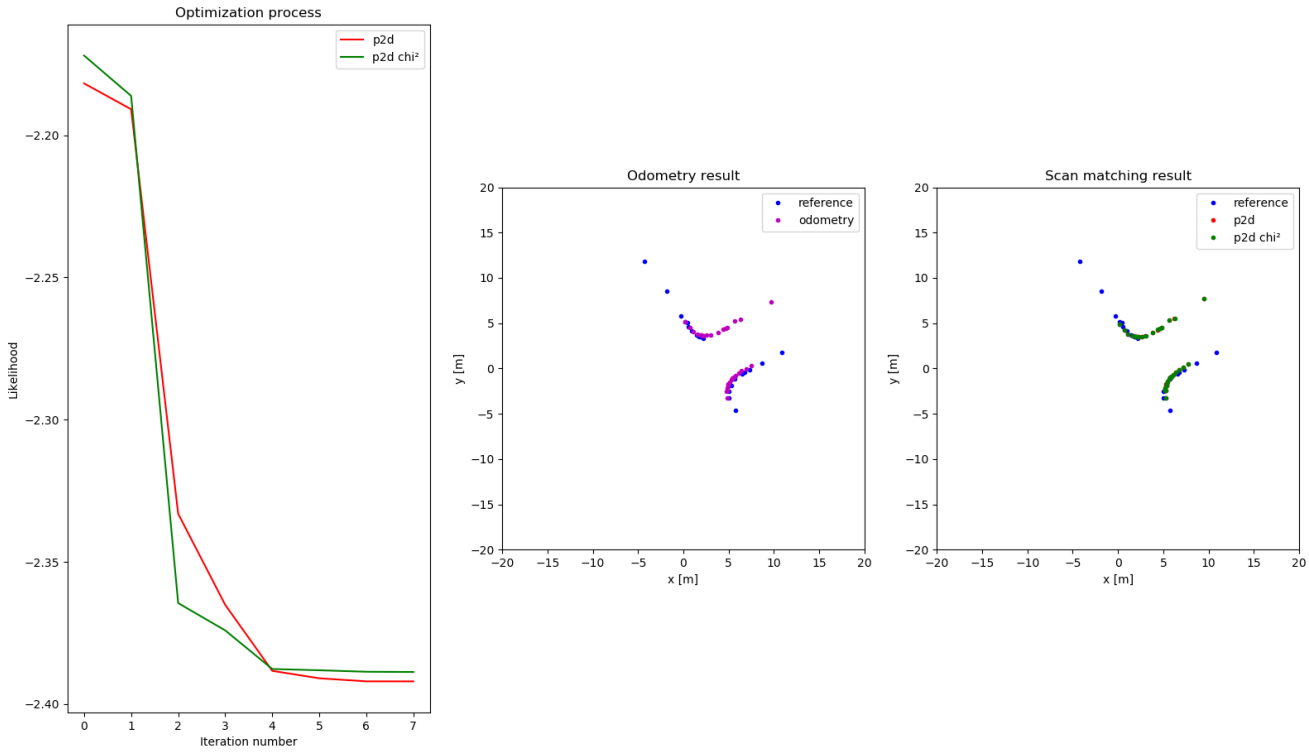

Figura 10.16: Exemple d'un registre particular pel dataset estructurat. Esquerra: Evolució del score durant el procés d'optimització. Centre: Associació d'escanejos provinent del sistema de navegació a la deriva. Dreta: Registre amb els mètodes P2D òptim i P2D no òptim.

En la figura 10.17 veiem un resultat semblant al de la figura 10.15 però en un entorn desestructurat. En les dues s'arriba a una solució en poques iteracions i les dues solucions són properes entre elles.

