FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Robot Self-Localization in Dynamic Environments

Carlos Miguel Correia da Costa

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Armando Jorge Miranda de Sousa (Ph.D.)

Second Supervisor: Germano Manuel Correia dos Santos Veiga (Ph.D.)

January 26, 2015

Robot Self-Localization in Dynamic Environments

Carlos Miguel Correia da Costa

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Name of the President
External Examiner: Doctor Name of the Examiner
Supervisor: Doctor Name of the Supervisor

January 26, 2015

Abstract

Mobile robot platforms capable of operating safely and accurately in dynamic environments can have a multitude of applications, ranging from simple delivery tasks to advanced assembly operations. These abilities rely heavily on a robust navigation stack, which requires a stable and accurate localization system.

This dissertation describes an efficient, modular, extensible and easy to configure 3/6 Degrees of Freedom (DoF) localization system, capable of operating on a wide range of mobile robot platforms and environments. It is able to reliably estimate the global position using feature matching and is capable of achieving high accuracy pose tracking using point cloud registration algorithms. It can use several point cloud sensing devices (such as LIght Detection And Ranging (LIDAR) or RGB-D cameras) and requires no artificial landmarks. Moreover, it can update the localization map at runtime and dynamically adjust its operation rate based on the predicted robot velocity in order to use the minimum amount of hardware resources. It also offers a detailed analysis of each pose estimation, providing information about the percentage of registered inliers, the root mean square error of the inliers, the angular distribution of the inliers and outliers, the pose corrections that were performed in relation to the expected position and in case of initial pose estimation it also gives the distribution of the acceptable initial poses, which can be very valuable information for a navigation supervisor when the robot is in ambiguous areas that are very similar in different parts of the known environment.

The Robot Operating System (ROS) implementation was tested in several dynamic indoor environments using two mobile robot platforms equipped with LIDARs and RGB-D cameras. Overall tests using sensor data from simulation and retrieved from the robot platforms performed in a high end laptop with an Intel Core i7 3630QM processor, 16GB DDR3 of memory and NVIDIA GTX680M graphics card, demonstrated high accuracy in complex dynamic environments, with less than 1 cm in translation error and less than 1 degree in rotation error. Execution times ranged from 5 to 30 milliseconds in a 3 DoF setup and from 50 to 150 milliseconds in a full dynamic 6 DoF configuration.

The sub centimeter accuracy achieved by the proposed localization system along with the dynamic map update capability and the need of no artificial landmarks will allow the fast deployment of mobile robot platforms capable of operating safely and accurately in cluttered environments. Moreover, the resilience to dynamic objects will grant the possibility to use robots as coworkers, helping humans perform their work more efficiently and thus reducing the overall production costs.

Resumo

Plataformas móveis robóticas capazes de operar com precisão e de forma segura em ambientes dinâmicos têm um alargado espetro de aplicações, desde simples entregas de objetos até operações complexas de montagem. Para atingir estes requisitos de operação é necessário um sistema de navegação robusto, que por sua vez requer um módulo de localização preciso e estável.

Esta dissertação descreve um sistema de localização para 3/6 graus de liberdade (DoF), que é eficiente, modular, extensível e fácil de configurar, capaz de operar num alargado conjunto de plataformas móveis e ambientes. É capaz de estimar a posição inicial de um robô usando métodos de associação de características geométricas e consegue seguir a sua pose com alta precisão através de algoritmos de registo de nuvens de pontos. A sua implementação consegue tirar partido de vários sensores laser e câmaras RGB-D e não necessita de marcadores artificiais ou modificação do ambiente. Possui ainda a capacidade de atualizar o mapa incrementalmente e ajustar a sua frequência de funcionamento de acordo com a velocidade do robô de forma a usar o mínimo de recursos computacionais possível. Para facilitar a avaliação da qualidade da localização para operações críticas, cada estimativa da pose do robô é acompanhada com a análise do registo da nuvem de pontos, contendo informação acerca da percentagem de pontos corretamente registados, a raiz quadrada do erro quadrático médio, a distribuição angular dos pontos classificados como pertencentes e não pertencentes ao mapa de referência, as correções aplicadas à estimativa da pose e no caso de ser efetuada localização global também é disponibilizada a distribuição das poses iniciais aceitáveis, o que pode ser informação bastante útil para um supervisor de navegação quando o robô está em posições ambíguas do ambiente nas quais existe geometria semelhante em sítios diferentes do mapa.

A implementação em ROS foi testada em vários ambientes dinâmicos recorrendo a duas plataformas móveis equipadas com LIDARs e câmaras RGB-D. Os resultados obtidos usando dados de simulação e recolhidos das plataformas robóticas realizados num portátil com CPU Intel Core i7 3630QM, 16GB DDR3 de memória e placa gráfica NVIDIA GTX680M demonstraram que o sistema consegue fazer a estimativa da pose do robô com um erro de translação inferior a 1 centímetro e um erro de rotação abaixo de 1 grau. Os tempos de execução oscilaram entre 5 e 30 milissegundos para uma configuração 3 DoF e entre 50 e 150 milissegundos para 6 DoF.

A alta precisão disponibilizada pelo sistema de localização proposto em conjunto com a sua capacidade para atualizar o mapa incrementalmente e de não necessitar marcadores artificiais, irá permitir o desenvolvimento de plataformas robóticas móveis capazes de operar em ambientes não estruturados. Por outro lado, a sua robustez em relação a objetos dinâmicos abre a possibilidade dos robôs colaborarem com humanos para melhorar a produtividade global de uma dada tarefa e assim reduzir os custos de produção.

Acknowledgments

I would like to express my gratitude to my supervisors and the INESC team, for their helpful contributions. Their experience and expertise significantly improved the quality of this dissertation.

I am also very grateful for the knowledge and experiences that I gather from my friends, teachers and colleagues over the years and for the brilliant work developed by the ROS and PCL community.

And of course, to my family, for their continuous support.

Carlos Miguel Correia da Costa

"Perfection is achieved, not when there is nothing more to add,
but when there is nothing left to take away."

Antoine de Saint-Exupéry

Contents

List of Figures

LIST OF FIGURES

List of Tables

List of Algorithms

Abbreviations

AGPS Assisted Global Positioning System.
API Application Programming Interface.

CAD Computer Aided Design.
CARLoS Cooperative Autonomous Robot for Large Open Spaces.

DoF Degrees of Freedom.

EKF Extended Kalman Filter.

GLONASS GLObalnaya NAvigatsionnaya Sputnikovaya Sistema.
GNSS Global Navigation Satellite System.
GPS Global Positioning System.

ICP Iterative Closest Point.
IP Internet Protocol.

LIDAR LIght Detection And Ranging.

MCL Monte Carlo Localization.

PCL Point Cloud Library.

RADAR RAdio Detection And Ranging.
ROS Robot Operating System.

SLAM Simultaneous Localization And Mapping.
SONAR SOund Navigation And Ranging.

TCP Transmission Control Protocol.
ToA Time of Arrival.
ToF Time of Flight.
TTFF Time To First Fix.

UDP User Datagram Protocol.
UKF Unscented Kalman Filter.

XML Extensible Markup Language.
XML-RPC Extensible Markup Language Remote Procedure Call.

Chapter 1

Introduction

This chapter provides an overview about the motivations and objectives of this dissertation along with its practical applications.

1.1 Context

Humanity has sought a reliable method of navigation ever since it started to explore the world. It began with simple landmark reference points for local travels, then perfected celestial navigation for global journeys, and when it finally conquered space, it deployed a global system for high accuracy localization. Autonomous robots face the same problem, because in order to be able to navigate with precision, they first need to know their location.

Over the years, several localization methods have been proposed and refined, according to the navigation environment and the accuracy requirements. Some are meant for high precision local navigation, while others provide an approximate global position.

A robot capable of operating safely and accurately in a dynamic environment can have innumerous applications, ranging from simple delivery tasks to advanced assembly. Besides improving productivity by performing repetitive tasks with precision and speed, robots can also act as coworkers, helping humans perform their jobs more efficiently and thus, reducing the overall production costs.

1.2 Project

Cooperative Autonomous Robot for Large Open Spaces (CARLoS)[1] is a European research project that aims to develop an autonomous robot capable of performing repetitive tasks alongside human co-workers in dynamic environments. The robot will operate in shipyards and is intended to perform fit-out operations, such as stud welding and projection mapping of Computer Aided Design

[1] http://carlosproject.eu/

(CAD) drawings. Stud welding is a repetitive task that provides structural support for other components, such as heat insulation layers or electrical systems. Projection mapping of CAD drawings or other important information will help human co-workers assemble components faster, because it will mark the exact position in which they must be installed.

1.3 Motivation and objectives

With the increase of competitiveness in the current globalized trading markets, companies are trying to reduce production costs and improve the productivity of their assets. Robots can help achieve these goals by performing the simple and repetitive jobs while giving humans more free time to perform the complex and creative tasks.

Mobile platforms equipped with robotic arms provide a flexible way to automate a wide range of tasks that must be performed over large areas. However, before performing the intended operations they first need to know where they are and how they can reach the desired location. Moreover, given their limited computational resources and energy storages, they require efficient, reliable and accurate control systems capable to operate in real time.

1.4 Contributions

This dissertation introduces an efficient, modular and extensible 3/6 DoF localization system for mobile robot platforms capable of operating accurately and reliably in dynamic environments. It is a multi-level registration pipeline that uses geometric features to estimate the initial position of a robot platform and point cloud registration algorithms to track its pose. The tracking subsystem can have two different configurations. One tuned for maximum efficiency used for the normal operation of the mobile platform and another for unlikely situations that may require more robust registration algorithms / configurations. It also supports incremental map update and can adjust its operation rate based on the estimated robot velocity. For critical operations, it provides a detailed analysis of the tracking quality and when initial pose estimation is required it gives the distribution of the acceptable poses, which can be very valuable information if there are several areas in the known map with very similar geometry.

1.5 Dissertation outline

The remaining of this dissertation is split over 5 chapters. Chapter 2 provides an overview of the main localization methods available for mobile robot platforms. Chapter 3 introduces the main frameworks used to implement the self-localization system. Chapter 4 details the architecture and ROS implementation of the proposed 3/6 DoF self-localization system. Chapter 5 evaluates the results achieved with the localization system in several testing environments. Finally, chapter 6 presents the conclusions of this dissertation and suggestions for future work.

Chapter 2

Localization methods

Self-localization is critical to any autonomous robot that must navigate the environment and requires the calculation of a 3/6 DoF pose in a given world coordinate system.

Over the years, several approaches were developed according to the precision requirements and the environment in which the robot is designed to operate. Some require support systems to calculate the position, while others are completely autonomous, allowing the robot to localize itself without any outside dependencies. Also, some systems have limited range while others can only be effective in open spaces. Moreover, several of these methods were designed to cope with errors in the localization sensors and tolerate temporary malfunctions.

The following sections will introduce some of the most used self-localization systems that can be employed in the estimation of a robot's pose.

2.1 Proprioceptive methods

Proprioceptive methods rely on internal information that the robot possesses about its own systems operation and movement in order to update the current pose estimation. As a result, they allow the robot to operate without an external support system.

Since these methods don't correct their estimations based on environment observations, they are bound to have significant cumulative errors. As such, in order to maintain an accurate estimation of a robot's pose, they are usually combined with exteroceptive systems.

2.1.1 Odometry

Odometry estimates the current pose by integrating the velocity of the robot over time. This velocity is usually calculated by measuring the number of rotations of the wheels (using optical encoders like the ones shown in figure 2.1). This method can provide a viable approximated location, as can be seen in [RKZ13].

