de Alcalá
COMISIÓN DE ESTUDIOS OFICIALES
DE POSGRADO Y DOCTORADO

ACTA DE EVALUACIÓN DE LA TESIS DOCTORAL

(FOR EVALUATION OF THE ACT DOCTORAL THESIS)

Año académico (*academic year):* 2016/17

DOCTORANDO *(candidate PHD)*: **MOLINOS VICENTE, EDUARDO JOSÉ**
D.N.I./PASAPORTE (*Id.Passport*): ******2388B**
PROGRAMA DE DOCTORADO *(Academic Committee of the Programme)*: **D332 DOCTORADO EN ELECTRÓNICA:SISTEMAS ELECTRÓNICOS AVANZADOS.SISTEMAS INTELIGEN**
DEPARTAMENTO DE *(Department)*: **Electrónica**
TITULACIÓN DE DOCTOR EN *(Phd title)*: **DOCTOR/A POR LA UNIVERSIDAD DE ALCALÁ**

En el día de hoy 03/07/17, reunido el tribunal de evaluación, constituido por los miembros que suscriben el presente Acta, el aspirante defendió su Tesis Doctoral **con Mención Internacional** (*In today assessment met the court, consisting of the members who signed this Act, the candidate defended his doctoral thesis with mention as International Doctorate*), elaborada bajo la dirección de (*prepared under the direction of*) MANUEL OCAÑA MIGUEL // .

Sobre el siguiente tema *(Title of the doctoral thesis)*: *DYNAMIC OBSTACLES AVOIDANCE ALGORITHMS FOR UNMANNED GROUND VEHICLES*

Finalizada la defensa y discusión de la tesis, el tribunal acordó otorgar la CALIFICACIÓN GLOBAL[1] de (**no apto, aprobado, notable** y **sobresaliente**) *(After the defense and defense of the thesis, the court agreed to grant the GLOBAL RATING (fail, pass, good and excellent)*: SOBRESALIENTE (EXCELLENT)

Alcalá de Henares, a 3 de Julio de 2017

Fdo. *(Signed)*: JOAQUIN LOPEZ Fdo

Fdo. *(Signed)*: VLADIMIR IVAN

Fdo. *(Signed)*: Mª Elena López Guillén

FIRMA DEL ALUMNO (candidate's signature),

Fdo. *(Signed)*: EDUARDO J. MOLINOS

Con fecha 24 de julio de 2017 la Comisión Delegada de la Comisión de Estudios Oficiales de Posgrado, a la vista de los votos emitidos de manera anónima por el tribunal que ha juzgado la tesis, resuelve:

☒ Conceder la Mención de "Cum Laude"
☐ No conceder la Mención de "Cum Laude"

La Secretaria de la Comisión Delegada

[1] La calificación podrá ser **"no apto" "aprobado" "notable" y "sobresaliente"**. El tribunal podrá otorgar la mención de "cum laude" si la calificación global es de sobresaliente y se emite en tal sentido el voto secreto positivo por unanimidad. (*The grade may be **"fail" "pass" "good"** or **"excellent"**. The panel may confer the distinction of "cum laude" if the overall grade is "Excellent" and has been awarded unanimously as such after secret voting.).*

INCIDENCIAS / OBSERVACIONES:
(Incidents / *Comments)*

Universidad de Alcalá

COMISIÓN DE ESTUDIOS OFICIALES
DE POSGRADO Y DOCTORADO

En aplicación del art. 14.7 del RD. 99/2011 y el art. 14 del Reglamento de Elaboración, Autorización y Defensa de la Tesis Doctoral, la Comisión Delegada de la Comisión de Estudios Oficiales de Posgrado y Doctorado, en sesión pública de fecha 24 de julio, procedió al escrutinio de los votos emitidos por los miembros del tribunal de la tesis defendida por *MOLINOS VICENTE, EDUARDO JOSÉ*, el día 3 de julio de 2017, titulada *DYNAMIC OBSTACLES AVOIDANCE ALGORITHMS FOR UNMANNED GROUND VEHICLES*, para determinar, si a la misma, se le concede la mención "cum laude", arrojando como resultado el voto favorable de todos los miembros del tribunal.

Por lo tanto, la Comisión de Estudios Oficiales de Posgrado **resuelve otorgar** a dicha tesis la

MENCIÓN "CUM LAUDE"

Alcalá de Henares, 27 julio de 2017
EL PRESIDENTE DE LA COMISIÓN DE ESTUDIOS
OFICIALES DE POSGRADO Y DOCTORADO

Firmado digitalmente por VELASCO PEREZ JUAN RAMON - DNI 03087239H
Fecha: 2017.07.30 18:40:34 +02'00'

Juan Ramón Velasco Pérez

Copia por e-mail a:
Doctorando: MOLINOS VICENTE, EDUARDO JOSÉ
Secretario del Tribunal: MARÍA ELENA LÓPEZ GUILLÉN.
Director de Tesis: MANUEL OCAÑA MIGUEL

Universidad de Alcalá

ESCUELA DE DOCTORADO
Servicio de Estudios Oficiales de Posgrado

DILIGENCIA DE DEPÓSITO DE TESIS.

Comprobado que el expediente académico de D./Dª EDUARDO JOSÉ MOLINOS VICENTE reúne los requisitos exigidos para la presentación de la Tesis, de acuerdo a la normativa vigente, y habiendo presentado la misma en formato: ☒ soporte electrónico ☐ impreso en papel, para el depósito de la misma, en el Servicio de Estudios Oficiales de Posgrado, con el nº de páginas: 252 se procede, con fecha de hoy a registrar el depósito de la tesis.

Alcalá de Henares a 26 de ABRIL de 2017

Rosa Carrillo Salas

Fdo. El Funcionario

Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Dynamic Obstacles Avoidance Algorithms for Unmanned Ground Vehicles

PhD. Thesis Presented by

Eduardo José Molinos Vicente

2017

Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Dynamic Obstacles Avoidance Algorithms for Unmanned Ground Vehicles

PhD. Thesis Presented by

Eduardo José Molinos Vicente

Advisor

Dr. Manuel Ocaña Miguel

Alcalá de Henares, 2017

Universidad de Alcalá

DEPARTAMENTO DE ELECTRÓNICA
Escuela Politécnica
Campus Universitario s/n
28805 Alcalá de Henares (Madrid)
Teléfono: 91 885 65 40
Fax: 91 885 65 91
dpto.electronica@uah.es

Dr. D. Manuel Ocaña Miguel, Profesor Titular de la Universidad de Alcalá

INFORMA: Que la Tesis Doctoral titulada "Dynamic Obstacles Avoidance Algorithms for Unmanned Ground Vehicles", presentada por D. Eduardo José Molinos Vicente, y realizada bajo mi dirección, dentro del campo de los sistemas de navegación autónoma, reúne los méritos de calidad y originalidad para optar al Grado de Doctor.

Alcalá de Henares, a 20 de abril de 2017.

Fdo.: Dr. D. Manuel Ocaña Miguel.

Universidad de Alcalá

DEPARTAMENTO DE ELECTRÓNICA
Escuela Politécnica
Campus Universitario s/n
28805 Alcalá de Henares (Madrid)
Teléfono: 91 885 65 40
Fax: 91 885 65 91
dpto.electronica@uah.es

Dr. Dña. Sira Elena Palazuelos Cagigas, Directora del Departamento de Electrónica de la Universidad de Alcalá

INFORMA: Que la Tesis Doctoral titulada "Dynamic Obstacles Avoidance Algorithms for Unmanned Ground Vehicles", presentada por D. Eduardo José Molinos Vicente, y realizada bajo la dirección del Dr. D. Manuel Ocaña Miguel, dentro del campo de los sistemas de navegación autónoma, reúne los méritos de calidad y originalidad para optar al Grado de Doctor.

Alcalá de Henares, a 20 de abril de 2017.

Fdo.: Dr. Dña. Sira Elena Palazuelos Cagigas.

"Which is the longest path? Any path is the longest when the feet go and the heart doesn't"
- Alejandro Jodorowsky

Agradecimientos

Tras mucho tiempo y sacrificio, pero también muchos momentos buenos y constructivos me encuentro escribiendo estos agradecimientos. Al final estas líneas resumen unos cuantos años de mi vida y una etapa de la que me llevo muchos gratos recuerdos.

Para empezar quiero agradecer a mi familia más cercana, porque ellos son los que más me han apoyado y han tenido que soportar el estrés y las tensiones que se derivan de la elaboración de una tesis. Desde luego sin ellos hubiera sido imposible.

Para continuar tengo que dar las gracias a mi tutor, Manuel. Primero, por confiar en mí cuando aún era un estudiante de segundo de carrera. Lo que empezó con un pequeño proyecto llevando la página web para complementar mis estudios ha terminado con muchos años de trabajo juntos. Y segundo, por todo el apoyo y el tiempo que me ha dedicado en esos años.

Siguiendo en orden de importancia quiero dar las gracias a toda la gente del Robe-Isis. Ángel por aguantarme todos los días, por los pocos (seguro que aún queda alguno más), viajes y momentos compartidos. Noelia que llevamos compartiendo pupitres (¡y al final hasta barrio!) más de una decada. Fer por todas las risas que nos hemos echado. Nacho por todo el tiempo compartido y la ayuda que ha proporcionado en esta tesis. Rober por surtirnos de jugosos pasteles siempre que la ocasión lo requería. A los chicos del Isis/Invett por tantos desayunos (Raúl, Carlos, Mario, Rubén, Javi, Carlota). También a todos los que han pasado por aquí estos años: Jorge, Javi, Balki, Augusto, Hugo. Y por último para los alumnos de TFG que terminaron siendo compañeros: Raúl, Rocío, Fran, Guillermo, Carlos (y seguro que para cuando este libro salga a la luz, Adrián, Alejandro,

Adolfo, Eduardo y David).

Gracias también a los profesores (Manuel, Dani, Elena, Carlos, Luismi) y los alumnos con los que he podido compartir labores docentes. Cada hora de clase ha sido una experiencia enriquecedora.

No quiero olvidarme tampoco de dar las gracias al fabuloso equipo con el que coincidí en Edimburgo. Ir a un país que no conoces no suele ser nada fácil, pero desde luego con un grupo así lo fue. Así que gracias a todos, especialmente a Sethu, Vlad, Jose y Yimming.

No puedo terminar estos agradecimientos sin pedir disculpas si me dejo a alguien. Son muchos años en la universidad entre unas cosas y otras y mi memoria cada vez es peor. Así que si estas leyendo esto y crees que mereces estar aquí, ¡gracias a ti también!

Resumen

En las últimas décadas, los vehículos terrestres no tripulados (UGVs) están siendo cada vez más empleados como robots de servicios. A diferencia de los robots industriales, situados en posiciones fijas y controladas, estos han de trabajar en entornos dinámicos, compartiendo su espacio con otros vehículos y personas. Los UGVs han de ser capaces de desplazarse sin colisionar con ningún obstáculo, de tal manera que puedan asegurar tanto su integridad como la del entorno.

En el estado del arte encontramos algoritmos de navegación autónoma diseñados para UGVs que son capaces de planificar rutas de forma segura con objetos estáticos y trabajando en entornos parcialmente controlados. Sin embargo, cuando estos entornos son dinámicos, se planifican rutas más peligrosas y que a menudo requieren de un mayor consumo de energía y recursos, e incluso pueden llegar a bloquear el UGV en un mínimo local.

En esta tesis, la adaptación de algunos algoritmos disponibles en el estado del arte para trabajar en entornos dinámicos han sido planteados. Estos algoritmos incluyen información temporal tales como los basados en arcos de curvatura (PCVM y DCVM) y los basados en ventanas dinámicas (DW4DO y DW4DOT). Además, se ha propuesto un planificador global basado en Lattice State Planner (DLP) que puede resolver situaciones donde los evitadores de obstáculos reactivos no funcionan.

Estos algoritmos han sido validados tanto en simulación como en entornos reales, utilizando distintas plataformas robóticas, entre las que se incluye un robot asistente (RoboShop) diseñado y construido en el marco de esta tesis.

Palabras Clave: Navegación autónoma, Evitación de obstáculos

dinámicos, Planificación de caminos, Vehículos terrestres no tripulados.

Abstract

In the recent years, unmanned ground vehicles (UGVs) are being increasingly used as service robots. Unlike industrial robots, which are situated in fixed and controlled positions, UGVs work in dynamic environments, sharing the space with other vehicles and humans. UGVs should being able to move without colliding with any obstacle, assuring its integrity and the environment safety.

In the state of the art, navigation algorithms for UGVs are able to plan routes in a safe way, with static obstacles and work in partially controlled environments. However, when the environment is dynamic, the paths planned are more dangerous and often result in more energy and resources consumption, or even can block the UGV in a local minima situation.

In this thesis, adaptation of state of the art algorithms for working in dynamic environments has been proposed. These algorithms take into account time information, such as based on curvature arcs (PCVM and DCVM) and based on dynamic window approach (DW4DO and DW4DOT). A global path planning algorithm based in Lattice State Planner (DLP) that can solve situations where an obstacle avoidance algorithm does not work is also proposed.

These algorithms have been validated in both simulated and real tests using several robotic platforms, including an assistant robot (RoboShop) that has also been designed and built during this thesis development.

KeyWords: Autonomous Navigation, Dynamic Obstacle Avoidance, Path Planning, Unmanned Ground Vehicles.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ABSYNTHE	Abstraction, Synthesis and Integration of Information for Human-Robot Teams.
AMCL	Adaptive Monte Carlo Localization.
AR	Augmented Reality.
ARA*	Anytime Repaired A*.
ARIA	MobileRobots' Advanced Robot Interface for Applications.
BCM	Beam Curvature Method.
CARMEN	Carnegie Mellon Robot Navigation Toolkit.
CPR	Cycles Per Revolution.
CVM	Curvature Velocity Method.
DARPA	Defense Advanced Research Projects Agency.
DCVM	Dynamic Curvature Velocity Method.
DLP	Dynamic Lattice Planner.
DOMap	Dynamic Obstacles Map.
DW4DO	Dynamic Window for Dynamic Obstacles.
DW4DOT	Dynamic Window for Dynamic Obstacles Tree.
DWA	Dynamic Window Approach.
DWA*	Dynamic Window Approach Star.
EKF	Extended Kalman Filter.

GPFs	Global Potential Fields.
GPS	Global Positioning System.
HMI	Human-Machine Interface.
IFR	International Federation of Robotics.
IMU	Inertial Measurement Unit.
LCM	Lane Curvature Method.
LIDAR	Light Detection and Ranging.
LTS	Long Time Support.
MRDS	Microsoft Robotics Development Studio.
MVSE	Microsoft Visual Simulator Environment.
ND	Nearness Diagram.
OSRF	Open Source Robotics Foundation, Inc..
PCVM	Predicted Curvature Velocity Method.
PFs	Potential Fields.
PI	Proportional Integral.
POMDP	Partially Observable Markov Decision Process.
PROPINA	Plataforma Robótica Para La Investigación.
PWM	Pulse-Width Modulation.
QR	Quick Response.
RobeSafe	Robotics and eSafety Research Group.
RoboShop	ROBOtic Guide for Shop.
ROS	Robot Operating System.
RPM	Revolutions per Minute.

RRT	Rapidly Exploring Random Trees.
SLAM	Simultaneous Localization And Mapping.
SND	Smooth Nearness Diagram.
STDR	Simple Two Dimensional Robot Simulator.
Tf	Transform.
TVDW	Time Variant Dynamic Window.
UAV	Unmanned Aerial Vehicle.
UGV	Unmanned Ground Vehicle.
UMBmark	University of Michigan Benchmark.
UUV	Unmanned Underwater Vehicle.
VFF	Vector Field Force.
VFH	Vector Field Histogram.
VFH*	Vector Field Histogram Star.
VFH*TDT	Vector Field Histogram with Time Dependent Tree.
VFH+	Vector Field Histogram Plus.
VOXEL	VOLumetric piXEL.

Chapter 1

Introduction

1.1 Motivation

For the past 50 years, robotics have been a key component in the manufacturing industry. Industrial robots are programmable mechanical devices designed to move materials, parts, tools, or specialized devices through variable programmed motions to perform a variety of tasks. Usually, industrial robots operate in controlled environments and they are situated in fixed positions, forming part of the environment itself. An industrial robot system includes, apart from the robot itself, any devices and/or sensors required by the robot to perform its tasks. Robots are generally used to perform unsafe, hazardous, highly repetitive and unpleasant tasks. Figure 1.1 shows a car assembly line where industrial robotic arms work.

Industries, especially the automotive and electrical ones, have invested on robots for the last decades. These investments have contributed to the growth of industrial robots, especially in the last decade. The International Federation of Robotics (IFR) collects all the information related to robots in order to analyse their growth. Figure 1.2 shows the estimated worldwide annual supply of industrial robots, which is increasing almost every year (with the exception of the years 2006-2009, due to the worldwide economy recession). Figure 1.3 shows the 2014-2015 industrial robot supply divided by regions (Asia/Australia, Europe and America) and the forecast for the next

years where the supply is expected to continue increasing.

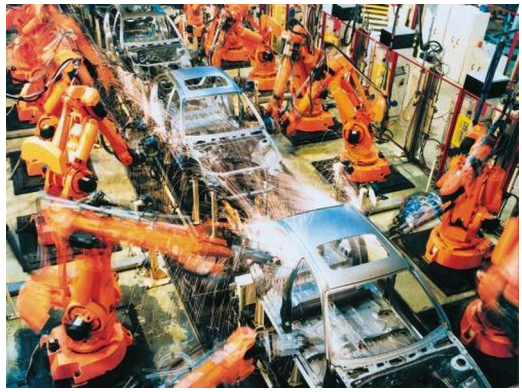


Figure 1.1: Assembly line.

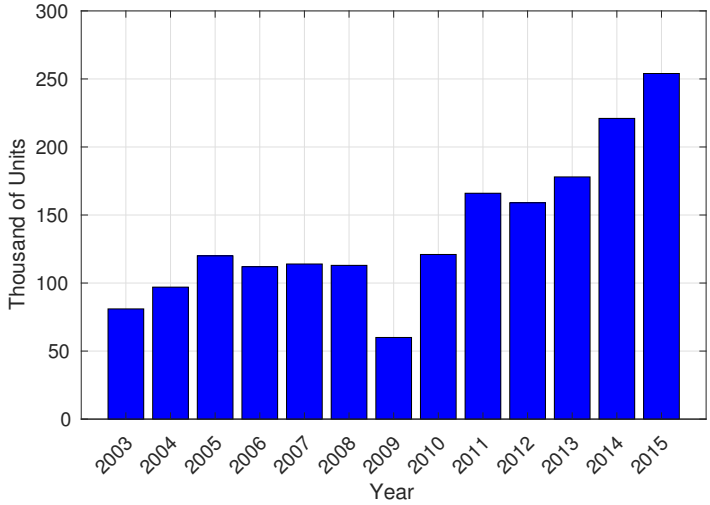


Figure 1.2: Estimated worldwide annual supply of industrial robots

While industrial robots are a fixture in the industry, advancements in safety systems, actuators and sensors are bringing robotics into new applications. Mobile robots promise to be the next frontier in service robotics. Although fixed robots will always have a place in the manufacturing industry, augmenting traditional robots with mobile robots adds flexibility to end-users in new applications.

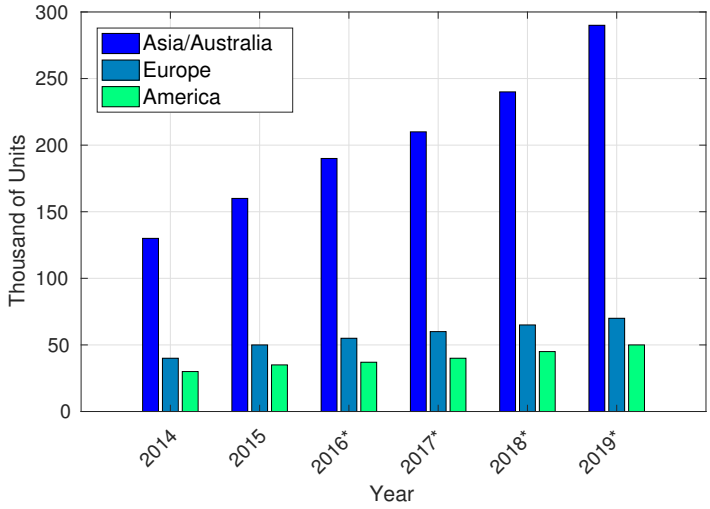


Figure 1.3: Forecast supply of industrial robots

These applications include medical and surgical uses, personal assistance, security and rescue, warehouse, distribution, and even ocean and space exploration.

Since 1998, about 220.000 service robots for professional use have been supplied. Due to the diversity lifespan of the products (underwater robots can have an average life time of 10 years in operation while defence robots may serve for short time), it is not possible to estimate how many robots are still in operation. The sales of service robots rose considerably in 2015, increasing 25 percent with respect to 2014 in professional use service robots, and 16 percent in domestic use ones. The most selling units are mobile platforms (domestic and professional use), and cleaning platforms (domestic use). Figure 1.4 shows how the supplies of service robots for professional use increase in every area in 2015 with respect to 2014. Figure 1.5 shows the forecast of service robots for domestic use and how the expected increment will be, from less than 10 million in the years 2014 and 2015 to more than 30 million in the period between years 2016 and 2019, only in household robots.

With this forecast, it is clear that mobile robotics is a field of interest, as more robots will coexist with humans in uncontrolled

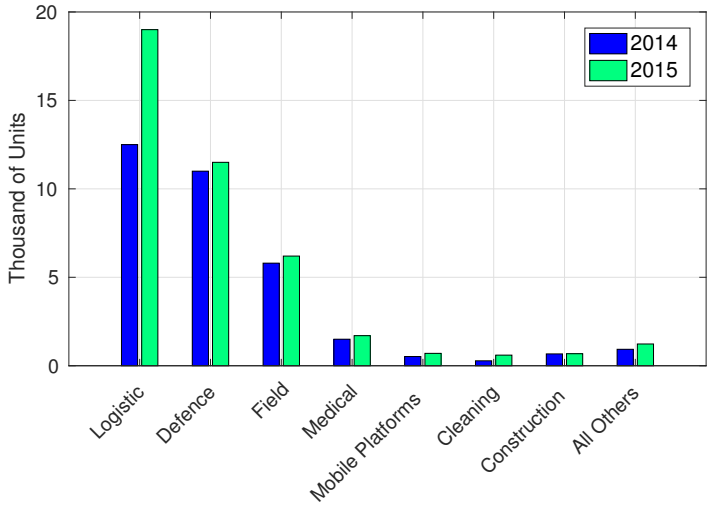


Figure 1.4: Estimated worldwide annual supply of service robots

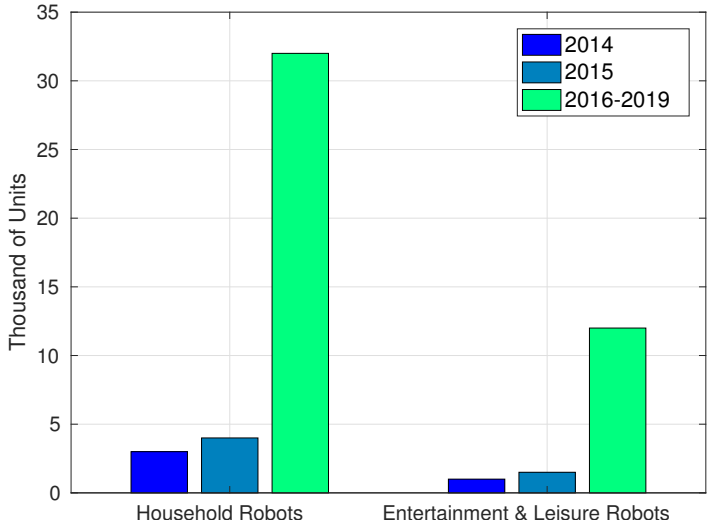


Figure 1.5: Forecast supply of service robots for domestic use

environments. In order to achieve their tasks, mobile robotic platforms need to move in the environment, at the same time that assure their security and the environment one. To achieve an autonomous navigation, a mobile robot needs to perform several stages that are interconnected: perception, localisation, planning and control. Figure

1.6 shows these stages and the connection between them.

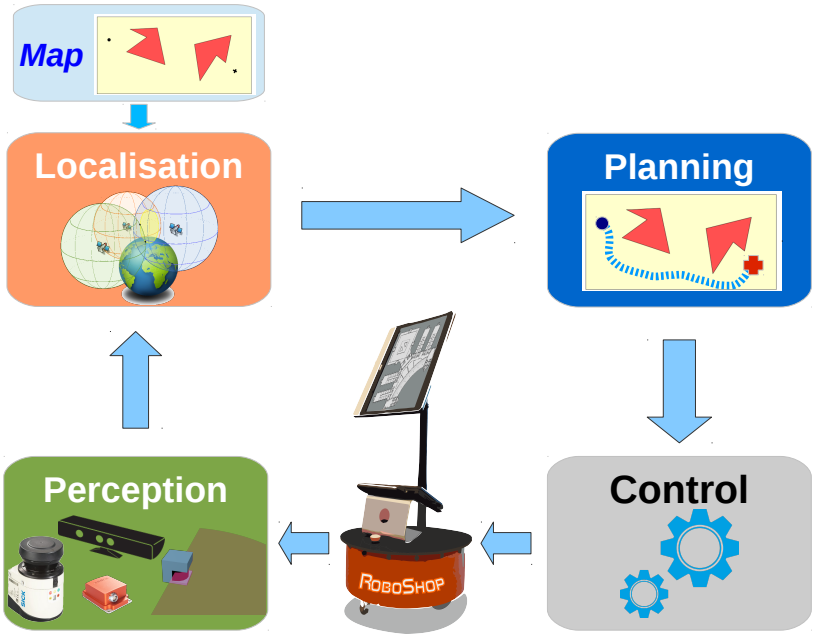


Figure 1.6: Autonomous Navigation Stages Diagram

An autonomous robot needs to **perceive** the environment, and it needs to know its own status, in order to perform the rest of stages. For this reason, it is equipped with two kind of sensors: proprioceptive and extereoceptive. Proprioceptive sensors, like odometers, gyroscopes or battery level sensors allow the robot to know its own state. Extereoceptive sensors, like Light Detection and Ranging (LIDAR), sonar, cameras or bumpers, allow the robot to perceive the environment.

Usually an autonomous robot needs different and redundant sensors to achieve the autonomous navigation task with enough safety. For example, LIDAR and sonar are range measurement sensors. While LIDAR is a more precise sensor than sonar, its limitations are different, and, for example, it can not detect glasses. For this reason, an indoor autonomous robot usually needs to be equipped with both sensors.

Sensor fusion techniques are generally used for working with dif-

ferent kind of sensors, improving the weakness of each one. Also, the information obtained from the sensors needs to be processed and interpreted in a way that the autonomous navigation systems can work with (for example, reducing the complexity of a depth image in a way that can be computed by the obstacle avoidance algorithm).

Localisation stage answers the question "Where am I?". The localisation can be performed with two different approaches, using a map or not. If the localisation is made without the use of maps, is commonly known as Dead Reckoning, and allows the system to know where the robot is at each time with respect to its origin. This localisation is made with one or more proprioceptive sensors like odometer, gyroscopes and accelerometers, or extereoceptive sensors like compass or Global Positioning System (GPS).

Localisation using maps needs extereoceptive sensors to know where the robot with respect the map coordinates. Common techniques used in this localisation are Adaptive Monte Carlo Filter [Thrun et al., 2001] and Iterative Closest Point [Biswas and Veloso, 2012] which use range information obtained from the sensors and match this information with an a prior built map, aided with the dead reckoning navigation, in order to obtain the position of the robot inside the map.

Maps can be built before the navigation, for example, using architectonic maps. But the environments, even the indoor ones, tend to slightly change with time. For this reason, maps can be more accurate if they are refined as the robot moves, adding information from the sensor measures. Also, there are techniques, known as Simultaneous Localization And Mapping (SLAM) [Newman et al., 2006] [Bekris et al., 2006], that build the map with the information from the sensors at the same time that localise the robot within the incrementally built map.

Once a map is built and the robot is localised into it, to achieve an autonomous navigation the robot needs to execute the planning stage. This planning stage can be divided into global path planning and obstacle avoidance (or local navigation). Global path planning provides a series of way points that the robot must reach in order to achieve its final goal. This can be a slow task, depending on the

complexity of the environment and the robot.

While the robot is moving, unexpected obstacles can be detected by the sensors. **Obstacle avoidance** algorithms are in charge of reacting to these unexpected obstacles and avoiding them, assuring the integrity of the robot and the surroundings, at the same time that the robot tries to reach the way points obtained from the **global path planning**. Classic obstacle avoidance algorithms, like Vector Field Force (VFF) [Borenstein and Koren, 1989] and Potential Fields (PFs) [Hwang and Ahuja, 1992], work in a merely reactive way with the distance readings obtained by the sensors. These algorithms are light and fast but tend to produce not optimal motions and get stuck in local minima. Modern algorithms, like Beam Curvature Method (BCM) [Fernández et al., 2004], Nearness Diagram (ND) [Mínguez and Montano, 2004] obtain more information from the surroundings in order to plan the movement avoiding the obstacles, while obtaining smooth paths and avoiding the local minima problem. There are also algorithms that fuses the global navigation and obstacle avoidance stages ([Montemerlo et al., 2009], [Kushleyev and Likhachev, 2009]) using the local information to replan the whole path in a faster way.

Finally, **Control** stage is the one in charge of executing the commands obtained from the planning stage. This stage is related to the actuators of the robot and its complexity depends on the kind of driving of the autonomous platform.

In the last years, autonomous navigation for robots and even vehicles has been improved with international competitions like DARPA Grand Challenge [Thrun et al., 2006] and DARPA Urban Challenge [Montemerlo et al., 2009], where car like robots were able to navigate autonomously reacting to unexpected obstacles. Figure 1.7(a) shows the winner of the 2003 edition and Figure 1.7(b) shows the winner of the 2005 edition. These robots are equipped with state of the art sensors and powerful computers and move in scenarios of known characteristics (for example, the traversable road is delimited with curbs). However, achieving autonomous navigation in previously unknown environments, and without this kind of expensive and redundant sensors, is still a challenge.

Figure 1.7: DARPA Challenge Autonomous Robots

1.2 Scope of this thesis

This thesis is placed within several projects: Abstraction, Synthesis and Integration of Information for Human-Robot Teams (ABSYNTHE)[1] [Alonso et al., 2012], Plataforma Robótica Para La Investigación (PROPINA)[2] and ROBOtic Guide for Shop (RoboShop)[3] [Ocaña et al., 2014].

ABSYNTHE project main goals are the development of concepts, tools, and approaches for the collaborative production and application of high-level semantic descriptions of computational objects with a view to facilitate the joint intelligent processing and exploitation of knowledge by mixed and heterogeneous teams of human and robots. Some of the problems that the project needs to deal with are:

- **Qualitative Object Description**: the collaboration between team members requires the ability to exchange information at various levels of abstraction.

- **Cooperative Robotics**: the collaboration between humans and robots requires the capacity to map of perceptual information into symbols describing environmental features (topological maps).

[1] Ministry of Economy and Competitiveness funded, code TIN2011-29824-C02-02

[2] University of Alcalá funded, code UAH2011/EXP-013

[3] University of Alcalá funded, code CCG2013/EXP-066

- **Autonomous navigation in complex environments**: it is needed that the robots in the team can navigate in realistic scenarios, with people moving around them and even changeable environments.

- **Integration of high-level sensor information**: the perceptual information can be collected by distributed, multi-modal and different characteristics sensors used by the team members (humans and robots). So the developing of techniques that integrate and fuse this information are needed.

This thesis is focused on the third point, the development of an autonomous navigation system that can react to moving and dynamic objects in the surroundings of the robot.

With the experience of the real experiments made in the Robotics and eSafety Research Group (RobeSafe)[4] with commercial robotics platform, PROPINA project proposes the design and development of our own robotic platform that can improve some of the flaws that the commercial platforms have.

RoboShop project goal is to build a Robotic Guide for Shops. For this reason, PROPINA platform is used and equipped with more sensors (LIDAR, depth cameras, RGB cameras, Inertial Measurement Unit (IMU), etc.) to improve its usability. This Robotic Guide needs to cooperate with humans and needs to navigate autonomously in challenging environments.

1.3 Proposal

Since 2004, the researchers of the RobeSafe research group at the Department of Electronics have been working in the autonomous navigation scope. Several important results were achieved in this scope. Some important works were, the development of a robot navigation system for indoor environments using signal strength from WiFi measurements and ultrasounds [Ocaña, 2005] [Ocaña et al., 2005], and the

[4] www.robesafe.com

development of a robot navigation system based on Partially Observable Markov Decision Process (POMDP) using vision and proximity sensors [López, 2004] [López et al., 2005].

The RobeSafe research group has focused its efforts in some important aspects in order to develop these robot navigation systems. RobeSafe research group is interested in developing non-invasive systems, which means to use the own infrastructure of the environment without adding extra devices or technologies. Finally, developing solutions that work in real scenarios is always a premise for RobeSafe research group.

The aim of this thesis is to develop a navigation system that can deal with dynamic obstacles and, even, changeable environments. This navigation system will not only assure the robot and environment safety, but also will produce smooth and fast trajectories. In order to do this, a perception system that can detect the variations in the robot's surroundings is needed. The navigation system only uses the onboard sensors of the robot (especially LIDAR, IMU and odometry) in a way that beacons or devices in the environment are not needed.

1.4 Document structure

After the introduction in Chapter 1, Chapter 2 contains a review of the most significant research in autonomous robot navigation, including obstacle avoidance and global path planning algorithms.

In Chapter 3 a review of several commercial robotics middlewares is done. And, the proposal, development and simulation of our own robotic platform RoboShop.

In Chapter 4, methods for autonomous navigation in dynamic environments are proposed. From local mapping stage, obstacle avoidance algorithms to global path planning ones. In Chapter 5 these algorithms are tested and evaluated, both in simulation and in real scenarios. Finally, Chapter 6 shows the conclusions and future work derived from this thesis dissertation.

Chapter 2

State of the Art

The main task of an autonomous mobile robot is to be capable of reaching several points in the environment, while assuring its own integrity and the environment safety. This task is known as navigation. Navigation can be divided into two different and complementary categories: global path planning and local navigation. Global path planning is the task that provides a series of positions or configurations that the robot must reach in order to reach a goal. To achieve this task the map needs to be known or, at least, partially known. As the robot moves, unexpected obstacles that have not been mapped before, or changes in the previously mapped environment, can happen and have to be measured by the available sensors (usually onboard the robot). These obstacles can block or affect the previously calculated path, so the robot must alter its route to the goal in order to avoid these unexpected obstacles, at the same time that the navigation has to be completed. This task is known as local navigation or obstacle avoidance.

The line that separates local navigation and global path planning is sometimes very blurry. In order to clarify the main differences between them, these characteristics are pointed:

- **Local navigation:**
  - It uses local information, usually obtained from onboard sensors.

  - It plans the next commands for the robot to execute (immediate or short time ahead).
  - It has near real time requirements. The algorithms are fast and work as reactive processes.
  - It allows the robot to travel safely.

- **Global path planning:**
  - It uses known or partially known maps.
  - It plans for long distance and time periods.
  - It is a slower and deliberative process.
  - It allows the robot to avoid getting trapped in local minima scenarios, and fulfil its task.