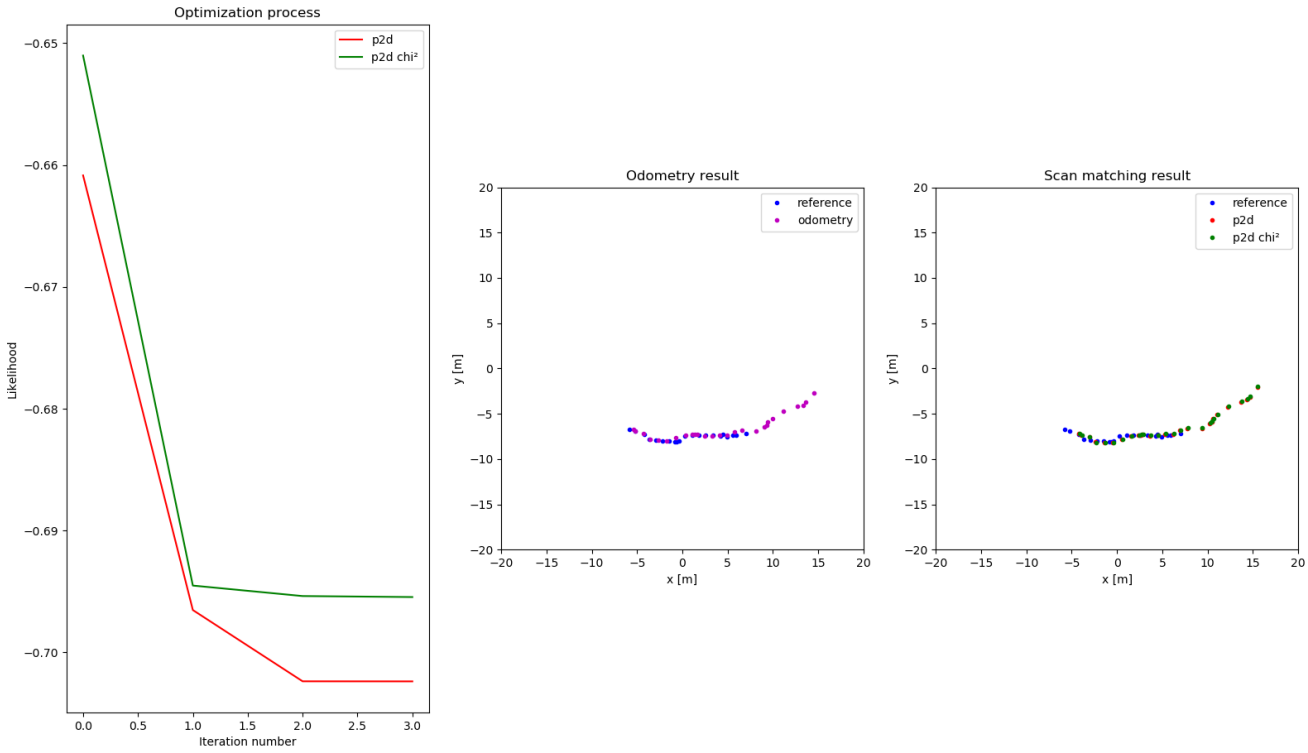

Figura 10.17: Exemple d'un registre particular pel dataset no estructurat. Esquerra: Evolució del score durant el procés d'optimització. Centre: Associació d'escanejos provinent del sistema de navegació a la deriva. Dreta: Registre amb els mètodes P2D òptim i P2D no òptim.

A la figura 10.18 veiem que les solucions donades pels dos mètodes són lleugerament diferents. Tot i que segons el valor del score podríem pensar que el mètode original dona una millor solució, l'òptim sembla donar una transformació millor.

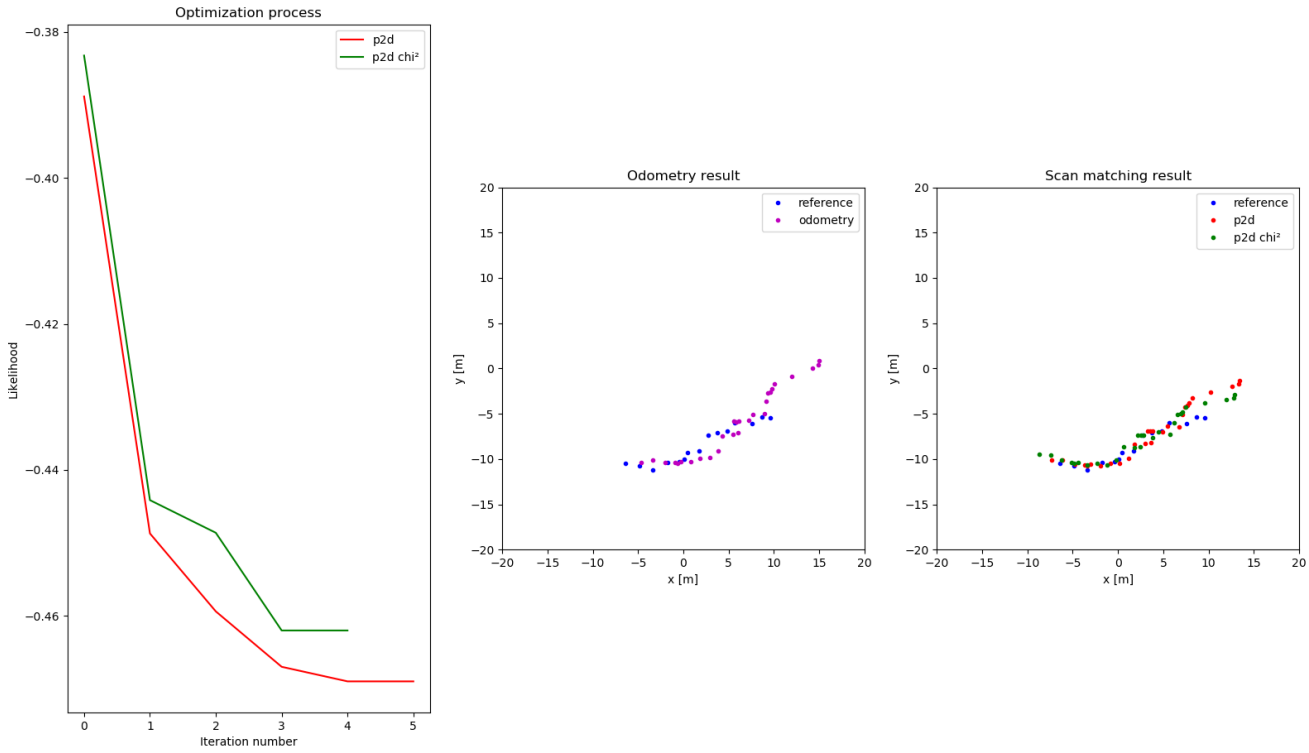

Figura 10.18: Exemple d'un registre particular pel dataset no estructurat. Esquerra: Evolució del score durant el procés d'optimització. Centre: Associació d'escanejos provinent del sistema de navegació a la deriva. Dreta: Registre amb els mètodes P2D òptim i P2D no òptim.