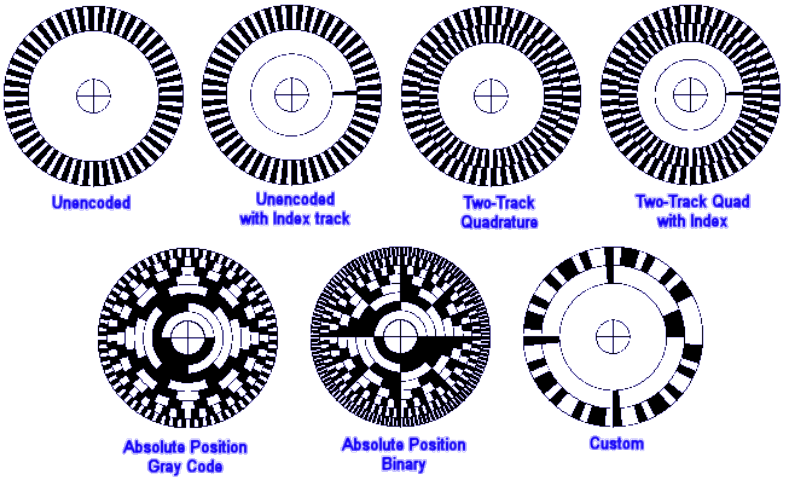

Figure 2.1: Different types of optical encoders used to measure distances[1]

2.1.2 Dead reckoning

Dead reckoning is an extension of the odometry method, in which the acceleration and angular velocity are used to improve the localization estimations.

Other sensors, such as accelerometers and gyroscopes, can also be used to improve the position estimation [Ibr10] and provide the robot orientation.

2.2 Exteroceptive methods

Exteroceptive methods use a range of sensors to analyze the environment and retrieve the necessary information to perform localization.

2.2.1 Time of Flight methods

Time of Flight (ToF) or Time of Arrival (ToA) methods can be used to calculate distances based on the amount of time that a given wave takes from the moment it is created to the moment it is received. This allows the construction of a 3D representation of the world that can be used in conjunction with geometric methods to estimate the pose of a robot.

Since these systems rely on active interaction with the environment, they can be used without being affected by lighting interferences. Nevertheless, they should take in consideration the conditions in which the waves propagate and also the geometry of the environment, because it can affect the path that the waves take, and as a result, can lead to the decrease of precision in the measurements.

[1] http://www.mindspring.com/~tom2000/Delphi/Codewheel.html

2.2.1.1 Light waves

Light waves generated with lasers can estimate distances with high accuracy. Systems like LIDAR take advantage of this fact and can be used to calculate a very detailed 3D point cloud of the environment (like the one showed in figure 2.2).

Given the high velocity in which light propagates, these methods allow the mapping of the environment with low latency, which can be a critical requirement in robots that must react very fast to changes in their surroundings, such as autonomous cars [MCB10].

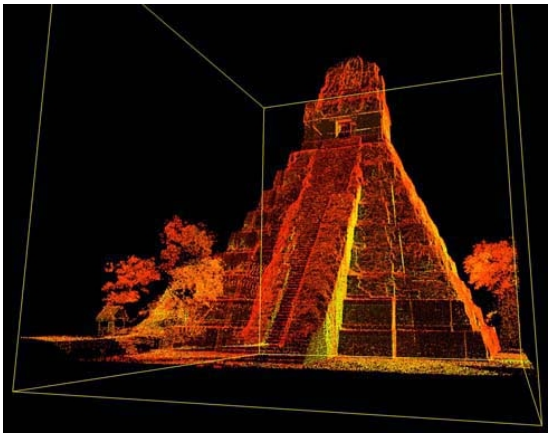

Figure 2.2: LIDAR scan[2]

2.2.1.2 Radio waves

Similar to LIDAR, radio waves can be used to calculate distances using the ToF technique. Systems like RAdio Detection And Ranging (RADAR) provide an effective way to calculate the distance, altitude, direction and speed of objects that can be used as landmarks in navigation.

Like any electromagnetic wave localization method, it must take in consideration ambient interferences and even jamming, in order to validate the obtained measures. Moreover, some types of materials with a given geometric configuration might be invisible to RADAR, and as such, critical localization systems might have to employ some additional techniques to ensure the correct mapping of the robot surroundings.

Since RADAR has a less focused beam than LIDAR, it can have considerable less accuracy, as can be seen in figure 2.3. Nevertheless, it can be an effective method to avoid obstacles [WY07].

2.2.1.3 Sound waves

Other localization approach that can be used to map under water environments relies in the acoustic analysis of the reflections of sounds in the surrounding objects.

Like the previous methods, SOund Navigation And Ranging (SONAR) can actively scan the environment to calculate the locations of the objects using a ToF technique (as can be seen in figure 2.4). Although this method is usually applied to underwater mapping, it can also be used in other sound propagation environments, such as air [GWG13].

2.2.2 Trilateration methods

Trilateration is a geometric technique that can be employed in the calculation of absolute or relative positions using distances from known points.

[2] http://blogs.scientificamerican.com/cocktail-party-physics/2012/03/12/l-is-for-lidar/

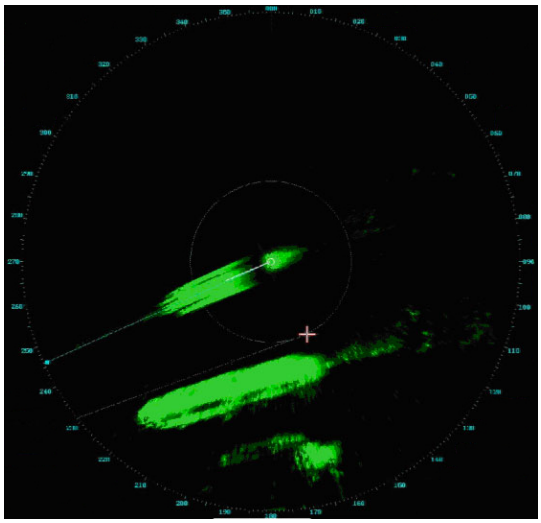

Figure 2.3: RADAR scan of two ships[3]

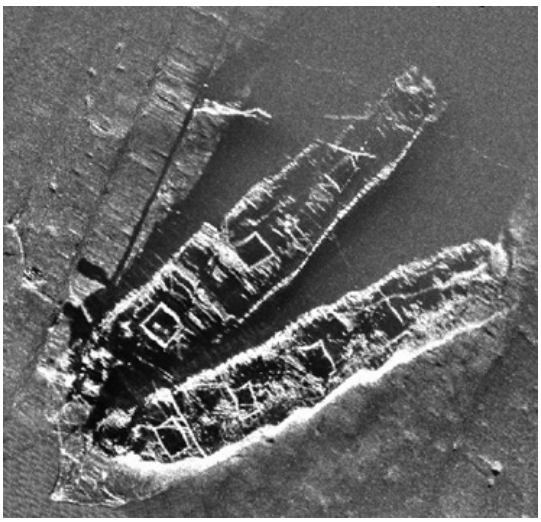

Figure 2.4: SONAR scan of two ships[4]

For 3D localization, it usually involves the intersection of 4 or more spheres, in which their radius is the distance to known positions.

2.2.2.1 Global Navigation Satellite System (GNSS)

Global Navigation Satellite System (GNSS) such as the Global Positioning System (GPS) or GLObalnaya NAvigatsionnaya Sputnikovaya Sistema (GLONASS), allow 3D positioning in the planet Earth using a trilateration method [DKU+07].

In these systems, a constellation of satellites broadcasts a radio signal with information about its position along with the time of the message dispatch. Using this data and knowing the speed of the radio waves, the distances to the satellites can be computed.

With at least 3 satellites distances, the 3D position can be calculated, since the Earth can be used as the 4^{th} sphere (figure 2.5 shows a visual representation of the trilateration technique used in a global localization system).

Given that the correct measurement of the distances to the satellites relies in the accurate computation of the elapsed time between the message dispatch and its reception, it is critical that both the satellites and the receiver have synchronized clocks. This is achieved by using high precision atomic clocks in the satellites and a clock reset technique in the receiver. This reset method relies on the fact that 3 satellite distances will only have a valid location if the clock of the receiver is synchronized. With this knowledge, the receiver can compute the necessary corrections to reset its internal clock to match the satellites time.

[3] http://www.sintef.no/Projectweb/STSOps/News/Operational-Aspects-on-Decision-making-in-STS-Lightering

[4] http://stellwagen.noaa.gov/maritime/palmercrary.html

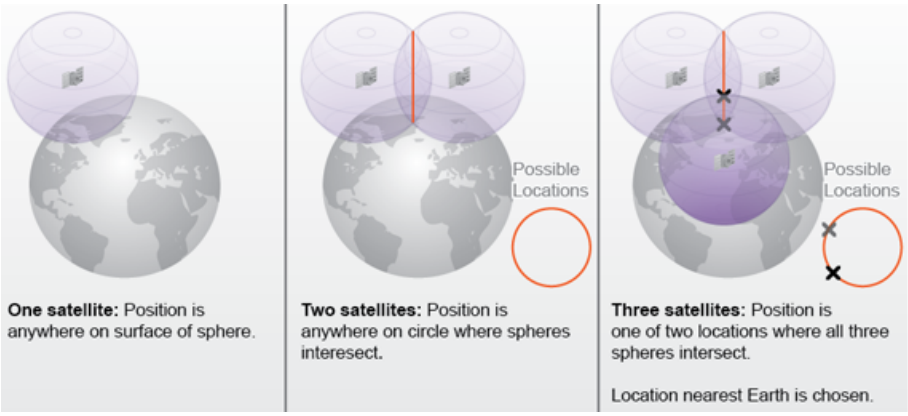

Figure 2.5: Trilateration technique in a global localization system[5]

2.2.2.2 Differential Global Positioning System (DGPS)

The accuracy of the GPS position can be increased with the help of local broadcast stations. These stations are fixed and provide information about the corrections that can be made to the satellite signals in order to improve the localization precision.

These corrections are useful to mitigate some of the ambient interference that the satellite signals face. These interferences can range from simple signal reflection in the environment landscape to the more complex interactions with the atmosphere, which can change the speed and path of the radio signals.

The computation of the corrections [KKOO07] is based on the fact that these stations are fixed, and as such, they can compare the location given by the satellite signals with their known location. With this position differential, the appropriate corrections can be calculated and broadcasted to the GPS receivers.

2.2.2.3 Assisted Global Positioning System (AGPS)

Assisted Global Positioning System (AGPS) systems are a common method used to speed up the Time To First Fix (TTFF) of a GPS receiver. They usually rely on the cellphone network to provide location estimation and signal corrections [R.48]. This information can greatly reduce the TTFF when there are few satellites visible or their signal is very weak and only temporary available.

2.2.2.4 Signal strength geolocation methods

Signal strength geolocation [CK02], also known as fingerprinting localization [BOG+10], is an approximate method that can be used to calculate relative positions.

It relies on the analysis of the signal attenuation from a given access point (like a Wi-Fi router or cellphone tower), to estimate distances. With enough access points (usually 4), an approximate position can be computed.

[5] https://www.e-education.psu.edu/geog160/node/1923

This type of distance estimation can be useful for indoor navigation, but requires a propagation model of the signal and the environment. If these models aren't accurate, then the localization precision of these methods will be very low.

Although this method is less accurate than the more recent global localization systems (such as GPS), it can be used without human made infrastructures, and as such, is a viable solution in case of temporary disruption of the GPS signal.

2.2.3 Celestial navigation

Celestial navigation [YXL11] relies on the observation of stars, planets or other reference objects, to calculate the latitude and longitude.

The calculation of the position on the surface of the Earth using celestial navigation is similar to trilateration, but in this case, angles are used instead of distances. These angles (delta), are measured between the Earth horizon and the center of the celestial object.