However, in recent studies some of these classical characteristics are relaxed. For example, some local navigation algorithms use cumulative sensors data, or fuse with known maps, in order to plan the next optimal movement. Also, some global path planning algorithms can react to unexpected obstacles and replan the path in real time scenarios, fusing the local and global navigation in the same process.

In the next sections a brief summary of several state of the art techniques that are used in local navigation and global path planning stages is performed. As the navigation task needs the perception and mapping stages, a brief summary of some helpful techniques is also included.

2.1 Perception and Mapping

Before the navigation stage can be achieved, the robot needs to know the environment where it moves. For this reason the perception and mapping stages must be performed. In this section a brief summary of these stages is done.

Perception is usually addressed with the onboard sensors of the robot, even when it can be complemented with intelligent environment sensors. Commonly used sensors in mobile robotics are: sonar

sensors, LIDAR sensors, colour cameras, depth cameras, infrared sensors, etc. For local navigation purposes, the main information needed is if an area is traversable or not, or, what is the same, where are the obstacles surrounding the robot. For indoor mobile robots, it is assumed that any detection in the same height of the robot is an obstacle and the robot should not collide with it. For this reason the most used sensors are range only sensors (sonar, infrared, LIDAR), situated parallel to the floor. Nevertheless, for outdoor mobile robots and for some complex robots or environments, 3D information is needed. This information can be obtained from 3D-LIDAR, moving 2D-LIDAR, 2D-LIDAR positioned not parallel to the floor, stereo cameras, etc.

In order to perform the local navigation or global path planning, a representation of the environment (**map**) is needed. Map can be local (it represents the surroundings of the robot, most suitable for local navigation) or global (it represents the whole environment where the robot moves). These maps can be built based on another map, based on measures from the sensors, based on expert knowledge, etc.

Usually, indoor mobile robots move in planar environments that can be represented as 2D maps. These representations can be classified into topological and metric maps [Thrun, 1998]. Topological maps represent important zones and the relations between them. These maps tend to be simple, close to the human interpretation of the environment, they are very useful for top level tasks or path planning, but they are not suitable for local navigation. On the other hand, metric maps represent a continuous space discretised in a way that can be used by a robot. Figure 2.1 shows the same map represented in both ways: topological map represents each room and the transition between them and metric map divides the continuous map in cells of the same size.

Building a metric map from continuous information needs a discretisation. One effective way to perform the discretisation in 2D environments are the cell decomposition methods. These methods divide the environment in cells that are connected with the adjacent ones and they allow to represent whether each cell is occupied (not traversable by the robot) or free (traversable by the robot). Even a

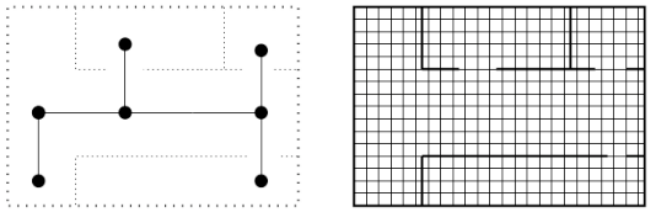


Figure 2.1: Topological (left) and metric (right) mapping

level of occupancy can be represented in a way that the loss of information caused by the discretisation, or the inaccuracies of the sensors measures can be directly used by the robot [Coué et al., 2006]. Some methods for cells discretisation are:

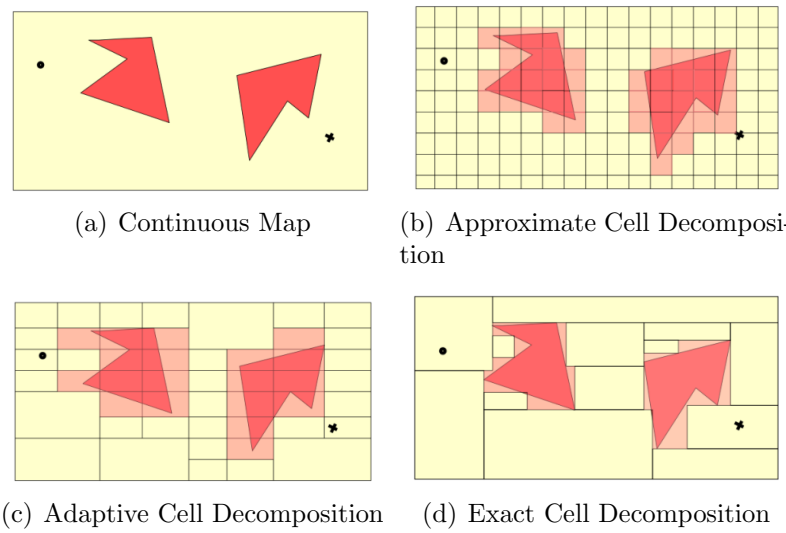
- **Approximate cell decomposition** [Latombe, 1991a]: the environment is divided into cells of the same size and shape. It is an easy process but needs a trade-off between loss of information (the smaller the cell is, the lesser information will be lost) and size of the map generated (the smaller the cell is, the bigger the resulting map will be).

- **Adaptive cell decomposition:** the environment is divided into cells of different size. One particular case of adaptive decomposition is the quadtree [Samet, 1988]. The method of dividing an environment into quadtrees begins with one cell that represents the whole map. If that cell is partially occupied, it is divided into four ones. That division is repeated on each cell until the resolution limit is reached or there are not partially occupied cells in the environment. This representation allows to have the same information as approximate cell decomposition using less memory.

- **Exact cell decomposition:** the cells does not have a predefined size or shape and are determined based on the environment. The joins between cells represent the free space in the map.

Figure 2.2 shows the three previously explained methods for cell

discretisation applied to the same map where yellow colour represents free space, red colour represents obstacles and light red colour represents occupied cells after the discretisation.



(a) Continuous Map

(b) Approximate Cell Decomposition

(c) Adaptive Cell Decomposition

(d) Exact Cell Decomposition

Figure 2.2: Cells Discretisation Techniques. Credits to M. Huber

The grid cell representation can be extended to full 3D maps, where the representation is usually based on VOLumetric piXELs (VOXELs) [Foley et al., 1994] instead of grid cells and it is useful for 3D navigation, as Unmanned Aerial Vehicles (UAVs) and Unmanned Underwater Vehicles (UUVs) need.

Also, the grid map representation can contain more information apart from the occupancy value, allowing to fuse topological and metric maps by means of adding information, for example, about features on the environment that are situated inside each cell coordinate limits.

2.2 Local Navigation

As a robot is moving, trying to reach a position or follow a path, unexpected obstacles, which have not been mapped before, could

appear and be detected. Therefore, the robot needs to modify its path in order to assure the surroundings and its own integrity, at the same time that the assigned task is addressed. In this section a review of the state of the art techniques used for local navigation is performed.

The simplest algorithms for local navigation are the **bug based algorithms**, such as Bug1, Bug2 (both algorithms in [Lumelsky and Stepanov, 1987]) and Tangent Bug [Kamon et al., 1996]. The behaviour of these algorithms is very simple: if an obstacle appears in the path of the robot, the robot follows the obstacle edge until it is avoided. These algorithms are very inefficient because the shortest or smoothest path to the goal is not calculated, and also potentially risky paths are performed because the robot do not react with enough anticipation to the obstacles. On the other hand, these algorithms can be used with very simple sensors such infrared or tactile ones. Figure 2.3 illustrates how the Bug2 algorithm works. The robot must follow the line that connects its initial position to the goal, but when an obstacle appears, it is surrounded until the robot crosses with the initial path and returns to it.

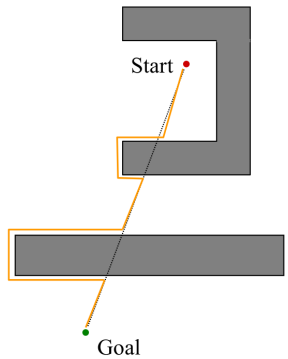


Figure 2.3: Bug2 Example. Credits to H. Choset

Another approach for local navigation is the **vector summation based algorithms**. In this approach, each obstacle create a repulsive force around it at the same time that the goal creates an attractive force toward them. The addition of all these forces creates a safety

path without obstacles which the robot can follow. Some implementations are VFF [Borenstein and Koren, 1989] and PFs [Hwang and Ahuja, 1992]. These algorithms are computationally simple and can work in controlled environments but they have several problems with some configurations of obstacles that can cause local minima and block the movement of the robot. In addition, they are suitable for omnidirectional robots, but, as they do not take into consideration the kinematics and dynamic restrictions, they do not work well in non-holonomic robots. Figure 2.4 shows the vector summation of the VFF algorithm, where the repulsive forces caused by the obstacles (F_r), the attractive force of the goal (F_t) and the resulting direction that the robot must follow R, are represented as vectors.

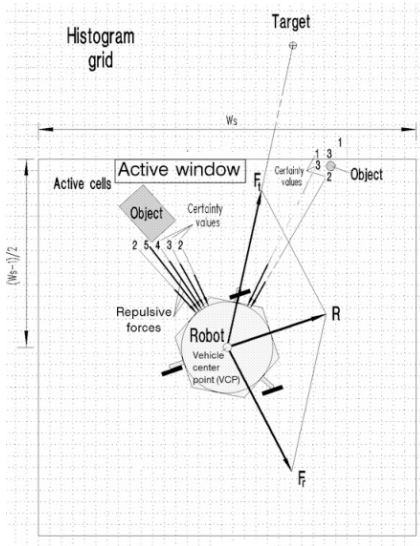


Figure 2.4: VFF Example. Credits to J. Borenstein et al.

As indoor robots move in planar floors, the local navigation can be calculated in a cartesian space (x,y coordinates). There are techniques that use other searching spaces, like the **histogram based methods**, which divide the surroundings of the robot into angular sectors that can be transformed into a polar histogram. In this histogram, the proximity of an obstacle on each angular sector is represented, and it is easier to calculate the next orientation of the robot

than in cartesian coordinates. Next orientation is obtained based on a cost function that have parameters such as security or distance to the goal. Algorithms of this kind are Vector Field Histogram (VFH) [Borenstein and Koren, 1991] and its extension Vector Field Histogram Plus (VFH+) [Ulrich and Borenstein, 1998]. These algorithms can deal better with uncertainties in the measurements and take into account kinematics restrictions of the robot. But they have some potential local minima problems, especially with "U" shaped obstacles. Figure 2.5 shows the transformation between the metric map (active window, shown below) to the polar histogram, where the height represents the proximity (and riskiness) of an obstacle, and the next action selection is performed.

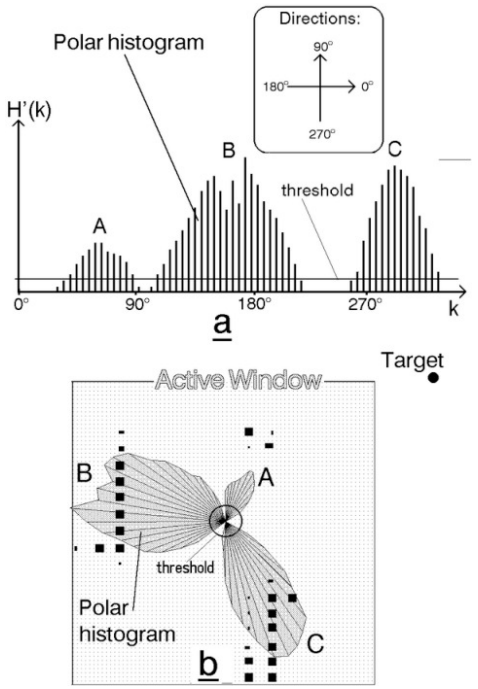


Figure 2.5: VFH Example. Credits to J. Borenstein and Y. Koren

Velocity Methods plan the next movement of the robot in the velocity space configuration instead of the cartesian one. In this way, velocity space maps the cartesian space (X-Y) into one that represents the linear and angular velocities of the robot, which made this

very suitable for navigation in differential and holonomic robots, as a point in the velocity space corresponds to a velocity directly executable to the robot. In this transformation, the obstacles in the environment block some velocities and reducing the searching space. At each iteration, the algorithm selects the next reachable velocity to command to the robot using a cost function with parameters like security or smoothness. Some well known implementations of this algorithms are Dynamic Window Approach (DWA) [Fox et al., 1997] and Curvature Velocity Method (CVM) [Simmons, 1996]. Figure 2.6 shows the transformation between the cartesian space and the velocity space in DWA implementation, where the obstacles detected in the cartesian space (semicircle in left image) block possible velocities in the velocity space (dark areas in right image). These velocity methods are ones of the most used in mobile robotics but, as purely reactive methods, they have some local minima issues.

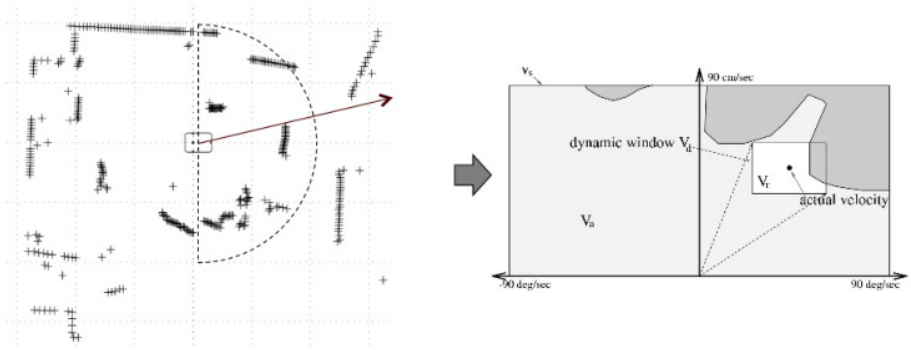


Figure 2.6: DWA Example. Credits to D. Fox et al.

As the previous algorithms are merely reactive ones, they have potential local minima issues. In order to tackle this problem, another family of algorithms **classifies the environment** into different scenarios and uses different strategies of avoidance for each one. Implementations of these algorithms are ND [Mínguez and Montano, 2004] and Smooth Nearness Diagram (SND) [Durham and Bullo, 2008]. Figure 2.7 shows the scenarios classification for ND algorithm. The big issue of these algorithms is the complexity, due to several avoidance strategies have to be implemented, and the needing of good

perception stage in order to classify the scenarios correctly.

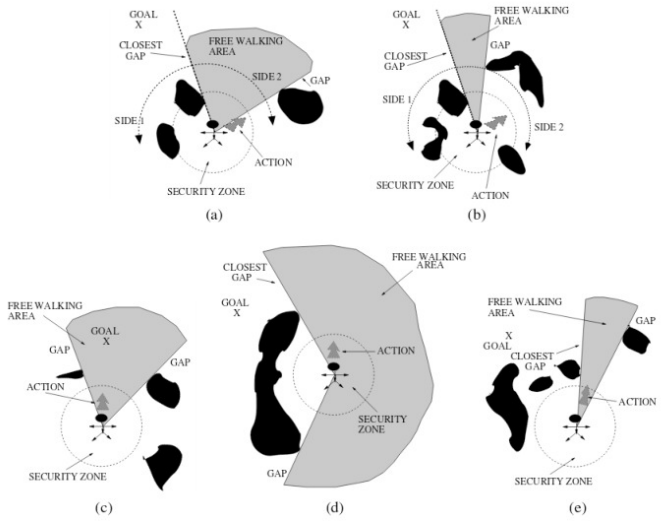


Figure 2.7: ND scenarios classification. Credits to J. Minguez and L. Montano

Another idea in order to avoid the local minima problem is to **mix different techniques** and create **two different planners**: one that selects a local goal within the surroundings of the robot and one that selects the next movement to reach this goal. Some of these methods are Lane Curvature Method (LCM) [Ko et al., 1998] and BCM [Fernández et al., 2004]. Both implementations obtain a local goal by means of dividing the surroundings of the robot (LCM into lanes, BCM into angular sectors) and they use CVM algorithm to reach this goal. Figure 2.8 shows the sectorisation of the free space (lighter areas) into lanes of the same direction using the LCM method. These methods are computationally more complex than the previous ones, as a two stage local navigation is performed.

Instead of these deterministic or probabilistic approaches, there are a family of algorithms that uses **soft-computing** techniques to perform the local navigation. Some of these algorithms are based on Fuzzy Logic like [Faisal et al., 2013] [Dongshu et al., 2011] [Liang et al., 2011] [Lee et al., 2012] [Mbede et al., 2012] [Tang et al., 2013] or Neuro-Fuzzy Logic like [Mohanty and Parhi, 2012]. These algo-

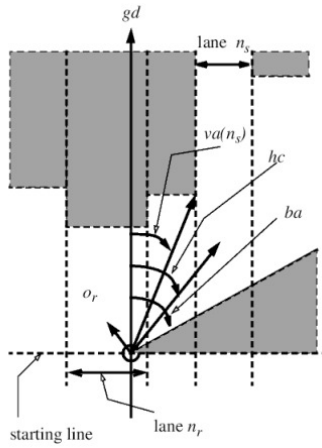


Figure 2.8: LCM Example. Credits to N. Y. Ko et al.

rithms are easy to understand and to code but usually perform a merely reactive behaviour without achieving the best path to avoid the obstacles.

It is assumed that the reviewed methods can achieve the local navigation, even in dynamic and changing environments where moving obstacles are. But this is only true under certain assumptions, especially that the robot can move faster than the obstacles. With this constrain, it is possible to avoid a moving obstacle in the same way than a static one. However, not dealing with the dynamic obstacles in a different way results in avoidance strategies that are not optimal and potentially risky, like moving toward approaching obstacles or crossing the robot and obstacle paths.

To tackle this problem, there are algorithms that avoid the dynamic obstacles in a different way than the static ones. Apart from the perception of the velocity of the obstacles (which is not a matter of study in this dissertation), the first step is to calculate when a collision with a moving obstacle is possible. Work in Velocity Space is a method to deal with it. In [Fiorini and Shiller, 1993] the **Collision Cone** concept is proposed, a transformation between the cartesian positions where two mobile objects can collide into the velocity space. Based on this idea, a family of implementations known as Velocity-Objects algorithms appears. Figure 2.9 shows graphically

the collision cone concept. The Collision Cone proposition only works with circular objects and it has been extended by [Chakravarthy and Ghose, 1998] to any irregular object. Some Velocity-Obstacle based methods are [Choi, 2014] where the collision cones are used to compute the path trajectory, [Masehian and Katebi, 2014] uses the collision cone to create angular sectors centred in the robot and choose the safer direction, [Alsaab and Bicker, 2014] where a simplified extension of the collision cones for irregular obstacles is used and [Zhong et al., 2011] where time constraints are introduced for rapidly planning. The big weakness of these methods is that the collision cone can only be calculated if the movement of the obstacles is linear.

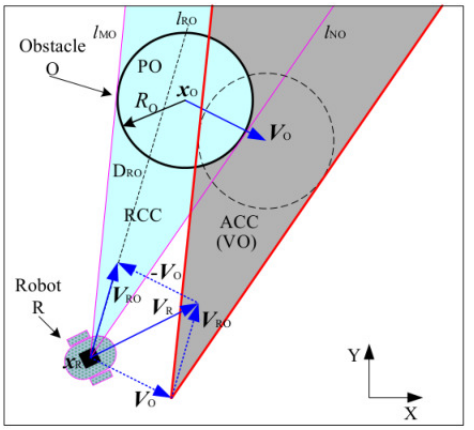


Figure 2.9: Collision Cone Example. Credits to X. Zhong et al.

There are also algorithms for local navigation that take dynamic obstacles into account without using the collision cone concept. [Yaghmaie et al., 2013] proposes an extension of the PFs algorithm, where the relative velocity of the obstacle affects the vector summation in order to avoid it. [Benzerrouk et al., 2012] uses the relative velocity of the obstacle to choose the direction of avoidance.

The previous dynamic obstacle avoidance algorithms work with linear and instantaneous velocities of the obstacles. [Seder and Petrovic, 2007] proposes an extension of the DWA algorithm (Time Variant Dynamic Window (TVDW)) where the collisions are checked cell by cell in a grid map. It allows the avoidance of moving obstacles even

if the movement is not linear, but increments the complexity of the searching (Figure 2.10 shows the cells intersections in the TVDW algorithm).

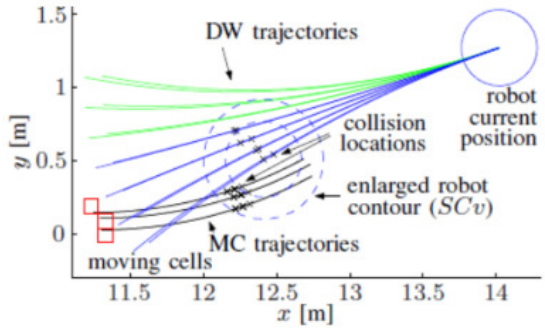


Figure 2.10: TVDW Example. Credits to M.Seder and I.Petrovic

The main disadvantage of these methods, is that they guarantee an avoidance of the dynamic obstacles, but the safer and usually longer avoidance path is not achieved. In that way, the perception error in the velocities of the obstacles can affect these algorithms in a critical way, crossing the robot with the obstacle's path. Therefore, if the obstacle increases its velocity or it is incorrectly perceived, it can crash with the robot.

Lastly, there are algorithms that mix the local and global navigation. They use not only the next movement according to the current state of the robot, but also the future possible positions (in a local window). Some algorithms of this kind are Vector Field Histogram Star (VFH*) [Ulrich and Borenstein, 2000] and Dynamic Window Approach Star (DWA*) [Chou et al., 2011] where a tree of classic algorithms executions (VFH+ and DWA respectively) is built and the next movement is selected according to the whole path cost, and Vector Field Histogram with Time Dependent Tree (VFH*TDT) [Babinec et al., 2014] that modifies VFH* including the time in the planning to avoid moving obstacles.

Table 2.1 shows a brief summary of the methods studied, and the pros and disadvantages of each family of algorithms.

Method	Pros	Disadvantages
Bug Based	Easy to implement. Simple sensors.	Non optimal path. Potentially risky.
Vector Summation	Easy to implement. Suitable for omni-directional robots	Local minima issues Not Suitable for non holonomic robots.
Histogram Based	Easy to calculate next direction. Includes kinematic restrictions.	Local minima issues
Velocity Space Based	Includes Kinematic restrictions. Suitable for differential and holonomic robots.	Local minima issues
Soft-Computing Based	Easy to implement. Human-like behaviour.	Non optimal path.
Environment Classification	Reduce local minima.	Complexity. Need good perception stage.
Two-Stage Planning	Reduce local minima problems.	Complexity. Slower methods.
Medium Time Planning	Most optimal path are calculated.	Complexity. Slower methods.
Velocity-Obstacles Based	Avoid dynamic obstacles.	Only linear velocities obstacles.

Table 2.1: Local Navigation Algorithms Summary

In this section a review of the local navigation techniques has been performed. However, it is important to remark that there is not a perfect algorithm that solves all the local navigation problems. The simpler algorithms can be used in smaller and computationally less powerful robots, even when the performance is worse than other more complex ones. It is also important to remark that in real scenarios is needed to avoid the dynamic obstacles taking into account their velocities and directions. In the literature some algorithms take this into account, but they deal with the immediate collision, and not the whole path that the obstacle is following.

2.3 Global Path Planning

Given initial and goal states, the global path planning calculates a sequence of configurations that connects both states and avoids colliding with any obstacle. Global path planning can be used in a variety of robotic fields (including industrial arms, for example) but, in this section, some techniques that are used for path planning in mobile robots are surveyed.

An Unmanned Ground Vehicle (UGV) can be considered as a solid rigid that moves in a 2D environment. So its configuration space is the cartesian coordinates (x and y) and its orientation. This configuration space can be increased if more variables are included like velocities, time, etc. The quality of a global planner can be measured with some parameters like:

- **Completeness:** One algorithm is complete, if it is guaranteed that the solution is found.
- **Optimality:** One algorithm is optimal, if it is guaranteed that the best solution (according to the evaluation function) is found.
- **Time/Space Complexity:** the time and memory that the algorithm needs as the number of possible paths/states grows.

In mobile robotics, time/space complexity is an important constraint, due to the systems real or near real time requirements. Also it is important that the completeness characteristic is achieved. The

optimality of a global planner is a characteristic that can be relaxed: in real applications it is better to find a path that is not optimal (sub-optimal) faster than a better path with time longer computation.

If it is assumed that the robot moves in a 2D environment represented as a grid map, path planning can be performed. The best path that connects the two points can be evaluated using different metrics: less cells traversed, less turns performed, more distance to the obstacles, etc.

One planning method for this grid maps is the Wavefront Based Algorithms. This family of algorithms calculates the value of each cell as the distance from the goal and propagates backward to the start point. Once the start position is reached, the planner can step back the cells with the less value in order to reach the goal. Some implementations are NF1 [Lengyel et al., 1990], NF2 [Latombe, 1991b] and Trulla [Murphy et al., 1999]. Figure 2.11 shows the path planned using a Wavefront algorithm in a grid map environment.

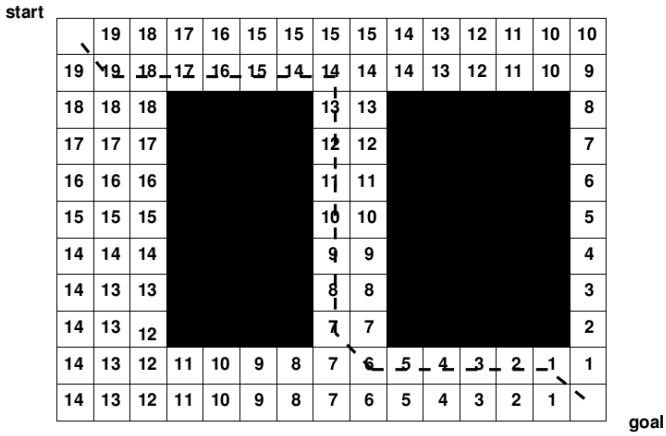


Figure 2.11: Wavefront Example

Another planning method for grid based maps is the Global Potential Fields (GPFs) [Khatib, 1986]. Like the PFs algorithms for local navigation, each obstacle creates a potential field around it (greater values near the obstacle) and the goal creates another attractive potential field (lesser value near the goal). All the potentials are

added, and the path is reconstructed as a descending gradient from the start to the goal. This method is prone to local minima problem. Figure 2.12 shows the local minima issue: when all the potential fields are added, there is a point with a lesser value surrounding with bigger values in the way from the goal to the beginning.

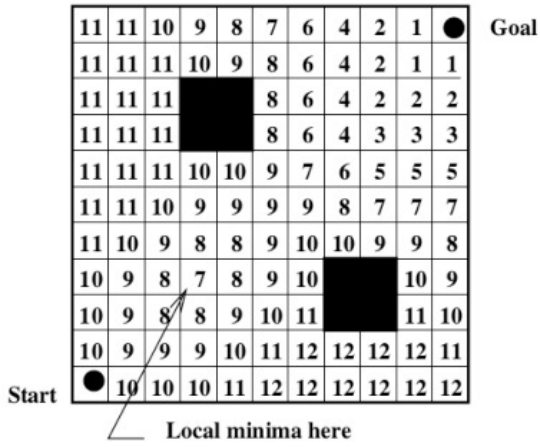


Figure 2.12: Global Potential Field. Local Minima Problem Example

One of the problems with the global path planning in grid maps is that the result is an unnatural path to follow, as it only considers 4 or 8 possibles changes of directions. There are techniques to smooth the path in order to be more natural to follow, but they need a two step algorithm and the computational cost is increased. On the other hand, the search space is reduced (x-y coordinates) and the path planning can be performed in a quick way.

To cope with the unnatural path problem, and to reduce the search space at the same time, roadmap based representations appear. These methods find the connection between the free space as a set of lines or curves. Some methods based on the roadmap representation are:

- **Visibility Graphs** [Nilsson, 1984]: connect each vertex of the obstacles in the environment with all the other vertex if the line does not cross any obstacle. Figure 2.13(a) shows an example of visibility graph and a path planned based on it.

- **Voronoi Diagrams** [Choset and Burdick, 1996]: the roadmap consists of paths which are equidistant from all obstacles in the region. These paths merges in vertices forming a full roadmap where a planning stage can be executed. Figure 2.13(b) shows a Voronoi diagram with the path planned.

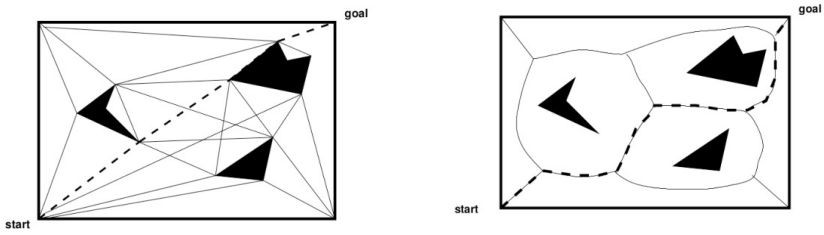


Figure 2.13: Visibility Graph and Voronoi Diagram Example

- **Probabilistic Roadmaps** [Kavraki et al., 1996]: this technique tries to reduce the complexity of a continuous space into a discrete space, but instead of using characteristics of the environment, like Voronoi Diagrams or Visibility Graphs, it works creating nodes in the space and checking if these nodes can be connected. One commonly used variation of this technique is the Rapidly Exploring Random Trees (RRT) [LaValle and James J. Kuffner, 2001] which are more suitable for a rapid search and it captures the dynamic or even non-holonomic constraints. [Pepy et al., 2006] uses an RRT with kinematic constraints to plan a directly executable path for a mobile robot.

- **State Lattice** [Pivtoraiko and Kelly, 2005]: this technique fuses the planning on configuration and cartesian states. The configuration space is discretised into configurations that are reachable from the previous one and they do not cross any obstacle. In that way, it is very useful to add dynamic and not holonomic constraints and to plan directly executable paths to a robot. Figure 2.14 shows a state lattice planner for a planetary rover.

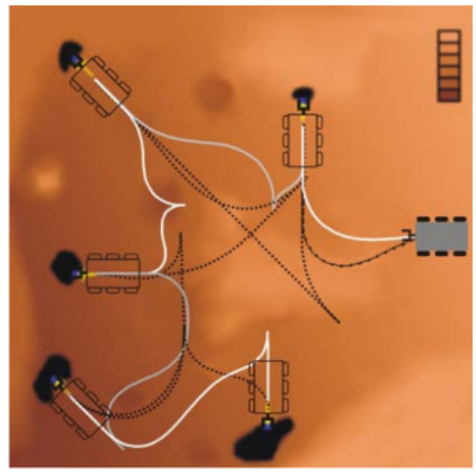


Figure 2.14: State Lattice Planning Example. Credits to M. Pivtoraiko and A. Kelly

The former methods (even the grid cell based ones) can be represented into a tree. This tree contains the possible configurations, the transition between them and the cost of performing this transition from a state to another. Figure 2.15 shows a typical decision tree, where each node can have an arbitrary number of successors, but each one only has one predecessor.

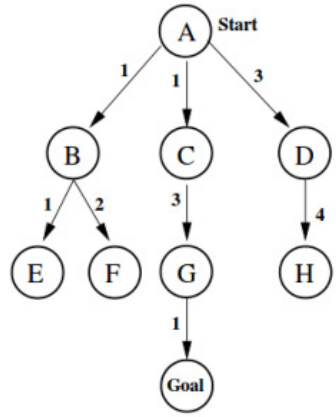


Figure 2.15: Decision Tree Example

In order to search the best path into the tree, a graph search algorithm is needed. These algorithms are also used in a wide variety of applications like artificial intelligence, computer networking, etc. Some of these algorithms are the following ones:

- **Depth-first search:** a branch is selected where the child is the better of its level and it is explored until the branch ends. If the goal is not reached, that particular branch is eliminated and the process is repeated. This search is simple and works well when there are multiple solutions to a problem, but does not guarantee that the best solution is found.

- **Breadth-first search:** this search evaluates all the nodes at the same level in parallel. It guarantees that the optimal path is found and it works better than the Depth-First Search when there are a small number of solutions. Nevertheless the whole set of possible paths needs to be evaluated increasing the time required.

- **Iterative deepening** [Korf, 1985]: this search is an iterative process of depth-first search with level limit. The optimal found tree can be found before the whole tree is explored, being faster than a breadth-first search, but it is slower than the depth-first search if there are many solutions.

- **Uniform-cost search:** if there is a different cost to travel from one node to another in the graph, this solution can be used. It is similar to the breadth-first, but the nodes of same cost are evaluated instead of the nodes within the same level in the tree. One particular case of this search is the Dijkstra's Algorithm [Dijkstra, 1959]. In that implementation (for global path planning) the cost of each node represents its distance to the goal.

In real applications, creating and computing the whole graph could be not possible due to the size or complexity of the environment. In these cases, an heuristic search is used. These algorithms can not guarantee to find the optimal path to the goal but find, at

least, a suboptimal executable path. They use an evaluation function to score each node or state so a guided search can be performed. Some heuristic search algorithms for path planning are the following:

- **Best-first search** [Winston, 1992]: it is depth-first algorithm where a heuristic is used, and indicates how far is each state from the goal. At each time, the best node in the frontier of evaluating nodes is selected and expanded. If the heuristic used is the distance from the node to the start the algorithm becomes an Uniform-Cost Search. If the metric is an estimate of the distance to the goal it becomes to a greedy search. Best-first search is faster than Dijkstra's algorithm but ignores the cost of the whole path generated.

- **A* Search** [Hart et al., 1968]: it is one of the most widely used search algorithm in robotics. It works in a similar way than the best-first search but the heuristic used is a combination of two metrics: the distance from the start and the estimated distance to the goal. The heuristic that estimates the cost of the node must be admissible. If the metric of the cost to the goal is stronger than the distance from the start, the algorithm works like a Dijkstra's algorithm and it results in the optimal shortest path to the goal, but the search process is longer. If the weight distance from the start is the strongest, it works like a Best-First Search, running faster but the optimal path may be not obtained. There are several variations of that heuristic that results in other algorithms, like Theta* [Daniel et al., 2014] and Block A* [Yap et al., 2011].

Even if these Heuristic searches are able to speed up the searching, when the environment is changing or partially unknown the replanning stage could not satisfy real time constraints. For this reason, algorithms for continuous or event driven replanning are used. One of the most used algorithm is the D* Algorithm [Stentz , 1994], an extension of the A* Algorithm. D* computes several paths at the start of the execution using an A* Algorithm and, if the map changes, is able to switch from one path to another and it repairs the previous one.

Several variants of A* and D* have been developed which are able to speed up the search, like Anytime A* [Hansen and Zhou, 2011], Anytime Repaired A* (ARA*) [Maxim Likhachev and Thrun, 2004], Focussed D* [Stentz, 1995], Framed Quadtree D* [Yahja et al., 1998], D*Lite [Koenig and Likhachev, 2002] and Constrained D* [Stentz, 2002].

The inclusion of dynamic obstacles into the global planning (or replanning) stage increments the search space. For example, if we consider the planning of a point-type agent in a grid type environment, it could be done in a 2D state space (x,y cartesian coordinates) but if we include the time, the complexity increments (3D state space, x,y cartesian coordinates and time). However, the previous techniques can be used but may not satisfy the time requirements of the real system. Nevertheless, works like [Kummerle et al., 2013] demonstrate that an autonomous robot navigation in real environments could be performed.