En conclusió, hem comprovat experimentalment com la millora introduïda compleix els objectius teòrics pels quals l'hem introduït. Amb el test de Chi quadrat es poden rebutjar les parelles punt-component que no aporten informació rellevant al registre i aconseguim reduir notablement el temps de càlcul sense afectar substancialment el resultat del registre obtingut. Per tant, aquesta millora ajuda a aconseguir una eina de registre més eficient des del punt de vista computacional.

10.4 Front-End 3D

En aquest experiment volem testejar com la funció “ndt_constructor()” processa un núvol de punts tridimensional com el de la figura 10.4 . Per fer-ho, s’ha parametritzat el constructor amb un minim de 20 punts per component i una mida de cel·la de 4 metres. Amb aquesta parametrització s’obté un GMM format per 62 components que comprimeix un núvol de punts de dimensió 40.680. Com es veu el poder de compressió és elevat. No obstant, el temps de càlcul és de 25 segons, fet que impossibilita l’execució en línia del registre.

Un cop après el model, es poden utilitzar els mètodes de plot per mostrar el model. A la figura 10.19 es pot veure la visualització que en fa el mètode “plot_gmm_density()” quan es fan 20 seccions al llarg de l’eix z. D’esquerra a dreta i de dalt a baix es pot veure que l’evolució de seccions en el pla xy concorda amb el núvol de punts de la figura 10.4. Per tant, es conclou que tant el mètode del Front-End com el mètode de visualització funcionen correctament.

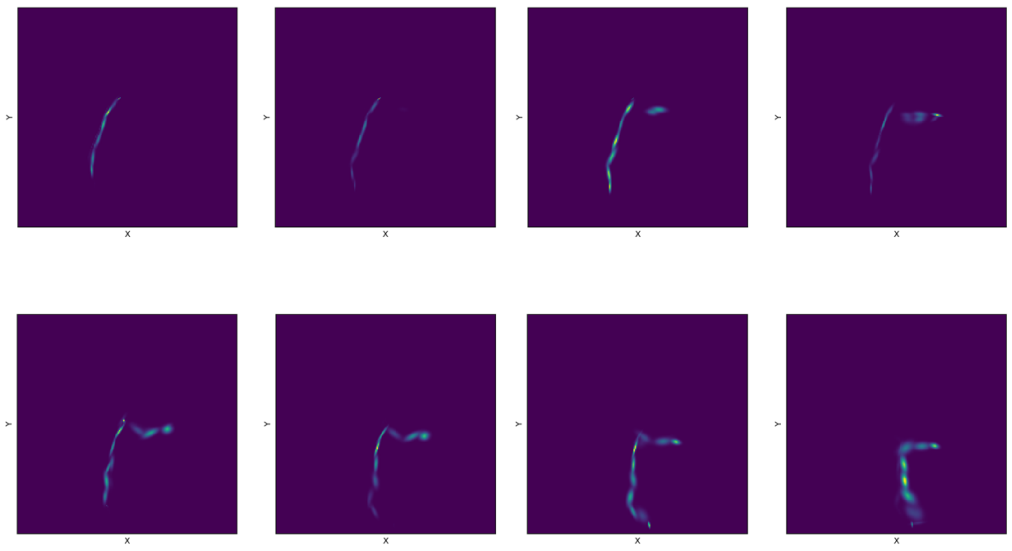

Figura 10.19: Desc.

A la figura 10.20 es pot veure la visualització del GMM aconseguida amb el mètode "plot_gmm_components()". A la figura s’observa que la visualització és coherent amb el núvol de punts de la figura 10.4 i que tots els el·lipsoïdes estan degenerats en la direcció normal al núvol de punts, tal i com és d’esperar. Per tant, es conclou que aquest mètode també funciona correctament.

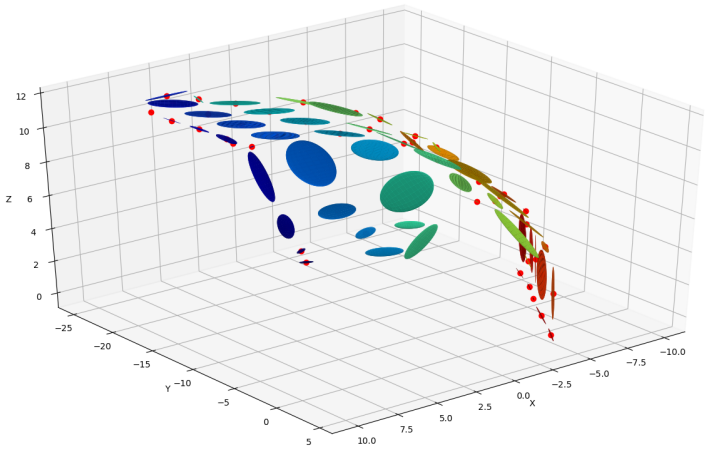

Figura 10.20: Desc.