Having the delta, and knowing the relative position of the Earth to the reference object, along with the Greenwich hour time, it is possible to calculate a circle of position (as shown in figure 2.6).

Having at least 3 circles of position, the latitude and longitude can be computed.

Although this method is less accurate than the more recent global localization systems (such as GPS), it can be used without human made infrastructures, and as such, is a viable solution in case of temporary disruption of the GPS signal.

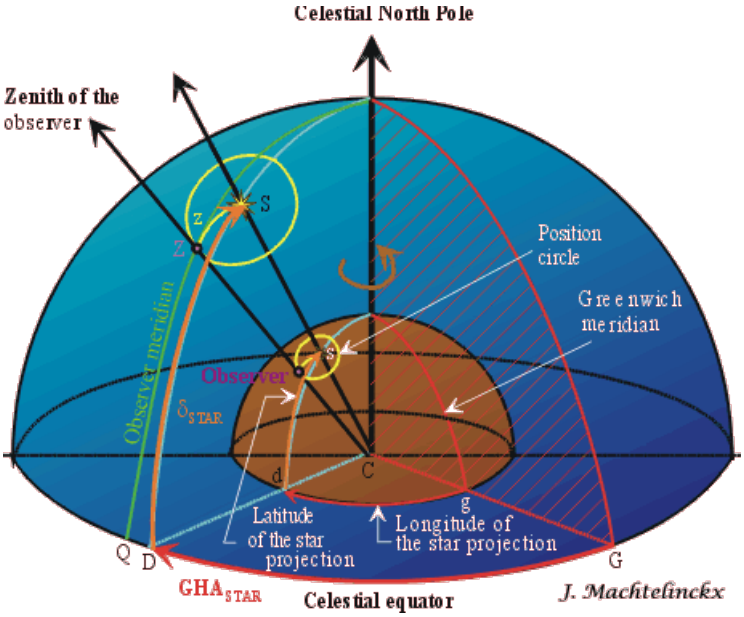

Figure 2.6: Circle of position in celestial navigation[6]

[6] http://onboardintelligence.com/CelestialNav/Celnav2.aspx

2.2.4 Landmark methods

Landmark methods [LKP06] can be used to perform relative localization, and are very useful to reduce the required information for navigation.

In these methods a database of markers / environment geometry is stored along with its location, and when the robot recognizes one of these markers, it corrects its proprioceptive methods measures.

It is a simplification of the method that will be presented in the next section, and it is useful for environments that have unique geometry in key positions of the navigation map.

2.2.5 Point cloud methods

Point cloud localization methods can be used to perform relative localization by finding the best point cloud match between the environment and the know map (figure 2.7 shows its application to small objects). These methods require a 2D or 3D representation of the environment and tend to be used in conjunction with proprioceptive methods (to have an estimation of movement), and also with probabilistic methods (when the point cloud acquisition location is not known).

One of the most used algorithms for 3D point cloud matching is the Iterative Closest Point (ICP) [BM92, Jez08, Zha94, BTP13, CSSK02, DZA+13, ZL11]. It is an iterative algorithm that finds the translation and rotation transformations that minimizes the distances of the corresponding points on both clouds.

There are several variants that optimize different parts of the algorithm [RL01].

The main steps for each iteration of ICP algorithm are presented below.

1. Selection of points in one or both point clouds (source and reference clouds)
2. Matching / pairing source points to reference points
3. Weighting the corresponding pairs
4. Rejecting low quality matches (outliers)
5. Assigning an error metric based on the point pairs
   (a) Usually mean square error based on points distance

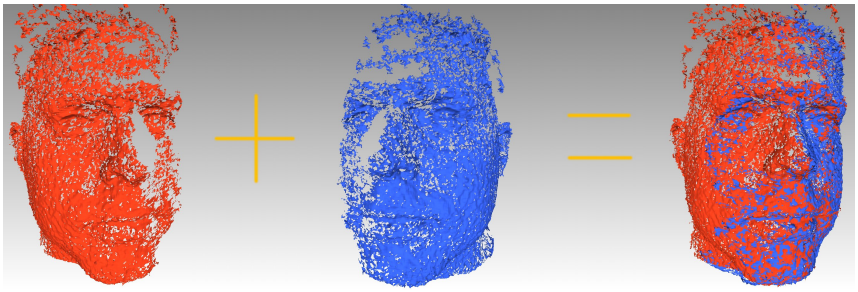

Figure 2.7: ICP point cloud matching[7]

[7] http://dynface4d.isr.uc.pt/database.php

2.2.6 Probabilistic methods

Probabilistic methods aim to reduce the impact of sensor accumulated errors or even temporary malfunctions by using Bayesian estimations and Markov processes.

2.2.6.1 Monte Carlo Localization (MCL)

Monte Carlo Localization (MCL) (also known as particle filter), is a global localization algorithm that estimates the position and orientation of a robot by analyzing and adjusting the distribution and weights of state particles on a given environment [BOG+10, AMGC02, BGFM10, Che03, FBDT99, SS09].

It starts by randomly distributing the state particles on the map, and over time it changes their position and weight according to new sensor readings. The probable location of the robot will be in the area of the map that has the largest cluster of state particles. The figures below show the evolution of the state particles distribution during the robot movement in the environment, and illustrates how the new sensor readings changed the particles clusters[8].

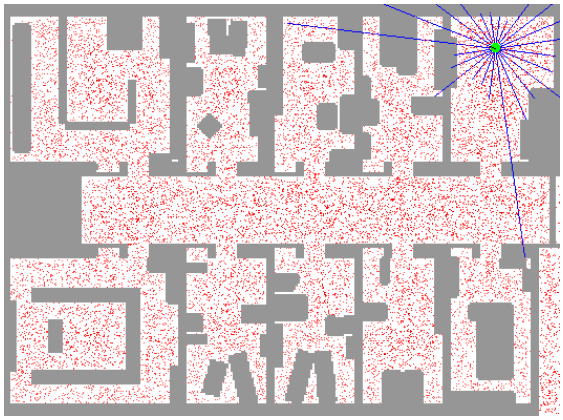

Figure 2.8: MCL particle distribution animation

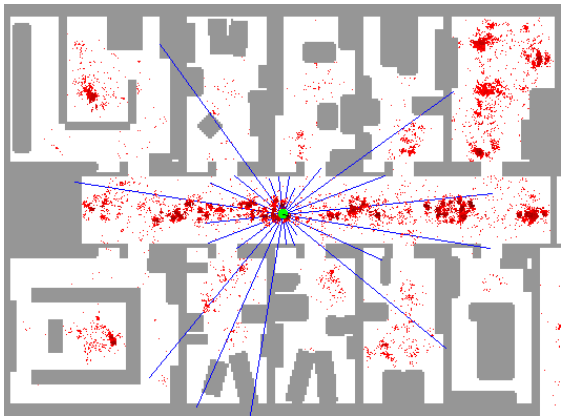

Figure 2.9: MCL redistribution of particles

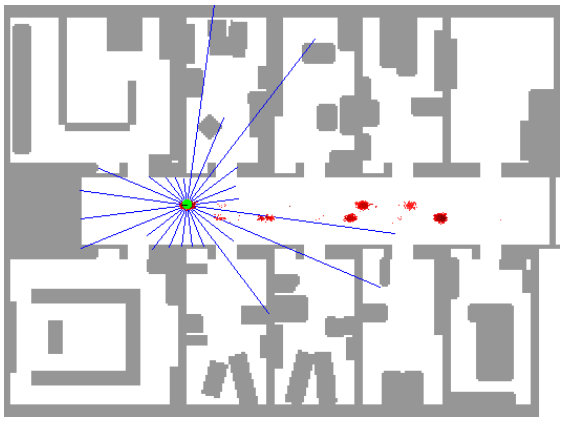

Figure 2.10: MCL position refinement

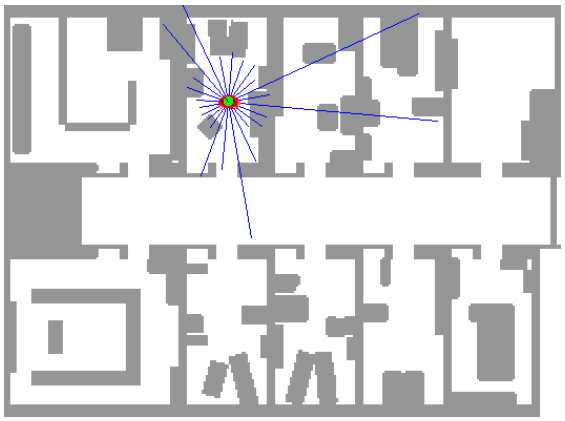

Figure 2.11: MCL position estimation

[8] http://www.cs.washington.edu/robotics/mcl/

2.2.6.2 Kalman filters

Kalman filters [Kal60] are probabilistic algorithms that estimate a given system state even when it is affected by noise or other errors. They perform linear quadratic estimations to achieve optimal results and can be efficiently implemented to be used in real time systems. They are recursive algorithms based on Marcov processes, and as a result, they only need to know the current system state in order to perform measurement corrections.

The Extended Kalman Filter (EKF) [EW99, Rib04, IKP10, LYL11] is a variant of the Kalman filter, designed to handle non-linear systems by performing linear approximations to the state variables. These approximations may lead to divergence in the estimations, and as such, the EKF can't guarantee optimal results.

The Unscented Kalman Filter (UKF) [JU97, WV02] is another variant of the Kalman filter that was designed for highly non-linear systems. It usually achieves better results than EKF due to its unscented transform.

For the particular case of localization, these algorithms start with an initial estimation of the system state, and for each new position (computed from the sensors data), they predict the estimated robot location (according to the Bayes estimation model and the Gaussian distribution of errors), and then update their internal model of the system (mean and covariance) to incorporate the system evolution.

In figure 2.12 can be seen that the UKF estimated position (red) is closer to the real position (blue) than the raw sensor measurements (green).

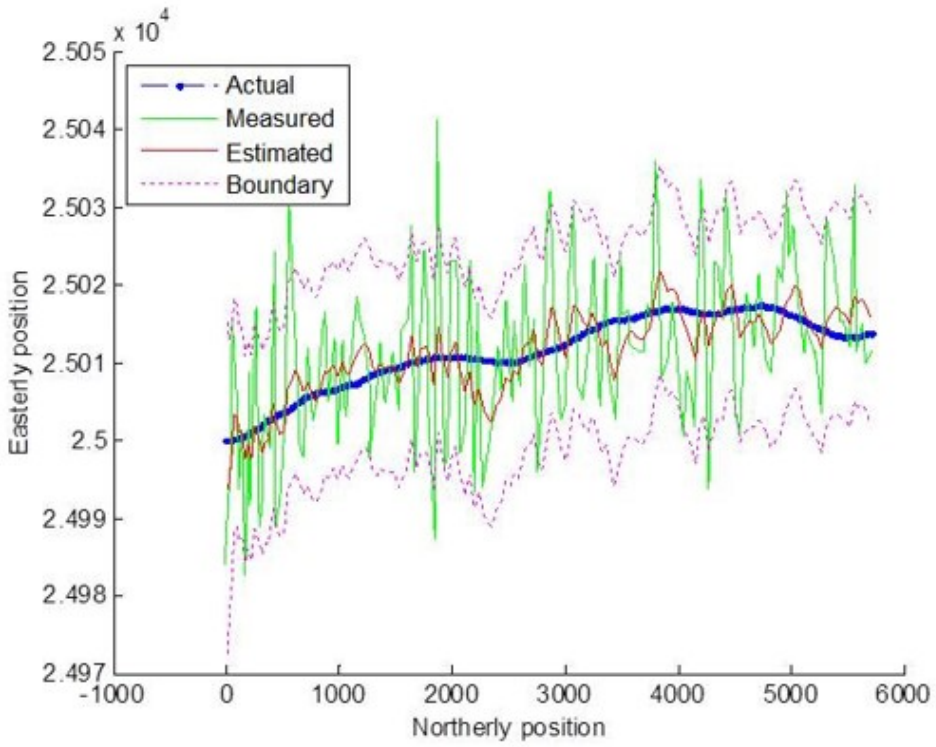

Figure 2.12: Unscented Kalman Filter[9]

2.2.6.3 Perfect Match

The Perfect Match [LLRM06, PM63] is an efficient self-localization algorithm that is largely used in the Robocup Robotic Soccer Mid Size League. Its main goal is to minimize the localization error by carefully analyzing the know map and selecting the most probable current position using a gradient descent approach. To improve tracking accuracy, the algorithm also uses a stochastic weighted approach.