During the Defense Advanced Research Projects Agency (DARPA) Urban Challenge, the Stanford Entry [Montemerlo et al., 2009] proposes the use of a state lattice for planning purposes mixing with an A* Search (Hybrid A*). As the time is implicit in the tree construction, planning in that kind of roadmap does not increases the complexity of the space from a cartesian one. Figure 2.16 shows the lattice planning of an autonomous car parking. [Kushleyev and Likhachev, 2009] and [Phillips and Likhachev, 2011] extend this lattice graph planning for indoor robots, where the transition space without moving in the space is needed.

In this section a review of the global path planning for mobile robots has been performed. Some of these techniques, especially the ones based on Lattice States, have demonstrated that a navigation with dynamic environments can be performed. However, these techniques need heavy computation (at least, heavier than the local navigation techniques).

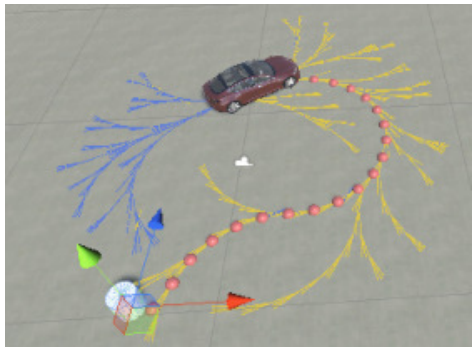


Figure 2.16: Hybrid A* Example.

2.4 Objectives

Once the state of the art has been summarized, and the flaws of the algorithms and improvement needed have been detected, the objectives of this thesis are the following:

- Study of the several robotics middlewares available. This studio will be used to select one standard development platform where the algorithms can be applied in the maximum different robotics platforms and sensors as possible.
- Study and comparison of local navigation algorithms. In this studio a real comparison of available local navigation algorithms will be performed, obtaining their pros and cons. Also, metrics for that comparison will be defined.
- Study if the local navigation algorithms can be adapted for dynamic environments.
- Obtaining the dynamics of the objects in the environment. Once the different objects are identified and classified, there is a need of modelling its movement and mapping that movement in a way that a local navigation algorithm can use it.
- Integration of local navigation algorithms for dynamic obstacles in a full navigation framework, and test in real scenarios.

Chapter 3

Architecture

In this chapter we are going to review the architecture that will be used as test bench for this thesis. First, robotics development platforms available on the market will be reviewed, compared and one of them will be selected to work with. Then, this platform will be explained in detail. We are also going to review the mobile robots available in the RobeSafe research group and we are going to explain the hardware and software development of our own mobile robot called PROPINA (and its extension as shop assistant RoboShop). Finally, a comparison between the available mobile robots will be made in order to check the advantages of our developed mobile robot.

3.1 Robotics Middlewares

Robotics has been a field of study and development through the last decades. One of the issues that robotics has is the standardisation. The standardisation, at least in the software layer, has been a matter of study in the recent years through the development of robotics middlewares.

In the past, the communication with a robot or a sensor had to be coded individually, usually using low level programming languages. This way of working with robotics has two main issues:

- The programmer must know the appropriate language and commands to communicate with each robot/sensor individually.

- Devices of similar characteristics (for example, two sonar sensors) are usually not interchangeable. If a device must be replaced by a similar one, from another manufacturer, or even with the same device with an updated firmware, the communication and coding process should be redone.

These two tasks increase the development cost and delay the implantation of robotics outside the industry, where this cost can be assumed. For this reason the robotics middlewares have appeared in recent years.

The idea of these robotics middlewares is to standardise the coding and communication with multiple robots and sensors, in a way where a device can be replaced with another one of similar characteristics without the need of re-coding. Usually a standard type of message for each type of device, and a communication protocol is defined. Several functions in a high level coding language (for example, C) have also been defined for common purposes (for example, sending velocity commands to a differential drive robot).

Nevertheless, each robot or device needs a *driver* that transforms its communication messages and protocol to the standard one defined by the robotics middlewares. If the programmer needs to code each driver, only one of the previous issues is solved. Luckily, if a robotics middleware becomes widely used and popular, the hardware manufacturers will provide drivers specifically for that platform, because it is a way of increasing their sales: the more users that can use your hardware, the more hardware you will sell.

These are the main two tasks that a robotics middleware must fulfil, but these platforms usually work also as software repositories. As the communication with similar devices is standardised, there is software of common use that is included with the development platform and, if it is open-source, this repository increases its size and functionality through the years.

Also, robotics middlewares usually include simulators. Performing tests with real robots is very costly, in money and time, for this reason, simulators that model the robots and the sensors are needed.

Through the years several projects have been released and maintained, such as Carnegie Mellon Robot Navigation Toolkit (CAR-

MEN) [1], Microsoft Robotics Development Studio (MRDS) [2], MissionLab [3], MobileRobots' Advanced Robot Interface for Applications (ARIA) [4], The Player Project [5] and Robot Operating System (ROS) [6]. There are many other projects, but these are ones of the most representative.

In order to choose one of the several robotics middlewares to use, some questions need to be answered:

- Is it an Open-Source or proprietary software?
- On how many operating systems does it work?
- Is the project maintained and updated?
- Has it got software repository? Is it a collaborative repository?
- How many coding languages are supported?
- How many robots and sensors are compatible with it?
- Has it got any simulator? 2D or 3D?

With these questions in mind we are going to briefly review the selected robotics middlewares:

- **CARMEN**: One of the first platforms, developed in the *Carnegie Mellon University*, and oriented towards autonomous robot navigation tasks. It is Open-Source software and it has an algorithm repository for navigation tasks. It also includes a 2D simulator and at least 9 robots are supported. Coding can be done in *C* and *Java* languages. It runs in Linux operating system (Red Hat and SUSE versions). The last version has been released in 2008.

[1] carmen.sourceforge.net/
[2] www.microsoft.com/en-us/download/details.aspx?id=29081
[3] www.cc.gatech.edu/ai/robot-lab/research/MissionLab/
[4] robots.mobilerobots.com/wiki/ARIA
[5] www.playerstage.sourceforge.net
[6] www.ros.org

- **MissionLab**: Developed in *Georgia Tech University*, this robotics middleware is oriented to top level task planning for robot teams. MissionLab includes some algorithms for navigation and a flowchart style programming. The coding is made in *C++*. A 3D simulator is included and at least 6 robots are supported. It runs on Linux (Red Hat and Fedora) and the last version has been released in 2006.

- **MRDS**: Developed by *Microsoft* and released in 2006, it is intended to be a general robotics middleware. The programming can be made in a *web-style* format or in *C#* language, making easy to code basic actions. It includes a 3D simulator and supports, at least, 25 robots. It only works in Windows and the last version has been released in 2012.

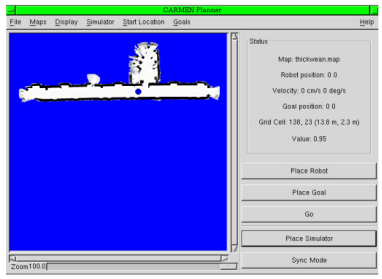
- **ARIA**: It was developed by *MobileRobots* and it only works with the Pioneer family robots (which are going to be reviewed in the next section). The platform can be downloaded for free, but the algorithm modules (such as mapping or navigation) have to be paid apart. It includes a multirobot 2D simulator. The coding is made in *C++* and it has compatibility with Matlab. It works in Linux and Windows and it is maintained nowadays (2017).

- **The Player Proyect**: It is free middleware oriented toward general robotics. It was released and maintained collaboratively by a group of researchers, since 2002. It includes a big repository of algorithms, from navigation to computational vision. It includes multirobot 2D and 3D simulators. The coding has officially support for C, C++, Python and Ruby languages, and many others supported by third party developers. It works with, at least, 19 families of robots. It works in Linux and OS-X and it was one of the most widely used robotics middleware till the last version released in 2010.

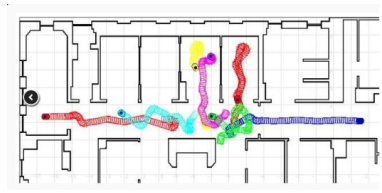
- **ROS**: It was developed by *Willow Garage* in the year 2007, ROS is a general robotics middleware. It can be installed

in many operating systems (the most widely used is Ubuntu-based distributions, but it has support for OS-X, Android and many others). It is coded in several languages, including *C++, Python, Ansic, Lisp, Java* and *Ruby*. At least 121 robots are supported (updated in January 2017). ROS is compatible with 2D and 3D simulators, especially with the 3D simulator Gazebo. It also has a big repository for robotics, with 2477 contributors (updated in January 2017) and increasing. It is now maintained by the non-profit organization Open Source Robotics Foundation, Inc. (OSRF) with planning support until, at least, 2021.

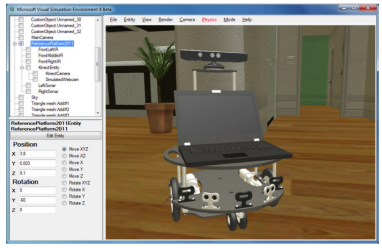
Figure 3.1 shows some of the simulators used by the development platforms analysed before. CARMEN planner (figure 3.1(a)) and *Stage* (used in both The Player Project and ROS) (figure 3.1(b)) are 2D simulators. Microsoft Visual Simulator Environment (MVSE) (figure 3.1(c)) used in MRDS and *Gazebo* (figure 3.1(d)) used in both The Player Project and ROS are 3D simulators.



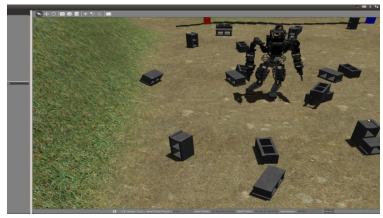
(a) CARMEN



(b) Stage



(c) MVSE



(d) Gazebo

Figure 3.1: Robotics Simulators

Table 3.1 summarises the characteristics of each robotics middleware. It is important to note that the number of robot families supported are approximated, because there are drivers developed for another robots not included directly in the platform repository. It is clear that ROS outperforms the other robotics middlewares evaluated, and fulfil all the requirements on our thesis:

Table 3.1: Robotics middlewares comparison

Middleware	**Robots**	**Maintained**	**Open Source**	**Simulator**
CARMEN	9	No	**Yes**	2D
MissionLab	6	No	**Yes**	**3D**
MRDS	25	No	No	**3D**
ARIA	1	**Yes**	**Yes**	2D
Player	19	No	**Yes**	**2D-3D**
ROS	**121**	**Yes**	**Yes**	**2D-3D**

- It is Open-Source, so it can be used in academy without increasing the cost of the project.
- It works with a large variety of operating systems, increasing its compatibility with other projects.
- It can be coded in several high-level languages.
- It includes a big, maintained and increasing repository of algorithms.
- It works with 2D and 3D simulators.
- It is compatible with a huge variety of commercial and researching robotics platforms. It is also compatible with all the platforms and sensors that we are going to use in this thesis.

For these reasons among many others, ROS is nowadays the standard robotics middleware and the one that is going to be used to test the proposals of this thesis.

3.1.1 ROS

Once a robotics middleware has been chosen, we are going to explain in detail how it works, its strengths and its weakness.

ROS [Quigley et al., 2009] was firstly released in 2007 by a group of researchers at *Willow Garage*, including some who have worked in the development of the *Player Project*. The philosophy of ROS is to develop a robotics middleware that can work *peer-to-peer*, is *multi-lingual, thin, free* and *Open-Source* oriented.

ROS is released in *distributions*, similar to the Ubuntu philosophy. Its first non-alpha distribution (*ROS Box Turtle*) was released in 2010. The first distributions were released without periodicity but, since 9^{th} distribution (*ROS Jade Turtle*, 2015), were released annually in may. The odd numbered years releases are considered *ROS normal* and will be supported for two years. The even numbered years releases are Long Time Support (LTS) releases and will be supported for five years.

ROS can be installed in several operating systems, such as, Linux, OS-X, Android and Arduino. The main installation support is for Ubuntu distributions of Linux and each LTS release of ROS has full support for one LTS distribution of Ubuntu.

As the project is Open-Source and it is the most popular robotics middleware nowadays, a huge repository of algorithms (from drivers to work with robots to artificial intelligence algorithms) is hosted in the ROS webpage. This repository is also divided between the several distributions. Each distribution has changes in some core libraries making not fully compatible with previous developed algorithms, so these algorithms need to be maintained. Some common algorithms (for example, laser grid mapping ones) are maintained by OSRF foundation and released with each version but, so many others, are not maintained by the community.

ROS can work with robot and sensors teams. A core node synchronizes the whole system but the computation of each algorithm can be executed in different machines connected to it.

In order to work with ROS some concepts and elements must be defined:

- **Nodes:** each individual program that is running in the system.

- **roscore:** a special node that must be always present in the system. That node is in charge of the synchronization of the whole system and contains the pre-requisites of a ROS-based system.
- **Package:** a collection of programs to perform one task (for example, the navigation package contains programs for local navigation and global path planning).
- **Topic:** a communication channel used by nodes to transmit data. These topics can be transmitted without a direct connection between nodes, meaning that the production and consumption of data are decoupled. A topic can have several subscribers and publishers associated to it.
- **Message:** it is the information unit that can be shared between nodes using a *topic*. In ROS, commons type of data are defined (for example, laser range information, image, odometry, etc.) and custom messages can be defined too. Usually, each message has the timestamps when the message is generated and its *frame*. This is the part of the system along with the instructions to access each type of message that standardise the communications task.
- **Frame:** It is the explicitly reference to a coordinate system in the environment. Frame information is contained on each *message* in order to increment the scalability and compatibility of the system.
- **Transform (Tf):** It is a special *topic* and *message* type that establishes the relationship between the coordinate *frames* of the whole system. It is necessary that these relationships are always sent, even if the relationship does not vary with time.
- **Services:** a communication between *nodes* of type "call/answer".

With these concepts explained we are going to describe a small application to demonstrate how the system works. In the example,

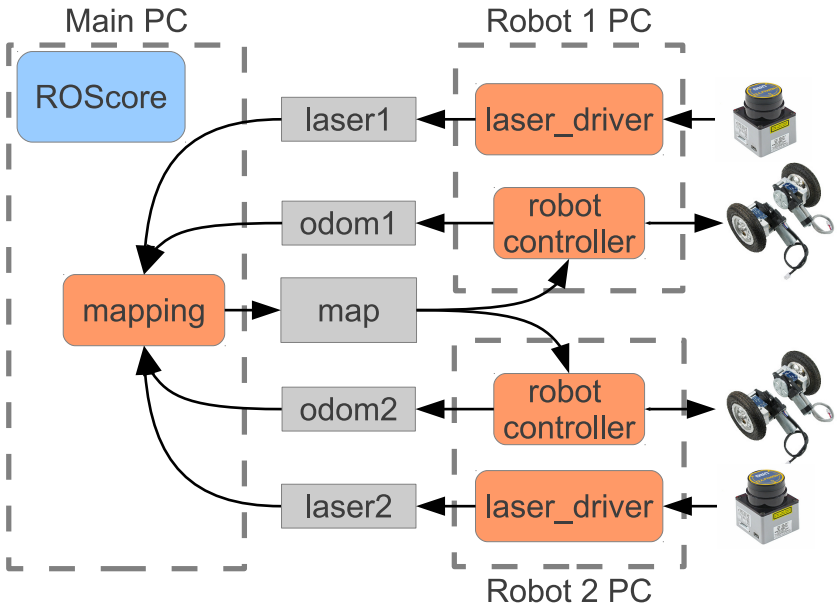


Figure 3.2: ROS architecture example

there are two mobile robots equipped with LIDARs that move in the environment. These two mobile robots build cooperatively a map and use this map to navigate in the environment. There is also a powerful computer that it is used to compute the map from the information of the robots. In the main PC, *roscore* is executed in order to synchronize the whole system. The robot's PCs are connected to that central node via *WiFi*. If the connection between any robot and the computer is interrupted, the computation in the mobile robot will halt. There are three types of nodes in the system: *mapping*, *robot controller* and *laser driver* nodes. Each robot executes an instance of *laser driver* and *robot controller*. Each *laser driver* produces the laser scan data (*message*) and sends it via *topic*. *Robot controller* generates the odometry data (*message*) of the robot and uses the map (*message*) to plan the movement of the robot. The mapping node needs the laser data and odometry of each robot and generates the map (*message*). It is important to note that each message is referred to a different coordinate system (laser data to each laser base,

odometry to each robot platform and map to the global coordinates) and the relationships between each coordinate system is sent in the system (in the Tf topic). Figure 3.2 shows the architecture of this example, where the grey boxes represent the topics in the system, orange boxes represent nodes and the lines represent if a node produces or consumes messages from a topic. For legibility the Tf topic and the connections of each topic to the *roscore* one are not represented.

Apart from the software repository, ROS includes a variety of useful software tools, such as **RViz** for data visualization (figure 3.3 shows the map, path planned, path followed and position of a robot), **RQT Tools** for system introspection and reconfiguration, **RosLaunch** for system starting simplification, and **RosBag** for data logging and replaying among so many others.

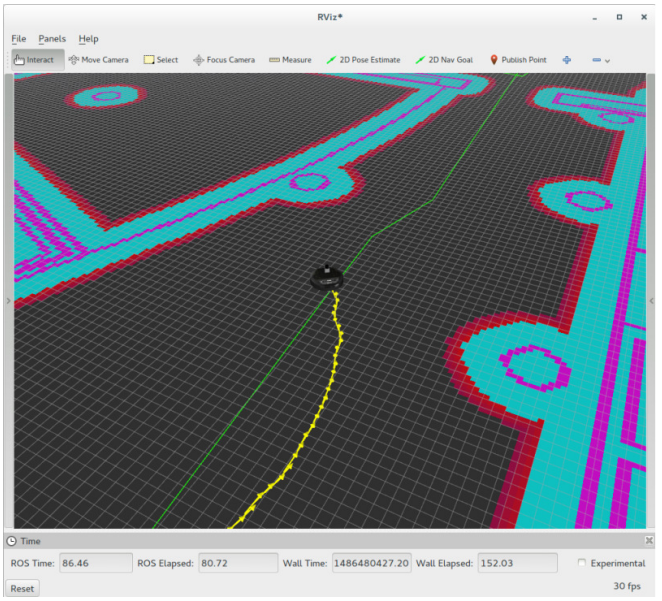


Figure 3.3: RViz example

ROS is not linked to a particular simulator and it has support for many of them, including Open-Source software like *Stage* or Simple Two Dimensional Robot Simulator (STDR), and proprietary ones like *Webots* and *VRep* among many others. However, the 3D simulator *Gazebo*, maintained by the same foundation that maintains ROS

becomes one of the widest used simulator.

Gazebo is a 3D simulator that includes rigid body physics (including friction and dynamics) and can work with several robots at the same time. *Gazebo* is a parallel project to ROS (it started in 2002 within the *Player Project*) but each ROS distribution has official support for one distribution of *Gazebo* (this does not mean that can not work with newer distributions, but does not have official support). That simulator can also work without ROS. In the next sections we are going to explain how to design a model and work with the *Gazebo* simulator.

It is clear that ROS has many strengths, even more when it is compared with other robotics middlewares. However we are going to highlight some of its flaws that can be conflictive when a robotics project is developed using this platform:

- Each distribution is not fully compatible with the algorithms released for the previous ones. In this way many submissions in the repository are not compatible with actual versions or they need maintenance.

- As the repository is free and collaborative, sometimes the testing and the information is not good enough or incomplete.

- A *roscore node* has to always be running in the system. As ROS support multi-robot and distributed systems, the communication with the master node is crucial. If the communication fails or it is not reliable enough, the whole system will fail. For this reason, some *Multi-Master* approaches has been made in a way that each robot or device is running its own instance of ROS and communicate with the others by TCP/IP or UDP services.

- The necessity of including the timestamp and *frame* of each *message*, and the relationship between all the coordinate systems (Tf), even when that relationships does not vary in time, increases the bandwidth needed for the system. On the other hand, if that information were not included, a deep knowledge of the whole system would be needed to work with robot teams.

- Each *message* that is sent can be read by any *topic*. So, for distributed networks, security measures should be implemented.

3.2 Robotic Platforms

In this section, the robotic platforms used in this thesis will be introduced. It is important to decide which platforms should be used on each scenario, for this reason, in the next sections the strengths and weakness of the commercial platforms and the platform developed in the RobeSafe Research Group will be analysed. Finally the different platforms will be compared in order to decide which is the best platform to use as a test bed for this thesis.

3.2.1 Commercial Platforms

There are some manufactures that provide robotic platforms that can be used in research, such as *Robotnik* [7]*, SoftBank Robotics* [8] (previously known as *Aldebaran), Boston Dynamics* [9]*, Clearpath Robotics* [10]*, Robotis* [11]*, Omron Adept MobileRobots* [12] (previously knowns as *MobileRobots* and *ActivMedia Robotics*) and some others. In the last years, many companies have started to develop and sell robots for industrial and research purposes. One of the most widely used platforms for researching are the robots by *Omron Adept MobileRobots*, such the well known *Pioneer family*, its little version *AmigoBot* and its big brothers, *Seekur* and *Seekur Jr.* RobeSafe Research Group has some of these robots to develop and test the projects (shown in Figure 3.4), from the *AmigoBot* to the *Seekur Jr.* In this thesis, the *Pioneer 3-DX, 3-AT* and *Seekur Jr.* have been used as Test Bed, due to they are the platforms available that accomplish the system requirements.

[7] www.robotnik.es
[8] www.ald.softbankrobotics.com
[9] www.bostondynamics.com
[10] www.clearpathrobotics.com
[11] en.robotis.com
[12] www.mobilerobots.com

Figure 3.4: RobeSafe Group's Robots

3.2.1.1 Pioneer Robots

Pioneer is a family of mobile robots, two-wheel and four-wheel drive, from the first *Pioneer 1* and *Pioneer AT*, followed by the second versions, *Pioneer 2-DX* and 2-AT, to the newest *Pioneer 3-DX* and *3-AT* mobile robots. These small research and development platforms, share a common architecture and core software with all other *MobileRobots* platforms, including *AmigoBot*, *PeopleBot V1*, *Performance PeopleBot*, *PowerBot*, *PatrolBot*, *Seekur Jr* and *Seekur*.

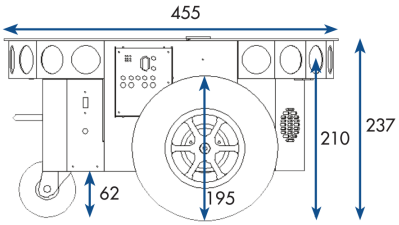
3.2.1.2 Pioneer 3-DX and Pioneer 3-AT

The *Pioneer 3-DX* and *Pioneer 3-AT* are durable, differential drive robots for academic and researching purposes. The Pioneer's versatility, reliability and durability have made them the most popular differential drive mobile robots in academic and researching for years.

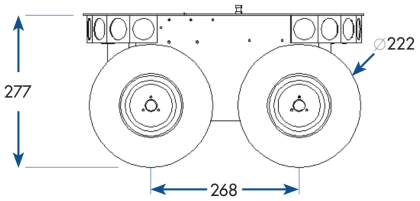
Unlike hobby and kit robots, Pioneer is ready to use, and will last through years of tough classroom and laboratory use. Both platform shares a common software architecture and sensors. Pioneer 3-DX (shown in Figure 3.5(a)) is a differential drive platform designed for indoor purposes and Pioneer 3-AT (shown in Figure 3.5(b)) is a skid steering drive platform designed for both indoor and outdoor purposes.

(a) Pioneer 3-DX

(b) Pioneer 3-AT



(c) Pioneer 3-DX: dimensions (mm)



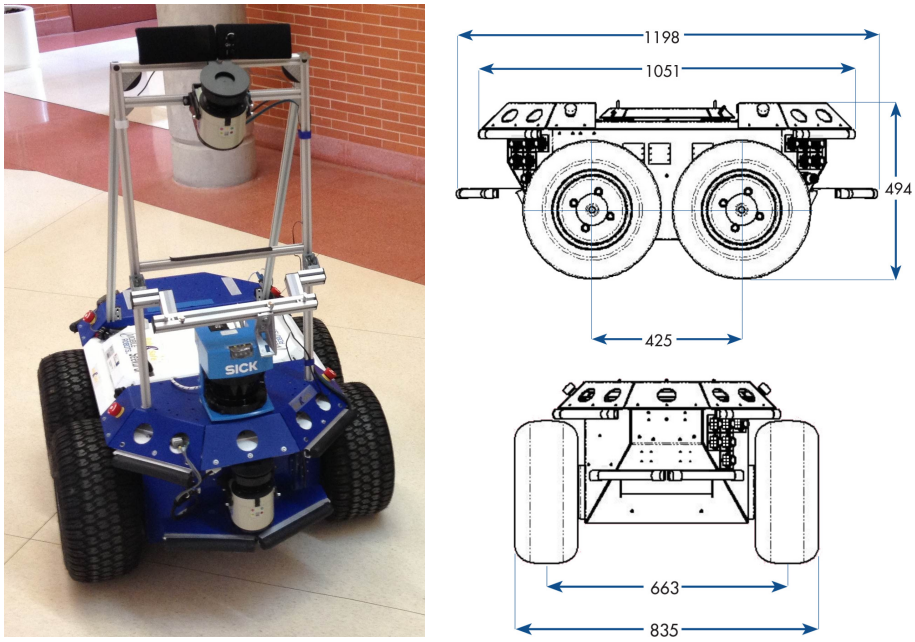
(d) Pioneer 3-AT: dimensions (mm)

Figure 3.5: Pioneer 3 Robots

3.2.1.3 Seekur Jr

The *Seekur Jr*, shown in Figure 3.6, is an outdoor robot platform, that can also operate in indoor scenarios with big open areas. Similar to the *Pioneer AT* models, this platform is four wheel skid-steer differential drive all terrain, though with a tires of 0.4 meters and it is

much stronger than the *Pioneer*, carrying a much larger payload and it is better protected against inclement weather. The body is made completely by sturdy aluminium. The robot includes a segmented bumper array in the front and the back and four emergency stop switches for safety.



(a) Seekur Jr (b) Seekur Jr: dimensions (mm)

Figure 3.6: Seekur Jr: platform and dimensions

3.2.2 Developed Platforms

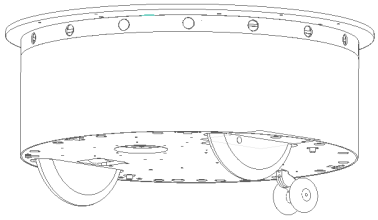
There is a necessity of improving the weaknesses of the available commercial platforms, such as better sensors, safety of the platform, weight, payload, etc. at the same time than the total cost is reduced. For this reason, a robotic platform has been developed from scratch with the improvements that will be described in the next sections. PROPINA has been developed in the RobeSafe Group, mainly in this thesis and in the thesis of Ángel Llamazares [Llamazares, 2017].

In the next sections the development of the platform will be explained. Also, its application and modifications as a shop guide RoboShop will be introduced. Finally, an experimental comparison with the commercial platforms, testing if the objectives are achieved, will be made.

3.2.2.1 PROPINA Platform

PROPINA platform has been developed as a robotic research platform to develop high-level applicatiosn using the robotics middleware ROS. The name is the Spanish acronym for "*robotic research platform*". It is a differential drive platform and it is equipped with odometry and range (ultrasonic and infrared) sensors. It is designed to work indoors. In addition, a 3D model for the Gazebo simulator (Figure 3.16) has been designed, which can be used as prior before designing the actual application.

The platform is equipped with embedded cards, that run ROS modules to control the motor, to perceive the information from sensors and to communicate with high level applications. In this way the perception and actuation is completely transparent to the remote control station. The modular design has been chosen to increase the functionality and autonomy.



(a) PROPINA design

PROPINA
PLATAFORMA ROBÓTICA PARA INVESTIGACIÓN

(b) PROPINA logo

Figure 3.7: PROPINA: design and logo

PROPINA platform is equipped with these onboard sensors: 16 maxbotics sonar sensors in a ring around the robot, 5 infrared sensors pointed to the ground in the front and the back of the robot to detect depressions of the terrain and optic encoders of 1000 Cycles Per

Revolution (CPR) in the motorized wheels. The logo and the preliminary design is shown in Figure 3.7. In the next sections the main components and characteristics of the platform will be explained.

3.2.2.2 PROPINA Platform: Mechanical design

Taking into account the previous experience using the commercial platforms, the objectives of the mechanical and structural design are:

- **Modular and versatile design:** In order to allow the addition of sensors or control modules, regular, symmetric and easy shapes have been selected. Aluminium has been used on the chassis and fibreglass in the cover to lighten the platform. Additionally, the cover on the top of the platform should facilitate the placement of additional sensors and structures.

- **Differential drive:** This traction system improves the manoeuvrability and agility of the platform, allowing it to turn without linear displacement (partial omni directionality). Rigid rubber wheels have been selected, similar to the ones used in scooters, due to several improvements that introduce to the platform, as the following ones:

  1. Increase the payload of the platform.
  2. Have a good grip reducing the slippage of the platform.
  3. Improve the odometry, compared to pneumatic wheels due to the diameter of these wheels do not change depending on the temperature, the carried load or the pressure. Also, the narrow wheels selected make the robot turns more precisely, due to less contact area with the floor, the distance between wheels keep more constant than the robots with wider tires.

- **Safety:** Any movable part of the robot should not be accessible to improve the safety. For this reason, the wheels must locate inside the structure, not allowing that the people around the robot can touch the wheels, or the wheels can trip over

some parts of the environment. Furthermore, the rounded shapes prevent it from getting stuck, consequently, improving the safety of the platform and its surroundings.

- **Robust:** The platform has been made for researching, which means, the design has to support an exhaustive use in several environments and scenarios.

- **Stability:** Due to the possibility of the addition of modules on the top of the platform, increasing the height, the design has to take this into consideration. The design maximizes the space between the wheels in order to increase the stability. Additionally, the main weight of the robot that are the batteries and the motors should be symmetrically located on the base to keep the mass centre close to the centre of the platform. In addition, a castor wheel has been placed in the back part of the robot's base to improve the stability.

Based on these objectives, a base and a sonar ring modules have been developed, both of them with cylinder shapes. The design of the platform is shown in the Figure 3.8.

Figure 3.8: PROPINA platform

3.2.2.3 PROPINA Platform: Electronic design

The first step in the electronic design is to chose the actuators. In this case, permanent magnet DC motor with a gearbox has been

chosen due to its high torque. *PM10-0033* by Parvalux[13] (shown in Figure 3.9) has been chosen for this purpose, with a speed of 195 Revolutions per Minute (RPM), and 2.5Nm of torque. The coupling from the motor to the wheels is made by flexible beam couplings to prevent that the torsions and impacts in the wheels damage the motor's gearbox.

Figure 3.9: PROPINA: PM10 Motor and gearbox

The second step is to decide the board that will control the motors and adapt to the information of the sensors. The motors are controlled by an *Arduino Mega*[14] board connected through a *Pololu VNH5019*[15] driver board. This driver board is a Dual H bridge for DC motors connected to the *Arduino Mega* as it is shown in the Figure 3.10. A software driver has been developed to run in the Arduino Mega board. This program reads the sensors and it commands the actuators at the same time that send and receive all the data in a ROS format through the *ROSserial libraries for Arduino*, allowing to control the robotic platform using a ROS remote client. The maximum linear and turn velocities have been limited by software to $1m/s$ and $100^{o}/s$ for safety, although the motors are capable of reach higher velocities.

The third step in the electronic design is to choose the appropriate sensors. The sensors that have been included in the platform are:

[13] www.parvalux.com
[14] www.arduino.cc/en/Main/arduinoBoardMega
[15] www.pololu.com

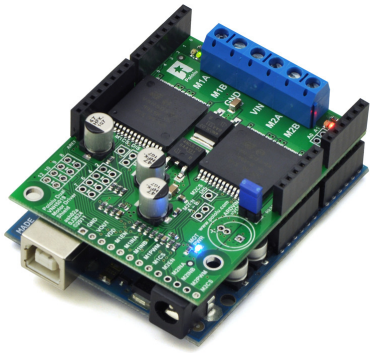


Figure 3.10: PROPINA: Motor's Driver and Arduino boards.

- **Encoder sensors:**

  The selection of a proper encoder is crucial, due to this sensor is needed to achieve a good odometry. Odometry is the base of the localisation, and it is also essential to achieve a good motor control stage. For both reasons, a high-resolution and optical (high immunity to interference) encoder has been chosen. For this platform, the *RI38 optical encoder by Hengstler*[16] (shown in Figure 3.11) has been selected. This encoder has an 1000 CPR resolution, which is two or five times (depending on the robot model) more resolution than the Pioneer ones have. This encoder has been the best election in terms of price, size and characteristics.

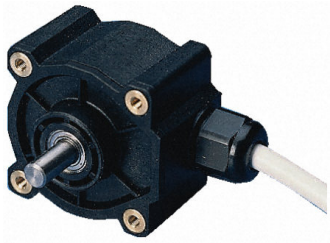


Figure 3.11: PROPINA: Wheel encoder RI38

- **Infrared range sensors:**

[16] www.hengstler.de/en

To keep the integrity of the robot, it is crucial to detect the ramps or gaps in the floor, and evaluate if the platform can overcome them or should avoid them. For this purpose, a total of five infrared range sensors pointed to the floor have been chosen. Three of them are located in the front of the base and two in the back. Ideally, with a cylindrical platform the sensor's positions should be symmetric, but the castor wheel makes it impossible as there is no space left in the back of the robot. With the sensors in that location, the robot knows in every moment the distance from the base to the floor in front and in the back of the wheels. The sensor chosen is the *SHARP GP2Y0A41SK0F*[17] (shown in Figure 3.12) due to its extended use in robotics and its low price.

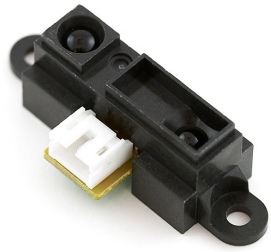


Figure 3.12: PROPINA: Infrared Sensor

This sensor has a range of 40 cm and a transfer function that is non linear. The five sensors have been connected to the analog inputs (A0-A4) of the *Arduino Mega* board. The transfer function has been approximated as a fifth order polynomial (due to the computation will be made outside the *Arduino Mega* board), given by Equation 3.1 and shown in figure 3.13, where d_{GP2Y} is the distance measured and V_{in} the voltage measured by the sensor.

[17] www.sharp-world.com

$$d_{GP2Y} = -0.076 \cdot V_{in}^5 + 0.78 \cdot V_{in}^4 - 3.2 \cdot V_{in}^3 + 6.76 \cdot V_{in}^2 - 7.8 \cdot V_{in} + 4.84 \tag{3.1}$$

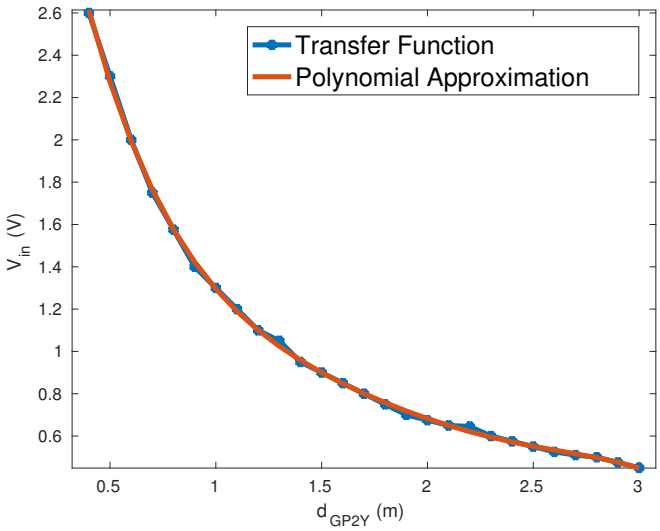


Figure 3.13: Infrared Sensor: Transfer function and Polynomial Approximation

- **Sonar range sensors:**

  To detect the environment around the robot, there is a necessity of including some range sensors. For this reason, a sonar module has been designed to include sixteen sensors *LV-MaxSonar-Ez2 (MB1010)* by *Maxbotics*[18], shown in Figure 3.14, selected due to its low price and very small size.