En vista dels elevats temps de càlcul necessaris per aprendre i visualitzar cada núvol de punts, com a treball futur, s'ha de considerar la vectorització del Front-End i del visualitzador "plot_gmm_density()"per poder-lo executar des d'una targeta gràfica. D'aquesta manera es podria reduir el temps de càlcul amb l'objectiu d'aconseguir una tècnica de registre tridimensional executada en línia en el robot.

10.5 Solver basat en la factorització de Cholesky modificada

Per comprovar el funcionament de la classe CholeskyNewtonMethod introduïda a la secció 9.8 s'ha fet un petit programa de proves on les funcions del mètode són públiques per facilitar el seu ús. Aquesta prova consisteix en 3 parts: comprovar que la matriu L sigui lower triangular, comprovar que s'hagi modificat la matriu correctament perquè tots els valors propis siguin positius i, finalment, que sigui capaç de solucionar un sistema amb una matriu triangular.

Per la primera part es genera una matriu aleatòria i es computa la matriu L amb el mètode, per comprovar que aquesta sigui triangular. Es mostra per pantalla a la figura 10.21.

```
CholeskyNewtonMethod<6> cho_nm(meth, 0.0001, 0.0001, 100, 1e-6);
```

```
Eigen::Matrix<double,6,6> rand = Eigen::Matrix<double,6,6>::Random();

cout << "Random matrix: \n" << rand << endl;
tuple<Eigen::Matrix<double,6,6>, Eigen::Matrix<double,6,6>,
    Eigen::Matrix<double,6,1>> result = cho_nm.compute_l_matrix(rand);

cout << "L:\n"<< get<0>(result) << "\nP:\n" << get<1>(result)<<
    "\ntau:\n" << get<2>(result) << endl;
```

Un cop comprovat que la matriu té la forma correcta es comprova fins a quin punt s'ha pertorbat la matriu restant la factorització a la matriu original permutada. Es pot observar que la diagonal del resultat coincideix amb la tau. Per tant, podem concloure que la factorització és correcte.

També es calculen els valors propis per comprovar que el mètode sí que fa el que es vol, que és modificar la matriu per a que aquesta sigui definida positiva. És a dir, que tots els seus valors propis siguin positius.

```
Eigen::MatrixXd LLt = get<0>(result) * get<0>(result).transpose();
cout << "PAPt - LLt =\n" << get<1>(result) * rand *
    (get<1>(result).transpose()) - LLt << endl;

cout << "eigv LLt\n" << LLt.eigenvalues() << endl;
```

Finalment, per comprovar que pot solucionar sistemes amb matrius triangulars correctament es crida el mètode amb un vector d'uns i el resultat obtingut és correcte.

```
Eigen::Vector<double,6> ones_v = Eigen::Vector<double,6>::Ones();
cout << "vector:\n" << ones_v << endl;
Eigen::Matrix<double,6,6> A = get<0>(result);
cout << "result:\n" << cho_nm.solve_system_l(A,ones_v);
```

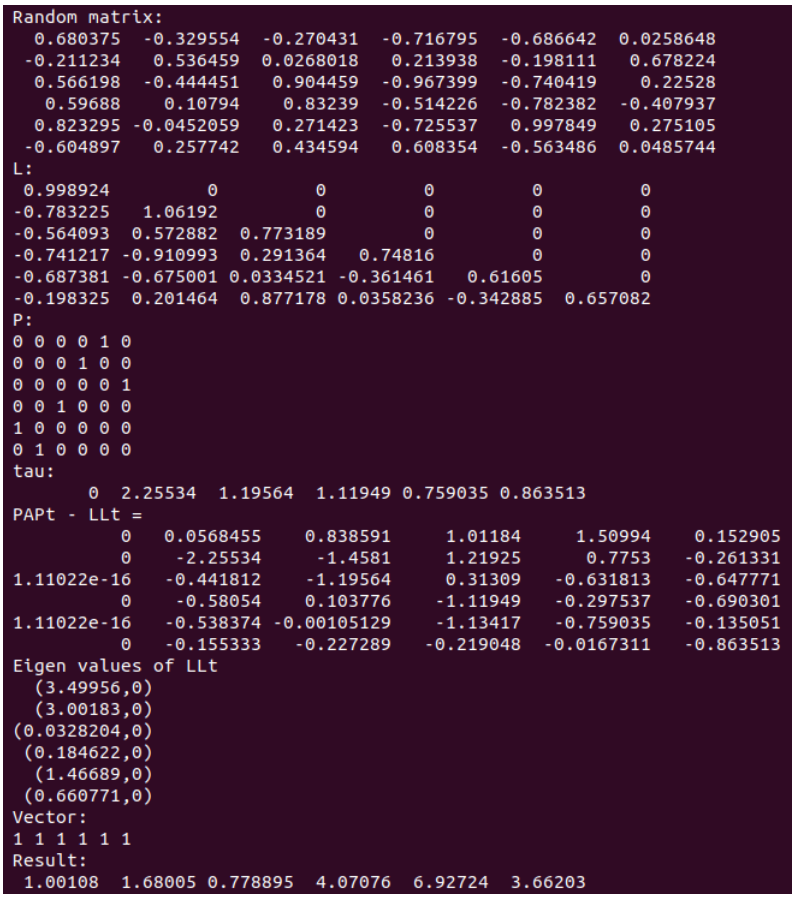

Figura 10.21: Output del programa per testejar el funcionament de la factorització de Cholesky