With the proper configuration, it can achieve a localization accuracy similar to the particle filter, while using about ten times less computations.

An example of the position estimation can be seen in the figures below. Figure 2.13 shows the probable locations when the robot detects a line on the floor, and figure 2.14 illustrates their associated errors (brighter areas indicate smaller error). By using a gradient descent, the most probable location was selected (black circle).

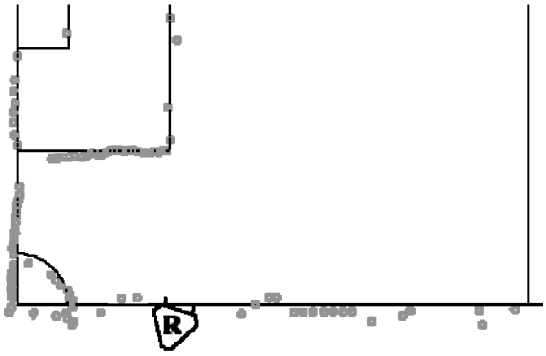

Figure 2.13: Position estimates

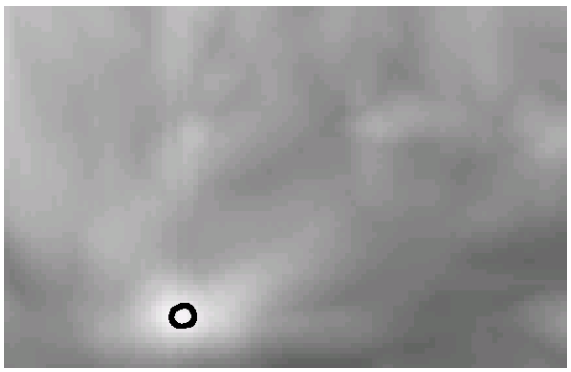

Figure 2.14: Positions associated error

2.2.7 Simultaneous Localization And Mapping (SLAM)

Simultaneous Localization And Mapping (SLAM) [Thr02] is a very effective approach to either explore unknown environments or update a known map. It relies on both proprioceptive and exteroceptive methods to perform localization and mapping. It is also very useful to make the robot navigation more robust in dynamic environments, in which the topology of the world may change considerably over time.

There are numerous approaches to perform SLAM [TGCV12]. Some optimized for exploration and others for map improvement. For the exploration tasks, proprioceptive methods play a critical role, and are usually paired with probabilistic methods, such as Kalman filters, in order to reliably map the environment. For map correction and improvement, several probabilistic methods can be employed according to the precision required. For high accuracy 3D mapping, the ICP algorithm can be used to build accurate point clouds of the environment.

[9] http://www.lzcheng.com/courseworks/kalmanfilter

2.3 Summary

This chapter introduced several localization systems that can be used in different types of environments and with different degrees of accuracy. It started with the simple proprioceptive methods and then moved on to the more robust exteroceptive approaches.

Some techniques can be combined to improve the pose estimate or to make the localization more efficient to a particular type of environment.

For outside tasks, a GNSS approach can give accurate localization estimations with very little computation cost. On the other hand, indoor localization requires more advanced techniques in order to infer the current position based on the analysis of the robot surroundings. These techniques usually start with an estimation of the robot movement and then refine it with probabilistic or geometric methods.

Table 2.1 presents an overview of the mentioned localization techniques.

Table 2.1: Overview of self-localization approaches

Method	*Ideal environment*	*Accuracy*	*Operational cost*	*Computational cost*	*Notes*
Odometry	Any	Low	Low	Low	Position estimation is affected by cumulative errors
Dead reckoning	Any	Low	Low	Low	Pose estimation is affected by cumulative errors
LIDAR	Any	High	Medium	High	Needs an auxiliary method to perform global localization
RADAR	Any	Medium	High	High	Needs an auxiliary method to perform global localization
SONAR	Any	Medium	Medium	High	Needs an auxiliary method to perform global localization
GNSS	Outside	Medium	Low	Low	Requires a clear line of sight to at least 3 satellites
Signal strength	Any	Low	Low	Low	Requires an accurate model for the signal attenuation
Celestial	Outside	Low	Low	Low	Requires a clear view of the celestial objects and a nautical almanac
Landmark	Any	Medium	Low	Medium	Requires a database of landmarks
ICP	Indoors	High	Medium	High	Requires a detailed 3D representation of the environment
MCL	Indoors	Medium	Medium	Medium	Inefficient for large areas
Kalman filters	Any	Medium	Low	Low	Useful to improve estimations of other methods
Perfect Match	Any	Medium	Low	Low	Not ideal for large or dynamic environments
SLAM	Indoors	Medium	Medium	High	Adapts well to dynamic environments

Chapter 3

Relevant software technologies

Robot self-localization in complex environments is a multidisciplinary problem that requires advanced computer software systems. In order to speed up the implementation and deployment of the localization system, several frameworks and libraries were used. Among the most important were ROS for the system architecture, Point Cloud Library (PCL) for the point cloud processing and Gazebo for simulation and testing.

3.1 ROS

ROS[1] [QC09] is a software framework designed to ease the development of robot systems. It provides seamless integration between hardware drivers and software modules, allowing a fast transition between simulation and deployment.

Figure 3.1: ROS logo

It's an open source project that offers a distributed computing framework with several core libraries and development tools that aims to speedup software prototyping, testing and deployment.

3.1.1 Architecture

The ROS architecture was designed from the beginning to be a distributed peer-to-peer software framework that could be deployed in several operating systems and implemented in a range of different programming languages. However, given that most of the ROS community prefers open source software, the Ubuntu[2]] operating system is the main developing and testing environment, and as such, the recommended choice for ROS developers. Moreover, considering that robotics

[1] http://www.ros.org/
[2] http://www.ubuntu.com/

research requires software with both performance and maintainability at its core, the C++[3] programming language is used in most of the available packages, along with Python[4] and Java[5].

Being a distributed computing framework, ROS relies in network connections and exchange of messages to perform the intended tasks. As such, its architecture was developed to follow a publish / subscribe pattern (ROS topics[6]) and request and reply communication paradigm (ROS services[7] and actions[8]). This allows ROS nodes[9] (operating system processes) to be deployed in different computing platforms with ease and simplifies testing and exchange of software modules.

The next sections provide a more detailed description of the main ROS architecture concepts.

3.1.1.1 Nodes

ROS nodes are operating system processes that are part of the peer-to-peer communication graph. They are the fundamental building blocks of any ROS system and can be spread among several computing platforms.

In order to manage the communications between nodes, the ROS framework provides a master node (roscore[10]) that uses Extensible Markup Language Remote Procedure Call (XML-RPC) to maintain a communication graph of the system. This allows nodes to be started without knowing the location (Internet Protocol (IP) address and port) of the other nodes in the network and greatly simplifies their integration and exchange. Also, by using ROS launch files[11] (Extensible Markup Language (XML) configuration files), the specification of a system communication data flow and configuration can be easily altered.

Besides handling communications, the master node also manages system configurations through the parameter server. This allows nodes to share and change the system configuration at runtime. However, given that the parameter server is usually queried only when a node starts up, the dynamic reconfigure[12] Application Programming Interface (API) can be used instead, if it is necessary to change the configuration of a node when it is already running. This is achieved by providing a callback that is asynchronously called when a configuration change is requested.

The flexibility provided by ROS in both module integration and exchange can greatly speedup testing and deployment and the possibility of changing the configuration of the system at runtime and restart software modules individually (nodes) is very useful when implementing system supervisors and recovery behaviors.

[3] http://www.cplusplus.com/
[4] https://www.python.org/
[5] https://www.java.com
[6] http://wiki.ros.org/Topics
[7] http://wiki.ros.org/Services
[8] http://wiki.ros.org/actionlib
[9] http://wiki.ros.org/Nodes
[10] http://wiki.ros.org/roscore
[11] http://wiki.ros.org/roslaunch
[12] http://wiki.ros.org/dynamic_reconfigure

3.1.1.2 Nodelets

A nodelet[13] is a special kind of node that aims to reduce the overhead of message exchange. This overhead can be significant when messages are very large, such as point clouds or video. To mitigate this problem, the nodelets exchange pointers to shared memory regions, instead of sending the entire messages between nodes.

To achieve this overhead reduction, some architecture changes are required. The most important being the use of threads instead of processes, and the creation of a superclass that all nodelets must inherit. This leads to the creation of plugin libraries for each nodelet instead of an executable for each node.

Another important change is the introduction of nodelet managers to allow loading and setup of nodelets in different threads inside the same process.

In terms of implementation, the transition from nodes to nodelets requires few code changes and can lead to a significant improvement of the overall system performance.

3.1.1.3 Topics

ROS topics are named communication buses that follow the publish / subscribe pattern. They provide a simple method for exchanging messages between nodes and allow the decoupling of information production and consumption. This is useful when there are multiple sources of the same information or there are multiple consumers that are interested in processing the same data for different purposes. Moreover, this communication architecture allows to log and replay the exchanged messages, which can be helpful to test different algorithms with the same data.

Currently, topics can use either the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) protocols to exchange messages. The TCP implementation is used by default and creates a bidirectional channel between each producer and subscriber while guaranteeing the delivery of all messages. The UDP implementation uses an unreliable and stateless transport approach, in which a subscriber listens to a given broadcast address, and has no guarantee that will receive all messages. As such, TCP should be used when all messages must be processed, and UDP should be considered when the latency and the TCP overhead are important issues.

3.1.1.4 Services

ROS services are named communication buses that follow the request and reply paradigm, in which a node asks for a given service and receives a response according to the data that was sent in the request message and the state of the service node. They are useful to query other nodes state or to request the execution of some behavior / action.

[13] http://wiki.ros.org/nodelet

3.1.1.5 Actions

ROS actions are a special kind of service in which the progress of the request can be queried. They are very useful when the request might take a long time, and gives the caller the necessary information to supervise the execution of the request and if necessary, terminate its execution.

3.1.2 Build system

The latest ROS build system is named catkin[14], and is the successor of the original rosbuild[15] system. It combines CMake[16] macros and Python scripts to allow building multiple dependent projects at the same time. It is a cross-platform build system, organized in packages and meta-packages (group of packages). Each package is a software module that can produce libraries or binaries from source code.

Catkin was designed to deal with complex build configurations, which in the case of ROS packages involves a considerable amount of build dependencies for each project. As such, catkin provides a build system that can easily find, build and link both ROS and system dependencies. Moreover, it provides install targets to allow faster code releases and simplifies builds from source for the final users.

Other useful features of the catkin build system are the concepts of workspace and overlays. Catkin uses a workspace with out-of-source builds to keep the source code separate from the build files. This allows the code directory structure to be clean of compiler generated files that are platform dependent. Moreover, it simplifies the concurrent usage of packages (overlay), because catkin gives priority to workspace packages (in relation to system packages). This is particularly useful when it is necessary to modify and test some package that has been released and is installed in the system (without having to uninstall the stable release of that package).