  These sensors have a range of 6.45m, are powered from 2.5V to 5.5V, work at 42kHz and provide output measurements up to 20Hz. It is possible to read the output in three different ways: analog, serial and Pulse-Width Modulation (PWM). In

[18] www.maxbotix.com

Figure 3.14: PROPINA: Sonar Range Sensor

this version of PROPINA platform, only eight of the sixteen sonar have been implemented, connecting these eight sensors to the analog inputs (A8-A15) of *Arduino Mega* board. The sensors form two groups of four each, as the manufacturer recommends, and the simultaneous transmissions are between the sensor most separated to minimize cross-talk. It is possible to enlarge the platform until sixteen sonars, due to can be placed all of them in the ring sonar structure, allowing their reading using the analog inputs of the second Arduino onboard.

3.2.2.4 PROPINA Platform: Software

As ROS has been chosen as the robotics middleware in this thesis, the drivers of the PROPINA platform are made fully compatible with this platform in order to increase the compatibility of PROPINA project with the algorithms developed for ROS development platform.

In the PROPINA prototype, there are two different *Arduino Mega* boards inside the robot, one that reads the encoders and sends velocity commands to the motors and one that reads the sensors. An embedded PC communicates with both boards through USB connection and executes a ROS node that provides the standard ROS messages along with the relationship between the coordinate system of the robot. Figure 3.15 shows the software diagram of PROPINA where light blue box is the central *roscore* node, orange boxes are the running nodes, grey boxes are custom messages and turquoise boxes are standard messages. In this section we are going to describe the nodes running on each arduino board and pc.

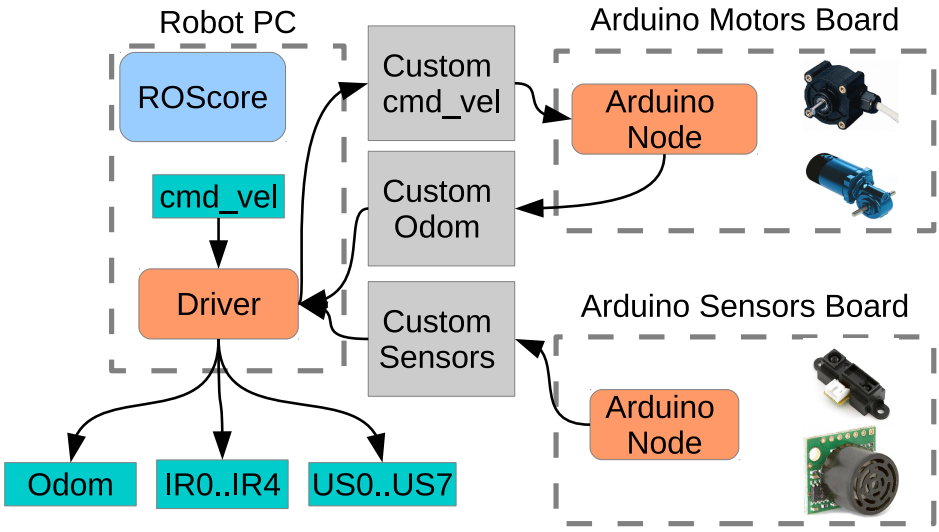


Figure 3.15: PROPINA: Software Diagram

- **Arduino Mega Sensors Board**: This board is in charge of transmitting the measurements of each sensor to the robot. As the sensors are connected directly to the analog inputs of the board their voltage measurement can be directly stored. In order to reduce the computational cost and the bandwidth, a periodical loop is executed which sends all the current sensors measurements in a custom message ROS topic that will be transformed into several ROS standard range sensors messages in the PC's driver node.

- **Arduino Mega Motors Board**: This board is in charge of transmitting the odometry of the robot and commanding the velocities to the motors. In order to reduce the bandwidth needed this information is sent and received through custom ROS messages with the minimum information instead of the standard message. Velocities command message is reduced to a two element array (linear and angular velocity) and odometry is reduced to a four element array (timestamp, and encoder position). The Arduino program executes two functions: a periodically (timely by a hardware interruption) loop that reads

the encoders, sends its information and controls the motors, and an event based reading function that waits for a velocity command and changes the reference commands to the control loop. The periodically 100Hz control loop work as follows:

1. It reads the encoders position and calculates its errors to the reference of each wheel. These references are calculated with the equations 3.2 and 3.3 for both wheels, where $wcod_0ref$ and $wcod_1ref$ are the encoders references, V_d and ω_d are the linear and angular velocity references, L is separation between the wheels, k is the velocity to RPM conversion factor and k_1 are a kinematic constant that depends on the radius of the wheel.

$$wcod_0ref = k_1 \cdot (V_d + \frac{L \cdot \omega_d}{2}) \cdot \frac{1}{k} \tag{3.2}$$

$$wcod_1ref = k_1 \cdot (V_d - \frac{L \cdot \omega_d}{2}) \cdot \frac{1}{k} \tag{3.3}$$

2. It calculates the control signals for each wheel, with a Proportional Integral (PI) controller that has been tuned experimentally in order to obtain an acceptable damping response and a setting time less than 100 ms. Also, an anti wind up block modifies the integral part to avoid saturation in the commands.

3. A security watchdog is implemented. If there are no new velocity commands in 2.5 seconds the velocity references are set to zero and the robot will be stopped in the next iterations.

4. A security stall warning is implemented. If there are commanded velocities and the encoder readings are the same for 2 seconds, the robot will be immediately stopped.

5. Update the velocity references. The velocity references will be increased or decreased to the commanded velocity in a linear way with constant intervals in order to reduce the abrupt accelerations and decelerations, increasing the

security of the platform. Also, there is a check to not command velocities under the maximum or minimum software velocity limits.

- **PC driver**: A ROS node that is executed in the robot's PC and its objective is to transform the custom messages sent and received by the arduino boards to standard ROS ones. Also, the node publishes the transformation between the sensors and the base coordinate frames. The node has several listeners to topics that work on an event based way:

  1. When a velocity command is received, it is transformed to the custom reduced message and it is sent to the arduino motor's board.
  2. When an odometry custom message is received the odometry of the robot is calculated and it is sent in a standard odometry ROS message referred to the *odom* coordinate frame. The odometry is integrated at 100 Hz using the following inverse kinematics equations 3.4 and 3.5, where x_t and y_t are the actual coordinates, x_{t-1} and y_{t-1} are the previous coordinates, pos_0 and pos_1 are the pulses counted by the encoders since the last odometry update, θ_{t-1} is the previous orientation of the robot and k_2 is a kinematic constant that depends on the radius of the wheel and the pulses per turn of the encoder. Equation 3.6 calculates the new orientation of the robot where L is the separation between the wheels. Also linear and angular velocities of the robot are calculated (equations 3.7 and 3.8) where ω_1 and ω_2 are the velocities of each wheel and R is the radius of the wheel.

$$x_t = x_{t-1} + \frac{pos_0 \cdot k_2 + pos_1 \cdot k_2}{2} \cdot cos(\theta_{t-1}) \tag{3.4}$$

$$y_t = y_{t-1} + \frac{pos_0 \cdot k_2 + pos_1 \cdot k_2}{2} \cdot sin(\theta_{t-1}) \tag{3.5}$$

$$\theta_t = \theta_{t-1} + atan(\frac{pos_0 \cdot k2 - pos_1 \cdot k2}{L}) \tag{3.6}$$

$$v = \frac{\omega_1 + \omega_2}{2} \cdot \frac{2\pi R}{60} \tag{3.7}$$

$$\omega = \frac{\omega_1 - \omega_2}{L} \cdot \frac{2\pi R}{60} \tag{3.8}$$

Also, a transformation between the *odom* frame and the *base_link* (robot base) coordinate frame is created.

3. When the sensors custom message is received it is transformed onto several standard ROS messages of *Range* type. Each voltage measured of the infrared sensors is transformed to the range measurement using the fifth grade polynomial described in the previous section and referred to each infrared coordinate frame. For the ultrasound measurements, a median filter with a 3 seconds window is used to obtain the range measurement due to the noisy sensors. Also, the transformations between each sensor coordinate frame and the robots *base_link* frame are created.

- ***RosSerial* nodes**: in order to execute the codes in the *arduino* boards, nodes that connect the *roscore* central node and the arduino ROS instances are needed to be executed as the system launch.

3.2.2.5 Modelling PROPINA in Gazebo

Simulators in robotics are crucial due to allow testing the design previously to the final application. In this way, it is possible to test with different sensors, configurations and environments, saving cost and time. For this reason, a simulation model of the developed platform is necessary to complete the thesis.

Gazebo is a 3D multi-robot simulator. This simulator is able to deal with several robotic platforms, sensors, objects and scenarios. *Gazebo* simulates rigid-bodies physics, including dynamics, interactions between several objects and realistic behaviour of the sensors.

The simulation is coded in *SDF* or *URDF* format. Both are *XML* based formats. The description of a model contains the following elements:

- **Links:** each part of the model. A model can be built using as many individual *links* as needed. The *link* contains information about:

  Collisions: describes the shape of a *link* and it is used by the physics engine to check collisions.

  Visual: describes the shape of a *link* and it is used by the rendering engine. It shows the appearance of a *link* to the user and some sensors such cameras. Usually the same *link* has a more complex shape for visual purposes and a simpler shape for collision purposes in order to reduce the computational load.

  Inertial: describes the physical properties of a *link* such as mass and moments of inertia.

  Sensors: the devices that obtain data from the scenario. These sensors can be, for example, a camera or a range sensor. One *link* can have more than one sensor attached.

- **Joints:** the union between different *links*. In *Gazebo* there are several type of *joints* such as fixed, revolution, prismatic, etc.

- **Plugins**: the libraries that are loaded by Gazebo allowing the user to control the model. These are also used for controlling a model in Gazebo from ROS.

The model in *Gazebo* (shown in Figure 3.16) has been built simplifying the model created in *SolidWorks* and has the following components:

- A base *link* that represents the robot base. For visual purposes it shows the mesh modelled in Solidworks and for collision purposes it is approximated as a solid cylinder.

- Two *links* that represent the differential drive wheels modelled as solid cylinders. These wheels are connected to the base *link*

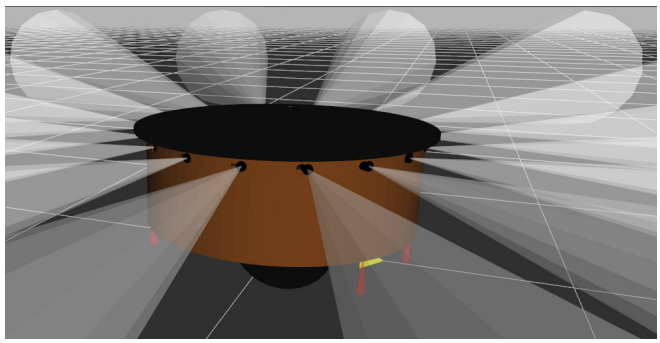


Figure 3.16: Gazebo model of PROPINA

with a *continuous joint* and can rotate around their axis to move the robot

- The castor wheel structure is approximated as a three prisms structure that rotates around the base *link* with two wheels attached.

- Five infrared sensors are modelled, heading downwards, attached to the base *link* and connected to ROS via GPU ray *plugin*.

- The sonar array is modelled as sixteen sonar sensors, attached to the base *link* and connected to ROS via GPU ray *plugins*.

- The model can be controlled via differential drive *plugin* that transforms the velocities commands (linear and angular velocities of the robot) in ROS to velocities in each differential drive wheels.

The diagram of the PROPINA model in *Gazebo* that contains the previous components is shown in Figure 3.17. The model has a form of a tree where each link (boxes in the diagram) has only one predecessor, but it can have many successors. The dashed lines represent *non-fixed joints* and solid lines, the *fixed joints*.

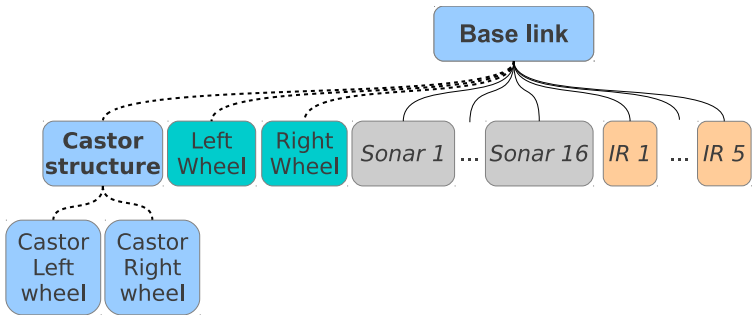


Figure 3.17: Gazebo diagram of PROPINA

3.2.2.6 RoboShop Platform

In order to increase the usability of our previous developed platform, an extension of the PROPINA platform has been developed through RoboShop project. In this project we developed a complete robot navigation system, that can be used as a *shop assistant* or a *robotic guide* in a museum or in a mall.

To develop the project, it is necessary to outfit the PROPINA platform with more sensors to measure the environment and a Human-Machine Interface (HMI) to communicate with the customers. The sensors selected are a LIDAR, an IMU, a depth camera, a RGB camera and a touch screen, as it is shown in Figure 3.18. The details of the design and equipment are shown as follow:

- Hokuyo URG-04LX LIDAR that can be used parallel to the ground to obtain a large view of the surroundings of the robot, or pitched to the ground to obtain measures in 3D and making the navigation system able to avoid depressions of the environment.

- Colibri IMU attached to the centre of rotation of the robot to improve the odometry.

- Depth camera *Asus Xtion Pro* that can provide 3D information along with RGB images. It can be used to recognize several objects and patterns in the environment.

Figure 3.18: RoboShop platform

- A camera pointed to the ceiling to obtain measures that are independent of the changes in indoor environments and it can be used to improve the localisation stage of the robot.

- Touch Screen that includes speakers, as HMI, mounted on a structure to increase the height, providing a comfortable interaction between the robot and the user.

- A powerful embedded PC that executes the perception, planning and control stages.

The RoboShop diagram of the whole proposed system is shown in the Figure 3.19. In this proposal, the PROPINA platform communicates and sends its information to the embedded PC. The embedded PC executes all the autonomous navigation system stages (perception, mapping, localisation and autonomous navigation) aided by the added hardware to the PROPINA platform. The communication with the user can be made by the Touch Screen integrated in the platform or by a mobile application. Also, there is a database which

stores several useful information like maps, positions, actions, etc., and can be used to coordinate the tasks that the robot must fulfil.

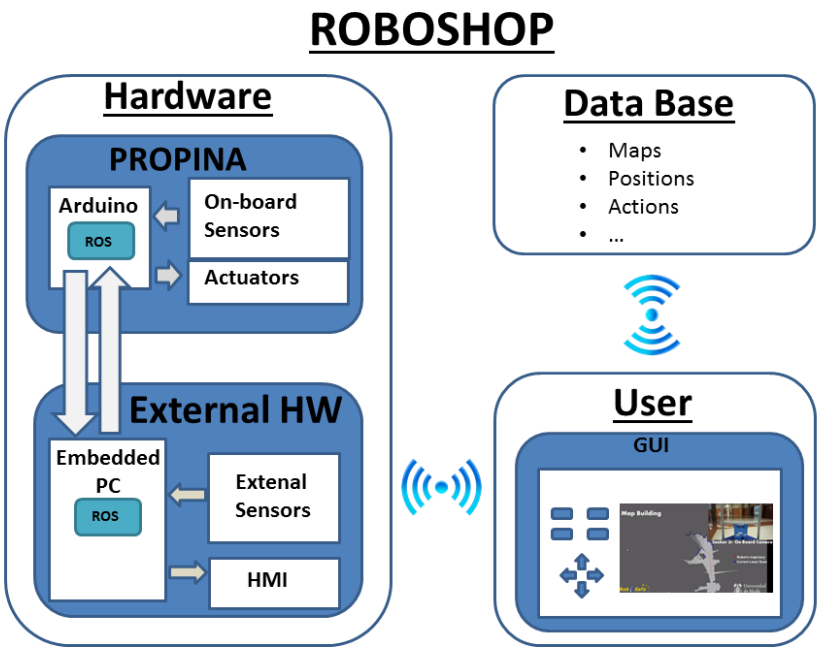


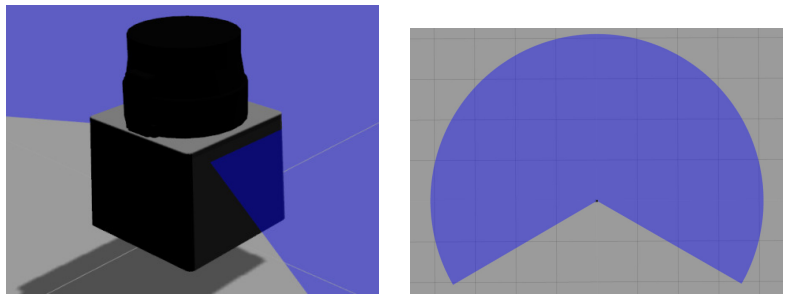
Figure 3.19: RoboShop Diagram

3.2.2.7 Modelling RoboShop in Gazebo

In order to simulate the RoboShop platform in Gazebo, an structure and three sensors have been added to the PROPINA platform.

- An structure that represents the pole and the monitor for HMI has been modelled. The structure is composed by the pole, approximated as a cylinder, the support of the monitor approximated as a prism and the monitor itself approximated as another prism.
- A Hokuyo URG-04LX LIDAR, situated on top of the robot and parallel to the floor. The LIDAR is modelled as a *link* with one

joint. For visual purposes it is represented as a complex mesh of the device. For collision purposes it is approximated by a box of 10 cm. The sensor itself has two versions, a *gpuray* one and a *ray* one, depending on the availability of a 3D graphics accelerator in the simulator. Figure 3.20(a) shows the laser model and 3.20(b) shows the field of view of the LIDAR (from a top view).

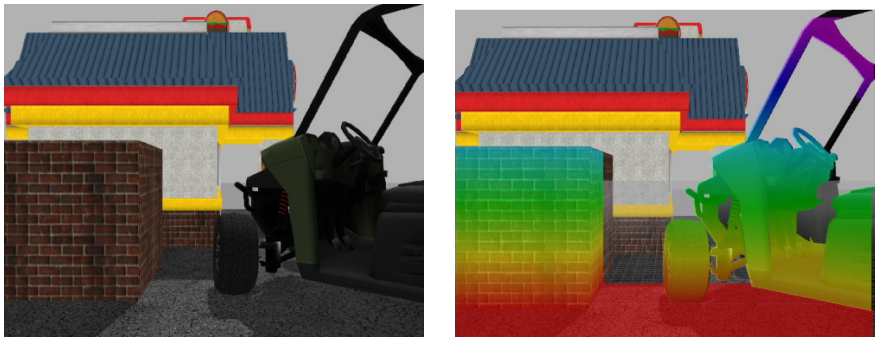


(a) Hokuyo URG-04LX Model (b) Field of view

Figure 3.20: Hokuyo URG-04LX in Gazebo

- An IMU situated in the centre of the robot, that is used to robust the odometry system.

- A depth-camera, situated on the pole and parallel to the floor. For visual purposes, the depth camera mesh is based on a *Kinect*, but its properties (field of view, resolution of the camera) have been modified, according to the *Asus Xtion Pro* specifications. It has two different sensors: an infrared sensor, that simulates the depth camera and produces a pointcloud and a RGB camera. Figure 3.21(a) shows an image of a simulated outdoor environment obtained from the RGB camera and figure 3.21(b) shows the pointcloud generated by the depth camera, coloured by its height axis value, superposed to the RGB image. It is important to notice that the *Asus* depth camera field of view is about 3.5 meters and this is the reason why the further part of the car and the house does not have correspondence in the pointcloud.



(a) RGB Camera Image

(b) Depth Pointcloud over RGB Camera Image

Figure 3.21: Depth Camera sensor in Gazebo

- A camera pointed to the ceiling. This camera is not going to be used for navigation purposes, but it will be used for localisation (although not in this thesis). The camera is attached to the monitor.

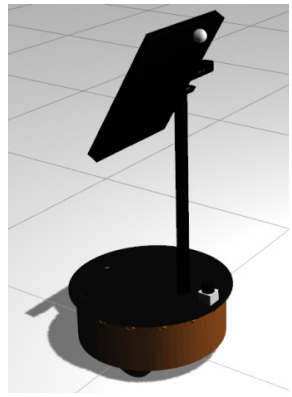


Figure 3.22: Gazebo model of RoboShop

Figure 3.22 shows the resulting model. The diagram of the RoboShop model that contains the previous components is shown in Figure 3.23. The dashed lines represent *non-fixed joints* and solid lines, the *fixed joints*. The right part of the diagram that is boxed is the added components to the PROPINA model.

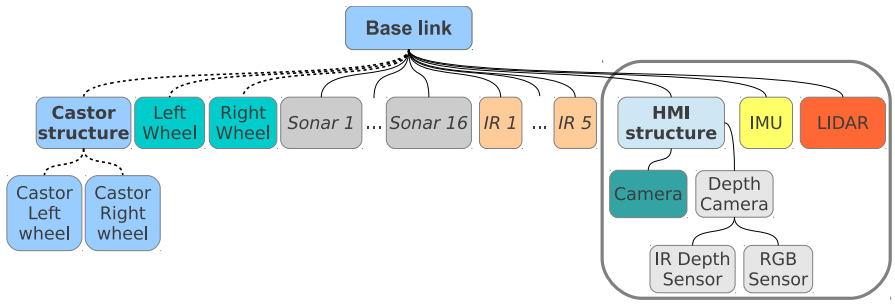


Figure 3.23: Gazebo diagram of RoboShop

3.3 Robotic Platforms Comparison

Once that the different commercial and developed robotic platforms have been introduced, it is necessary to compare them to test the performance of each one. Table 3.2 shows the main characteristics of the previous explained platforms. These platforms have a partial omnidirectional drive system (differential or skid steering one) so they can turn without translation movement and its turn radius is 0 cm. Swing radius is the space needed to perform a turn without translation and it is related to the size and the morphology of the robot.

PROPINA platform and P3-DX are indoor oriented robots, and P3-AT and Seekur Jr are outdoor and indoor oriented robots. This difference is caused by the dimensions of the wheels and the torque of the motors: outdoor robots have bigger tires and their maximum traversable step (the highest step that the robot can climb), maximum traversable gap (the longest gap in the ground that the robot can cross) and maximum traversable grade (the maximum ramp that the robot can ascend) are bigger.

Skid steering drive robots usually can have more payload than differential drive ones, as it is the case of the comparison. PROPINA robot weight can be reduced in future prototypes of the robot and its maximum velocities are software limited for safety purposes. PROPINA price does not include the industrial benefits as the other ones do, but, in the other hand, it is the price of a single unit prototype, and can be reduce in massive production.

Table 3.2: Pioneer 3-DX, 3-AT and PROPINA Comparison

	P3-DX	**P3-AT**	**Seekur Jr**	**PROPINA**
Drive	Differential	Skid Steering	Skid Steering	Differential
Tires	19 cm Foam-filled rubber	22,2 cm Reenforced Pneumatic	40,64cm (16") Pneumatic	17,5 cm Rigid rubber
Turn Radius	0 cm	0 cm	0 cm	0 cm
Swing Radius	26.7 cm	34 cm	52 cm	33 cm
Max Lin. Speed	1.2 m/s	0.7 m/s	1.2 m/s	1 m/s (lim. by software)
Max Ang. Speed	300 °/s	140 °/s	100 °/s	100 °/s (lim. by software)
Max Trav. Step	2.5 cm	10 cm	12 cm	2 cm
Max Trav. Gap	5cm	15 cm	∼ 20cm	3 cm
Max Trav. Grade	25%	35%	75 %	25%
Traversable Terrain	Indoor, wheelchair accesible	Asphalt, flooring, sand and dirt.	All terrain	Indoor, wheelchair accesible
Dimensions	381 mm width 455 mm length 237 mm height	467 mm width 508 mm length 277 mm height	835 mm width 1198 mm length 494 mm height	560 mm diameter 245 mm height
Robot Weight	9 Kg	12 Kg	77 Kg	19 Kg
Payload	17 Kg	5 Kg (Asphalt) 12 Kg (Tile)	50 Kg	20Kg
Autonomy	8-10 hours (3 batteries, no accessories)	2-4 hours (3 batteries, no accessories)	3-5 hours (no accessories)	4-6 hours (2 batteries)
Encoders	500 CPR	500 CPR	1024 CPR	1000 CPR
Sensors Included	8 Frontal Sonar	8 Frontal Sonar	segmented bumper array, IMU	16 Sonar Ring, 5 IR (floor)
Approx. Price	∼ 4800€	∼ 8900€	∼ 28000€	∼ 3000€

With respect to the indoor suitable commercial platforms, PROPINA platform has some advantages:

- The payload is a 20% higher, then the range of application is wider.
- The top part of the platform (black cover) has different screw holes, allowing to attach easily, and in any direction, any kind

of sensor or structure. Also, the symmetry of the platform make easier to place the sensors than the non-symmetric Pioneer robots.

- It is a cheaper platform, almost the half price that the Pioneer 3DX and the third part of Pioneer 3AT.
- It incorporates infrared sensors point to the floor, to avoid the stairs or steps higher than the platform could overcome.
- It is safer, due to the wheels are located inside the structure, not allowing that the people around the robot can touch the wheels, preventing possible damages, neither the wheels to trip over some parts of the environment. Additionally, the rounded shapes prevent it from tripping with the corners or getting stuck.

The last part to compare, is the odometry performance. To evaluate this parameter, the University of Michigan Benchmark (UMBmark) [Borenstein and Feng, 1996] has been used. UMBmark, is a *Bidirectional Square Path* experiment, with a length of 4x4 m square path, as shown in Figure 3.24, that has to be performed five times on each direction: clock-wise (*cw*) and counter-clockwise (*ccw*) directions, defined as follows:

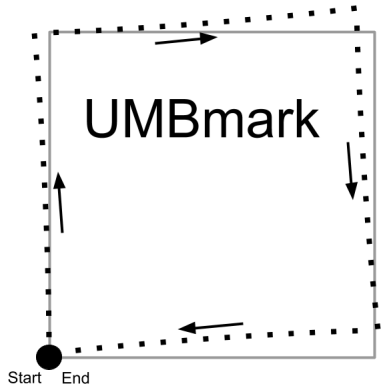


Figure 3.24: UMBmark path

1. At the beginning of the run, measure the absolute position (and, optionally, orientation) of the vehicle and initialise the onboard odometric starting position to this position.

2. Run the robot through a 4x4 m square path in *cw* direction, making sure to:

   - Stop after each 4 m straight leg.
   - Make a total of four 90º turns on the spot.
   - Run the vehicle slowly to avoid slippage.

3. Upon return to the starting area, measure the absolute position (and, optionally, orientation) of the vehicle, using a fix reference, such as walls.

4. Compare the absolute position to the robot's calculated position, based on odometry. The result is a set of return position errors caused by odometry and denoted εx, εy, $\varepsilon\theta$, shown in equations 3.9.

$$\begin{aligned} \varepsilon x &= x_{abs} - x_{calc} \\ \varepsilon y &= y_{abs} - y_{calc} \\ \varepsilon\theta &= \theta_{abs} - \theta_{calc} \end{aligned} \tag{3.9}$$

   Where, εx, εy, $\varepsilon\theta$ are the position and orientation errors due to odometry. $x_{abs}, y_{abs}, \theta_{abs}$ are the absolute position and orientation of the robot. $x_{calc}, y_{calc}, \theta_{calc}$ are the position and orientation of the robot as computed from odometry.

5. Repeat steps 1-4 for four more times.

6. Repeat steps 1-5 in *ccw* direction.

In the experiment a path of 3 x 3m , instead of 4 x 4m, has been used due to the lack of free space and precision in some platforms, to

perform 4 x 4m square with the robots. In addition, only 4 runs could be performed, due to the odometry errors present in some platforms, forcing to stop the experiment due to the deviation from the ideal path, leaving the free space, in the fourth run.

After conducting the UMBmark experiment, the authors suggest to consider the centre of gravity of each cluster of position and orientation errors due to odometry, obtained by Equation 3.9, as representative for the odometry errors in *cw* and *ccw* directions. These centres of gravity are given by Equation 3.10:

$$\begin{aligned} x_{c.g.,cw/ccw} &= \frac{1}{n}\sum_{i=1}^{n} \varepsilon x_{i,cw/ccw} \\ x_{c.g.,cw/ccw} &= \frac{1}{n}\sum_{i=1}^{n} \varepsilon x_{i,cw/ccw} \end{aligned} \tag{3.10}$$

where $n = 4$ is the number of runs in each. The absolute offsets of the two centres of gravity are defined by Equation 3.11:

$$\begin{aligned} r_{c.g.,cw} &= \sqrt{(x_{c.g.,cw})^2 + (y_{c.g.,cw})^2} \\ r_{c.g.,ccw} &= \sqrt{(x_{c.g.,ccw})^2 + (y_{c.g.,ccw})^2} \end{aligned} \tag{3.11}$$

Finally, the authors define the larger value among $r_{c.g.,cw}$ and $r_{c.g.,ccw}$ as the measure of odometric accuracy for systematic errors:

$$E_{max,syst} = max(r_{c.g.,cw}, r_{c.g.,ccw}) \tag{3.12}$$

This thesis focuses on real applications and such as authors suggest, the average of the centres of gravity must not be used, instead taking into account the largest possible odometry error, that means

the worst scenario. In addition, the final orientation $\varepsilon\theta$ is not explicitly considered to obtain $E_{max,syst}$, because the systematic orientation errors are implied by the final position errors.

Figure 3.25 shows the results of UMBmark performed by a Pioneer 3DX, Pioneer 3AT and RoboShop platforms, where the *stars* are the odometry errors in each run, the *circles* are the clusters of the odometry errors and the *crosses* are the centre of gravity of each cluster. It is clear that the error distribution of each RoboShop runs is, at least, in the same magnitude level as the better run of the other two platforms. Also, Table 3.3 shows the measure of odometric accuracy of each platform, where the better performance of the developed platform RoboShop is shown.

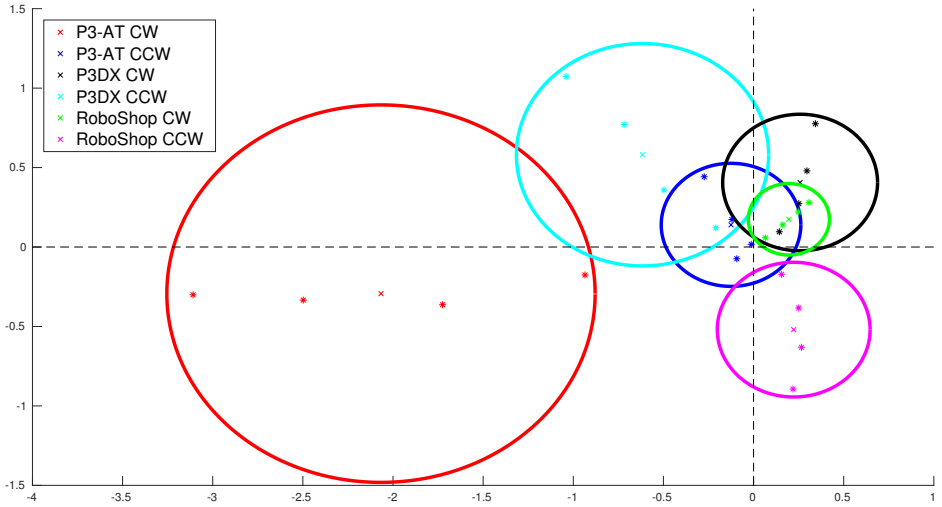


Figure 3.25: Comparison of platform's odometry based on UMBmark

	Pioneer 3DX	**Pioneer 3AT**	**RoboShop**
$E_{max,syst}$	0.8466	2.0875	**0.5656**

Table 3.3: Measure of odometric accuracy based on UMBmark

Also, a more realistic scenario has been used to measure the odometry accuracy. A manual run around the Polytechnic University second floor of about 300 meters has been performed with each robot,

with the same linear and angular velocities limits and trying to avoid drastic velocity changes. Finally, the position error with respect to the initial point is measured in euclidean distance and angular error. Figure 3.26 shows the three runs, where crosses represent the initial positions, stars the end positions and arrows the initial and final orientations. Table 3.4 shows the measured errors.

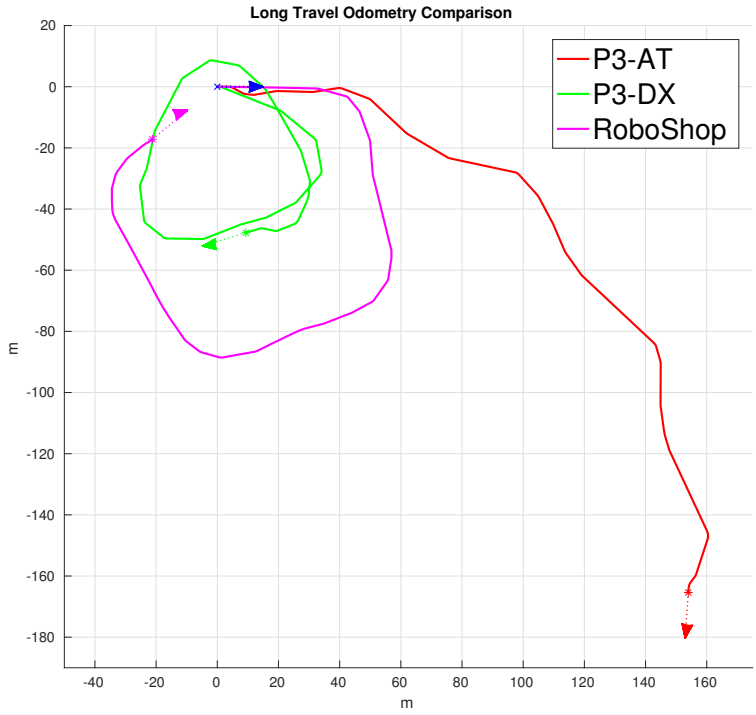


Figure 3.26: Comparison of platform's odometry on real runs

Table 3.4: Measure of errors on real runs

	Pioneer 3DX	**Pioneer 3AT**	**RoboShop**
$E_d(m)$	48.6674	225.6688	**27.5756**
$E_\theta(^o)$	159	-264	**-33.33**

With these experiments performed, it is clear that the odometry accuracy of the platform outperforms the commercial platforms available in the RobeSafe group, and, subsequently, makes the platform

more suitable to test perception and navigation algorithms. This improvement is due to several factors:

- The better odometer sensors chosen (1000 CPR, instead of 500 CPR of the Pioneer),
- The narrow rigid rubber wheels chosen that have a good grip, preventing the slipping effect. Its diameter does not depend on the carrier load or the pressure. Also, the narrow wheels keep the distance between wheels more constant, which is especially important during the turns.