Capítol 11

Conclusions

En aquest treball s'ha fet la implementació d'una llibreria en C++, que permet fer el registre de núvols de punts obtinguts amb sonar en un entorn submarí. Aquesta implementació està basada en la tècnica de registre Normal Distribution Transform, que s'ha adaptat per les condicions del domini submarí.

A l'inici del treball, per tal d'habituar-nos a l'entorn de programació, es va començar per fer una implementació de prova de la tècnica NDT a partir de l'article científic on es va proposar per primera vegada [cita]. Aquesta es va fer de manera molt simple per així entendre com funciona l'algorisme. Alhora, va servir per identificar les parts clau de la tècnica per poder, posteriorment, fer un bon disseny de les classes que estructuren la llibreria.

L'estructura de classes que proposem es basa en tres blocs conceptuals: Front-End, mètode de registre i solver. El Front-End fita un GMM donat un núvol de punts. Els mètodes de registre defineixen les funcions de cost que defineixen el problema de registre com un problema d'optimització, juntament amb les seves derivades primera i segona. Finalment, el solver s'encarrega de solucionar numèricament el problema d'optimització que planteja el mètode de registre aplicant mètodes del gradient.

El Front-End és una llibreria de funcions formada per funcions que donat un núvol de punts retornen un GMM. De moment només s'ha implementat la funció ndt_constructor(), basada en la tècnica NDT. Aquesta clusteritza un núvol de punts projectant-hi una graella. Per cada cel·la de la graella amb un mínim de punts, s'estableix una component del model ajustant una distribució gaussiana als punts.

Els mètodes de registre utilitzen la interfície ScanMatchingMethods que defineix els mètodes que s'han d'implementar en totes les subclasses. Els constructors de les seves subclasses depenen de la implementació, però sempre es necessita donar dos scans (en forma de GMM o núvol de punts segons el mètode) que s'utilitzen per fer el registre. Aquests dos scans, donats al constructor, els utilitzen els mètodes que calculen les funcions de cost i derivades. Els mètodes compute_score, compute_score_and_gradient, compute_score_gradient_and_hessian, donada per paràmetre una transformació que s'aplica al segon scan, retornen el valor de la funció de cost, el valor de la

primera derivada de la funció de cost, és a dir el gradient, i el valor de la segona derivada de la funció de cost, és a dir la matriu hessiana. Finalment, el mètode plot permet visualitzar les dades en mode de debug. A aquest mètode se li ha de donar per paràmetre l'evolució del cost en el procés d'optimització, la transformació original de la que parteix el solucionador i la transformació òptima a la que arriba el solucionador. Amb aquesta informació genera un gràfic que permet visualitzar el procés d'optimització.

S'han implementat dues subclasses de ScanMatchingMethods: Point2Distribution i Distribution2Distribution. A la primera s'implementa el mètode P2D pel cas bidimensional, on es registra un scan representat per un GMM contra un altre scan representat per un núvol de punts. En canvi, a la segona s'implementa el mètode D2D pel cas bidimensional, on es registren dos scans representats amb GMMs. Com que les funcions de cost són diferents entre els mètodes, les implementacions són completament diferents. Per això, s'ha creat una interfície que permet agrupar els dos mètodes, ja que no es poden implementar juntes templaritzant segons el cas.

Els solucionadors utilitzen la interfície Solver que defineix els mètodes que s'han d'implementar en totes les subclasses. Aquesta conté atributs min_norm_, min_score_inc_ i max_NM_it_ per definir els criteris de parada de l'algorisme implementat. També conté l'atribut method_ de la classe ScanMatchingMethods per definir les funcions de cost i les seves derivades i l'atribut process_ que és un vector on es guarda la score de cada iteració del solver. Els mètodes de la interfície són compute_optimum, process i plot_process. Tot el codi de Solver ha d'estar templaritzat amb la dimensió del vector que defineix la transformació, per així poder ser utilitzar tant en problemes de dues o tres dimensions. D'aquesta manera l'estructura del codi no varia segons la dimensió d'aquest vector.

Com a solucionadors s'han implementat algorismes numèrics basats en el gradient de la funció de cost. S'han implementat dues subclasses de Solver: NewtonMethod i LineSearchNewtonMethod. El primer és el mètode de Newton pur que només utilitza el vector gradient i la matriu hessianna per determinar el pas i la direcció d'optimització. El segon conserva les característiques de l'anterior, però afegeix un mètode de line_search per determinar la mida òptima del pas en cada iteració. Aquest mètodes es basa en les condicions de Wolfe, implementades a les funcions first_Wolfe_condition i second_Wolfe_condition. Per tant, cal incloure nous atributs per definir el criteri de parada de l'algoritme de linesearch max_LS_it establint un màxim d'iteracions; beta1_ i beta2_ que són constants necessàries per les condicions de Wolfe; i gamma_ una altra constant per l'algoritme de LineSearch.