Finally, catkin is a cross-platform build system that can be used to build other projects that use CMake and are not related to ROS.

3.1.3 Development tools

ROS provides several development tools that allow introspection and visualization of the system state. They are very useful for testing, debugging and profiling.

The next sections give an overview of the most important ROS tools that are currently available.

3.1.3.1 Graphical User Interface tools

The ROS development tools that have graphical user interfaces are aggregated in the rqt framework[17], and are loaded as plugins at runtime (some can be started as standalone applications).

[14] http://wiki.ros.org/catkin
[15] http://wiki.ros.org/rosbuild
[16] http://www.cmake.org/
[17] http://wiki.ros.org/rqt

Currently there is plugins to visualize sensor data (rviz[18]); introspect the contents of topics; log and replay ROS messages (rosbag[19]); display node, package and coordinate systems graphs; list and filter debug messages; change configuration of running nodes (dynamic reconfigure); monitor nodes memory and processor usage and much more.

3.1.3.2 Command line tools

The ROS command line tools[20] are split across several executables and can be used for advanced introspection (rosnode, rostopic and rosservice), system configuration (rosparam), package building and management (catkin, rosdep and rosinstall) and also to search ROS message types documentation (rosmsg and rossrv).

Finally, it is available a diagnostics tool (roswtf), that can detect packages / dependencies issues and configuration problems.

3.2 Gazebo

Gazebo[21] is a 3D multi-robot simulator capable of generating hardware sensor data for different kinds of robots while providing a realistic environment with physics simulation and 3D visualization. It is very useful to speedup testing of algorithms with different types of robots and environments.

Figure 3.2: Gazebo logo

Figure 3.3 shows how Gazebo can be used instead of a real robot, without requiring any implementation code modification (because it implements the same ROS interfaces that the hardware drivers use).

3.3 PCL

The PCL[23] [RC11] is an open source project that provides algorithms for processing point clouds. These algorithms can be used to filter and register point clouds as well as perform object segmentation, recognition and tracking.

Figure 3.4: PCL logo

Figure 3.5 gives an overview of the main modules currently available in PCL.

[18] http://wiki.ros.org/rviz
[19] http://wiki.ros.org/rosbag
[20] http://wiki.ros.org/ROS/CommandLineTools
[21] http://gazebosim.org/
[22] http://gazebosim.org/tutorials?tut=ros_control
[23] http://pointclouds.org/
[24] http://pointclouds.org/about/

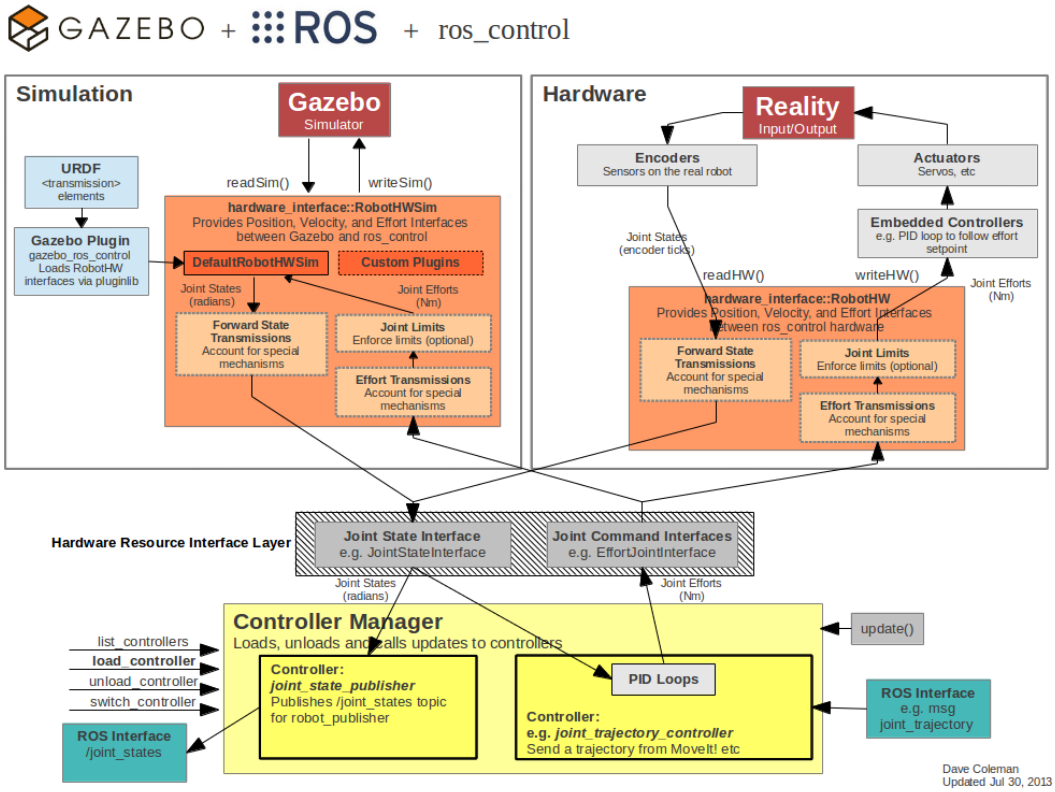

Figure 3.3: Integration of ROS and Gazebo[22]

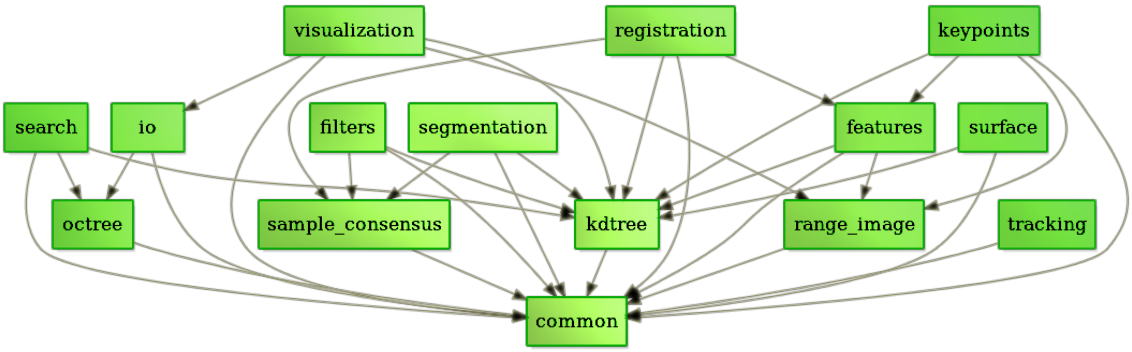

Figure 3.5: Point Cloud Library[24]

Chapter 4

Localization system

This chapter details the ROS implementation of the proposed 3/6 DoF self-localization system[1]. It starts with an overview of the main processing stages and then details the control flow and algorithms within the localization pipeline.

4.1 Overview

The self-localization system was implemented as a ROS package and provides 3/6 DoF localization by publishing *geometry_msgs::PoseStamped*[2] and *geometry_msgs::TransformStamped*[3] messages along with a detailed analysis of the pose estimation and registered point cloud (split into inliers and outliers).

The system can receive sensor data through *sensor_msgs::PointCloud2*[4] messages and as a result it can directly use data from RGB-D and ToF cameras. To use LIDARs, it provides an assembler that can produce point clouds by merging measurements from several sensors using spherical interpolation. As such, if the LIDAR sensors are mounted on tilting platforms, they can emulate a 3D sensor and retrieve a very detailed view of the environment.

The self-localization system has a modular software architecture and was implemented as several C++ templated shared libraries that can be easily used for other applications besides robot self-localization. As can be seen in figure 4.1, it is an extensible and flexible system able to fit the needs of a wide range of mobile platforms. It can be configured as a tracking system, with or without pose recovery and can also have initial pose estimation using feature detection and matching. Moreover, it can also dynamically create and update the map if necessary.

To allow fast deployment of robots in large environments it supports two configurable processing pipelines. One to process new reference maps and another to localize a mobile robot platform

[1] https://github.com/carlosmccosta/dynamic_robot_localization
[2] http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html
[3] http://docs.ros.org/api/geometry_msgs/html/msg/TransformStamped.html
[4] http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html

using ambient point clouds. This enables the loading of either processed or unprocessed referenced point clouds and allows a navigation supervisor to dynamically provide the relevant map sections based on the robot position (in order to reduce the computation resources needed).

The next sections explain in detail the architecture and algorithms used in each of the processing modules present in figure 4.1.

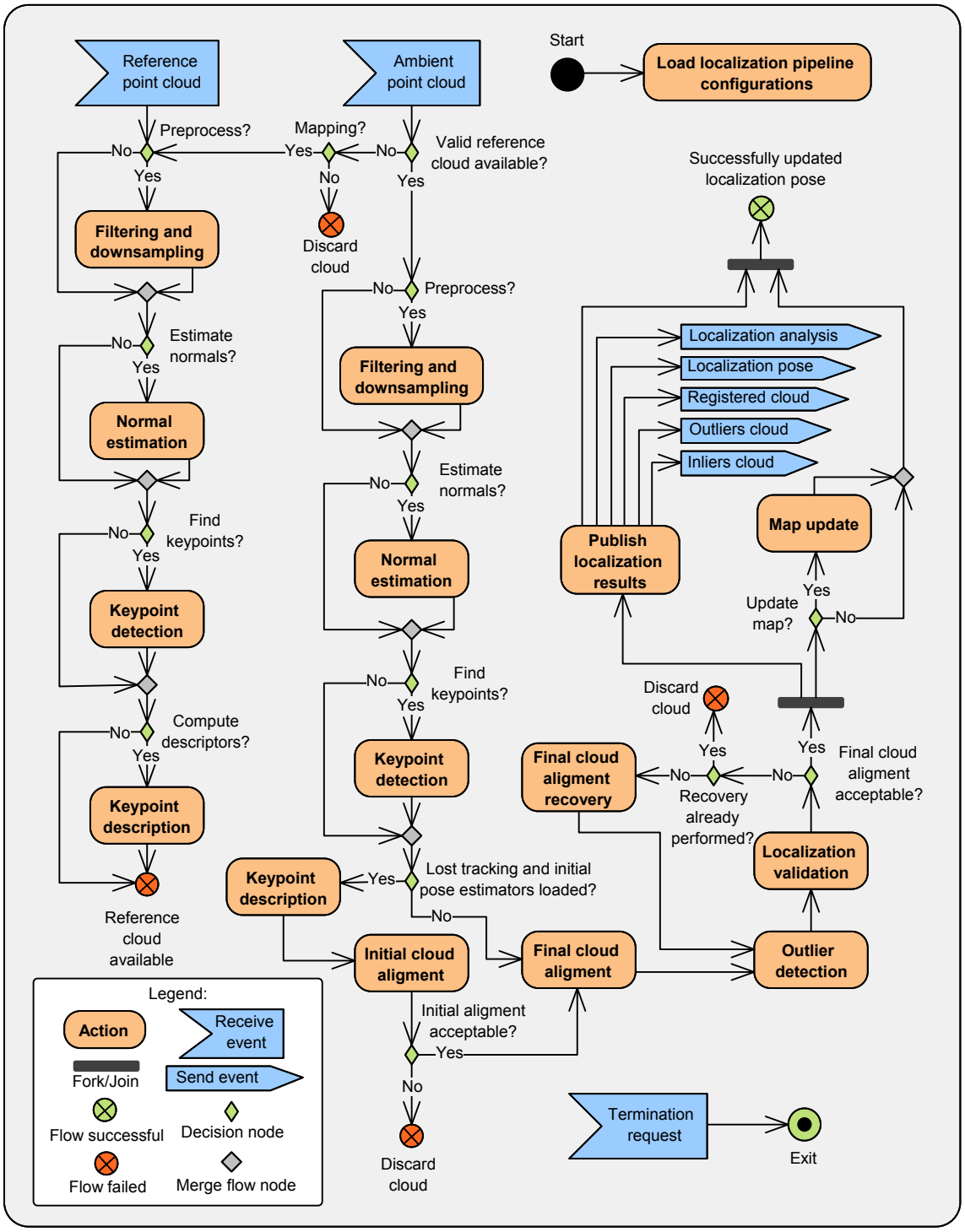

Figure 4.1: Localization system overview

4.2 Pipeline configuration

The self-localization system was designed to allow fast reconfiguration and parameterization through the use of yaml[5] files and the ROS parameter server[6]. This gives the possibility to quickly tune the localization system to the specific needs of a given mobile platform moving in a particular environment, in order to use the least amount of computational resources possible and without requiring any reprogramming or source code modification. Nevertheless, the system can use a generic configuration if hardware resources are not a concern.