Several improvements have been introduced in the development of the RoboShop platform, compared to the commercial Pioneer platforms, for these reasons, mentioned along the Section 3.3, this is the main platform chosen to develop and test this thesis.

Chapter 4

Development

One of the main goals of this thesis is the adaptation and/or development of obstacle avoidance algorithms that can work in dynamic environments.

In our previous works [Molinos, 2013] and [Molinos et al., 2013] the state of the art about obstacle avoidance algorithms has been compared and tested. Therefore, algorithms that are suitable for being modified and working with dynamic obstacles were selected.

First of all, a local mapping stage that can work with dynamic obstacles in an efficient way, and that can be provided as input for all the algorithms is proposed. Then, the algorithm CVM that is suitable for being extended has been implemented and tested under the ROS framework, and two extensions for dynamic avoidance are proposed: Predicted Curvature Velocity Method (PCVM) and Dynamic Curvature Velocity Method (DCVM).

Due to the fact that these extensions are able to avoid dynamic obstacles, but they have some flaws, another algorithm based on Dynamic Window Approach Dynamic Window for Dynamic Obstacles (DW4DO) is proposed and, therefore, extended to medium time planner Dynamic Window for Dynamic Obstacles Tree (DW4DOT). Finally, a mixture of the local and global planner based on the conclusions obtained Dynamic Lattice Planner (DLP) is proposed.

4.1 Local Mapping

Local navigation algorithms need information from the sensors, on each iteration, in order to react and plan the next movement of the robot. This information can be directly the raw measurements of range sensors (like LIDAR or ultrasound sensors) but, in this way, the algorithm can only work in a reactive way.

To avoid this problem, local mapping algorithms are proposed. The local mapping algorithms store the measurements from the sensors in a way that the local navigation algorithms can plan the next movements taking into account information not only measured in the actual time, improving its stability and allowing the algorithms to plan movements beyond the actual one. This is known as spatial memory.

Also, local mapping algorithms give more advantages, like coping with the uncertainties of the measurements, adding more information than the raw measurements and improving the computational efficiency of the mapping.

One kind of algorithm that has all of these characteristics is the occupancy grids. This algorithm divides the environment into cells (in 2D dimension spaces) or cubes (in 3D dimension spaces) and increases the occupancy value of each cell (or cube) with the measurements of each sensor that impact into the cell. In that way, the map can cope with the uncertainties of the sensors: the more impacts located into a cell, the higher the probability that an obstacle will be in this cell.

If we want to work with dynamic environments, planning in the cartesian space (2D or 3D) could not be enough, and the time dimension needs to be added, becoming a three dimensional grid in 2D spaces, or a four dimensional grid in 3D spaces. This dimension adds memory consumption to the algorithm, and increases the complexity of the search to the local navigation algorithms.

Our proposal is an occupancy grid, where the time is taken into account, but instead of adding it as a dimension of a fixed size, it is implicit in the information saved on each cartesian position.

The map created is always referred to the robot position (the robot is at the centre of the map) and divides its surroundings (carte-

sian coordinates) into cells of the same size. In this way, the continuous space is discretised and the search of the local navigation algorithms can be sped up. Also, dividing the environment into same size cells improves the map building, as the map can be stored as a two dimensional array, where each cell is referred to a cartesian coordinates limits, that can be calculated directly without needing information from the surrounding cells.

In the classical occupancy grid algorithms, each cell in the cartesian space stores its occupancy value. In our proposal, each cartesian space cell stores a list of cells that each one has information about its occupancy values and the time when it is stored. In this way, the time dimension is stored in that list, which limits only depends of the actual information and it does not have a fixed size, improving the memory consumption of the algorithm.

Each cell stores the following information:

- The occupancy value of the cell. If its occupancy value increases, it is more certain than the cell is occupied by an obstacle.
- The time when the occupancy value is valid.
- The position (without discretisation) of the point stored in the cell. It is useful to move the map information to the robot coordinate system, as the robot moves, without losing information by the discretisation method.
- The velocities of the point. It is used to predict the next point positions.
- The label of the obstacle to which the detected point belongs, if it is available.

The map needs measurement data (it can be obtained from one or more sensors) containing the following information:

- The cartesian position of the measurement point with respect to its coordinate frame.

- Current velocities of the obstacle. If the perception stage can not detect the velocities of the environment, the local mapping stage works as each obstacle was an static one.

- Number of obstacle where the point belongs, if the perception stage performs a clustering and data association of the environment. It is useful for the map building.

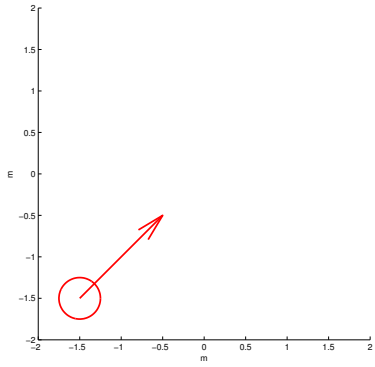
Figure 4.1 shows the differences between a classic 2D cartesian and time occupancy grid (4.1(b)) and our proposed occupancy grid (4.1(c)). In this example, a moving obstacle 4.1(a) with linear movement is mapped. Both grids have the same resolution (1x1 m cells in cartesian space and 1 second in time space). In this way, classic grid is a cube of 4x4x4 cells, even if only four of them are occupied by the obstacle (shown as red cubes). In the other hand, our proposed grid stores the 4x4 cartesian coordinates, but only the occupied cells are stored as cells. It is clear in this example that the memory consumption of our proposal is reduced with respect to the classic approaches.

The building of the local map is going to be explained. In order to achieve the spatial and temporal memory, two local maps are stored simultaneously, one that contains the information of the current iteration and one that contains the information of the previous iteration.

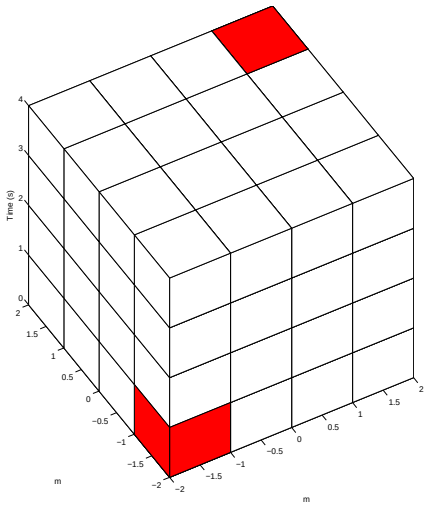
The algorithm to build the map is going to be explained along an example of each stage. The example situation is shown in figure 4.2 where a robot is situated in the centre of the map and orientated to the north (black box) and moving with linear velocity. Three obstacles are detected, two static obstacles with two detections each one (red circles) and one moving obstacle (blue circle, with a velocity represented as blue arrow).

The stages that the algorithm performs in order to build the map are the following:

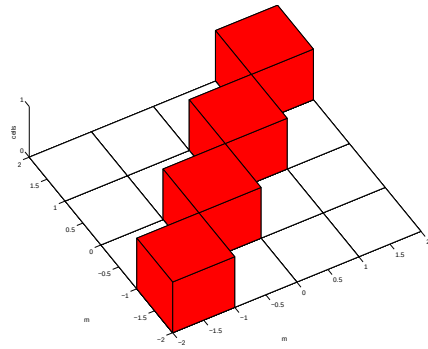
1. Current map is reset and its limits are set according to the localisation of the robot, in a way that the map is always centred in the robot and with the same orientation.



(a) Moving Obstacle



(b) Cartesian Grid + Time



(c) Proposed Grid

Figure 4.1: Local Mapping Example

2. Insert static obstacles: every point of the current point cloud that does not have any velocity detected is considered as static obstacle. These points are stored as a cell in the grid map. First, it is calculated the 2D coordinates position of the grid where the cell is going the be stored, and then it is stored with a maximum occupancy value. More than one point can be stored in the same 2D coordinates position grid.

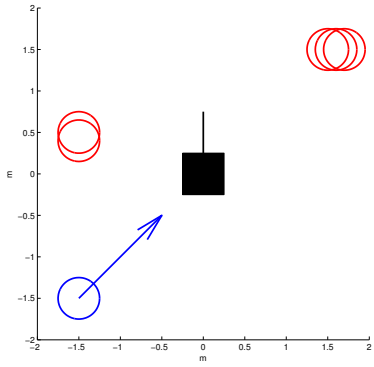


Figure 4.2: Local Mapping Example: Current Situation

Figure 4.3 shows the result of inserting the static obstacles of the example. Each obstacle has two detections that was impacted in the same 2D grid position, so two cells for each obstacle are stored (red boxes). The position and orientation of the robot is shown as a green arrow.

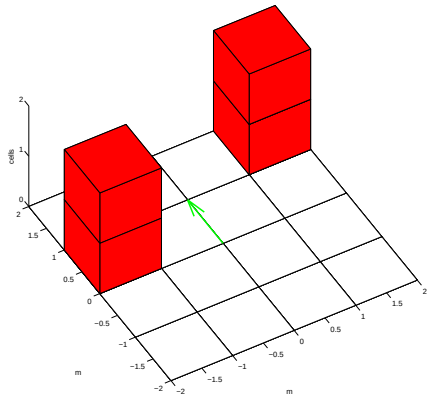


Figure 4.3: Local Mapping Example: Insert Static Obstacles

3. Insert dynamic obstacles detections and predictions: every point evaluated that has any velocity detected is marked as a dynamic obstacle. That point is stored in a cell with a time stamp of zero (current time). Then, the future positions of the point are predicted using the velocities associated to the

point. These future positions are calculated until the prediction lied outside the local map limits or it is beyond the time limit, avoiding in this way inserting positions in the map that are too far away in time. These predictions are stored with its associated time stamp. Also, if the detection has an obstacle label associated it is stored.

Figure 4.4 shows the result of inserting the dynamic obstacle detected in the example. The obstacle produces four cells (its current position and its future positions). These positions are represented as blue boxes, with lighter colour as its time stamp increases.

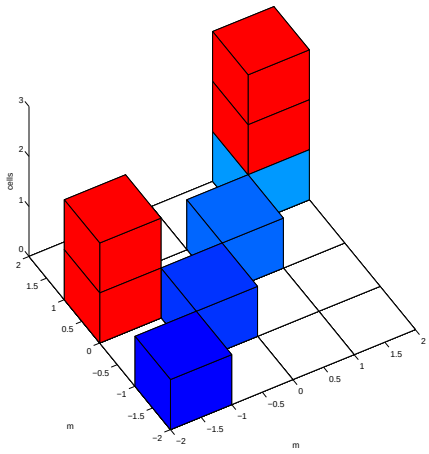


Figure 4.4: Local Mapping Example: Insert Dynamic Obstacles

4. Order and filter cells. In this stage of the algorithm, the list of cells on each cartesian grid position is ordered and filtered. First, it is ordered by timestamps (from closer to farther time), firstly static obstacles, and then dynamic obstacles. Once the list is sorted, if there is more than one point with the same time stamp in the cell (and the same label if the sensory information is labelled), they are fused, reducing the number of cells and saving memory space, and keeping a mean of its real positions avoiding the loss of information.

   In the example (4.5), the two processes are performed. First,

the cells are ordered by time stamp (predicted moving cell goes to the top of the list). Then, cells with the same time stamp and occupancy values (each static obstacle produces two cells in the same cartesian coordinates with the same time stamp) are fused, keeping only one cell per static obstacle.

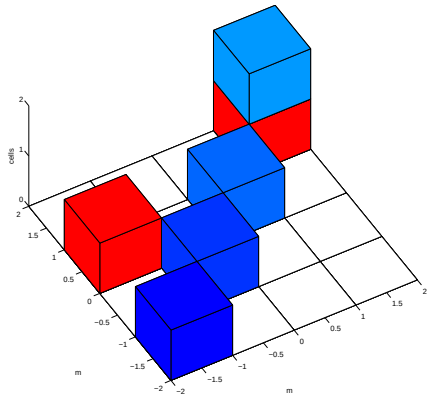


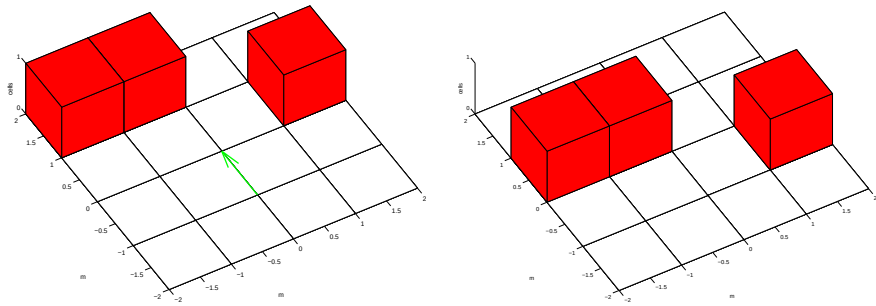
Figure 4.5: Local Mapping Example: Order and Filter Cells (First Time)

5. Move the previous local map. The local map is always centred in the robot, so the previous map needs to be translated and rotated as the robot moves to be referred to the robot frame. In order to do this operation, two different methods are performed, depending on whether the obstacle is static or dynamic.

    - If the cell has a high occupancy value (it is sure that a point has been detected inside the cell), the point stored in the cell is moved. First, the new cartesian grid where the point is located in the current map is calculated and, if the grid is between the local map limits, then the cell is stored.
    - If the cell has a high occupancy value and velocities associated, the new time stamp (subtracting the time stamp difference between the current time stamp and the previous iteration time stamp) is calculated, and if this time stamp is positive, the velocities of the cell are rotated and

the cell is moved as the static one. If the point is also labelled, and this obstacle was detected in the current iteration, the point is deleted. This deletion is made due to the dynamic obstacles can change their velocities during time, so it is better to track the current velocities than the last time prediction ones.

In the example (before moving 4.6(a) and after moving 4.6(b)), three static grids where detected in the previous iteration. As the robot is moving with linear velocities, these detections are moved, in this case, with a displacement only in the x axis.



(a) Previous Local Map (b) Previous Local Map Moved

Figure 4.6: Local Mapping Example: Move the Previous Local Map

6. Raytrace: Before adding the previous local map and the current one, a raytrace in the field of view of the robot is performed. Following lines from the robot, as a ray sensor does, if a point is detected now behind a previous detection, this one is deleted as it is impossible that the obstacle is there (maybe the obstacle has moved or was an erroneous detection). If an obstacle in the previous iteration is behind a current obstacle, it is kept because there is not new information about this point. Also, all the points in the previous iteration that are located outside the field of view of the sensor are kept.

   Figure 4.7 shows the raytrace performed in the moved previous local map. The cells covered by the field of view of the robot are

shown in pale blue. The raytrace from the current detections to the robot are shown as blue lines. One of the lines traverses one previously detected cell (shown in light red) and it is deleted, as it is impossible that an obstacle is there because in the current perception stage one obstacle is detected behind it.

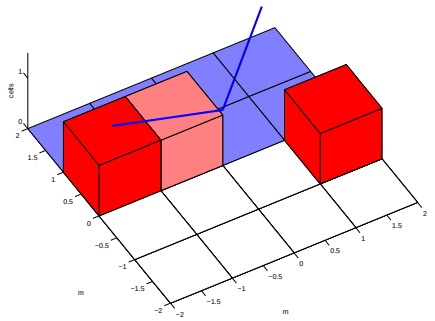


Figure 4.7: Local Mapping Example: Previous Local Map after Raytrace

7. After moving and raytracing the previous iteration local map, both local maps are added. Figure 4.8 shows the result in our example.

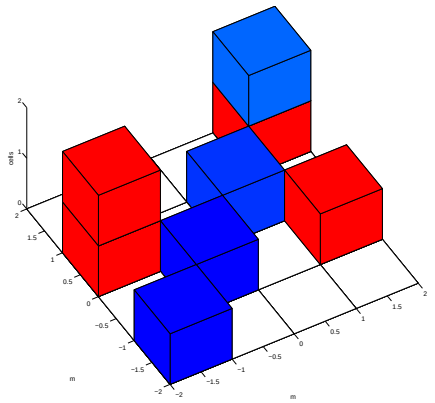


Figure 4.8: Local Mapping Example: Add Previous and Current Local Maps

8. Order and filter the added local grid. As the addition of the two local maps can result in more cells in each cartesian coordinate grid, the ordering and filtering process is repeated. In this case, the ordering is performed in the full local map, but the filtering process is performed only in the field of view of the robot (the surface from the robot that are covered by the sensors that are used. For example, commonly used LIDAR for mobile robots has a field of view of 180 degrees in front of the robot). This is because if the points that are located outside the field of view of the robot are merged, this will cause loss of information through the algorithm iterations.

   Figure 4.9 shows the result in the example, as one of the static obstacles detected is merged (because it is located inside the field of view of the robot).

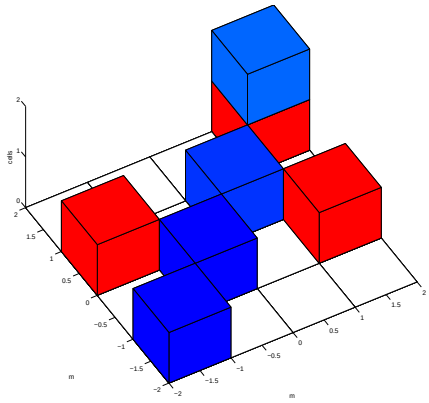
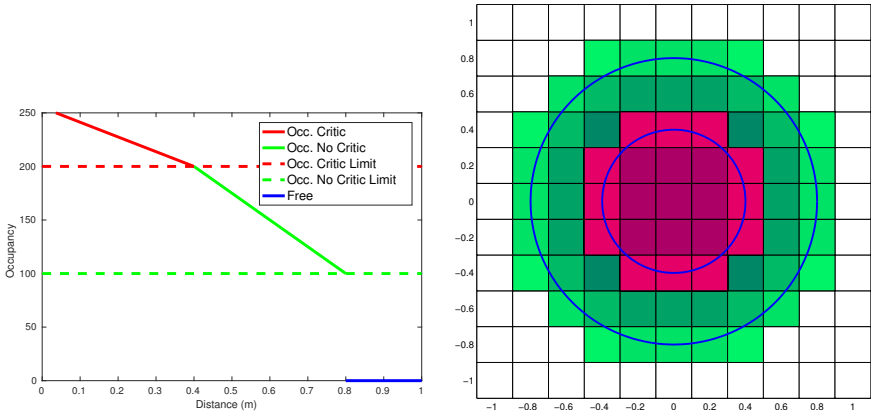


Figure 4.9: Local Mapping Example: Order and Filter Cells (Second Time)

9. Enlarge the obstacles: for navigation algorithms, it is useful that the obstacles are enlarged in order to avoid getting the robot too close to an obstacle. In our proposal each cell will be enlarged as circles with two different limits: critic and non critic. Critic limits are cells too close to the obstacles that should be avoided by robots (classical obstacle enlargement technique). Non critic limits cells are cells close to the obstacles

which the robot can traverse. This enlargement is useful to calculate the nearness of obstacle by local navigation algorithm, improving its computational cost.

Figure 4.10 shows an example of the enlargement of an obstacle. The occupancy value of each cell decreases with the distance, with two different ramps depending on the limit, as figure 4.10(a) shows. Cells which occupancy is below the non critic limit are considered as free cells. Figure 4.10(b) shows the enlargement of the obstacle in the cartesian grid. The obstacle is at the centre of the grid and the critic and non critic limits are represented as circles. The grids around the centre show the occupancy value (red ones has occupancy values above the critic limit and green ones above the non critic limit) of each cell, from darker colour (higher values) to lighter colours.



(a) Occupancy-Distance Function (b) Obstacle Enlarged

Figure 4.10: Obstacle Enlargement Example

Figure 4.11 shows the enlargement of obstacles in our example. To clarify the figure, the obstacles are enlarged only in the orthogonal adjacent cells, instead of a circular enlarged obstacle. The enlargement of static obstacles is shown as orange boxes and the enlargement of dynamic obstacles is shown as light purple boxes.

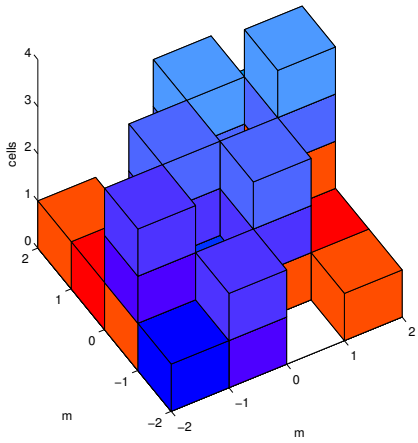


Figure 4.11: Local Mapping Example: Enlarge Obstacles

10. Order and filter cells, within the field of view of the robot, to obtain the final current local map. With the difference of the previous order and filter cells steps, the enlarged obstacle cells are also taken into account.

    First, each cartesian grid position needs to be ordered by time stamp, because in the cell enlargement stage, the new cells inserted can be added in the top position of each cartesian grid, making this position not ordered. Then, the static obstacles are evaluated and the maximum occupancy cell is kept (or the mean of the positions if there are more than one with the same occupancy level). Static cells with less occupancy value are deleted reducing the number of cells stored in the map.

    Then, the dynamic obstacles are evaluated. If there are more than one cell in the same cartesian position with the same time stamp, the same process of the static obstacles is executed (keeping the cells with maximum occupancy value). The dynamic cells are also compared with the static cells if there are some in the same cartesian position. If the static cell has more occupancy value than the dynamic cell, the dynamic one is deleted. If the dynamic cell has more occupancy value than

the static one, both cells are kept.

Figure 4.12 shows the result of the filtering in the example. As the enlargement of dynamic obstacles has the same occupancy value of the enlargement of the static ones in this example, all the grid positions where an enlarged dynamic and static obstacle are at the same time, the dynamic obstacle is deleted. Also, there are two static cells (resulting of the enlargement of the static obstacles in $x = -0.5, y = 1.5$ and $x = 1.5, y = 1.5$) in the position $x = 0.5, y = 1.5$ with the same occupancy value that are fused. The resulting map has eleven less cell occupied than before this stage.

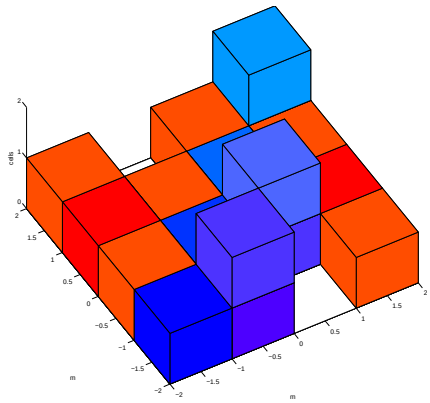
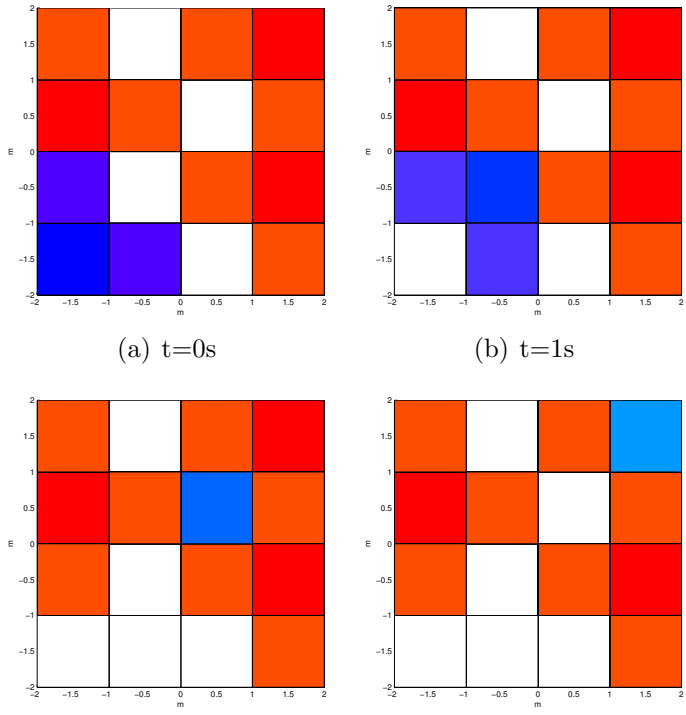


Figure 4.12: Local Mapping Example: Order and Filter Cells (Third Time)

After all these steps, Figure 4.13 shows the current local map result divided by the time stamp of the cells. Static obstacles (shown as red boxes and its enlarged cells as orange boxes) are stored in a way (with a special -1 time stamp) that they are present in all the time stamps evaluated. As the time changes, dynamic obstacles occupied cells (shown in blue and in lighter blue its enlarged cells) change with the velocity prediction.

Figure 4.14 shows the summarized flowchart of each iteration local mapping building, where blue boxes represent actions made in the current iteration local map and green boxes represent actions made in the previous iteration local map.



(a) t=0s (b) t=1s

(c) t=2s (d) t=3s

Figure 4.13: Local Mapping Example: Final Map

This proposed local mapping allows the local obstacle avoidance algorithms to take into account the dynamic of the environment. It reduces the memory consumption with respect to the time and cartesian coordinates grids, as the third dimension of the map varies its dimension dynamically as obstacles are detected by the sensors. The two level enlargement obstacle technique along with the raytrace technique allows the map to deal with measurement errors and provides information about the nearness of each obstacle to the local obstacle avoidance algorithms. Finally, as the real position of each detection is also stored in the map, it can be recovered if any algorithm needs it (like CVM does).

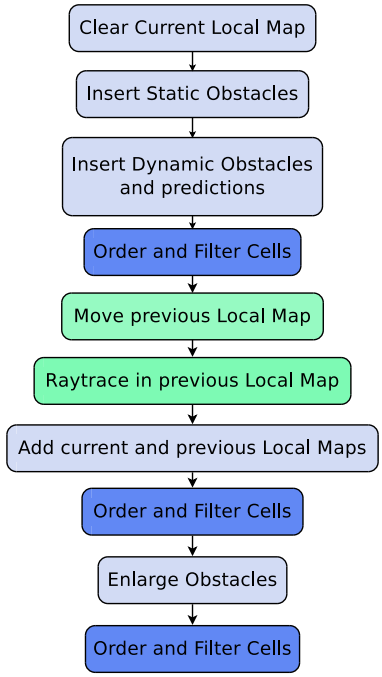


Figure 4.14: Local Mapping Flowchart

4.2 Algorithms Based on Curvature Arcs

In this section, CVM [Simmons, 1996] is analysed and implemented into the ROS framework. Then, the study and extension of the algorithm in order to work with dynamic obstacles is done, resulting in two proposed algorithms: PCVM and DCVM.

4.2.1 Curvature Velocity Method

CVM [Simmons, 1996] was the first method selected to work with. This algorithm has a set of characteristics that makes it suitable to be adapted to work with dynamic indoor scenarios:

- The method works with differential or non-holonomic robots.
- It takes into account the dynamic restrictions of the robot.

- It is able to plan paths where the time is implicit.

The main idea of the algorithm is that a differential or non holonomic robot can move following curvature arcs, defined by a pair of constant linear and angular velocities. The search space of possible curvature arcs can be reduced taking into account some constraints. Some of these constraints are set by the robot, like its kinematic restrictions (acceleration and velocities limits). Other constraints are imposed by the environment (arcs that leads to an unavoidable collision). CVM algorithm sectorises the environment in curvature intervals of the same distance free of obstacle in order to reduce the search space and increase the speed of the algorithm.

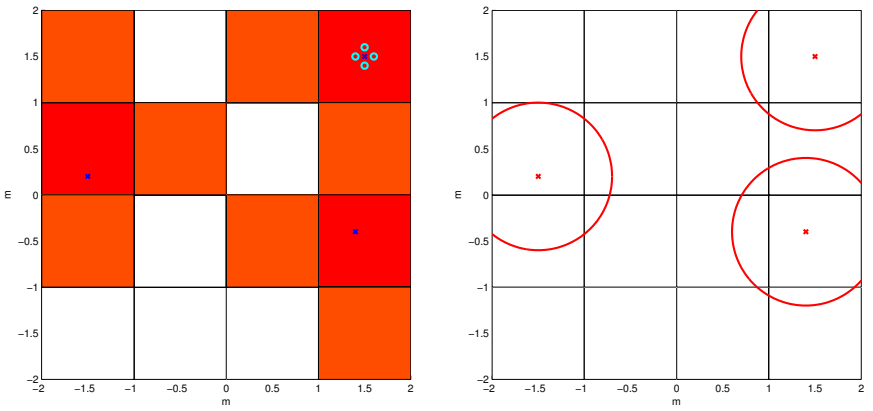
The algorithm has been implemented into ROS platform based on the previous work by Reid Simmons. Four stages are performed at each iteration of the algorithm: local mapping, curvature sectors creation, set of curvature arcs to be evaluated and curvature arc selection.

- **Local Mapping**: The algorithm needs a pointcloud as input, because it works with the current range detections (that can be provided by any distance sensor, like UltraSound, LIDAR, etc.). In our implementation the local mapping grid is used because it discretises the environment and reduces the number of points that have to be taken into account. Also, it allows the algorithm to store the previous detections, making the mapping stage more precise and improving the stability of the algorithm.

  Each cartesian grid in the local mapping stage contains the real point position (or the mean of the points if various points are located in the same grid), their positions are taken as an input by the CVM algorithm. Then, each obstacle is enlarged as a circumference of a fixed radius.

  Figure 4.15 shows the transformation between the local mapping stage (figure 4.15(a)) and the CVM local mapping stage (figure 4.15(b)). In the local mapping stage three cells have been marked as occupied and enlarged (red and orange boxes respectively). The real impacts are shown as blue cyan circles

and the mean position of detections inside a cell as blue circles. These positions are used as input by the CVM mapping stage (red crosses) that are enlarged as circles (not necessarily of the same radius as the previous local mapping stage).



(a) Local Mapping Stage (b) CVM Mapping Stage

Figure 4.15: CVM: Local Mapping Stage

- **Curvature Sectors Creation**: The environment around the robot is divided into sectors of the same distance to the obstacles limited by curvature arcs. In this way, the curvature arcs evaluation and selection is sped up. Each sector is defined by its curvature limits (minimum and maximum) and its free of obstacles distance that can be travelled by the robot.

  As the obstacles are modelled as circumferences in the local mapping stage (each detection is enlarged with a security radius), the tangent curvature arcs from the robot to the obstacle, if the robot keeps constant velocities, can be calculated. Equations 4.1 and 4.2 are used to calculate the maximum and minimum curvatures respectively, where x_{obs} and y_{obs} are the cartesian positions of the obstacles centre, and r_{obs} is the enlargement radius of the obstacle.

$$curv_{min} = \frac{x_{obs} - r_{obs}}{((x_{obs}^2 + y_{obs}^2 - r_{obs}^2)/2)} \tag{4.1}$$

$$curv_{max} = \frac{x_{obs} + r_{obs}}{((x_{obs}^2 + y_{obs}^2 - r_{obs}^2)/2)} \tag{4.2}$$

Each curvature arc defines a circumference of radius $|1/curv|$ and centre $x = 0$, $y = r$ or $y = -r$, depending on whether the curvature is positive or negative. The distance that the robot can travel along each arc until the obstacle can be obtained if the intersection between the obstacle circumference and the curvature arc is known. As the curvature arcs are tangent to the obstacle circumference, the tangent points can be obtained using the equations 4.3 and 4.4 for $curv_{min}$ and 4.5 and 4.6 for $curv_{max}$.

$$x_{min} = \frac{x_{obs} - r_{obs}}{1 - curv_{min} \cdot r_{obs}} \tag{4.3}$$

$$y_{min} = \frac{y_{obs}}{1 - curv_{min} \cdot r_{obs}} \tag{4.4}$$

$$x_{max} = \frac{x_{obs} + r_{obs}}{1 + curv_{max} \cdot r_{obs}} \tag{4.5}$$

$$y_{max} = \frac{y_{obs}}{1 + curv_{max} \cdot r_{obs}} \tag{4.6}$$

Once the intersection point is calculated, the angle where the point is with respect to the robot is calculated using equation 4.7, where y and x are the calculated points (maximum or minimum). To calculate the effective distance travelled until reach the point, 4.8 is used.

$$\phi = atan(\frac{y}{x - 1/curv}) \tag{4.7}$$

$$d = |\frac{1}{curv} \cdot \phi| \tag{4.8}$$

Figure 4.16 shows an example of the curvature arcs calculation, from the robot to the obstacles. The robot is at the origin and pointing to the north (represented as a black box). Obstacles are shown as red circles of different radius. Minimum and maximum tangent curvatures (shown as semicircles) and its intersection points with the obstacles (shown as small circles) are represented in green and blue colour respectively. The angular sector between the curvatures are represented as solid different colours for each curvature interval sector.

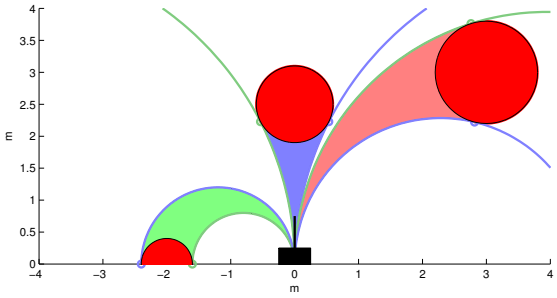


Figure 4.16: CVM: Tangent curvature arcs to obstacles

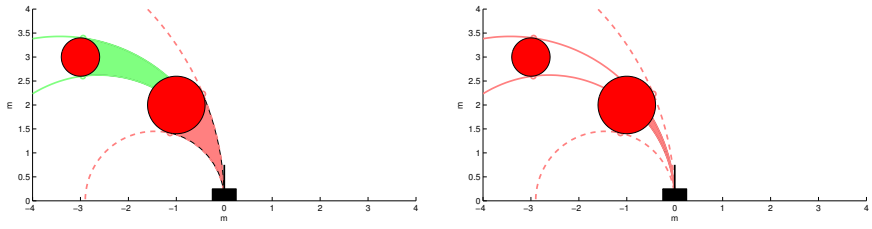
To calculate the minimum distance that the robot can travel inside a curvature sector until it collides with the obstacle, all the points between the tangent curvatures should be calculated. It is computationally expensive and could not satisfy the time constraints in real robotics scenarios. Simmons proposes an approximation where the obstacle is divided into quadrants using the line that joins the robot with the obstacle centre. Then the distance to the limits of each quadrant is calculated using the previous equations and the resulting distance that the robot can travel is the minimum of the calculated distances. If the obstacle is divided into more quadrants, the distance calculated is more precise, but more computation load is required.

Once all the curvature intervals are created, an algorithm to insert the curvature intervals, reducing its number, is executed.

The interval creation begins with a single interval with infinite curvature as limits. Each new interval (defined by d_{new},$cmin_{new}$ and $cmax_{new}$) is evaluated and compared with all the existing intervals. Depending on the relationship between the evaluated interval and compared existing interval (defined by d_{old}, $cmin_{old}$ and $cmax_{old}$), four different actions are performed:

- If $cmin_{new} \leq cmin_{old}$ and $cmax_{new} \geq cmax_{old}$ the compared interval is contained by the interval. If $d_{new} < d_{old}$, the compared existing interval is modified setting its distance as the minimum of d_{new} or d_{old}.