Per tal de publicar el codi en un repositori s'ha fet la documentació de

tota la llibreria amb el programa doxygen. Aquest permet generar un html o pdf amb tota la informació necessària del codi escrita en aquest amb un format específic de comentaris.

Un cop feta la implementació de la llibreria, la feina ha seguit posant el focus en l'optimització de l'eficiència del codi, proposant solucions particulars a problemes particulars. Per tant, primer, s'identifiquen punts ineficients i, a continuació, se'n proposa una solució.

En primer lloc, s'ha decidit millorar el temps d'execució dels càlculs de la funció de cost i de les seves derivades a la interfície ScanMatchingMethods. Les funcions que fan aquests càlculs es criden a cada iteració del solucionador i cada vegada han de recórrer dos scans niuats. Ens hem adonat de que aquests càlculs són fàcilment paral·lelitzables, ja que per cada punt i/o component del GMM es fa un càlcul independent que forma part d'un sumatori. Això ens permet utilitzar l'API OpenMP [OpenMP 022] per afegir la paral·lelització.

La implementació s'ha provat amb diferents quantitats de fils de la CPU concurrents. Després de fer proves amb 8, 4 i 2 fils s'ha conclòs recomanar utilitzar només 2 fils, ja que la utilització de més fils provoca pics de temps. Aquests són deguts a que el sistema operatiu també en fa ús, provocant retards en part dels càlculs.

En segon lloc, observant els dos mètodes de registre es pot veure que el mètode P2D treballa amb una major quantitat d'associacions, ja que compara un núvol de punts amb un GMM. En canvi, aquest problema no es troba al mètode D2D ja que es comparen dos GMM. El nombre d'associacions és molt menor ja que un GMM actua com una compressió d'un núvol de punts. Per reduir el cost computacional del mètode P2D es proposa retallar de manera automàtica el nombre d'associacions a l'hora de fer els càlculs. Per fer-ho, es proposa que cada parella punt-distribució es sotmeti a un test de Chi quadrat. Aquest test determina si un punt es troba a una distància rellevant d'una distribució a partir de la distànca de Mahalanobis, la qual indica la rellevància d'un punt concret a un altre on hi ha definida una distribució gaussiana. Si la parella passa el test s'utilitza per avaluar les derivades. Si no el passa, es descarta.

Aquest retall en el nombre d'associacions podria provocar que els resultats obtinguts deixin de ser correctes per la pèrdua d'informació. Per comprovar que aquest no sigui el cas, es mesura la diferència entre les transformacions obtingudes i s'inspecciona de manera qualitativa si els resultats són correctes. L'experimentació permet concloure que és una opció viable i que permet obtenir una millora de temps considerable, d'un 40% en la funció compute_score_gradient_and_hessian.

Un cop millorada l'eficiència computacional del codi, s'ha volgut comprovar com es comporta la llibreria al processar núvols de punts tridimensionals. Fins a la realització d'aquest treball, la comunitat del CIRS encra no havia aplicat la tècnica de registre de GMM-Registration a núvols de punts tridimensionals. Només s'havien fet registres 3D aplicant l'algorisme ICP sense aconseguir resultats rellevants, ni molt menys determinar de manera automàtica la incertesa del registre. Per fer-ho, ha calgut templaritzar l'objecte GaussianMixturesModel i la funció ndt_constructor del Front-End. Alhora, ha calgut implementar les classes P2D3D i D2D3D per escanejos tridimensionals.

La implementació de la part tridimensional de la llibreria ha començat pel Front-End, fitant GMMs a núvols de punts de tres dimensions. Per templaritzar el codi, s'han canviat totes les instàncies on es declaren vectors i matrius d'Eigen amb dimensions concretes, a una variable definida per la templarització de cada classe. Alhora, s'ha modificat la implementació de la graella de suport al constructor, ja que amb un vector de vectors, només es poden representar 2 dimensions i si s'afegeix una capa més als vectors, en el cas bidimensional s'estarà ocupant molta memòria innecessàriament. Per tant, s'ha acabat afegint la nova capa però en el cas 2D la primera dimensió de la graella està limitada a 1 i, en conseqüència, s'ha d'adreçar com [1,x,y].

Aquest mètode genera GMM tridimensionals i, per tant, també cal modificar la classe GaussianMixturesModel perquè els pugui emmagatzemar. Per fer-ho s'ha templaritzat la classe amb un enter per representar la dimensió de 2 o 3. Aquest enter s'utilitza per la definició dels vectors i matrius d'Eigen que representen les mitjanes i les covariàncies. No obstant, els mètodes de plot no tenen sentit tal com estan fets per al cas tridimensional. El mètode plot_density s'ha modificat perquè a cada GMM es generin imatges com les del 2D per n capes al llarg d'un eix especificat per paràmetre. En canvi, pel mètode plot_components no es pot fer directament en C++, ja que la llibreria Matplotlib per C++ no permet dibuixar superfícies tridimensionals. Com que aquest mètode no s'aplica al registre, només s'utilitza per debugar i visualitzar les dades, s'ha decidit generar un fitxer de text amb totes les dades necessàries perquè un script de Python faci els gràfics aplicant la llibreria Maatplotlib que sí que inclou eines tridimensionals. Aquest script dibuixa el·lipsoides a partir de la descomposició en vectors i valors propis de la covariància amb el centre a la mitjana de la component.