In a typical configuration (following the *Yes* paths of the activity diagram in figure 4.1), the first time the localization system is used, it receives a raw reference point cloud that is preprocessed and saved to long term memory along with its associated keypoints and keypoint descriptors. This allows a much faster startup the next time the localization system is initialized. After having a reference point cloud, the localization system will estimate the robot pose periodically by analyzing the ambient point cloud sensor data. This data can be preprocessed with several filters and can be associated with computed surface normals. The robot pose estimation is performed by applying a matrix transformation correction to the current robot pose and is based on the registration of the ambient point cloud with the know map. This registration can use a tracking algorithm configuration tuned for efficiency and a second configuration for tracking recovery purposes. These tracking algorithms require a initial pose estimation, and as such, if one isn't available, a third configuration can be employed to estimate the global position of the robot based on geometric features of the environment. The switch between these configuration is based on the analysis of the registered cloud metrics, such as outlier percentage, inliers root mean square error, inliers angular distribution and the registration corrections performed to the ambient point cloud. After successfully performing the robot pose estimation, the map can be updated by either integrating the full registered point cloud, its inliers or its outliers. Finally, like most ROS nodes, the localization system will stop its execution when it receives a termination signal request.

4.3 Point cloud acquisition

Point clouds can be retrieved with a wide range of sensors with varying levels of precision and assembly time. They usually rely on time of flight or trilateration / triangulation techniques to compute distances to environment objects, which in combination with the sensor position and orientation allows to retrieve the Euclidean coordinates of the scene points. Some sensors can also capture the point's reflectance or color, which can be very useful in segmentation techniques and point cloud feature algorithms.

[5] http://yaml.org/
[6] http://wiki.ros.org/Parameter%20Server

4.4 Point cloud search data structures

Most of the point cloud algorithms use neighbor searches to analyze the surroundings of a given point. As such, efficient data structures are needed to speedup these operations, in order to execute the algorithms efficiently.

4.4.1 Voxel grids

A voxel grid is a three dimensional space partition data structure that splits the Euclidean space into regular voxels (volume pixel). It can be built very fast but is not very efficient for sparse point clouds. Figure 4.2 shows how the voxel size affects the level of detail of point clouds.

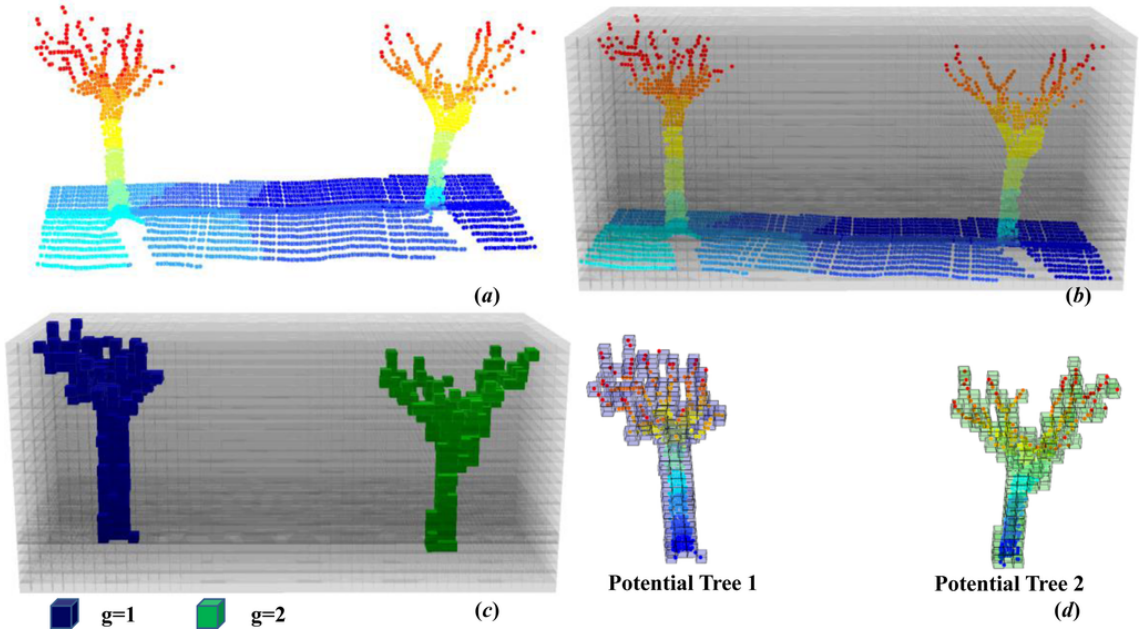

Figure 4.2: Voxel grid applied over trees point clouds [WYY+13]

4.4.2 Octrees

An octrees is a hierarchical space partition technique that adapts its tree data structure to the distribution of points in the cloud. It accomplishes this by recursively dividing each voxel in 8 octants until the tree depth is reached or when there is no more points in that region of space. This can be seen in figure 4.3 in which areas with no points have large voxels while areas with high point density have much smaller voxels.

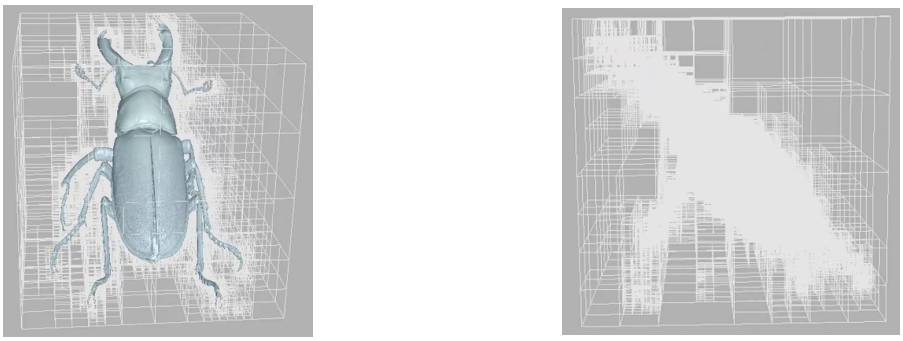

Figure 4.3: Octree of a stag beetle[7]

4.4.3 k-d trees

A k dimensional tree is a space partition technique that can organize points with k dimensions. It is a generic data structure that can be used for 2D and 3D points (examples in figure 4.4 and figure 4.5) as well as any other type of data that have an arbitrary number of dimensions (such as point cloud feature descriptors).

The binary k-d tree is built by successively selecting the median point in each axis until all points are inserted in the tree (the selection of the next axis is performed in a circular way, which in the case of three dimensional data means that after processing the z axis, the x axis would be selected).

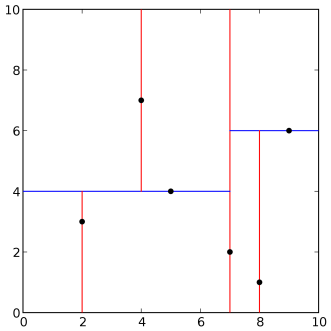

Figure 4.4: 2-d tree[8]

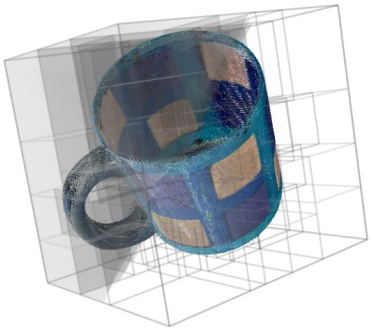

Figure 4.5: 3-d tree[9]

4.5 Reference map

The reference point cloud can be loaded from a CAD file, point cloud file or dynamically arrive trough a ROS topic as either a 3 DoF occupancy grid or 6 DoF point cloud. This allows a localization supervisor to give only sections of a global map in order to use the least amount of memory

[7] http://blog.mpanknin.de/?p=753
[8] http://en.wikipedia.org/wiki/K-d_tree
[9] http://docs.pointclouds.org/trunk/group__kdtree.html

and processing power (very large maps have deeper search structures, such as kd-trees, and should be avoided in order to improve the system efficiency).

4.6 Point cloud assembly

The laser assembler converts laser measurements in polar coordinates into Cartesian coordinates and projects the points using spherical interpolation in order to account for laser scan deformation that occurs when the robot is moving and rotating. It can merge scans from several lasers (sequentially or with a circular buffer) and it will publish the final point cloud after assembling a given number of scans or periodically after a specified duration. These assembly configurations can be changed at runtime through the use of the ROS dynamic reconfigure Application Programming Interface (API), which allows a localization supervisor to control the rate at which the localization system operates.

4.7 Filtering and down sampling

The time it takes to perform cloud registration is proportional to the amount of points in the ambient point cloud and in the reference map. As such, adjusting the level of detail of the point clouds by using voxel grids gives some control over the desired localization accuracy and the computational resources that will be required. This stage is also useful to mitigate the measurement errors of the depth sensors since the centroid of a voxel that contains points from several laser scans will be closer to the real surface (if the voxels have dimensions slightly larger than the expected laser measurement errors).

Point cloud data that comes from laser sensors can have a considerable amount of noise and an unnecessary level of detail. To cope with these problems, several preprocessing algorithms can be applied, ranging from simple point cloud downsampling to the more advanced outlier removal and surface reconstruction methods.

4.7.1 Point cloud downsampling

Downsampling methods aim to reduce the number of points while maintaining the surface structure of a given point cloud. They can be used to adjust the level of detail according to the point cloud registration precision required.

4.7.1.1 Voxel grid sampling

A voxel grid is a uniform space partition technique that can be used to cluster points according to their Euclidean coordinates. As can be seen in figure 4.6, it is a very effective method to control the level of detail of a point cloud because it gives the ability to specify the maximum number of points that a region in space should have.

The point cloud downsampling is achieved by replacing each cluster with a single point. The selection of this point can be very fast if the voxel center is used, but computing the centroid of the cluster yields better results because it represents the underlying surface with more accuracy and it attenuates errors in the sensors measurements.

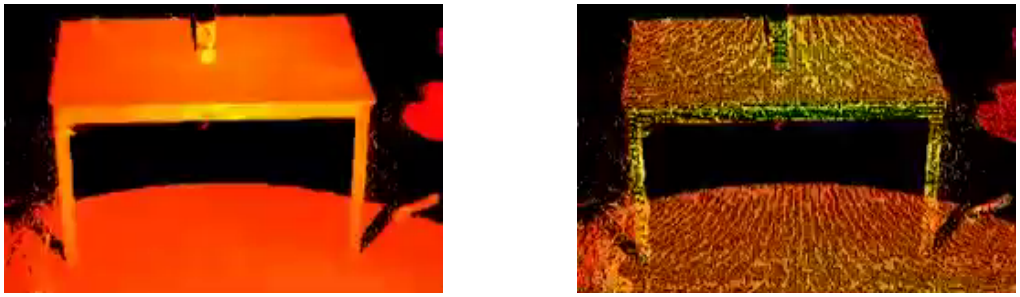

Figure 4.6: Table point cloud before (left) and after (right) voxel grid downsampling[10]

4.7.1.2 Random sampling

Random sampling [Vit84] is a fast downsampling method that randomly selects points from the input cloud until the specified number of samples is reached. This has the advantage of using real measures instead of downsampled approximations, but also means it is more sensible to sensor measurements noise. However, for outdoor environments or very complex scenes, using the real measurements can be preferable than using cluster centroids because the voxels may not have the necessary resolution or may have a prohibitive computational cost.