  Figure 4.17 shows an example of this case. Dashed red lines represent the evaluated interval and green lines represent the compared existing interval. As the evaluated interval free distance is less than the compared existing interval, it contains the compared existing interval. Before the comparison, the existing interval distance is set to the new one (red interval in figure 4.17(b)).

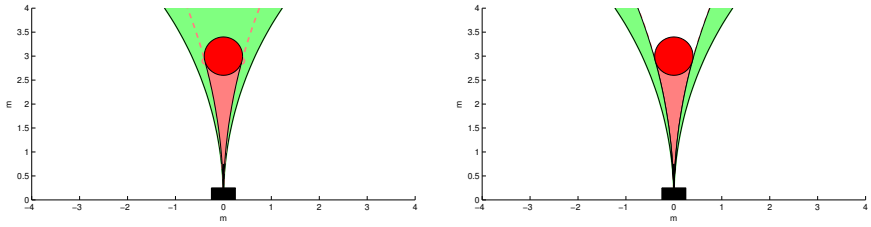


(a) Before comparison (b) After comparison

Figure 4.17: CVM: Curvature Interval contained in the current one

- If $cmin_{new} \geq cmin_{old}$, $cmax_{new} \leq cmax_{old}$ and $d_{new} \leq d_{old}$ the existing compared interval contains the new interval. In this case the new interval is inserted in the list and the existing interval is split, resulting in three intervals: one with $cmin = cmin_{old}$,$cmax = cmin_{new}$ and d_{old}, the new interval $(cmin_{new}, cmax_{new}, d_{new})$ and one with $cmin = cmax_{new}$,$cmax = cmax_{old}$ and d_{old}.

Figure 4.18 shows an example, where the existing evaluated interval (solid green lines) contains the evaluated interval (dashed red lines) and its distance is less than the existing one. The result after the comparison (Figure 4.18(b)) are three intervals, one from the minimum curvature of the existing interval to the minimum curvature of the evaluated interval (left green one) with the greater distance, one from the maximum curvature of the evaluated interval to the maximum curvature of the existing one, with the greater distance (right green one) and the evaluated interval (red central one).



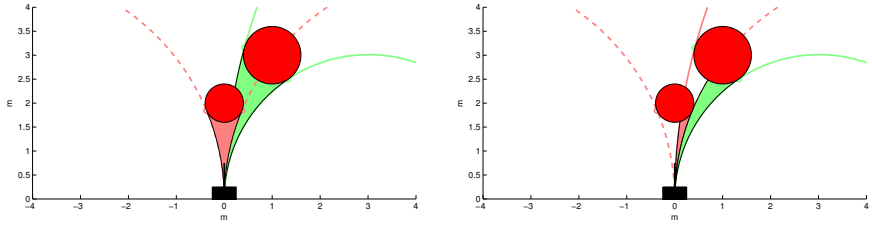
(a) Before comparison (b) After comparison

Figure 4.18: CVM: Curvature Interval contains the current one

– If $cmax_{new} > cmax_{old}$ and $cmin_{new} > cmin_{old}$, or $cmin_{new} < cmin_{old}$ and $cmax_{new} < cmax_{old}$ the intervals are overlapped. If $d_{new} < d_{old}$ the existing interval is split into two: if $cmax_{new} > cmax_{old}$ has $cmin = cmin_{old}$,$cmax = cmin_{new}$,d_{old} and $cmin = cmin_{new}$,$cmax = cmax_{old}$,d_{new}, otherwise $cmin = cmin_{old}$,$cmax = cmax_{new}$,d_{new} and $cmin = cmax_{new}$,$cmax = cmax_{old}$,d_{old}.

  Figure 4.19 shows an example of overlapped intervals. Dashed red lines represent the evaluated interval, and green arc is the existing interval (figure 4.19(a)). As both arcs are overlapped, the existing arc is split into two (figure 4.19(b)): in the area where both arcs are overlapped the minimum distance is kept (red left interval), and where

both intervals are not overlapped the existing distance is kept (green right interval).



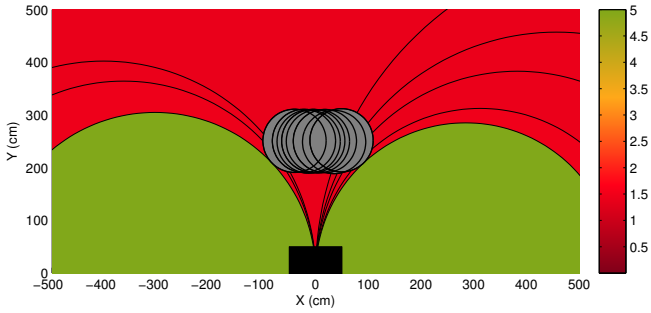
(a) Before comparison (b) After comparison

Figure 4.19: CVM: Curvature Interval Overlapped

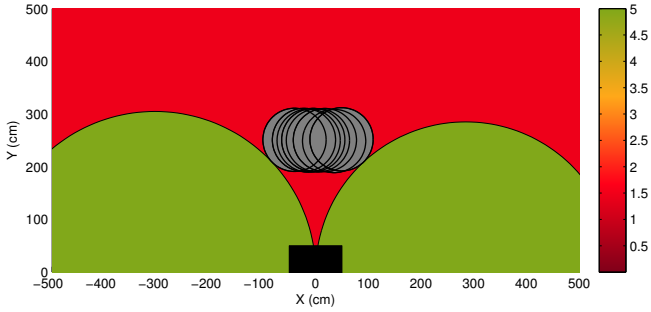
– If none of this cases occurs, the curvature sectors are considered disjoint, and no actions need to be performed.

Once all the sectors are inserted in the curvature sector list, an algorithm is executed to reduce the number of sectors to compute. The algorithm deletes sectors that are too narrow (difference between c_{min} and c_{max} below a threshold), and merges consecutive sectors that have similar distances. The distance similarity has a threshold that varies with the distance: it is more important to keep the difference of distance in curvature sectors with less distance to travel than in the ones that have a lot of distance free of obstacles.

Figure 4.20 shows the interval reduction algorithm. The robot is at the origin and pointing to the north (represented as black box). There is an obstacle in front of the robot, which has multiple detections (represented as grey circles). Figure 4.20(a) shows the curvature intervals after all of them are inserted. Colour represents the distance free of obstacles (from red to green). It is clear that the distance free of obstacles of the curvature sectors that lead to the obstacle are very similar. After the reduction (figure 4.20(b)) only three curvature interval are in the list: one interval that leads to the obstacle where multiple intervals were merged, and one interval on each side of the obstacle.



(a) Before reduction



(b) After reduction

Figure 4.20: CVM: Curvature Interval Reduction

- **Set of curvature arcs to be evaluated**: The sectorisation of the environment helps to reduce the number of velocities to be evaluated. All the possible velocities are limited by the robot maximum speeds and accelerations at each time. As the algorithm prizes travelling at high speeds, and the distance free of obstacles is set to the same value along a curvature interval, a subset of the possible speeds is selected to guarantee candidates with a higher value in the evaluation function on each interval.

  One curvature arc is evaluated among each curvature limits of the interval (maximum and minimum curvature arcs) where the linear velocity is set as the maximum velocity and the rotational velocity is calculated in a way that the curvature arc has the same curvature than the curvature arc limits of the interval. If

the rotational velocity is reachable by the robot these velocities pairs are evaluated.

Then, the maximum and minimum rotational velocities are evaluated. Depending on the sign of the rotational velocity, the maximum linear velocity that lies inside the curvature arcs limits are evaluated.

Finally, a third rotational velocity that moves the robot towards goal is evaluated.

To avoid travelling at maximum speed when the obstacles are too close to the robot, the maximum linear velocity on each interval is limited by the minimum of the maximum reachable linear velocity of the robot and the division between the interval distance free of obstacles and a time to impact previously set, in a way that the robot goes slower when it is approaching to obstacles.

Figure 4.21 shows an example of this set of candidate velocities selection. In the example, there are three curvature sectors with different distance free of obstacles (green surfaces have more distance free of obstacles than red one). The actual velocity of the robot is represented as a black circle and the reachable linear and angular velocities are the ones bounded by the black rectangle. Magenta dashed line represents the curvature arc that moves the robot towards goal and black dashed line represents the maximum linear velocity boundary of the more dangerous curvature sector.

For the right curvature sector only two curvatures are selected: the maximum linear velocity along the minimum curvature limit (purple circle) and the maximum linear velocity along the maximum rotational velocity limit (red circle). The pair of linear and angular velocities that moves the robot towards the goal coincides with the velocity along the minimum curvature arc. The maximum curvature arc is not reachable (bounded by infinitum curvature), neither the minimum angular velocity.

For the central curvature sector, two curvatures are selected along the minimum and maximum curvatures (blue circles)

bounded by the maximum velocity limit (dashed black line). The curvature that approaches the goal coincides with the velocity selected along the minimum curvature limit and the maximum and minimum angular velocities are not reachable.

For the right curvature sector three velocities are selected, one along the maximum curvature sector (purple circle), one with the minimum angular velocity (red circle) and one that moves the robot towards the goal (grey circle). The maximum angular velocity and the minimum curvature interval are not reachable for this curvature interval.

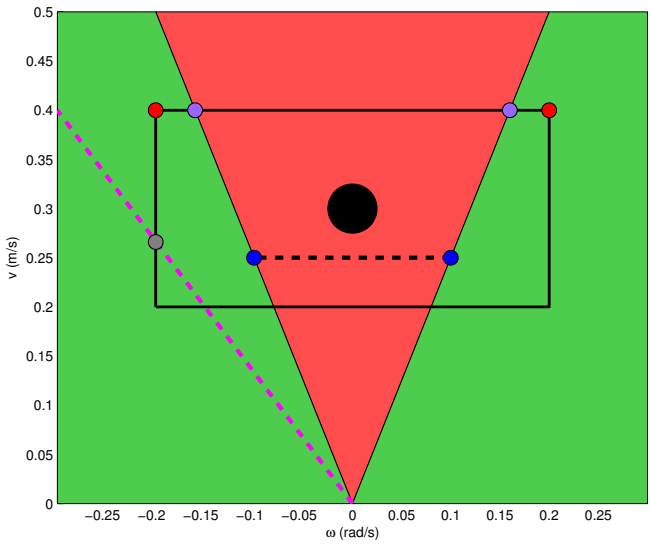


Figure 4.21: CVM: Set of curvature arcs

- **Curvature Arcs Selection**: Once the subset of linear and angular velocities is created, all of them are evaluated using a cost function (equation 4.9) that needs to be maximised. This function has three terms: velocity, heading and distance free of obstacles, each of them with configurable weights (α).

$$f(v,\omega) = \alpha_{speed} \cdot speed(v) + \alpha_{dist} \cdot dist(v,\omega) + \alpha_{head} \cdot head(\omega) \tag{4.9}$$

Velocity term $speed(v)$ is calculated using the equation 4.10 where v is the evaluated linear velocity and v_{MAX} is the maximum linear speed of the robot. This equation is monotonically increasing with the linear velocity and does not depend on the angular velocity.

$$speed(v) = \frac{v}{v_{MAX}} \tag{4.10}$$

Distance free of obstacle $dist(v, \omega)$ is calculated using the equation 4.11 where $d(v, \omega)$ is the distance free of obstacles of the curvature interval and d_{MAX} is the maximum distance free of obstacles, used to normalize the equation. There can be special cases, especially in the curvature arcs with infinite curvature radius limits boundaries, where d_{MAX} implies that the robot performs more than one turn. In these cases $d(v, \omega)$ is set to the distance travelled by the robot when performs one full circumference. This function is constant on each curvature interval.

$$dist(v, \omega) = \frac{d(v, \omega)}{d_{MAX}} \tag{4.11}$$

Orientation term $(head(\omega))$ is calculated using the equation 4.12 where $dHead$ is the heading difference to the goal, t is a time constant and ω' is a rotational velocity, calculated from the linear velocity, in a way that the result of $\omega' \cdot t$ of all the pairs of linear and angular velocities that follows the same curvature is the same. The function will be increasing or decreasing in function of the curvature.

$$head(\omega) = 1 - \frac{|dHead - \omega' \cdot t|}{\pi} \tag{4.12}$$

There are two specific situations where the heading weight (α_{head}) is modified: when the arc leads the robot towards the goal and when the goal is too far away.

When the goal is too far away from the orientation of the robot α_{head} is modified using the equation 4.13 where α_{far} is another

weight that needs to be set. This function increases the orientation weight with the orientation difference to the goal, increasing the chance to turn towards the goal.

$$\alpha_{head} = \alpha_{head} \cdot (1 + \alpha_{far} \cdot \frac{dHead^2}{\pi^2}) \tag{4.13}$$

When the curvature arc leads the robot towards the goal, its weight is drastically increased in a way that the selection of an arc that goes to the goal is almost sure. In order to do that, the goal is enlarged as an obstacle (with the goal tolerance radius) and a curvature sector is built towards it. If the evaluated curvature arc is contained by this curvature sector (it is between the curvature limits and its distance to travel is less or equal than the distance to the goal) α_{head} is multiplied by another weight (α_{goal}).

Also, a escape strategy is performed by the robot, when the selected interval has a very small distance free of obstacles. In this case the robot begins turning without translational speed until a valid curvature interval is selected.

A total of five weights are involved in the selection of the best curvature arc to follow, therefore a tuning of these weights is needed. The algorithm always maintains the safety of the robot (it does not select curvature arcs that goes towards an unavoidable collision) but it works with different behaviours:

- If α_{vel} is the dominant weight, the robot will choose curvature arcs that allows it to travel at high speeds without regards of the orientation of the goal.
- If α_{dist} is the dominant weight, the robot will travel in curvature intervals with high distance free of obstacles (safer intervals). As the maximum velocity is in relation with the distance free of obstacles, the robot will also travel at high speeds.
- If α_{head} is the dominant weight, the robot will head to the goal more aggressively, even if it needs to travel slowly or in more dangerous intervals.

- If α_{goal} is set, and with a weight larger than the rest, the robot will head directly towards the goal if it is reachable (without obstacles in the way to the goal), which is a desired behaviour.
- If α_{far} is not set, the algorithm tends to fail if the goal is behind the robot. As this weight is increased, the robot will turn towards the goal in a more aggressive way.

There is not an optimal weight configuration, as it depends of the behaviour intended, the environment where the robot is moving and the robot itself (its velocities and acceleration limits).

Finally, the curvature arc that has the higher value in that cost function will be the one commanded to the robot. As the curvature arcs are limited by the velocities and accelerations of the robot, this curvature arc (pair of linear and angular velocities) is directly executable in differential and non holonomic robots.

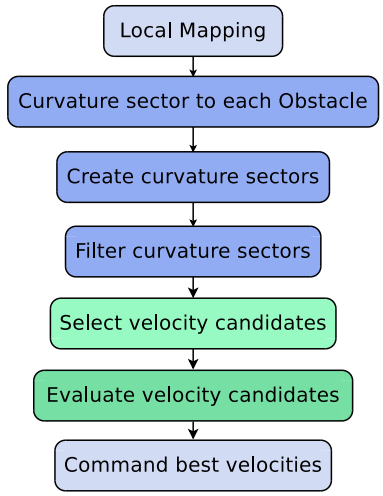


Figure 4.22: CVM: Flowchart

Figure 4.22 shows the summarized flowchart of each iteration of the algorithm, from the local mapping (light blue box), environment

sectorisation (blue boxes), curvature selection (green boxes), to the velocities commanded to the robot.

4.2.2 Predicted Curvature Velocity Method

CVM algorithm is able to avoid some dynamic obstacles under certain assumptions, especially if the velocity of the obstacles is lower than the robot one. In this case, dynamic obstacles are avoided like static ones on each iteration.

PCVM is an extension of the CVM algorithm for dynamic obstacles avoidance. In order to do that, the perception stage is modified. This implementation was tested and presented in our work [Llamazares et al., 2014].

The modified perception stage uses the position of the obstacles (after a fixed time t) within the world frame, to create the curvature arcs from the robot. In this way, the robot can plan the next movement using the future position of the obstacles, getting ahead of the dynamic obstacles movement and anticipating the avoidance manoeuvre.

In order to achieve this algorithm, the perception system must provide the prediction of the dynamic obstacles position after a fixed time, or, at least, the actual velocity of the obstacles, from which the next position of the obstacle can be calculated assuming that the velocities will be kept constant between iterations.

PCVM can deal better with dynamic obstacles than the original CVM algorithm, however the use of only one future position without the knowledge of the obstacles path can cause problems. Figure 4.23 shows one possible risky situation: the robot is situated at the origin and orientated to the north. One future position is taken into account in the algorithm (red circle). After the PCVM evaluation, the curvature arc (shown as green line) is selected by the cost function. This curvature arc is safe if the obstacle is static, however, as the obstacle is moving, the curvature arc crosses the path of the obstacle and can collide with it as the time where the robot and obstacle crosses at the same point is not evaluated. The actual position of the obstacle is shown as dashed blue circle and the movement between the two time iterations is shown as dashed cyan lines. In order to avoid

these unwanted and risky situations, the time where the positions are predicted should have small value, even though when the algorithm reacts later and in a more reactive way to the dynamic obstacles.

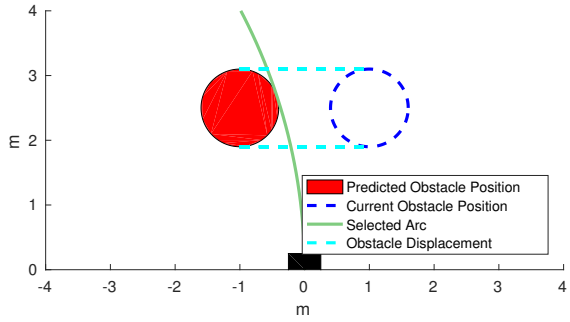


Figure 4.23: PCVM: Risky Situation

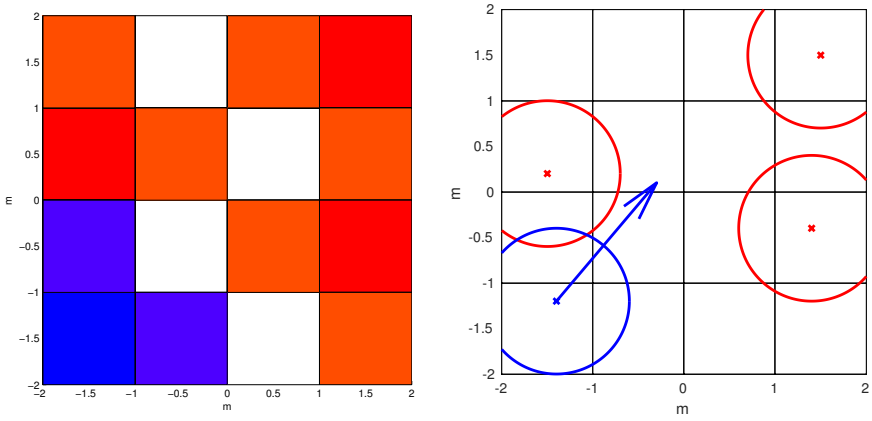
4.2.3 Dynamic Curvature Velocity Method

The using of future positions of obstacles improves the avoidance of the dynamic ones, but it has the mayor issue discussed before, as the curvature arc selected can collide with obstacles position between the times evaluated. Also, even when the path planned is safe, traverse curvature arcs that crosses the obstacles path can be dangerous if the future position of the obstacles has some prediction errors, or the velocities of the obstacle change during time. For this reason DCVM is proposed.

DCVM is a modification of CVM algorithm, where the movement of the dynamic obstacles is taken into account. In order to achieve this, the local mapping and the curvature selection stage are modified. This algorithm has been presented in our previous work [Molinos et al., 2014].

The first change is in the local mapping stage. The algorithm needs to store the points detected by the laser and its velocities, that are considered constant between time steps, in order to predict its future positions. This information can be obtained by the proposed local mapping stage, where the point and velocities of each cartesian grid are stored.

Figure 4.24 shows how the information from the local mapping stage is transformed into the DCVM mapping stage. 4.24(a) shows the actual detection and the prediction in the current time: cells occupied by static obstacles are represented from red to orange colour, depending on its occupancy level, and dynamic ones are represented in blue colours. As the real position of the obstacle into each cell is saved, it is used by the DCVM mapping stage (figure 4.24(b)) to enlarge the obstacles instead of working with the cell discretisation. Red crosses represent static obstacles, blue cross the dynamic obstacle and the circles the enlargement radius of each one. Also, the velocities of the dynamic obstacles are stored (blue arrow shows the magnitude and direction of the velocities).



(a) Local Mapping Stage (b) DCVM Mapping Stage

Figure 4.24: DCVM: Local Mapping Stage

The curvature sectors building, from the static obstacles, and the candidate curvature selection, are performed in the same way than the CVM algorithm does (to the static obstacles and the current position of the dynamic ones). Then, the predicted movement of the obstacles is computed and checked against each evaluated curvature.

If the predicted obstacle positions and the curvature that the robot should follow intersect (both future positions are calculated considering that the velocities will maintain constant during time), there is a potential risk of collision. This intersection is calculated

considering that the obstacle will move only with linear velocity forming a line, and the robot will move along a circumference of radius $\frac{v_r}{\omega_r}$ and its centre at the left or right of the robot, depending on the sign of its angular velocity ω_r. In order to evaluate if the collision will be probable, the time when the robot and the obstacle will be in the point is calculated. Equation 4.14 is used to calculate the time where the obstacle will be in the point if it is moving only with linear velocities, where x_p, y_p is the intersection position between the robot and the obstacle, x_{obs}, y_{obs} the current position of the obstacle and v_{obs} its linear velocity. In the case of the robot, equation 4.15 is used, where x_c, y_c is the centre of the circumference described by the curvature radius and x_r, y_r is the robot position.

$$t_{obs} = \frac{\sqrt{(x_p - x_{obs})^2 + (y_p - y_{obs})^2}}{v_{obs}} \tag{4.14}$$

$$t_r = \frac{2 \cdot \pi \cdot (atan(\frac{y_c - y_r}{x_c - x_r}) - atan(\frac{y_c - y_p}{x_c - x_p})}{|\omega_r|} \tag{4.15}$$

If the time difference has a low value, the collision between the obstacle and the robot (if both keep the same constant velocity) is probable, and the algorithm should take it into account. Also, even if the time difference has not a low value the algorithm should try to avoid situations win which both path cross, because, as it has been discussed in the state of the art section, if the velocity measures are not exact or these velocities change during time, potentially risky situations can happen. For this reason a new cost function in the curvature arc calculation is proposed (equation 4.16).

$$f_{DCVM}(v_r, \omega_r) = f_{CVM}(v_r, \omega_r) - \alpha_{dyn} \cdot f(v_r, \omega_r/3) \cdot (riskiness - T_{impact}) \tag{4.16}$$

In this equation, the cost value of each curvature interval is reduced if the *riskiness* parameter increases. The *riskiness* value can take three different values (0, 1 or 2), depending on the minimum $|t_{obs} - t_r|$ that is obtained after the evaluation of all the dynamic obstacles intersections. Two time difference threshold are set, a not risky threshold and a collision threshold. The three possible cases are:

- If there are no intersections or the intersection time is over the not risky threshold, the curvature arc is considered secure, therefore the value of f_{DCVM} is the same as f_{CVM}, as the cost function is not modified.

- If the intersection time is over the collision threshold and below the not risky threshold, the paths cross and there will be a possible intersection, so the value of f_{CVM} is reduced.

- If the intersection time is below the risky threshold, a collision is probable and the cost function is reduced in a more drastic way.

T_{impact} is defined by the equation 4.17 which is the time where the robot will be at the risky position. This function is normalised by a maximum time value. In this way, if the risky situation is close to the robot (time near zero) the curvature arc is more dangerous, and if the risky situation will be far in the future (near the t_{max} set) the dangerous of the arc will be decreased.

$$T_{impact} = t_r / t_{max} \tag{4.17}$$

In this way, if there are no intersection with any dynamic obstacles, the algorithm works in the same way than the CVM does. If the planned curvature arc crosses an obstacle path, the cost of that curvature arc is decreased, making it less probable to be selected, with less probability as the risky situation happens closer to the robot. αdyn is a weight added to the cost function that needs to be tuned. If its value increases, the probabilities of selecting curvature arcs that does not crosses any obstacle path increases, increasing the safety of the algorithm, but it does not guarantee that optimal paths to the goal were planned, because paths does not cross any obstacle path will be selected, even if following these paths moves the robot away from the goal.

Figure 4.25 shows a simplification of the algorithm: X_r and Yr are the cartesian positions of the robot, X_g and Y_g are the cartesian positions of the goal. Each curvature arc can be represented as a circumference of centre (X_c,Y_c) and radius r. In this simplification,

curvature arcs (dashed and dotted lines) are created by the obstacles limits and the goal. As the dynamic obstacle (red box) has a linear movement (blue dotted line), the possible collision points are represented as stars. Once the curvature arcs are computed, two of them do not have any *riskiness* value because they do not cross with moving obstacles path (shown with green dashed lines), one of them goes toward a possible collision (shown with red dotted line) and have the higher *riskiness* value, and another one crosses with the obstacle path, although the collision is not probable if the obstacle and the robot maintain the same velocities (yellow dashed line, less chance of choice because of the *riskiness* value).

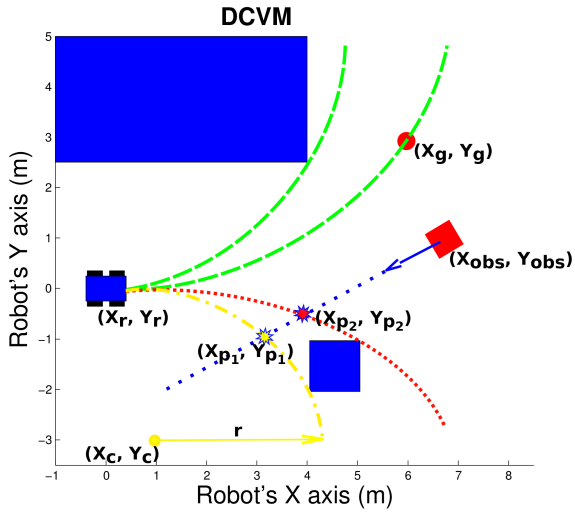


Figure 4.25: DCVM Diagram

4.2.4 Conclusions

In this section, the CVM algorithm has been implemented into the ROS framework. Due to its limitation avoiding dynamic obstacles (the velocities of the obstacles are not taken into account), the algorithm has been extended into two new proposed algorithms: PCVM and DCVM, which are able to deal with the dynamics of the environment.

DCVM improves the security of the robot and the environments, and it not only takes into account the dynamic obstacles to avoid them, but also tries to not cross the robots path with the dynamic obstacles path. Using the proposed cost function the security is improved with respect to algorithms that only checks the unavoidable collision scenarios in order to plan the next movement.

But these algorithms still has some flaws. As the sectorisation of the environment is created from the enlarged obstacles, the nearness of the obstacles is not taken into account resulting in paths that usually go close to the enlarged obstacles, and that can be blocked if the obstacle was not correctly detected.

Also, as the possible arcs to be evaluated are derived from the sectorisation of the environment, its number could change drastically during the execution, especially in dynamic environments. This results in a variation in the time needed to perform each iteration of the algorithm (it increases as more curvature sectors are detected). Also, this curvature arcs evaluation could result in unstable paths because the current velocity of the robot is not taken into account in the cost function.

Even with these flaws, DCVM algorithm has proven to be secure and works in real dynamic environments as will be shown in the results chapter.

4.3 Dynamic Window based Algorithms

Methods based on the curvature arc approach can avoid dynamic obstacles assuring its security, but have some problems inherited from the evaluated curvature selection and its cost function. For this reason, another algorithm based on the Dynamic Window Approach is presented: Dynamic Window for Dynamic Obstacles (DW4DO). Therefore, its extension to a medium time search planner Dynamic Window for Dynamic Obstacles Tree (DW4DOT) will be proposed with the aim to increase the optimality of the path planned, although the computational cost will be increased.

4.3.1 Dynamic Window for Dynamic Obstacles

The Dynamic Window Approach DWA [Fox et al., 1997] is one of the most used methods for local obstacle avoidance in mobile robots. The idea of this method is that if it is executed with a fixed frequency, only a set of velocities can be commanded to the robot due to its acceleration and velocity limits. Between this set of velocities (which can change on each iteration) a cost function is proposed in order to select the best velocities to follow. DWA can be adapted for differential, non-holonomic and holonomic mobile robots.

Our proposed implementation of this approach, called DW4DO has some variations from the original method:

- Local mapping has been modified to include dynamic obstacles and time information.
- Collision checking is made taking time into account, in a way that dynamic obstacles can be avoided.
- Two different velocity windows are evaluated at the same time, in order to help the algorithm to reach the best trajectories, even when its acceleration limits are very low.
- A new cost function is proposed that takes into account the two velocities windows and the dynamic obstacles.

The input of the algorithm (local mapping stage) is directly the local mapping stage proposed at the beginning of this chapter, in a way that dynamic obstacles can be taken into account.

DW4DO is oriented to differential or non-holonomic robot, which moves in pair of linear and angular velocities (curvature arcs), although it can be adapted (with slightly modification in the window building and the weight function) for holonomic robots. Two set of velocities are evaluated on each iteration of the algorithm, from two simultaneous velocity windows centred in the robot:

- One window with velocities that are reachable by the robot in the next iteration (limited by its acceleration and maximum velocities), with a small velocity step. This window is called "local window".

- One window with velocities bounded by the velocity limits of the robot, with a velocity step bigger than the *local window*. This window is called "best window" and calculates curvature arcs without taking into account the kinematic constraints (accelerations) of the robot.

The reason to use that two level window is because if the algorithm iterates at high rates and/or the accelerations of the robot are very low, the local window where the algorithm searches possible trajectories could be really small, therefore paths that are reachable in the next iterations could be discarded, and trajectories would be planned in a very reactive way. In our implementation, the best curvature arc between the velocity limits of the robot is selected first and then used as an input to the cost function of the arc in the local window, in a way that the selected arc can move the robot near the best path (in the current situation) even if it is not reachable in the next iteration.

The set of velocities to evaluate is a two dimensional matrix with different limits and steps between its coordinates depending whether it is the *best window* or the *local window*.

The "best window" is fixed during the execution of the program. Its limits are: the maximum and minimum angular velocities, maximum linear velocity and zero. Negative linear velocities are not considered to avoid calculating trajectories that makes the robot go backwards (even it can be added if the robot has 360 degrees field of view sensors.). The step between the linear and angular velocities (can be different for both dimensions) is fixed and it is recommended to be bigger than the *local window* to avoid increasing the computation cost of the program.

The limits of the *local window* are calculated on each iteration of the algorithm. The window is centred in the robot and its limits are defined by the equations 4.18 and 4.19 for angular velocities, and equations 4.21 and 4.20 for linear velocities, where $\omega_{interval}$ and $v_{interval}$ are the maximum acceleration to be considered divided into the granularity to evaluate (the discretisation of the accelerations), and ω_{step} and v_{step} are the number of intervals, in a way that $\omega_{step} \cdot \omega_{interval}$ are the maximum velocity possible in that iteration.

This local window is also bounded by the maximum and minimum angular velocities, and the maximum and zero linear velocities, avoiding trajectories that makes the robot moves backwards.

$$\omega_{min} = -\omega_{step} \cdot \omega_{interval} + \omega_t \tag{4.18}$$

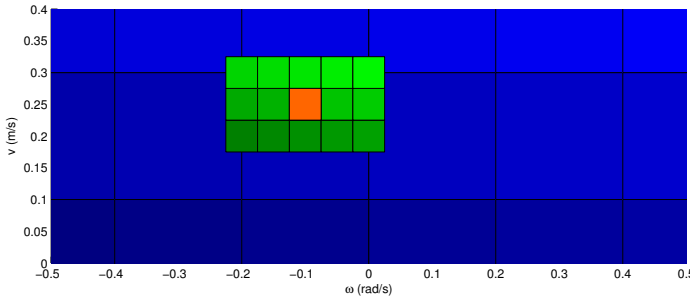
$$\omega_{max} = \omega_{step} \cdot \omega_{interval} + \omega_t \tag{4.19}$$

$$v_{min} = -v_{step} \cdot v_{interval} + v_t \tag{4.20}$$

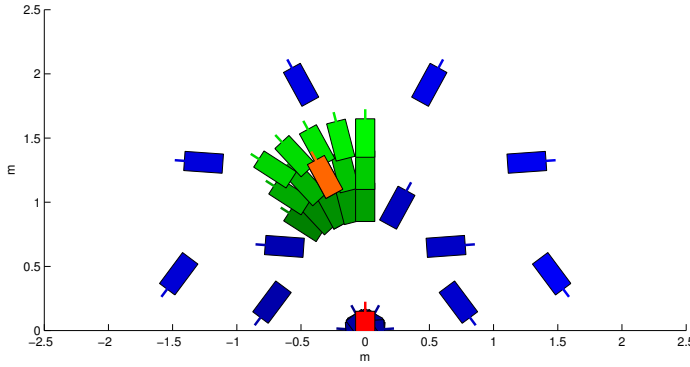
$$v_{max} = v_{step} \cdot v_{interval} + v_t \tag{4.21}$$

Figure 4.26 shows an example of the execution of the two step dynamic window in velocity space (4.26(a)) and cartesian space (4.26(b)). Red rectangle represents the robot, that is at the coordinates origin, oriented to the north, and moving with linear and angular velocity (represented as an orange square in velocity space). Orange rectangle represents where the robot will be if it keeps its velocities after five seconds. Blue rectangle in velocity space represents the *best window* discretised in squares. If the centre velocity of each square is applied to the robot, the robot will be in five seconds in the blue rectangles in cartesian space. Green rectangle in velocity space represents *local window*, which is discretised in smaller squares than the *best window*. If each velocity pair is applied, the robot will be at green rectangles (in cartesian space) in five seconds. It can be appreciated that the discretisation of the "local window" is in smaller steps than the "best window".

Once the windows are created, each pair of linear and angular velocities (a curvature arc) is evaluated. To evaluate this curvature arc positions along the curvature arc are calculated until the maximum time limit that is set to evaluate. In order to simplify, instead of having a fixed step between the positions, the step is adapted in a way that each cell that the curvature arc traverses in the local grid map has relation with only one position, and the positions are connected to adjacent cells, avoiding "gaps" between positions that can cause errors in the evaluation of the curvature arc.



(a) Velocity Space



(b) Cartesian Space

Figure 4.26: Dynamic Window Example

Each curvature arc is evaluated by a cost function that needs to be maximized. The function has several terms that prizes different behaviour of the algorithm:

- **Collision**: The first check of each curvature arc is whether the movement along the arc is possible or it leads the robot to an unavoidable collision. On each curvature arc, iterative position checking across the arc is performed.