Amb els mètodes de plot es comprova que el Front-End generi els GMM de manera correcta contrastant tant el gràfic de components com el de densitat al núvol de punts original. L'experimentació demostra que els resultats obtinguts són bons. Aquest fet valida el funcionament de les funcions, però

es comprova que no són executables en temps real, trigant 25s per fitar un GMM a un núvol de 40.680 punts. Aquest fet impossibilita l'execució en línia del registre.

Un cop implementat el Front-End, s'ha implementat la classe PointsToDistribution3D. Es decideix començar pel mètode P2D ja que la funció score és més simple analíticament i, en conseqüència, les seves derivades també. No obstant, després de diferents proves no s'aconsegueix la convergència de cap registre. En primer lloc, es pensa que els problemes poden ser provocats per una mala inicialització del solucionador. Per això s'implementa i s'intenta utilitzar el solucionador CholeskyNewtonMethod. Aquest solucionador aplica una factorització de Cholesky modificada a la matriu hessiana per pertorbar numèricament la matriu en cas que aquesta sigui indefinida, forçant que sigui definida positiva. D'aquesta manera s'assegura una direcció de minimització al mètode de Newton. No obstant, l'aplicació d'aquest solucionador provoca pertorbacions molt agressives que no afavoreixen la convergència, mostrant que el problema no és la inicialització del solucionador. En segon lloc, es dedueix que el problema està causat per les derivades de l'acció de la rotació, fet molt més complicat de solucionar. Creiem que parametritzar l'acció de la rotació amb matrius de rotació és una mala idea i que s'obtindrien millors resultats parametritzant amb quaternions. No obstant, aquesta prova no s'ha efectuat per manca de temps.

En conclusió, en aquest treball s'ha aconseguit implementar una llibreria que permet el registre de núvols de punts bidimensionals obtinguts amb sonar en un entorn submarí. Les dades de sonar són molt sorolloses, provocant que les tècniques de registre convencionals aplicades en entorns terrestres i aeris deixin de funcionar correctament. Alhora, s'ha assegurat que la implementació de la tècnica de registre sigui suficientment ràpida per poder ser executada en temps real en un AUV, donant diferents variants dels mètodes per efectuar el registre. Finalment, s'ha iniciat la implementació de la llibreria per ser utilitzada també amb núvols de punts 3D, tot i que encara no és completament funcional.

Capítol 12

Treball futur

Després de la realització d'aquest treball es detecten dues línies de treball futur. En primer lloc, resoldre els problemes tècnics amb les derivades de l'acció rotació parametritzant-la amb quaternions. Això hauria de permetre aconseguir registrar núvols de punts tridimensionals. En segon lloc, ja hem vist que tant aprendre el GMM com registrar els núvols de punts en el cas tridimensional no és viable fer-ho en temps real des d'un punt de vista computacional. Per millorar l'eficiència computacional del cas tridimensional caldria fer-ne una nova implementació en una targeta gràfica, ja que tots els càlculs sobre el núvol de punts es poden vectoritzar.

Bibliografia

[Atom 022] Atom. *atom*, (Accedit: Abril 2022). Disponible a https://atom.io/. (Cited on page 18.)

[Bailey 2006] T. Bailey and H. Durrant-Whyte. *Simultaneous localization and mapping (SLAM): part II*. IEEE Robotics & Automation Magazine, vol. 13, pages 108–117, 2006. (Cited on page 10.)

[Biber 2003] P. Biber and W. Strasser. *The normal distributions transform: a new approach to laser scan matching*. Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pages 2743–2748, 2003. (Cited on page 12.)

[Bierlaire 2015] M. Bierlaire. Optimization: Principles and algorithms. EPFL Press, 2015. (Cited on pages 13, 41 and 50.)

[Bitbucket] Bitbucket. *Built for professional teams*. Disponible a https://bitbucket.org/. (Cited on page 17.)

[Boost 022] Boost. *boost*, (Accedit: Maig 2022). Disponible a https://www.boost.org/. (Cited on page 18.)

[Durrant-Whyte 2006] H. Durrant-Whyte and T. Bailey. *Simultaneous localization and mapping: part I*. IEEE Robotics & Automation Magazine, vol. 13, pages 99–110, 2006. (Cited on page 10.)

[Eigen] Eigen. *C++ template library for linear algebra*. Disponible a https://eigen.tuxfamily.org/index.php?title=Main_Page. (Cited on page 18.)

[Electron 022] Electron. *electron framework*, (Accedit: Abril 2022). Disponible a https://www.electronjs.org/. (Cited on page 17.)

[Gallego 2015] G. Gallego and A. Yezzi. *A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates*. Math Imaging Vis, vol. 51, pages 378–384, 2015. (Cited on page 52.)

[Gedit 022] Gedit. *gedit*, (Accedit: Abril 2022). Disponible a https://wiki.gnome.org/Apps/Gedit. (Cited on page 18.)