4.7.2 Outlier removal

Laser range finders can perform measurements with millimeter accuracy, but they have some limitations that can lead to the creation of outliers [Sot06].

One of those limitations can produce shadow points around objects boundaries. This is due to the fact that a portion of the laser beam may hit the object boundary and other part may hit other areas in the object background. And given that most laser range finders use a weighted sum of several beams, this can yield measurements that are not associated with any real object (outliers). Figure 4.6 shows a considerable amount of these shadow points close to the front table legs. Another issue is related to the angle in which the laser beam hits the objects. If the incidence angle is very low, then it may be difficult to detect if the beam had ambient reflections. This can significant increase the measurements noise or even lead to the creation of outliers. Other less common problem is associated with the material properties of the surfaces. For example, objects with very high or very low reflectance, such as metals or glass, can increase the measurements noise. Moreover depending on the combination of surface geometry, material and incidence angle, some objects may even be undetectable by laser range finders.

[10] http://pointclouds.org/documentation/tutorials/voxel_grid.php

Given the negative effect that outliers have in object segmentation and registration algorithms, they should be removed in a preprocessing stage. There are several approaches to perform outlier detection and removal [Yan10], ranging from simple distance thresholds to more robust statistical analysis. The next sections present some of then that can be useful in a localization system.

4.7.2.1 Distance filter

Given that laser range finders have a maximum distance for their measurements, it is wise to remove points that are close or beyond this limit. Moreover, it may be useful to remove points that are too close to the sensor, because they may belong to the robot itself and not the environment.

This can be achieved by applying a minimum and maximum threshold to the distances returned by the laser sensor (before converting then to Euclidean points).

4.7.2.2 Passthrough filter

A passthrough filter can select or remove points according to their properties.

For outlier removal, it can be used to select points that are within a given bounding box (useful when we already know what area of the environment we want to analyze) or remove points that don't have the appropriate intensity or color.

4.7.2.3 Radius outlier removal filter

The radius outlier removal filter deletes points that don't have a minimum number of neighbors within a specified radius distance. It can be useful when the point density is known and is very effective in removing isolated points.

4.7.2.4 Statistical outlier removal filter

The statistical outlier removal filter [Rus10] performs a global analysis of the distances between points and discards the ones that don't follow the global distance distribution.

It is a robust filter that adapts itself to the point cloud density and is very effective in removing shadow points. To do so, it computes the mean distance that each point has to a given number of neighbors and builds a global distance distribution (example in figure 4.7). Then, assuming that the distribution is Gaussian, it discards the points that have a distance higher than a given threshold (that is a percentage of the standard deviation of the distance distribution).

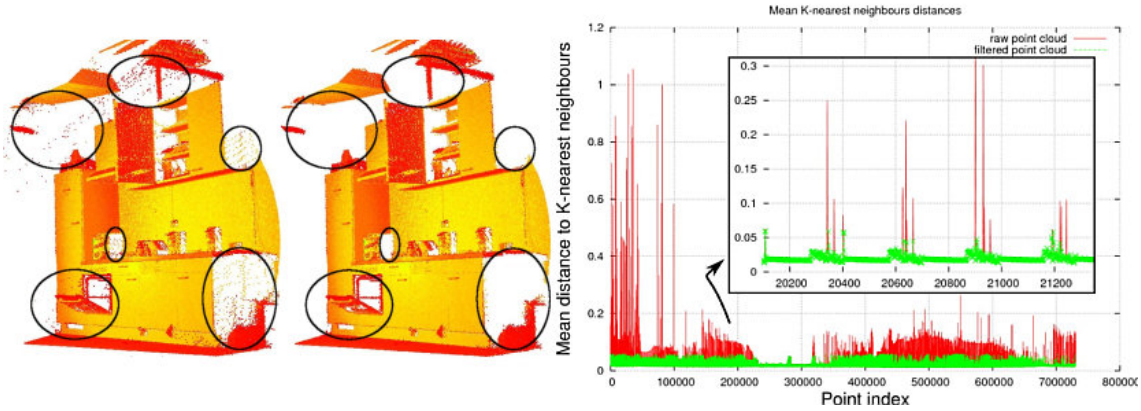

Figure 4.7: Statistical outlier removal filter[11]

4.7.3 Surface and object reconstruction and resampling

Depending on the level of sensor noise and amount of outliers present in a given point cloud, it may be necessary to employ surface reconstruction techniques to fill gaps in sensor data or correct measurements errors. The next sections introduce some of the most common techniques to achieve these goals.

4.7.3.1 Moving Least Squares

Moving least squares [ABCO+03] is a surface reconstruction algorithm that uses higher order bivariate polynomials to fit surfaces to a given set of points. It can be used to fill possible gaps in sensor data, smooth the point cloud (shown in figure 4.8), refine surface normals (shown in figure 4.9) and perform downsampling or upsampling.

Surface reconstruction can also be useful when the point cloud is built from several laser scans with different origins and registered with some alignment errors. In figure 4.9 can be seen that the normal estimation is not very accurate in the regions of overlap between different scans. This can be solved by estimating the normals using the surfaces computed by the moving least squares algorithm instead of using the point's neighbors.

[11] http://pointclouds.org/documentation/tutorials/statistical_outlier.php

Figure 4.8: Surface smoothing using moving least squares algorithm[12]

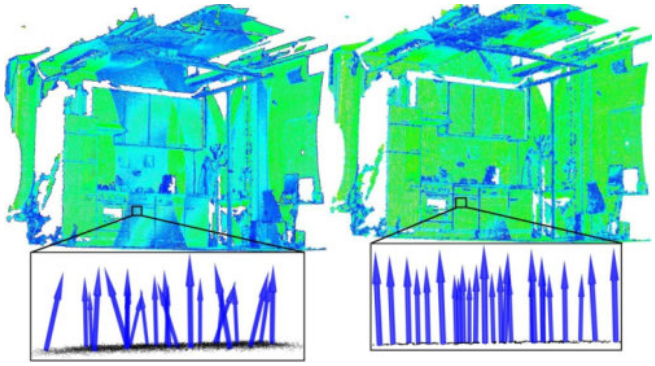

Figure 4.9: Surface normals refinement using moving least squares algorithm [Rus10]

4.8 Normal estimation

Most of feature detection, description and matching algorithms along with some registration methods rely on the point's surface normal and curvature. As such, it was developed a robust 3 DoF normal estimation algorithm that uses Random Sample Consensus (RANSAC) [14] to fit lines to the sensor data and then orient the normals to the sensors origin. For 6 DoF data, the Principal Component Analysis (PCA) [15] or the Moving Least Squares (MLS) [16] approaches available in PCL can be used.

4.8.1 3D surface normal estimation

Surface normals provide information about the orientation of their underlying geometry and are widely used as the basis for other point descriptors. They can be computed using plane fitting methods or using more advanced techniques such as the one presented in section 4.7.3.1.

These algorithms analyze the neighborhood of a given point in order to compute the surface normal, and as such, the correct specification of what points should be included in the estimation is crucial to achieve accurate results. This depends on the environment geometry and the level of

[12] http://pointclouds.org/documentation/tutorials/resampling.php

detail that is required, and is usually done by specifying a radius distance (example in figure 4.10) or by limiting the number of neighbor points to use.

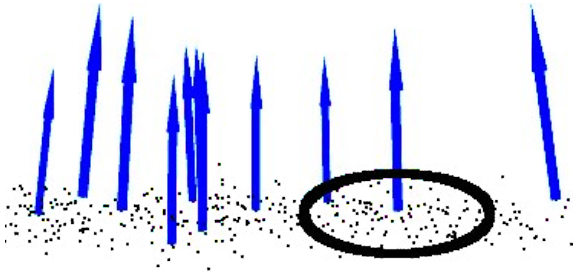

Figure 4.10: Point neighborhood for normal estimation [Rus10]

4.8.2 2D normal line normal estimation

FF.

4.9 Keypoint detection

Keypoint detection aims to find interesting points in a given point cloud in order to compute the cloud registration faster or to perform feature description and matching. There are several approaches to find keypoints that have different definitions of what kind of points are worth analyzing. But they all should be able to detect the same keypoints given similar data and they aim to select the points that best describe the underlying point cloud geometry. Currently the localization system can use the Scale Invariant Feature Transform (SIFT) [17] algorithm on the points curvature or the Intrinsic Shape Signatures (ISS3D) [18] keypoint detector.

Aligning two point clouds with overlapping views of the environment requires the establishment of point correspondences. If both point clouds have similar sensor origins, these can be determined with nearest neighbor's searches and filtered with correspondence rejectors (using other point properties such as reflectance and color). But if they were acquired in two very different positions, then more advanced techniques must be employed.

One of those techniques uses histograms to describe the geometric properties of the environment around a given point. This allows points to be matched even if they have completely different Euclidean coordinates. Also, by using histograms and sampling techniques, these descriptors are much more robust against sensor noise and varying level of point density. However, these advantages come with a heavy computational cost, and as such, point descriptors should only be computed on the most descriptive areas of the environment.

Identifying these environment points is known as feature / keypoint detection, and usually involves finding interesting points, such as corners and edges. Besides uniqueness, these points must also be repeatable. This means that the detection algorithms should be able to find the same points even if they are present in different point clouds with sensor noise and varying point density.

This is of the utmost importance, because if the same keypoints are not identified on both clouds, then matching the point descriptors will likely fail.

The next sections introduce some of the most used algorithms for normal estimation, keypoint detection and point description.

4.9.1 Intrinsic Shape Signatures

FF.

4.9.2 SIFT keypoints

FF.

4.10 Keypoint description

Describing a keypoint usually involves analyzing its neighboring points and computing a given metric or histogram that quantifies the neighbor's relative distribution, their normals angular relation, associated geometry or other metrics that are deemed useful. Several approaches were suggested over the years according to different recognition needs and they are the basis of feature matching algorithms used in the initial pose estimation.

The localization system can use most of the keypoint descriptors available in PCL, namely the Point Feature Histogram (PFH), the Fast Point Feature Histogram (FPFH), the Signature of Histograms of Orientations (SHOT), the Shape Context 3D (SC3D), the Unique Shape Context (USC) and the Ensemble of Shape Functions (ESF).

4.10.1 Point Feature Histograms

FF.

4.10.2 Fast Point Feature Histograms

FF.

4.10.3 Shape Context Estimation

FF.

4.10.4 Unique Shape context

FF.

4.10.5 SHOT estimation

FF.

4.10.6 Spin image estimation

FF.

4.10.7 Ensemble Shape Functions

FF.

4.10.8 Rotational Projection Statistics

FF.

4.11 Cloud registration

Point cloud registration is the process of finding the transformation matrix (usually translation and rotation only) that when applied to a given ambient cloud will minimize an error metric (such as the mean square error of the ambient point cloud in relation to a given reference point cloud). Several approaches were suggested over the years and they can be categorized as point or feature cloud registration.