  Three different approaches to perform the collision checking are evaluated:

  – To treat the robot like an infinitesimal point, enlarging the obstacles to avoid collisions. It is computationally low

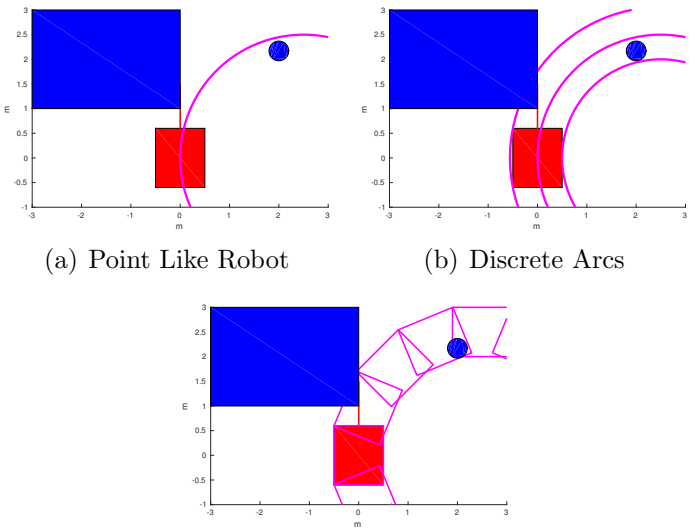
demanding and can work if the robot has a circular like footprint, but it has problems with other type of robots.

- Due the indoor mobile robots usually have nearly rectangular or circular shapes, another approach is checking the reference point of the robot (the curvature that a "point like" robot follows over its reference axis) and the maximum and minimum curvature arcs that the robot body follows. These curvature arcs need to be calculated depending on the morphology of the robot and its motion system. For example, the maximum and minimum radius of curvature of a circular differential drive are the central point adding or subtracting the radius of the robot.

  This approach is computationally more demanding than the infinitesimal point one, because more curvature arcs need to be calculated and evaluated. In addition, the approach can fail in detecting collisions if the robot is big, or the grid precision is high, because obstacles can appear between the evaluated arcs.

- To calculate all the occupy cells at each time. It can result in a high computer demanding algorithm, that can not run with the time requirements of a mobile robot. The computational cost can be reduced creating a discretisation of possible orientations of the robot, its occupied cells, and storing these configurations in a table or database. This approach works with all kind of footprints and robots sizes.

Figure 4.27 shows the three methods applied to the same situation. The robot (represented as a red rectangle) is a differential drive one, with a square footprint, and it is moving in a curvature arc to its right and pointing to the north. In the Figures, obstacles are represented in solid blue, and magenta lines and polygons represent the collision checking using the three different methods explained before. In 4.27(a) the path is clear because the central point of the robot does not collide with any obstacles, but in the reality the robot will collide with both obstacles. In 4.27(b) central point, maximum and minimum cur-



(a) Point Like Robot (b) Discrete Arcs

(c) Occupy Positions

Figure 4.27: Collision Checking

vature radius are evaluated, and the big obstacle is detected, but the second one is not detected because it is between the curvature radius evaluated. In 4.27(c) the footprint of the robot is evaluated in discrete positions along the curvature arc, and both obstacles are detected. The approach selected will depend on the local mapping method used (enlarging obstacles or not), the morphology of the robot and the computational power of the computer that runs the algorithm.

Regardless of the approach used, the checking is performed evaluating positions from the robot till one of two possible limits: the position is outside the local window or it is above the time limit to evaluate. On each checking, if at least one position (one of the arcs evaluated, or one of the cells occupied by the footprint) is in a occupied cell, the evaluation is stopped and the time and position of the collision are stored. A collision can happen when one of these situations occurs:

– The cell is occupied by an static obstacle (occupancy value

above the dangerous cell limit).

- The cell is occupied by a moving obstacle (occupancy value above the dangerous cell limit) and the time where the cell is occupied and where the position is evaluated are similar.

With that check the algorithm knows that if the robot follows that curvature arc, one collision state will happen, but that is not enough to discard this trajectory, because the robot can vary its trajectory on each iteration. Discard all the trajectories that lead to an obstacle can make the algorithm unable to move in crowded environments. So, to discard a curvature arc, one of this two situations must also happen:

- The collision will happen in less than a "security time".
- The distance that the robot can travel along the curvature arc is less than a "security distance".

If at least one of these situations happen, it is considered that following that trajectory will lead to a very dangerous scenario or even an unavoidable collision. So this trajectory will be discarded. All the trajectories that do not lead to a collision are evaluated.

- **Speed term**: This term prizes moving at high speeds, in a way that the global time of reaching the goal can be decreased. There are two possible scenarios to evaluate: the goal is in front or behind the robot. If the goal is in front of the robot equation 4.22 calculates the cost term (*speed*) as a division between current linear velocity v_t and maximum linear velocity v_{MAX}, in a way that the value is normalized between 0 and 1 and increases as linear speed increases. The surface of this term is constantly increasing with the linear velocity regardless of the angular one.

$$speed(v_t) = \frac{v_t}{v_{MAX}} \tag{4.22}$$

If the goal is behind the robot, prizing highest linear velocities can result in the performing of long curves to turn the robot towards the goal, pushing it away from the goal and consuming a lot of time and energy. For this reason, if the goal is behind the robot, the speed term formula is changed to Equation 4.23. This equation has two different terms, linear and angular velocities. The optimal behaviour for a differential robot if the goal is behind it could be to turn to the goal without translational movement. For this reason lower linear speed (v_t) is prized (divided by v_{MAX} to normalize it) and higher angular velocity (ω_t) is prized (divided by maximum angular velocity ω_{MAX} to normalize it). The surface of this term is symmetrical with respect to the angular velocity axis, and has two maximums, at $v = 0$,$\omega = \omega_{MAX}$ and $v = 0$,$\omega = -\omega_{MAX}$. The surface decreases until its minimum at $v = v_{MAX}$,$\omega = 0$.

$$\begin{aligned} speed(v_t, \omega_t) =& (1 - \frac{v_t}{v_{MAX}}) \cdot 0.5 \\ &+ (\frac{|\omega_t|}{\omega_{MAX}}) \cdot 0.5 \end{aligned} \tag{4.23}$$

- **Distance to goal term**: this parameter prizes the curvature arcs that move the robot to the goal. In order to evaluate this distance, all the positions along the curvature arcs (until a fixed time) are evaluated, and the minimum euclidean distance to the goal is saved. Once the minimum distance to the goal is selected, Equation 4.24 is used to calculate this term, where d_{goal} is the distance to the goal and it is normalized to a fixed maximum distance d_{MAX}. Beyond this distance all the curvature arcs are equally bad and need to be selected based on the other terms of the cost function. Also, the cost function is normalized between 0 and 1, if the distance to the goal is greater than d_{MAX} the value of the term is zero. If the goal is between the local window and reachable by a curvature arc, the arc that crosses the goal has the maximum value of the surface, decreasing its value symmetrically to the curvature arc radius

(v_t/ω_t). Also, all the curvature arcs with the same radius have the same value on this term.

$$dgoal(v_t, \omega_t) = 1 - \frac{d_{goal}(v_t, \omega_t)}{d_{MAX}} \tag{4.24}$$

- **Orientation to goal term**: this parameter prizes the curvature arcs that head the robot towards the goal, with independence of the distance to the goal, because turns to head the goal tends to approach the robot to it. The cost of this parameter is calculated using the equation 4.25, where x_{goal} and y_{goal} are the goal cartesian positions. The evaluated position ($x_{t'}$, $y_{t'}$ and $\theta_{t'}$) is a position along the curvature arc in a time t'. In order to obtain the same cost for all the curvature arcs of the same curvature this time t' is calculated using the equation 4.26 where t_{ev} is a fixed time value and the value of t' increases as the actual linear velocity decreases. The surface has a maximum along the line of same curvature and decreases symmetrically (among curvature lines) as the orientation to the goal increases.

$$ogoal(v_t, \omega_t) = 1 - \frac{|atan(\frac{y_{goal} - y_{t'}}{x_{goal} - x_{t'}})|}{\pi} - \theta_{t'} \tag{4.25}$$

$$t' = t_{ev} * \frac{v_{MAX}}{v_t} \tag{4.26}$$

- **Distance to obstacles term**: this parameter prizes that the robot travel far from the obstacles. When the local map is created, each cell can have three different types of occupancy:
  - Under the non critical obstacle limit: cell is secure and traversable.
  - Between the non critical obstacle and the critical obstacle limits: cell is traversable, but potentially dangerous because it is close to an obstacle.

– Above critical obstacle limit: cell is dangerous because there is an obstacle and therefore it is not traversable.

Each curvature arc is evaluated within the local window, saving the highest occupancy value of the cells that the path crosses until a time limit. Therefore equation 4.27 calculates the value of the term, where d_{ob} is the maximum occupancy value of the cells traversed by the curvature arc, $limit_{nc}$ is the non critical obstacle limit and $limit_c$ is the critical obstacle limit. The equation is normalized between zero and one using the maximum value of traversable cells. If the curvature arc traverses a dangerous cell (more than critical obstacle limit occupancy value) the equation is normalized to zero. Using this equation all curvature arcs that not cross any occupied cell have the maximum value and if the curvature arcs crosses an occupy cell have a value of zero. If the curvature arc crosses cells close to the obstacles, the value decreases as the cell is closer to the obstacle.

$$dobs(v_t, \omega_t) = 1 - \frac{d_{ob}(v_t, \omega_t) - limit_{nc}}{limit_c - limit_{nc}} \tag{4.27}$$

- **Distance to the best arc term**: this parameter prizes curvature arcs that are near the best curvature arc, even if it is outside the local window (velocities not reachable by the robot in the next iteration). Equation 4.28 obtains the value of this term, where $curv(v_t, \omega t)$ is the curvature of the evaluated arc, $curv(v_{best}, \omega_{best})$ is the curvature of the best arc, and $curv_{MAX}(v_{min}, \omega_{MAX})$ is the maximum curvature the robot can follow. If the evaluated curvature arc has a value of infinitum ($v_t = 0$) equation can not be calculated and the value of the term is set to minimum.

  There is a particular scenario where the best curvature arc has a value of infinitum (the robot turns without translational movement). In this case the equation used is 4.29 where the robot linear velocities that are near zero are prized, and if the robot is turning in the same direction that the best curvature

arc it is prized too. The surface generated by this scenario is continuously decreasing with the linear velocity, and it has one planar surface of 0.25 added in the angular velocities of the same sign. There are multiple maximums of the surface in $v_t = 0$,$sign(\omega_t) = sign(\omega_{best})$ and multiple minimums in $v_t = v_{MAX}$,$sign(\omega_t) \neq sign(\omega_{best})$.

$$dbest(v_t, \omega t) = 1 - \frac{|curv(v_t, \omega t) - curv(v_{best}, \omega_{best})|}{2 \cdot curv_{MAX}(v_{min}, \omega_{MAX})} \tag{4.28}$$

$$\begin{aligned} dbest(v_t, \omega_t) = 0.75 \cdot (1 - (\frac{|v_t - v_{best}|}{v_{MAX}})) \\ +0.25 \cdot (sign(\omega_t) == sign(\omega_{best})) \end{aligned} \tag{4.29}$$

- **Oscillation term:** this parameter prizes the stability of the movement, increasing the possibility of selecting an arc of the same curvature than the current travelled arc. The equation is similar to the "distance to the best arc" term, using the last velocities values instead of the best curvature arc values (equation 4.30 and 4.31).

$$osc(v_t, \omega t) = 1 - \frac{|curv(v_t, \omega t) - curv(v_{t-1}, \omega_{t-1})|}{2 \cdot curv_{MAX}(v_{min}, \omega_{MAX})} \tag{4.30}$$

$$\begin{aligned} osc(v_t, \omega_t) = 0.75 \cdot (1 - (\frac{|v_t - v_{t-1}|}{v_{MAX}})) \\ +0.25 \cdot (sign(\omega_t) == sign(\omega_{t-1})) \end{aligned} \tag{4.31}$$

- **Distance to a moving obstacle:** this term increases the security of the trajectories prizing the trajectories that do not cross any path of a moving obstacle, even if this trajectory does not lead to a collision. This term is added to the equation to deal

with the uncertainties of the movement of an arbitrary object in the environment. If the algorithm only checks collisions with moving obstacles the robot can approach the obstacles and cut its path, and an error in the detection or a variation in the obstacle velocity can lead to an unavoidable collision. The value of this term is calculated using the equation 4.32 where dt_{int} is the minimum difference time along the curvature arc where an obstacle path and the curvature arc intersects, dt_c is the critic difference time value (used in the evaluation of the collision with a moving obstacle) and dt_{nc} is the non critic difference time value, used to normalize the equation and also to avoid discarding arcs that crosses the path of obstacles but with a high time difference between them.

$$movobs(v_t, \omega t) = \frac{dt_{int} - dt_c}{(dt_{nc} - dt_c)} \tag{4.32}$$

- **Curvature arc crosses the goal:** if the curvature arc crosses the goal, the weight of the curvature arc is increased by a constant value in order to select one of the curvature arc that leads to the goal, even if the total weight is less than other curvature arcs (for example, it can happen if the goal is close to an obstacle). To check that, the goal is enlarged to a circle, with the goal tolerance radius value and is checked if at least one cell of the arc is inside the goal. In addition, some terms of the equation are varied:

  - The distance to the goal is set to zero ($W_{dgoal} = 1$) to avoid the discretisation errors.
  - If the position where the orientation to the goal is calculated is behind the goal, this term is changed to the maximum value. If the position where the orientation to the goal is calculated is before the goal cell, its value is kept. In this way, the algorithm avoids obtaining low cost values on this term when the robot is close to the goal.

– The distance to a static obstacle or moving obstacle is only checked if the cell is before the goal. If the obstacle is behind the cell that crosses the goal, it is not computed. Varying these terms, the algorithm avoids lowering the value of curvature arcs that leads to the goal if there are obstacles behind it.

The cost value of each curvature arc is calculated by equation 4.33, where each term is multiplied by a weight α. If the curvature arc leads to a collision the value of the equation is set to zero and if the goal is traversed the cost function is increased by a multiplier value (α_{goal}).

$$\begin{aligned} f(v_t, \omega_t) = & \alpha_{speed} \cdot speed(v_t, \omega_t) + \alpha_{dgoal} \cdot dgoal(v_t, \omega_t) \\ & + \alpha_{ogoal} \cdot ogoal(v_t) + \alpha_{dobs} \cdot dobs(v_t) \\ & + \alpha_{dbest} \cdot dbest(v_t) + \alpha_{osc} \cdot osc(v_t) \\ & + \alpha_{wmovobs} \cdot movobs(v_t) \end{aligned} \tag{4.33}$$

Each term of the equation is normalized between zero and one. The sum of the weights is also normalized between zero and one. The maximum value of the equation, if the goal is not crossed, is the unity, and if the goal is crossed, is the goal multiplier term α_{goal}.

This set of parameters need to be tuned, depending on the scenario, the robot, and the behaviour that is intended. If α_{speed} is the predominant weight, the algorithm prizes curvature arcs with high linear speeds (or angular speeds if the goal is behind the robot). If α_{dgoal} is the predominant weight the algorithm will prize getting close to the goal. If α_{ogoal} is the predominant weight, the algorithm will prize moves towards the goal. If α_{dobs} is the predominant weight, the algorithm will move away the enlarged obstacles, even when possible paths to the goal will be discarded. If α_{dbest} is the predominant weight, the algorithm will tend to follow the "best window" arc even if the arc is not good in the rest of terms evaluated. Setting α_{osc} to higher values increases the stability of the robot but can cause that the robot does not turn at all once a velocity is selected. The higher $\alpha_{wmovobs}$ is set, less chance of selecting paths that crosses dynamic

obstacles will be. If α_{goal} is set to high values, the robot will go aggressively to the goal when it is reachable in the robot nearby.

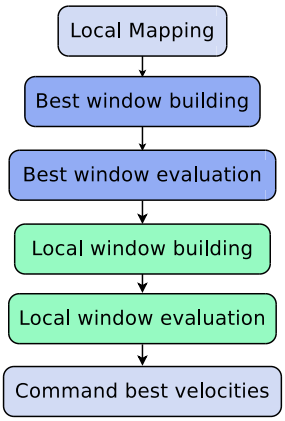


Figure 4.28: DW4DO: Flowchart

Figure 4.28 summarises the algorithm workflow on each iteration. First the local map is built, then the two velocities window are evaluated: first the "best window" and then the "local window". Finally, the selected pair of velocities are commanded to the robot.

4.3.2 Dynamic Window for Dynamic Obstacles Tree

DW4DO is able to guide the robot towards the goal avoiding dynamic obstacles, at the same time that keeps it away from static obstacles, but the path followed can be improved if information about future paths is available.

In this proposed extension DW4DOT, instead of considering the velocity of the robot constant, a DW4DO algorithm is executed in several future times. In this way, the algorithm is able to plan more than one unique velocity to command the robot. However, due to the initial position of each iteration varies, and the local map does not, the information of the subsequent iterations are not reliable enough and must not be commanded to the robot directly in an open loop mode.

Using the previous method, instead of searching the next commanding curvature arcs, the proposed algorithm will select the best commanding arc, but taking into account the next ones in the search, in a way that can select a curvature arc that is not the best in the current situation but leads to better path in the future.

In order to do that, a tree of a predefined depth is built following the algorithm:

- DW4DO is evaluated at each time for each branch and level of the tree. On each iteration the initial position and velocities used to build the dynamic window vary: if the depth is 1 (initial branch) they are the current position and velocities of the robot, if the depth is more than one the initial position and velocities of this arc are the final position and the velocities of its parent arc (that has been evaluated for a fixed time).

- After the evaluation of the DW4DO algorithm for each level, a set of the evaluated arcs are selected to create the next level to be evaluated.

After the tree creation, each path is evaluated using a backward algorithm and the cost of each path is the sum of each individual arc traversed cost.

In order to build the arcs tree, different strategies are proposed:

- **Depth First technique**: It creates a tree of single width, selecting the best curvature arc each time. This can be used for commanding future positions to the robot, or even extending the algorithm to a global planner, but it is not useful for our approach, due to the selected curvature arc will be the same as only evaluates the first branch of the tree, and the subsequent evaluation only consumes computational resources.

- **Breadth First technique**: It extends the tree and keeps all the possible curvature arcs evaluated, selecting the best path after the tree is built using a backtracking algorithm. This technique assures that the algorithm find the best path, nevertheless, the computational load increases a lot and can not

satisfy the time requirements for a local obstacle avoidance algorithm.

- **Set of Best Arcs**: It builds the tree from a set of the best arcs on each branch. This technique reduces the complexity of the tree and it works as a "guided search". However, selecting the best curvature arcs when the accelerations of the robot are low, usually yields in very similar paths, which is not a significant improvement with respect to DW4DO algorithm, and the computational requirements increase.

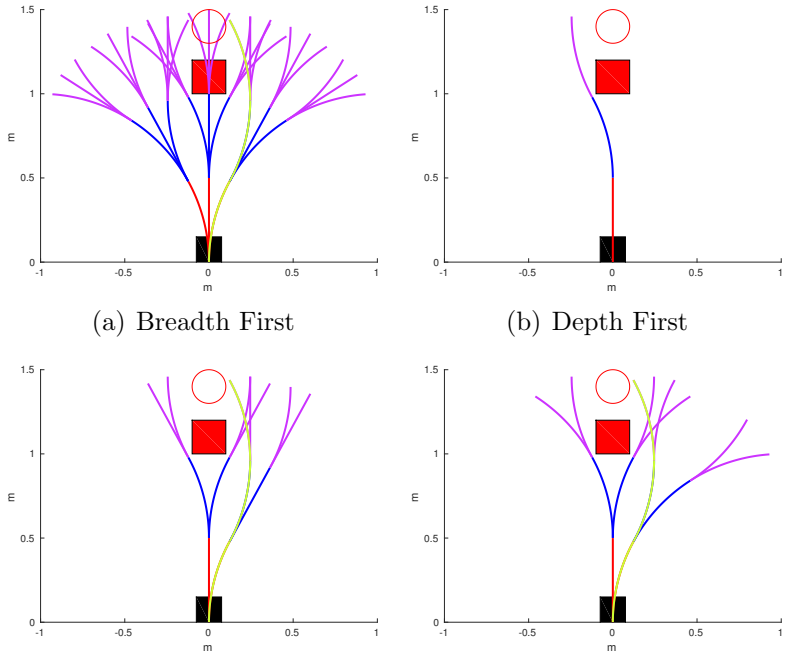
- **Set of Weighted Best Arcs**: It builds the tree from a set of the best arcs on each branch. However, in order to increase the difference between the paths and to increase the chance to find the best solution, even when the first arcs of the tree are not the best, the arcs to create the next branch are selected using this method:

  1. The best cost of each node is always kept.
  2. The second arc is selected as one which cost is, at least, $1/k$ times the best interval and its curvature radius is at least c away from the best arc.
  3. If there are none curvature arcs that satisfies that restrictions, the search is repeated reducing the distance iteratively until a second curvature arc is selected.

  In this way, the complexity is reduced with respect to the Breadth First Technique. In addition, the chance of finding the optimal path, when the accelerations of the robot are low, increases with respect to the set of best arcs technique.

Figure 4.29 shows an example of the tree building using the techniques explained before. The maximum level of the tree is set to three (branches of each level are shown in red, blue and purple). From each position a maximum number of three child are calculated. The selected path on each scenario is represented in light green colour. The red box is an obstacle and the red circle is the goal to reach. Breadth first technique (figure 4.29(a)) is able to obtain the best path, but all

the paths need to be evaluated (in the example, 27 possible paths). Depth First technique (figure 4.29(b)) only evaluates one path but the result is not optimal (the best arc in the first branch does not produce the best whole path). Set of best arcs technique (figure 4.29(c)) is able to find the best path to the goal (it is obtained from the second best arc in the first evaluation) with less paths evaluated than the breadth first technique (in the example, 8 paths instead of 27). Set of weighted best arcs technique (figure 4.29(d)) is also able to reach the optimal path, and the difference of all the generated paths is bigger than the set of best arcs technique, which can find the best path in more situations when it is not found within the first best arcs.



(a) Breadth First

(b) Depth First

(c) Set of Best Arcs

(d) Set of Weighted Best Arcs

Figure 4.29: DW4DOT: Tree building techniques

The proposed method allows the algorithm to command velocities to the robot that can reach better positions in the future steps, even when the current arc followed is not the best one. Keeping intervals, with a curvature difference from the best arc, increases the differ-

ence of the paths evaluated, resulting in more possibilities of finding different and better paths than the straightforward best one.

4.3.3 Conclusions

In this section, a proposal to avoid dynamic obstacles based on the DWA has been explained (DW4DO). This algorithm is able to work in real environments improving some flaws of the CVM based algorithms:

- It improves the security of the avoidance manoeuvre working with a two limits occupancy grid. In that way, the obstacles do not need to be enlarged more to keep the robot away from it. The robot will travel away from the obstacles but can cross the second level occupancy cells if it is necessary in order to reach the goal.
- The algorithm prizes the stability of the path planned, taking into account the previous velocities of the robot.
- The computational time required by the algorithm is more stable, as the same number of arcs are evaluated on each iteration.

Also, an issue of the obstacle algorithms that takes dynamics into account has been detected: paths that are not reachable in the next iteration are not evaluated. This could be a big issue if the algorithm works at high frequencies or the robot has low acceleration limits. For this reason, a two level dynamic window is proposed to calculate better paths to the goal.

The algorithm has proven to work in real scenarios, improving the performance of the CVM based algorithms in most of the tests (shown in results section). This algorithm can be extended to holonomic robots that can move generating other movement primitives (not only curvature arcs) changing some terms in the cost function, which is an extension that can not be done in CVM based algorithms.

DW4DOT extends this algorithm to a planner beside the local window of the robot. It calculates possible paths further in time using a proposed tree building method in order to reduce the number

of paths evaluated. This algorithm has proven to work, however the computational requirements increase, making it not suitable for some real applications where the computers has not enough computational power or the time requirements can not been satisfied.

4.4 Dynamic Lattice Planner

The previously proposed obstacle avoidance methods are able to avoid dynamic obstacles and they can work in complex environment. But their local goals are provided by global planners that do not take into account the dynamics of the environment.

In this section a global planner that can produce routes for the robot, taking into account the dynamic of the environment (or even more semantic information and restrictions) is proposed: Dynamic Lattice Planner (DLP).

The idea of this algorithm is to use a State Lattice Planner, that generates future positions state of the robot. The validity of this positions can be evaluated in a similar way than DW4DO collision checking works. Also, the next positions reachable from each position are calculated based on the Dynamic Window Approach. This kind of planner has the time information implicit on each evaluated position, in a way that dynamic obstacles can be taken into account. In addition, the route planned can be directly followed by the robot with a local control stage.

However, planning in this state dimension could be very complex and it does not guarantee the time constraints of our application. For this reason, firstly a 2D A* path search is performed and each goal obtained is provided to the DLP as local goals in order to reduce the search space.

- **A* path searching**: Searching a path to the goal in a grid map can be an unbound problem. For that, heuristic algorithms like A* are proposed. In DLP a guided search algorithm A* in 2D cartesian space is executed to obtain a path that is used as a guideline for the Lattice State Planner.

  The input of the algorithm is an occupancy grid map of a fixed

squared grid size, which each cell can have a value between -1 and 255. The cell has the value -1 if it is unknown, 0 if it is totally free of obstacles, 255 when it is sure than a obstacle occupies the cell, and a number that decreases between 255 and 0 as the cell is further the obstacle.

Each node evaluated has a cost to the goal and the path can be recovered as a gradient that begins in the goal and travels to the origin. The problem is that calculating the cost of each node can be computationally expensive.

The solution proposed by the A* algorithm is to calculate the cost of the nodes that have influence on the path, leaving as unknown the other nodes. Prior to the calculation of the whole path it is impossible to know what cells have influence or not in the path, so this algorithm always calculates the cost of the best candidate.

The A* algorithm orders the nodes in a tree. Each node, in a 2D map, is inside a cell of the map, and it is connected to 4 or 8 nodes/cells (depending if we consider moving diagonally a valid motion or not) and it is reached from another cell (parent node). So, the algorithm works as follows:

- Evaluate the cells connected with the first candidate (starting node).
- Move the evaluated node to a evaluated list.
- Select minimum cost node from the not evaluated list.
- Evaluate nodes connected with the candidate and move this candidate to the evaluated cells list.
- Repeat the step until the goal cell is reached.
- Recover the path from the evaluated nodes list, from the last node (goal cell) to the beginning.

The cost of each cell is calculated using the following cost function (equation 4.34):

$$cost_{node} = dist_{goal} + dist_{travel} \tag{4.34}$$

- If the cell where the node is evaluated is occupied by an obstacle it is considered as not reachable (its cost is infinite) and is not put inside the candidate list.
- The euclidean distance to the goal ($dist_{goal}$).
- The distance travelled in order to reach the cell ($dist_{travel}$).

Using this cost function, the cost to the goal will be kept constant between cells if the goal is reachable in a straight line (the distance travelled increases at the same time that the distance to the goal decreases). If the goal is not reachable in a straight line, the cost of the path increases, and the total length of the path is taken into account (distance travelled term) in order to select the best candidates. Figure 4.30 shows an example where the goal is in front of the robot blocked by an obstacle. The evaluated cells are shown as boxes, and their cost to the goal goes from red (low cost) to purple (high cost). It can be appreciated how the cells that directs the robot to the goal in a straight line has the same values (red cells in the left side of the image). However, in order to reach the goal, cells with more cost to traverse need to be evaluated to surround the obstacle.

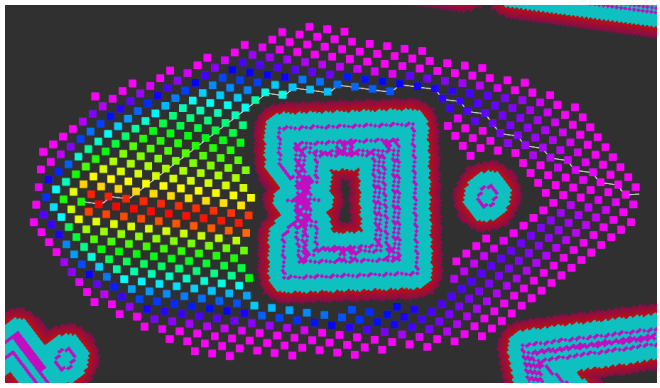


Figure 4.30: A* Path Planning Example

A problem that this algorithm has is how to avoid the infinite loops as a cell can be reached following an unbounded number of different paths. In order to avoid the infinite loops, if there

are two nodes located in the same cartesian cell, the algorithm only keeps the one with the best cost. If the cost is equal, the less distance travelled is the one that is kept. Using this algorithm, if a cell is reached following a longest path than one that has already been evaluated, this path is discarded.

Using this algorithm a path to the goal is obtained, but can not be guaranteed that is the best possible path as all the possible path combinations has not been evaluated. In addition, the time employed in obtaining the path can not be known prior to the calculation, it could be between the time consumption in evaluate the cells that connect the goal to the origin in a straight line (best case) till the evaluation of all the grid possible paths (worst case).

One of the problems of this approach is related to the map resolution. Ideally, the higher possible resolution of a map is desired, but the time consumption of the algorithm is directly related to it. As the A* search is used as an input for DLP it could be not affordable. On the other hand, if the resolution is too low, possible paths to the goal can be blocked.

Our proposal is to use a multi-resolution map according to this technique: the algorithm plans in the low resolution map available, but when a cell is occupied, it searches in the next resolution map iteratively until the cell is free of obstacles or the plan is done in the higher resolution. Using this technique, there are two important improvements:

- If the environment is free of obstacles, the A* search time decreases (as the cells are squared, and each resolution is divided by two, there are four times less cells to evaluate on each incrementally lower resolution).
- As the path planning is performed in the higher resolution maps when a zone is full of obstacles, narrower paths to the goal can be evaluated. Using fixed low resolutions this paths can not been achieved.

In order to build the multi resolution map, several layers need

to be calculated. The first layer is the higher resolution one, and each layer above it has 4 time less cells than the previous one, due to each cell of this layer has the size of 4 cells of the lower layer. The occupation of each cell is the mean value of the corresponding cells in the lower layer.

Figure 4.31 shows an example of a three level resolution map. In the figure the occupancy value goes from darker colours (high occupancy) to lighter ones (less occupancy). White cells are free of obstacles. It can be seen how there are more possible paths to plan in the higher resolution maps (more cells free of obstacles), because each cell of the lower resolutions has the mean value of four cells in the higher one.

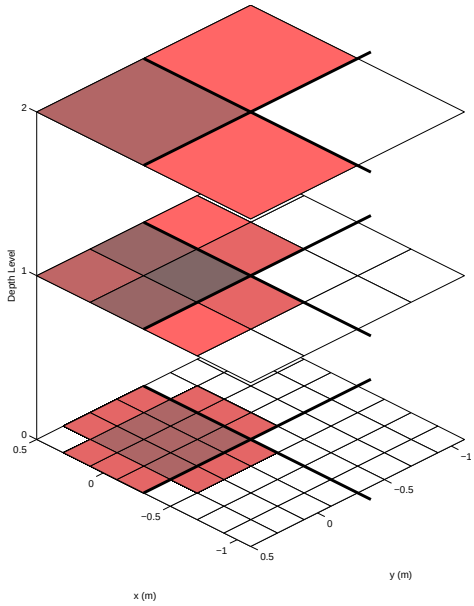


Figure 4.31: Multilevel Grid Map

The A* search algorithm is adapted in order to change dynamically between lower and higher resolution maps. The proposed method works as follows:

– Algorithm begins planning in the maximum resolution to obtain the accurate initial position of the robot, evaluates

the cells around it in the same way than the explained A* does.

- If a evaluated cell is free, check the upper level (lower resolution) corresponding cell. If the cell is free continue searching the lower resolution until a cell is occupied or the minimum resolution map is reached. The candidate cell will be the lower resolution free cell evaluated. Using this method the algorithm is able to plan in lower resolutions when the area is free of obstacles.
- If there is a cell blocked by an obstacle around the current cell, the algorithm evaluates the cells that correspond to it in the lower layer (higher resolution). If there is a free cell in this level, select this as a candidate child. If not, continue evaluating the lower layers (higher resolutions) till one free cell is detected or the maximum resolution is reached. Using this method the algorithm is able to plan paths that reach the goal in narrow environments.

With these improvements, the cells that are far from the obstacles are evaluated faster, and the narrow paths are not blocked. In the worst case scenario (a environment really full of obstacles) this implementation works as the A* planner in the maximum resolution and in the best case scenario (an environment where there is not obstacles between the goal and the origin) it works several times faster than the previous one.

Another issue of discussion is how to select the number of layers, because the computational cost increases as the number of layers increases. Planning in the multi-resolution map could be very computationally expensive if there are many layers, the map is big and crowded of obstacles or the higher resolution is really small. Our algorithm searches in the map building which is the lower and higher gap in the map (in vertical and horizontal coordinates) and sets the maximum and minimum levels of the map to this values.

- **DLP Route Search**: A* was able to calculate a path in a fast way, but this path could not be directly followed by the

robot, unless it was an holonomic one, due to its kinematic constraints. Also, the time is not implied in the path calculus. Non-holonomic robots has to plan their movements in more state dimensions than the cartesian ones in order to be directly commanding to the robot: x-y coordinates, heading of the robot, linear and angular velocities of the robot and, even, time. If the result of the planning algorithm produces directly commanding inputs to the robot, it is called *route* instead of path.

Planning in this dimensional space can be very computational and time demanding. For this reason the space is discretised and a lattice state planner is used which fit to all the possible motions of non holonomic robots.

To build a tree of feasible commands of the robot, there are some parameters that are need to be set, in a very similar way than the DW4DO works:

- Time step where the velocities are considered constant.
- Velocity and acceleration limits, in order to calculate reachable commanding velocities.
- Linear and angular velocities steps. How many velocities are evaluated each step.

The state lattice planner tree is built and evaluated using an A* search algorithm. Each evaluated state in the tree has a cost, and can be already evaluated or not. On each iteration, the state with the best cost value, from the list that has not been evaluated yet, is selected and its connected states are created in the tree. Once the final state is reached, the path can be obtained using a backtracking algorithm.

The cost function (equation 4.35) of each position has four terms:

$$cost_{state} = dist_{goal} + dist_{travel} + or + v \tag{4.35}$$

- Euclidean distance to the goal ($dist_{goal}$). This term reduces as the path is close to the goal.
- Cost to reach the state ($dist_{travel}$). The sum of distance travelled to reach this state. This term increments with the path length.
- Orientation to the goal (or). This term reduces the cost of the state when the robot is oriented towards the goal, which usually produces commands that move the robot close to the goal.
- Linear velocity of the state (v). This term increases the cost of the state as lower velocities are produced in a way that the algorithm tends to calculate faster paths to the goal.

A collision checking (using the discretised footprint of the robot as was explained in the DW4DO section) is performed on each arc that joint two consecutive states. If there is a collision in the arc followed, the resulting state is discarded.

Using the proposed cost function, planning can be done using only three dimensions: the cartesian states and the time steps. The infinite loops are avoided keeping only the best cost state on each cube. Introducing the orientation to the goal term in the cost function avoids the discretisation of the map into angular positions, because if two states with the same euclidean distance to the goal and cost to reach the state are in the same cube, the algorithm will keep the one with the best orientation to the goal.

DLP is able to plan a route that is directly commanding to the robot. However, the creation of the connected state to each one is computationally more complex than the cells evaluation in the A* path search. Planning an State Lattice Planner when the goal is far from the robot results in the evaluation of a high number of states which could not satisfy time constraints. For this reason, the euclidean distance is calculated to each A* path position, using this path as a guide for the DLP route.

To avoid dynamic obstacles the time has to be also taken into account. In order to do that, the range of our proposed local mapping algorithm is extended to the whole environment, obtaining the static and dynamic obstacles from a global information system instead of directly from the sensors. Using this map, collision checking can be made taking into account dynamic obstacles and the route planner is able to avoid them.