[Generalized-ICP 022] Generalized-ICP. *Generalized-ICP*, (Accedit: Abril 2022). Disponible a https://www.robots.ox.ac.uk/~avsegal/resources/papers/Generalized_ICP.pdf. (Cited on page 11.)

[Git 022] Git. *git*, (Accedit: Abril 2022). Disponible a https://git-scm.com/. (Cited on page 17.)

[Gnuplot 022] Gnuplot. *gnuplot*, (Accedit: Abril 2022). Disponible a http://www.gnuplot.info/. (Cited on page 18.)

[ICP 022] ICP. *A method for registration 3-D shapes*, (Accedit: Abril 2022). Disponible a https://www.robots.ox.ac.uk/~avsegal/resources/papers/Generalized_ICP.pdf. (Cited on page 10.)

[Jiang 2011] B. Jiang and B. C. Vemuri. *Robust point set registration using Gaussian Mixture Models*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, pages 1633–1645, 2011. (Cited on page 12.)

[Matlab 022] Matlab. *matlab*, (Accedit: Maig 2022). Disponible a https://www.mathworks.com/products/matlab.html. (Cited on page 18.)

[Matplotlib 022] Matplotlib. *matplotlib-cpp*, (Accedit: Abril 2022). Disponible a https://github.com/lava/matplotlib-cpp. (Cited on pages 18 and 48.)

[OpenGL 022] OpenGL. *opengl*, (Accedit: Maig 2022). Disponible a https://www.opengl.org/. (Cited on page 18.)

[OpenMP 022] OpenMP. *Pàgina principal de l'API d'OpenMP*, (Accessed: May 2022). Available at https://www.openmp.org/. (Cited on pages 42 and 79.)

[PCL 022] PCL. *The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing.*, (Accedit: Maig 2022). Disponible a https://pointclouds.org/. (Cited on page 18.)

[Python 022] Python. *python*, (Accedit: Abril 2022). Disponible a https://www.python.org/. (Cited on page 18.)

[ROS] Open Robotics ROS. *ROS - Robot Operating System (Instal·lació en Ubuntu)*. Disponible a http://wiki.ros.org/noetic/Installation/Ubuntu/. (Cited on page 18.)

[Stoyanov 2012] T. Stoyanov, M. Magnusson, H Andreasson and A. J. Lilienthal. *Fast and accurate scan registration through minimization of*

the distance between compact 3D NDT representations. The International Journal of Robotics Research, vol. 31, pages 1377–1393, 2012. (Cited on page 12.)

[SublimeText 022] SublimeText. *sublime text*, (Accedit: Abril 2022). Disponible a https://www.sublimetext.com/. (Cited on page 18.)

[VisualSC 022] VisualSC. *visual studio code*, (Accedit: Abril 2022). Disponible a https://code.visualstudio.com/. (Cited on pages 17 and 18.)

Annex A: Repositori

En aquest annex, s'afegeix tan l'enllaç que porta cap al repositori que conté tot el codi del projecte com l'enllaç de la documentació que s'ha generat amb Doxygen.

Un cop dins, es podrà veure tota la informació referent a aquest repositori, les diferents branques, còmits realitzats, modificacions fetes i el mateix codi.

Pàgina de Bitbucket on es troba guardat el repositori: NDT_ScanMatching_CPP

Dins del Bitbucket es pot escollir quina branca es vol mirar, la part desenvolupada en el meu treball es troba en 2 branques.

- Branca anomenada 'part_conjunta', en aquesta s'ha desat tot el codi base per al funcionament de la llibreria.
- Branca anomenada 'master', en aquesta es troba tota la part implementada individualment.

Enllaç web per a accedir a la documentació generada del projecte amb Doxygen: scanmatchingdoxygentfg.bitbucket.io

Annex B: Resultats

En aquesta carpeta de drive, es troben els resultats obtinguts de les diferents proves fetes. Hi ha 4 carpetes amb diferents resultats.

- **Chi quadrat**: Hi ha 2 subcarpetes pels 2 entorns diferents on s'han recollit les dades, on trobem una primera imatge amb els temps que triguen els mètodes de registre amb i sense optimització. Una altra amb la diferència en la transformació optima. I un seguit de gràfics amb el procés d'optimització per cada parella d'escanejos.
- **Estadístiques Front-End GMM 2D**: Trobem tres gràfics sobre els 2 entorns diferents i la combinació d'aquests, per veure les relacions entre el núvol de punts original, el numero de components generades i el temps d'execució.
- **Gràfics GMM 3D**: En aquesta hi ha diferents subcarpetes que representen diferents GMM generats amb el nuvol de punts que diu el primer número de la subcarpeta, els que tenen afegit '_4m' es perquè s'ha establert un tamany de cel·la a la graella de 4 metres. Dins d'aquesta subcarpeta es troben els gràfics de densitat del GMM al llarg dels 3 eixos separats en 20 capes, també trobem els dos arxius de text per generar el gràfic 3D que es veu al vídeo amb un script de Python.
- **Test paral·lelització**: Aquí es troben les probes de paral·lelització comentades a la secció 10.2.