4.12 Initial alignment with keypoint descriptors matching

Feature registration is the process of matching keypoint descriptors in order to find an initial alignment between two point clouds. The proposed localization system uses a feature registration method similar to the Sample Consensus Initial Alignment algorithm presented in [20]. It uses a sample consensus approach to select the best registration transformation after a given number of iterations. In each iteration a subsample of randomly selected descriptors from the ambient cloud is retrieved. Then for each of these descriptors, k best matching descriptors in the reference point cloud are searched and one of them is chosen randomly (this improves robustness against noise in the sensor data and changes in the environment that are not yet integrated into the map). Later after having filtered these correspondences between ambient and reference descriptors, the registration matrix is computed. If this registration matrix results in a point cloud overlap that has a minimum of inliers percentage (a point in the ambient cloud is an inlier if it has a point in the reference cloud closer than a given distance), then it is considered an acceptable initial pose and is saved (to allow a localization supervisor to analyze the distribution of the acceptable initial poses). In the end of all iterations, the best initial pose (if found) is refined with a point cloud registration algorithm.

4.13 Final alignment with point cloud error minimization

Point cloud registration algorithms such as the Iterative Closest Point [19] (with its several known variations like ICP point-to-point, ICP point-to-point non-linear, ICP point-to-plane and generalized ICP) and the Normal Distributions Transform [6] are among the most popular algorithms to

register point clouds. They can achieve very accurate cloud registration but they require an approximate initial pose for the registration to successfully converge to a correct solution (otherwise they may achieve only partial cloud overlap or even fail to converge to a valid solution). To solve this problem, a complete autonomous registration pipeline must also include the computation of this initial alignment through the usage of feature registration.

4.13.1 Iterative Closest Point

FF.

4.13.2 Normal Distributions Transform

FF.

4.14 Outlier detection

FF.

4.15 Localization validation

After a point cloud is registered, several metrics are calculated in order to evaluate if a valid pose can be retrieved using the registration matrix.

The first computed metrics are the percentage of inliers and the root mean square error of these inliers. If a minimum number of points was registered and the inlier percentage and root mean square error are acceptable, then the registration is considered successful. However, these registered points can be agglomerated in a small area and may not be representative of the robot location. As such, a second metric is computed that takes into account the angular distribution of these inliers. This metric gives a measurement of how reliable is this registration and is based on the fact that there is high confidence in a given estimation when there are correctly registered points all around the robot.

The last metrics are the corrections that the registration matrix introduced. Given that the localization system will be in tracking mode most of the time, it is possible to define how far a new pose can be in relation to the previous accepted location and discard new poses that exceed a given threshold. This is useful to discard pose corrections that are very unlikely to happen, such as the robot moving half a meter between poses when it is expected to move only at 30 cm/s. These situations can happen when there is a sudden decrease in the field of view (that can occur due to sensor occlusion or malfunction) or when large unknown objects very similar to sections of the map appearinto the field of view of the robot.

If all these metrics are within acceptable thresholds, then the robot pose can be computed by applying the matrix correction to the initial pose associated with the ambient sensor data. If any of these metrics are not acceptable, then the system can be configured to simply discard

this pose estimation and try to estimate the pose in the next sensor data update or it can apply a tracking recovery attempt with a different registration algorithm (or the same algorithm with different parameters). If several consecutive pose estimations are discarded, the system can have a second level of recovery that can be configured to use the initial pose estimation algorithms in order to finally estimate the robot pose and reset the tracking state.

4.16 Dynamic map update

After performing a successful pose estimation, the localization system can be paired with OctoMap [21] in order to update the localization map by either integrating only the unknown objects or the full registered point cloud. Integrating only the unknown objects is the recommended approach when there is a known map and the environment is expected to change gradually. This is also more efficient as only the points that need to be integrated are processed and ray traced in OctoMap. On the other hand, integrating the full registered cloud can be desirable if the map of the environment is very incomplete, very outdated or expected to change considerably during the operation of the robot.

Chapter 5

Localization system evaluation

FF

5.1 FF

FF

Chapter 6

Conclusions and future work

The implementation of an efficient and reliable self-localization method will allow the deployment of robots in dynamic environments to help human co-workers improve the overall productivity and reduce the time and costs of manufacturing.

There are several approaches that can be used to estimate a robot pose according to the target environment and accuracy requirements. This dissertation tackled this problem by providing a modular and configurable localization system that can be used for pose tracking and initial localization estimation.

Appendix A

Appendix 1

FF

A.1 FF

FF

Appendix 1

References

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, January 2003.

[AMGC02] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[BGFM10] J.-L. Blanco, J. Gonzalez, and J.-a. Fernandez-Madrigal. Optimal Filtering for Non-parametric Observation Models: Applications to Localization and SLAM. *The International Journal of Robotics Research*, 29(14):1726–1742, May 2010.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[BOG+10] Mussa Bshara, Umut Orguner, Fredrik Gustafsson, Leo Van Biesen, and Student Member. Fingerprinting Localization in Wireless Networks Based on Received-Signal-Strength Measurements : A Case Study on WiMAX Networks Fingerprinting Localization in Wireless Networks Based on Received-Signal-Strength Measurements : A Case Study on WiMAX Networ. . . . *IEEE Transactions on*, 59(1):283–294, 2010.

[BTP13] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. Sparse iterative closest point. *Computer Graphics Forum*, 32(5):113–123, August 2013.

[Che03] Z H E Chen. Bayesian Filtering : From Kalman Filters to Particle Filters , and Beyond VI Sequential Monte Carlo Estimation : Particle Filters. *Statistics*, 182(1):1–69, 2003.

[CK02] Yongguang Chen Yongguang Chen and H. Kobayashi. Signal strength based indoor geolocation. *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*, 1(1):436–439, 2002.

[CSSK02] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The Trimmed Iterative Closest Point algorithm. *Object recognition supported by user interaction for service robots*, 3:545–548, 2002.

[DKU+07] Zhen Dai, Stefan Knedlik, Pakorn Ubolkosold, Junchuan Zhou, and Otmar Loffeld. Integrated GPS navigation for civilian vehicles in challenging environment. *2007 IEEE International Conference on Vehicular Electronics and Safety, ICVES*, pages 1–6, December 2007.

REFERENCES

[DZA+13] M. Djehaich, H. Ziane, N. Achour, R. Tiar, and N. Ouadah. SLAM-ICP with a Boolean method applied on a car-like robot. *Proceedings of the 2013 11th International Symposium on Programming and Systems, ISPS 2013*, pages 116–121, 2013.

[EW99] Garry a. Einicke and Langford B. White. Robust extended Kalman filtering. *IEEE Transactions on Signal Processing*, 47(9):2596–2599, 1999.

[FBDT99] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. *American Association for Artificial Intelligence*, (Handschin 1970):1–6, 1999.

[GIRL03] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy. Geometrically stable sampling for the ICP algorithm. *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 260–267, 2003.

[GWG13] Francesco Guarato, James F C Windmill, and Anthony Gachagan. A new sonar localization strategy using receiver beam characteristics. *IEEE International Ultrasonics Symposium, IUS*, pages 445–448, July 2013.

[Ibr10] Modar Ibraheem. Gyroscope-enhanced dead reckoning localization system for an intelligent walker. In *ICINA 2010 - 2010 International Conference on Information, Networking and Automation, Proceedings*, volume 1, pages 67–72, 2010.

[IKP10] Edouard Ivanjko, Andreja Kitanov, and I Petrovic. Model based Kalman Filter Mobile Robot Self-Localization. *Robot Localization and Map ...*, (1996):59–91, 2010.

[Jez08] O Jez. Evaluation of a factorized ICP based method for 3D mapping and localization. *Atpjournal.Sk*, pages 29–37, 2008.

[JU97] Sj Julier and Jk Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. *Int Symp AerospaceDefense Sensing Simul and Controls*, 3:26, 1997.

[Kal60] Re E Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KKOO07] Yong Shik Kim, Bong Keun Kim, Kohtaro Ohba, and Akihisa Ohya. Bias estimation of DGPS multi-path data for localization of mobile robot. *Proceedings of the SICE Annual Conference*, pages 1501–1505, September 2007.

[LKP06] Sukhan Lee Sukhan Lee, Eunyoung Kim Eunyoung Kim, and Yeonchool Park Yeonchool Park. 3D object recognition using multiple features for robotic manipulation. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, (May):3768–3774, 2006.

[LLRM06] Martin Lauer, Sascha Lange, Martin Riedmiller, and Martin Riedmiller Martin Lauer, Sascha Lange. Calculating the perfect match: an efficient and accurate approach for robot self-localization. *Robocup 2005: Robot soccer world cup ...*, 4020(c):142–153, 2006.

[LYL11] Jianshen Liu, Baoyong Yin, and Xinxing Liao. Robot self-localization with optimized error minimizing for soccer contest. *Journal of Computers*, 6(7):1485–1492, July 2011.

REFERENCES

[MCB10] Julien Moras, Véronique Cherfaoui, and Phillipe Bonnifait. A lidar perception scheme for intelligent vehicle navigation. In *11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010*, number December, pages 1809–1814, 2010.

[PM63] Miguel Pinto and Ap Moreira. Novel 3D matching self-localization algorithm. *International Journal of . . .* , 5(1):1–12, 1963.

[QC09] Morgan Quigley and Ken Conley. ROS: an open-source Robot Operating System. *. . . on open source . . .* , 2009.

[R.48] W. R. Introduction to Wireless. *Nature*, 162:911–911, April 1948.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011.

[Rib04] Maria Isabel Ribeiro. Kalman and Extended Kalman Filters : Concept , Derivation and Properties. *Institute for Systems and Robotics Lisboa Portugal*, (February):42, 2004.

[RKZ13] Michal Reinstein, Vladimir Kubelka, and Karel Zimmermann. Terrain adaptive odometry for mobile skid-steer robots. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4706–4711, 2013.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on. IEEE*, pages 145–152. IEEE Comput. Soc, 2001.

[Rus10] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Technische Universität München, August 2010.

[Sot06] S Sotoodeh. Outlier Detection in Laser Scanner Point Clouds. *Iaprs*, Volume XXX(September):297–302, 2006.

[SS09] Ptm Saito and Rj Sabatine. An analysis of parallel approaches for a mobile robotic self-localization algorithm. *International Journal of . . .* , 2(4):49–64, 2009.

[TGCV12] Gurkan Tuna, Kayhan Gulez, V. Cagri Gungor, and T. Veli Mumcu. Evaluations of different Simultaneous Localization and Mapping (SLAM) algorithms. *IECON Proceedings (Industrial Electronics Conference)*, pages 2693–2698, 2012.

[Thr02] Sebastian Thrun. *Probabilistic robotics*, volume 45. 2002.

[Vit84] Jeffrey Scott Vitter. Faster methods for random sampling. *Communications of the ACM*, 27(7):703–718, 1984.

[WV02] E a Wan and R Van Der Merwe. The unscented Kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 7:153–158, 2002.

[WY07] Shun-xi Wu Shun-xi Wu and Ming Yang Ming Yang. Landmark Pair based Localization for Intelligent Vehicles using Laser Radar. *2007 IEEE Intelligent Vehicles Symposium*, pages 209–214, June 2007.

[WYY+13] Bin Wu, Bailang Yu, Wenhui Yue, Song Shu, Wenqi Tan, Chunling Hu, Yan Huang, Jianping Wu, and Hongxing Liu. A voxel-based method for automated identification and morphological parameters estimation of individual street trees from mobile laser scanning data. *Remote Sensing*, 5(2):584–611, January 2013.

[Yan10] Yang Zhang. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys*, 12:159–170, 2010.

[YXL11] Peng Yang, Li Xie, and Jilin Liu. Angular velocity current statistical model based real-time celestial navigation for lunar rover. *Proceedings of the 30th Chinese Control Conference*, pages 4011–4016, 2011.

[Zha94] Z Zhang. Iterative point matching for registration of free form curves and surfaces. *International Journal of Computer Vision*, 12:119–152, 1994.

[ZL11] Jia Heng Zhou and Huei Yung Lin. A self-localization and path planning technique for mobile robot navigation. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pages 694–699, 2011.