On the other hand, using this mapping technique, increases the search state as the time dimension is included, but allows the robot to plan movements when it should turn without linear displacement or even being still. This derives in a big improvement with respect to local navigation scenarios because it allows to solve a very singular scenario: the best route to the goal is to keep the robot stopped during a time until an obstacle unblocks the route to the goal.

As the map is built in the same way than the local navigation algorithms do, it could be dynamically improved with the measurement sensors of the robot, and even using the same map (with two different windows) to follow the route planned using a local obstacle avoidance algorithm.

The cell map proposed along with the route planning also allows including more information apart from the occupancy (in space and time) values: direction allowed in the cell (which it is useful for autonomous car), cost of the cell (if there are zones in the environment that is preferred not to be traversed by the robot), etc.

Figure 4.32 summarises the algorithm workflow. The pale orange boxes are stages related with the multi resolution map building process. Blue boxes represents the cartesian A* path planning which is used as an input to the DLP Route planning. Finally, a control algorithm (explained in Appendix A.3) generates valid commands for the robot in order to follow the route generated.

In this section, a global route planner which can take dynamic obstacles into consideration has been proposed. The algorithm is based on a Lattice State Planner and due to the cost function and the node

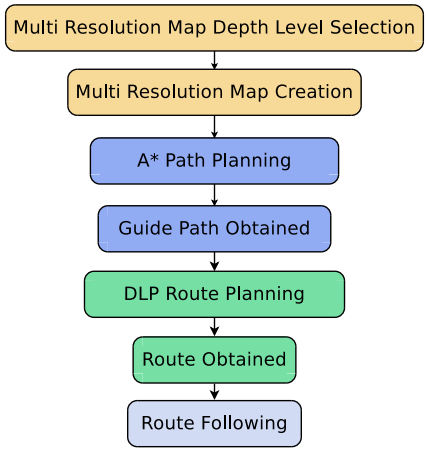


Figure 4.32: DLP: Flowchart

expansion technique proposed it is able to calculate possible routes reducing the complexity of the searching space. The performance of the algorithm is improved using a simpler A* Path Planning to generate a valid path that can be used as a guideline for the route.

The proposed local mapping technique has been extended to be used in global maps allowing the algorithm to avoid dynamic obstacles. Also, a technique to build multi resolution maps has been developed, improving the performance of the path search.

The DLP is able to solve some situations where obstacle avoidance algorithms can not solve, like waiting for an obstacle to move in order to reach the goal. Tests to prove the performance of DLP algorithm have been presented in the results section.

DLP can be extended to take semantic information into account, and it is suitable to be used in car like environments where route must take into account restrictions like lane directions, velocity limits or semaphores.

Chapter 5

Results

In this chapter experiments using the algorithms proposed in the development section will be performed. Firstly, experiments will be carried out in simulation (using the Gazebo 3D simulator) in order to test the performance of each algorithm without localisation and perception errors. Secondly, tests will be achieved under real conditions with a real platform, testing the navigation algorithms along localisation and perception algorithms. Finally, tests in the ABSYNTHE and RoboShop projects will be explained.

5.1 Simulation Results

In this section experiments will be performed using differential drive platforms under Gazebo simulation environment, in order to test the behaviour of each algorithm proposed in the chapter 4.

5.1.1 Parameters Sensitivity Studio

Before performing tests in challenging scenarios, the sensitivity of parameters for each obstacle avoidance algorithm is tested. Tests are carried out where the goal is near the robot in a way that a global planner is not needed.

The set up for all the algorithms is the same: each algorithm is executed in the simulated RoboShop differential drive robot, which

velocities are limited to 0.4 m/s (linear velocity) and 40º/s (angular velocity) for security in indoor scenarios. The range sensor used is a Hokuyo URG-04LX with 5 meters coverage and its field of view limited to 180º. This laser is mounted approximately at 0.5m from the floor. The local map has the same dimensions for all the simulations, which are 10mx10m with a cell size of 10 cm. Each obstacle has a critic enlargement radius of 0.6 m and an additional 0.2 m of non critic enlargement radius.

For considering that an execution has reached the goal, the robot should be closer than 30 cm to it.

Some parameters are evaluated for each test: travelled distance d, time expended t, mean linear velocity $\overline{v}$, mean absolute angular velocity $|\overline{\omega}|$ and velocities standard deviation σ_v^2 and σ_ω^2. Each parameter is affected by the environment (for example, in scenarios with curve walls it is more probable that the angular velocity increases) but allows to compare several configurations in the same scenario. These parameters give an idea about the efficiency of the algorithm (d), its speed ($\overline{v}$, and t) and its smoothness ($|\overline{\omega}|$, σ_v^2 and σ_ω^2). Images of the path performed by each experiment are also shown in figures, in a way that the behaviour of each algorithm can be explained.

The sensitivity of parameters is studied under static scenarios conditions. Only CVM and DW4DO algorithms are deeply studied in these experiments, because the other algorithms (PCVM, DCVM and DW4DOT) use the same cost function with some additions to avoid dynamic obstacles (PCVM,DCVM) or to plan further in time (DW4DOT).

The three scenarios are:

- First scenario: an empty environment with the goal at 6 metres in front of the robot and a box of 1m x 1m situated in the natural path of the robot. In this scenario the ability to avoid obstacles is tested.

- Second scenario: a goal at 6 metres in front of the robot blocked by a big obstacle formed by a box of 1m x 1m and a wall of 4 metres, and symmetrical to the axis that joins the robot and the goal. The goal is in the middle of a "corridor" formed by

two walls. In this scenario the ability to avoid big obstacles, and the ability to enter corridors are tested.

- Third scenario: an empty environment with four consecutive goals, one behind the robot and the other three at the left or right of the previous one, forcing the robot to turn in order to reach each one. In this scenario, the ability to follow different goals is tested.

The first algorithm to be tested is CVM. The cost function is affected by three weights (α_{dist},$\alpha_{heading}$ and α_{speed}) so seven configurations are tested: three configurations that set one of the weight to zero in order the check the effect of this parameter in the avoidance, three configurations that set one of the weights as the predominant weight of the function (it doubles the second one) and a configuration where all the weights are equally set.

- **First Scenario**: In this scenario we tested how setting to zero any of the parameters makes the algorithm unable to avoid obstacles in the path of the goal. Figure 5.1 shows the path performed by the robot in several colours. If $\alpha_{distance}$ is set to zero (plain blue line in figure), the algorithm tries to drive the robot regardless of the distance that can traverse and finally gets stuck close to an obstacle. If $\alpha_{heading}$ is set to zero the robot does not go towards the goal and falls into a local minima traversing in circles (red line in figure). If α_{speed} is set to zero it is equally good moving or not in a curvature sector, and the robot finally get stuck close to the obstacle (light green line in figure).

  In table 5.1 a summary of the working executions is shown. The best results are obtained setting α_{speed} as the predominant value, however the avoidance manoeuvres performed by the predominant α_{dist}, α_{speed} and equally set weights are very similar. In the case of $\alpha_{heading}$ being the predominant weight, the avoidance of the algorithm is harmed, because there are local minima between avoiding the obstacle and turning away from the goal resulting in the robot continues its movement until the obstacle is close. Finally the robot is able to avoid the obstacle but

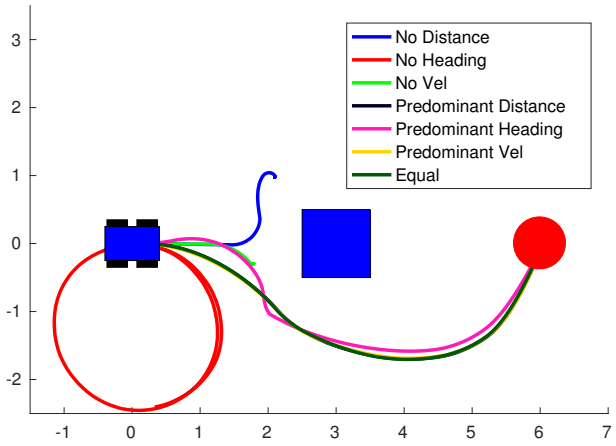


Figure 5.1: CVM Parameters Test: Scenario 1

spending most time and with more velocity changes (pink line in the figure).

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Predominant α_{dist}	7.059	19.15	0.37	0.064	8.72	9.33
Predominant $\alpha_{heading}$	7.141	24.85	0.30	0.146	10.39	13.45
Predominant α_{speed}	**7.037**	**18.91**	**0.38**	**0.050**	**8.76**	**9.25**
Equal α	7.038	19.05	0.37	0.062	**8.76**	9.35

Table 5.1: CVM Parameters Test: Scenario 1: Results

- **Second Scenario**: In this scenario, only the successful tests from the previous one are tested. Paths performed are shown in figure 5.2 and the data collected in table 5.2.

  Configurations that have more weight in the $\alpha_{heading}$ (predominant $\alpha_{heading}$, shown in pink line, and equal set weights, shown in grey) have problems with the obstacle avoidance, because the robot needs to get to an orientation really away from the goal in order to avoid it. One of the configurations gets stuck in a local minima and the other one is able to reach the goal but

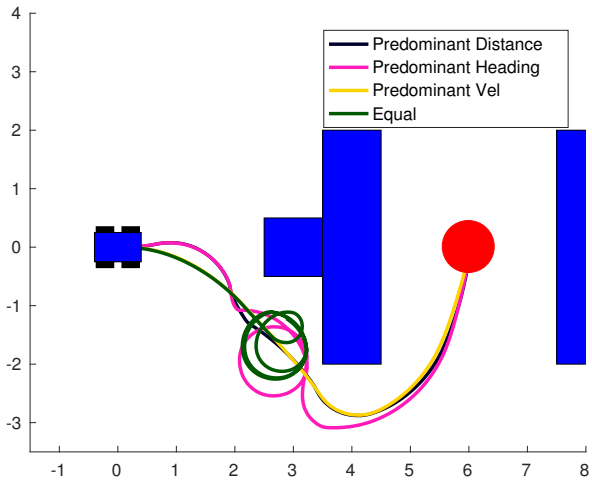


Figure 5.2: CVM Parameters Test: Scenario 2

performing a non desirable path. Predominant α_{dist} and predominant α_{speed} configurations are able to reach the goal, and predominant α_{speed} achieve the best result. However, predominant α_{dist} keep more away from obstacles performing a safer avoidance (although in this scenario the avoidance manoeuvre is very similar).

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Predominant α_{dist}	8.89	26.93	33.21	10.715	12.67	15.19
Predominant $\alpha_{heading}$	13.40	41.16	32.30	12.311	20.85	23.58
Predominant α_{speed}	**8.55**	**22.87**	**37.77**	**6.058**	**8.47**	**11.53**

Table 5.2: CVM Parameters Test: Scenario 2: Results

- **Third Scenario**: In this scenario, only the configurations that have one weight as the predominant one are tested (the other configurations have previously failed). Paths performed by the robot are shown in Figure 5.3 and the results in Table 5.3.

  In this scenario the three configurations are able to reach the goal and the manoeuvres performed are very similar, only with

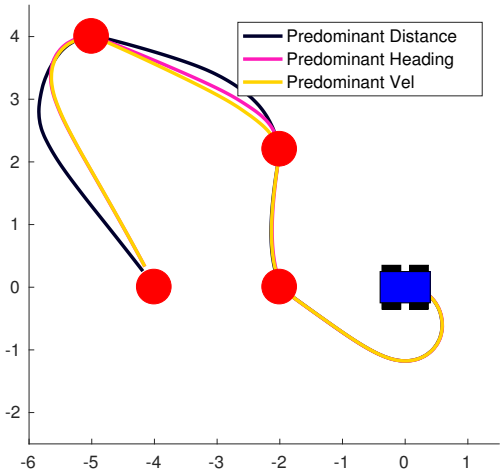


Figure 5.3: CVM Parameters Test: Scenario 3: Results

the predominant α_{dist} configuration making a slightly longer path, especially in the last turn.

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Predominant α_{dist}	14.79	40.33	36.80	8.020	12.87	**18.30**
Predominant $\alpha_{heading}$	14.46	39.66	**36.53**	8.346	**12.63**	19.08
Predominant α_{speed}	**14.34**	**39.22**	36.79	**7.839**	13.14	19.78

Table 5.3: CVM Parameters Test: Scenario 3: Results

With the information obtained from the performed tests, it is concluded that the configurations where α_{dist} or α_{speed} are predominant in the cost function are valid. For the next tests, the configuration where α_{dist} has a high value will be selected, because it is the valid one that produces more safer avoidance in the presence of obstacles, even when it is not the faster configuration.

Cost function of DW4DO algorithm has more terms involved, so twelve configurations are tested where each weight of the cost function (α_{speed}, α_{dgoal}, α_{dobs}, α_{dbest}, α_{osc} and α_{ogoal}) is set to zero and to the predominant value of the cost function, in order to test the influence of each parameter in the cost function.

Among with these configurations, a configuration proposed (proposed α) is tested, where the weights are the following adjusted to an expected behaviour: $\alpha_{wmovobs} = 0.5$, $\alpha_{dbest} = 0.25$, $\alpha_{dobs} = 0.25$, $\alpha_{ogoal} = 0.2$, $\alpha_{speed} = 0.15$, $\alpha_{dgoal} = 0.1$, $\alpha_{osc} = 0.1$. Using this configuration the robot tries to reach the "best window" arc at the same time that keeps away from obstacles. Then the algorithm selects an arc that is oriented to the goal, with high speed, and that approaches the goal. Finally the α_{osc} is set to a low value in order to aid the stability of the algorithm without avoiding varying the trajectory. All these configurations evaluate each possible arc travelled during ten seconds. In addition, a configuration where the paths are evaluated during five seconds is tested to determine how this time evaluation affects the algorithm.

- **First Scenario**: the paths performed by the robot in these tests are shown in Figure 5.4 and the results obtained in Table 5.4. In this scenario, almost all configurations are able to reach the goal. Failing configurations are:
  - No α_{speed}, shown as red line in the figure. This configuration fails when the robot approaches the goal, because all the possible velocities are equally good, and avoiding the obstacle moves the robot away from the path to the goal, reducing the rest of the parameters. Finally the robot is stuck close to the obstacle and it is unable to continue.
  - Predominant α_{osc} is shown as turquoise line in the figure. With this weight set the robot tends to keep moving in the same direction once is selected, and it fails trapped in circular movement.
  - Predominant α_{ogoal}. In this case, it is getting away from the line that joins the robot and the goal decreases the value of the cost function, getting the robot stuck close to the obstacle.

  All the other configurations perform a similar obstacle avoidance with the exception of two of them: predominant α_{speed} (shown in bright green) that gets too close to the obstacle,

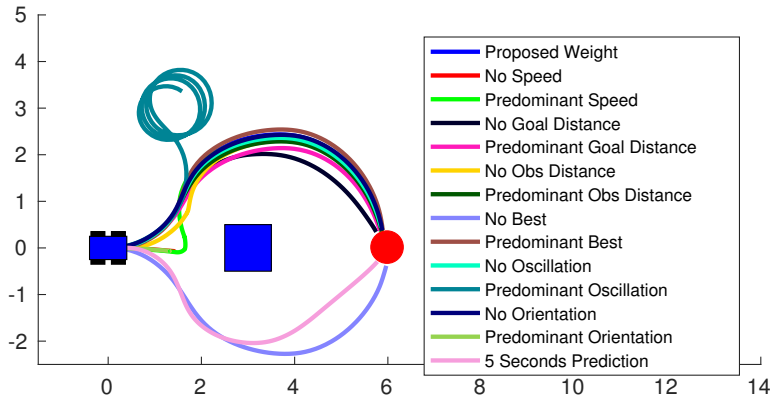


Figure 5.4: DW4DO Parameters Test: Scenario 1

forcing an escape manoeuvre and delaying to reach the goal, and prediction in 5 seconds that goes in a straighter path. The faster configuration has been the no α_{dgoal} one.

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Proposed α	7.98	21.73	0.36	0.086	0.17	0.17
Predominant α_{speed}	8.71	34.54	0.23	0.171	0.21	0.25
No α_{dgoal}	**7.21**	**19.60**	0.36	0.087	0.16	0.17
Predominant α_{dgoal}	7.49	20.49	0.36	0.089	0.16	**0.16**
No α_{dobs}	7.90	26.26	0.28	0.149	**0.15**	0.17
Predominant α_{dobs}	7.75	20.88	0.36	0.086	0.16	0.17
No α_{dbest}	7.77	21.08	0.36	0.090	0.16	0.16
Predominant α_{dbest}	8.24	22.59	0.35	0.095	0.17	0.18
No α_{osc}	7.82	21.23	0.36	0.087	0.17	0.17
No α_{ogoal}	7.99	21.43	0.36	0.083	0.17	0.17
Prediction 5 seconds	7.31	19.67	**0.37**	**0.063**	0.17	0.20

Table 5.4: DW4DO Parameters Test: Scenario 1: Results

- **Second scenario**: Paths performed by the robot are shown in Figure 5.5 and the data collected in Table 5.5. In this more complex scenario more algorithms failed. Predominant and no α_{dgoal} fails to avoid the big obstacle (shown in bright pink and black respectively). Predominant α_{dgoal} is unable to avoid the obstacle because all the possible paths gets the robot away from the goal, and no α_{dgoal} fails because it gets very close to the obstacle. In the same way, no setting α_{dobs} results in the robot getting stuck in the enlarged obstacle (yellow line). Setting the α_{dbest} as the predominant one can result in failure in the algorithm, because the robot will always try to reach the best arc even when it is impossible (brown line). No setting the α_{ogoal} also results in a failure: the robot begins avoiding the big obstacle but it is unable to turn again to the goal getting stuck in a local minima.

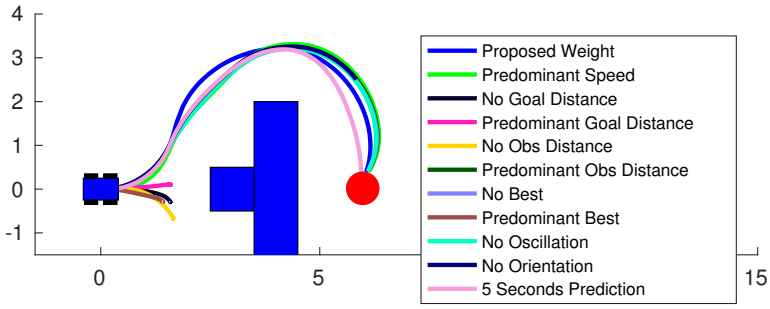


Figure 5.5: DW4DO Parameters Test: Scenario 2

The rest of the configurations are able to reach the goal. In a

similar way, only with the 5 seconds prediction the performance results in a straighter path (light pink line). Predominant α_{speed} and no α_{osc} result in straighter but more riskier paths. While the other ones perform a very similar manoeuvre.

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Proposed α	9.59	25.57	0.37	0.079	0.17	0.17
Predominant α_{speed}	9.86	26.23	0.37	0.076	0.17	0.18
Predominant α_{dobs}	9.66	25.89	0.37	0.078	**0.16**	**0.16**
No α_{dbest}	9.62	25.82	0.37	0.079	**0.16**	**0.16**
No α_{osc}	9.58	25.64	0.37	0.079	**0.16**	**0.16**
Prediction 5 seconds	**9.21**	**24.28**	**0.38**	**0.052**	**0.16**	0.18

Table 5.5: DW4DO Parameters Test: Scenario 2: Results

- **Third Scenario**: the paths followed by the different configurations are shown in Figure 5.6 and the results in Table 5.6. In this scenario two configurations failed: no setting α_{dbest} it can result in fail reaching the goal (bright turquoise line) and keep the robot going in circles around it. In the same way, setting α_{speed} to a high value it can result in similar fails, because the robot is not able to turn enough to reach the goal getting stuck in a local minima.

  The other configurations are able to reach the goal, but with different behaviours. Proposed α and predominant α_{dobs} works in a very similar way because there are no obstacles in the scenario (α_{dobs} does not affect the velocities selection) and the differences are caused by the simulator variations. Setting α_{osc} to zero will cause more unstable manoeuvres (black line) and traversing more distance. Setting the prediction to five seconds cause a path significantly more straight and short than the rest.

With the information obtained from these tests it is concluded that the proposed weight configuration is a right choice and it will be used. However, setting the prediction time to 5 seconds instead of 10 seconds will improve the followed path, but it can have problems with moving obstacles. So, in the next tests both configurations will be used, depending on if a global planner is involved (reducing the prediction time) or not.

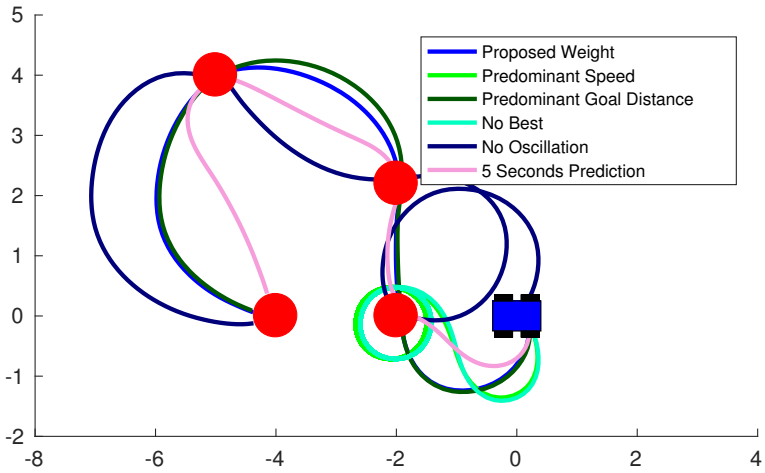


Figure 5.6: DW4DO Parameters Test: Scenario 3

Configuration	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Proposed α	15.05	55.30	0.27	0.164	**0.16**	0.19
Predominant α_{dobs}	15.29	54.27	0.28	0.162	0.17	0.20
No α_{osc}	21.67	69.41	**0.31**	**0.146**	0.21	**0.16**
Prediction 5 seconds	**12.96**	**53.20**	0.24	0.173	0.19	0.27

Table 5.6: DW4DO Parameters Test: Scenario 3

5.1.2 Dynamic Obstacle Avoidance

In this section, tests are performed where the robot must avoid dynamic obstacles simulating real life scenarios.

Dynamic obstacles are simulated as robots in order to control their movement, with a box on the top of 0.7m x 0.4m, and moving with constant velocities. As the position and velocities of each robot can be obtained from the simulator, an algorithm has been developed to associate the laser impacts to the obstacles, subsequently obtaining

the velocities of each point of the laser as the algorithms need.

The parameters that has been used for each algorithm, and kept during the different tests are the following:

- CVM cost function has the weights: 0.6 for distance (α_{dist}), 0.3 for heading ($\alpha_{heading}$) and 0.1 for speed (α_{speed}), where the α_{dist} is the predominant one as has been obtained in the parameter sensitivity section. Using this parameters the robot selects a curvature arc based on three criteria in order of importance: maximizing the chances of selecting an arc that faces the goal, then selecting a curvature arc with maximum distance free of obstacles increasing the security of the navigation and finally selecting a path that maximizes the linear velocity of the robot.

- PCVM algorithm uses the same cost function, and the inputs of the local mapping are the predictions of 2 seconds ahead the current iteration.

- DCVM uses the same cost function adding the weight of the moving obstacles (α_{dyn}) which is set to 1 in order that the resulting cost had one third of the CVM cost function value if there is a possible collision close to the robot. To consider that a collision is probable the robot and the obstacle have to be in the same point within 2 seconds difference. To consider a medium risk value the robot and the obstacle must be in the same point within 5 seconds difference.

- DW4DO uses the proposed α weight explained before, adding the $\alpha_{movobs} = 0.5$. Using this configuration the algorithm avoids crossing obstacles path in an aggressive way, then it tries to reach the "best window" arc at the same time that keeps away from obstacles. Then the algorithm selects an arc that is heading the goal, with high speed and that approaches to the goal. Finally the α_{osc} is set to a low value in order to help the stability of the algorithm without avoiding vary the trajectory.

- DW4DOT uses the same cost function, and the tree is built with a fixed depth of 3. The local window map is also extended

to 30 m x 30 m in order to calculate curvature arcs within its dimensions.

The scenarios evaluated are the following:

1. **Two obstacles crossing**: in this scenario, the final goal is in front of the robot and two objects cross its direct path at 0.4 m/s (Figure 5.7). The numerical results are shown in Table 5.7.

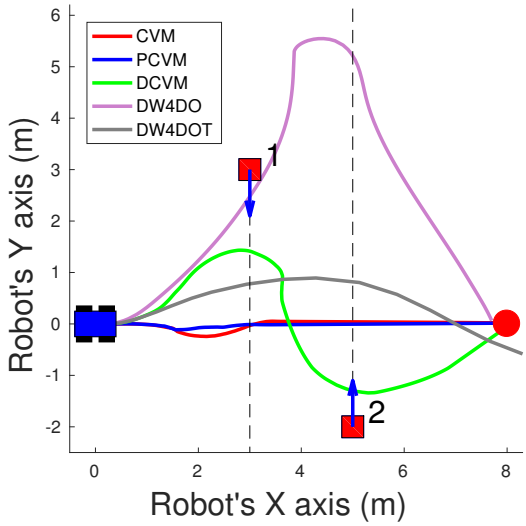


Figure 5.7: Dynamic Obstacle Avoidance: Two obstacles crossing

CVM and PCVM try to avoid the first obstacle turns to its right, which is a risky situation because the robot crosses the obstacles path and it can cause a unavoidable collision situation. The algorithms that work with velocities and predictions of obstacles (DCVM, DW4DO and DW4DOT) are able to avoid both obstacles without crossing its path. However, DW4DO algorithm falls in a local minima with the second obstacle, as it is unable to avoid it from behind without reducing its speed. Finally it is able to escape the local minima when the goal is behind the robot and its speed's term changes (the robot prioritizes turns to the goal instead of travelling at high speed).

2. **Corner and two obstacles**: in this experiment (Figure 5.8 and numerical results in Table 5.7) the goal is at 45 degrees from the start orientation of the robot and it is blocked by a big obstacle that must be rounded. There are two moving obstacles, one crossing the natural path of the robot from right to left at 0.4 m/s and one overtaking the robot at its left side at 0.75 m/s. Both obstacles cross at the big obstacle corner.

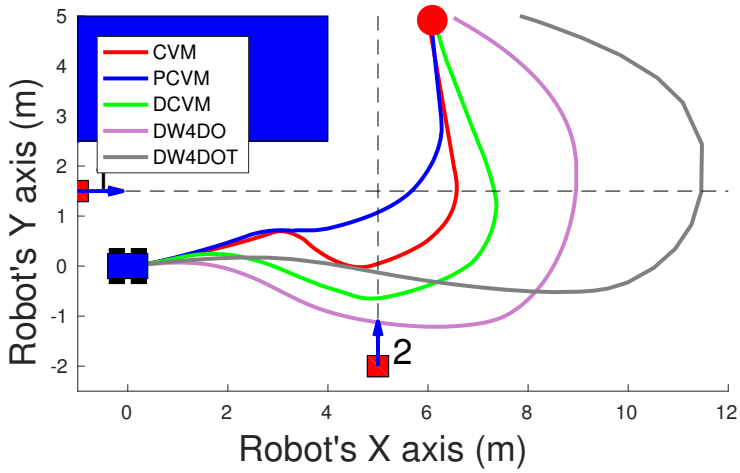


Figure 5.8: Dynamic Obstacle Avoidance: Corner and Two Obstacles

In this scenario all the algorithms are able to reach the goal but performing slightly different avoidance manoeuvres. Both CVM and PCVM begin the avoidance turning to the direction of the goal because they do not know where the moving obstacles will be after several seconds, and they get stuck in the corner until both obstacles move away from the robot path. PCVM prediction improves the performance of the algorithm with respect to the original CVM. The other algorithms are able to plan with the information of the moving obstacles and they begin the avoidance turning to their right in order to avoid the cross-

ing moving obstacles. DCVM performs the shortest path of the three, but it is less smooth than the DWA based approach. DW4DOT performs a less natural avoidance manoeuvre than DW4DO because of its low frequency. In this scenario the best performance is obtained by DCVM (it employs less time but travels along less smooth path) and DW4DO. CVM and PCVM reach the goal in less time but their avoidance of obstacles are too risky.

3. **Corner and two obstacles approaching the robot**: this experiment is set up in the same environment as the previous one with different positions of the moving obstacles (Figure 5.9 and numerical results in Table 5.7): one crosses the natural path of the robot from left to right at 0.4 m/s and the other one approaches the robot at its left side at 0.75 m/s.

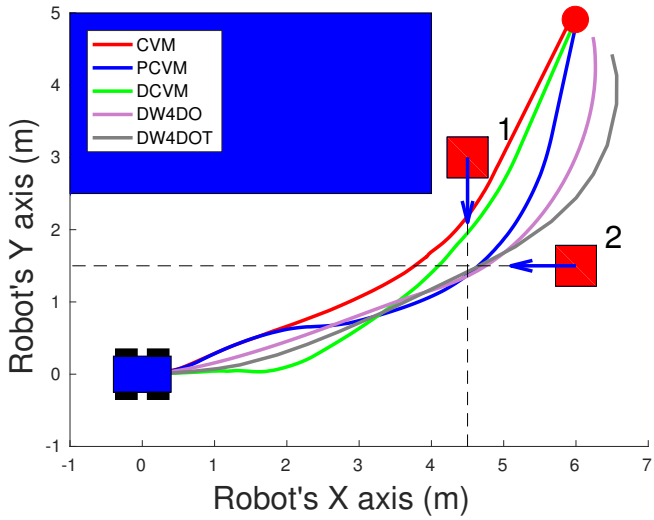


Figure 5.9: Dynamic Obstacle Avoidance: Corner and Two Obstacles Approaching the Robot

In this experiment, all the algorithms begin turning to the corner, but the algorithms that use the obstacles velocities and positions (DCVM, DW4DO and DW4DOT) are able to cor-

rect the turn, going behind the obstacle and performing a safer manoeuvre. In this scenario DWA based algorithms perform better, with DW4DO achieving the faster and smoother path.

4. **Multiple obstacles and moving obstacle approaching the robot**: This scenario is a more complex environment that mixes static and dynamic obstacles (Figure 5.10 and numerical results in Table 5.7). Two more static obstacles are added to the previous scenario: one box in the path of the robot that forms a corridor between the big obstacle and the box and another box in the corner that forms another and wider corridor. In addition there is a moving obstacle in collision course with the robot by its diagonal and through the first corridor.

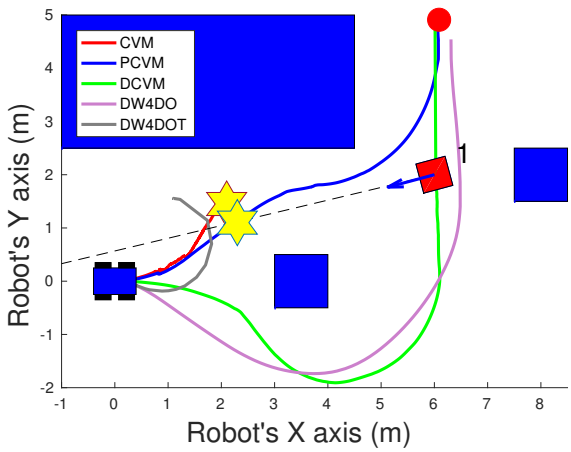


Figure 5.10: Dynamic Obstacle Avoidance: Multiple Obstacles and Moving Obstacle approaching the robot

In this scenario, CVM and PCVM are not able to avoid the approaching obstacle. These algorithms go through the first corridor and when the obstacle affects the path planned it is too close to the robot to be avoided. As the obstacle is in collision course with the robot and due to its low working frequency, DW4DOT needs to turn a lot to avoid the obstacle and falls in a local minima (the robot it is too close to the wall to keep

moving). DCVM and DW4DO are able to avoid the obstacle, with DW4DO performing a faster and smoother path.

5. **Crossroad**: In this scenario there are no static obstacles and the goal is at 45º of the robot. Two obstacles moving in opposite directions simulate a crossroad in the street (Figure 5.11 and numerical results in Table 5.7). In this scenario the first obstacle is not detected until the robot turns and it is close to it.

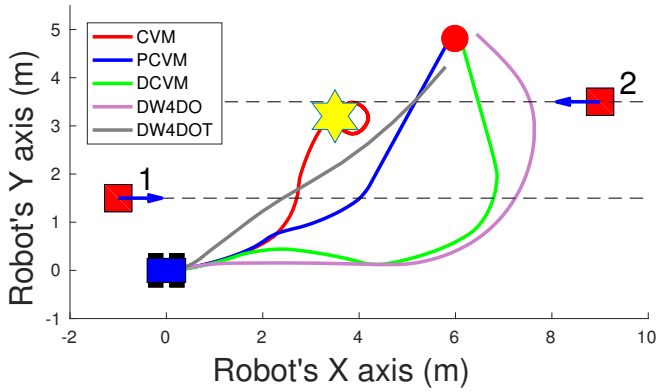


Figure 5.11: Dynamic Obstacle Avoidance: Crossroad

As CVM does not have any information of the velocity of the obstacles and the detected obstacle is very close to the robot, it is not able to perform an avoidance and ends crashing with the second one. PCVM uses the future prediction of the obstacles and it is able to avoid them, but stopping because the obstacles are detected very close to the robot, resulting in a potentially risky situation. DCVM and DW4DO are able to avoid both obstacles in a different way: DCVM waits until the first obstacle crosses its path and then goes to the goal, while DW4DO avoids both obstacles from behind, avoiding the risky situation of crossing their path. DW4DOT is also able to reach

the goal but its avoidance manoeuvre is very different due to its low working frequency.

6. **Lane Changing**: this scenario is a variation of the previous one, but the moving obstacles travel in the same direction of the robot, so it simulates a lane changing in a road (Figure 5.12 and numerical results in Table 5.7). The obstacle closer to the robot travels at 0.75 m/s and the further one at 0.4 m/s.

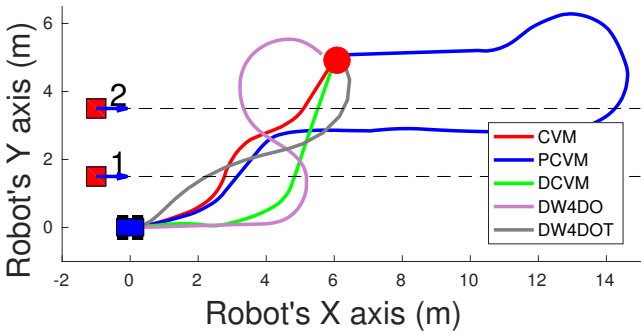


Figure 5.12: Dynamic Obstacle Avoidance: Lane Changing

Since there is no information about the obstacles, all the algorithms begin turning left to face the goal. When CVM detects the obstacle is very close to the robot but it is able to avoid it slowing down its velocity and turning until the obstacle is not blocking the path. PCVM uses the future prediction of the obstacles, but almost fails into a local minima as the obstacle moves parallel to the the robot and with the same velocity. Finally it is able to escape to that local minima and reach the goal. DW4DO, DW4DOT and DW4DOT are able of avoid the obstacles from behind, avoiding potentially risky situations. In that scenario DW4DO performs a long, but faster and smoother

trajectory.

Table 5.7 shows the results of each successful execution. From this table and the path performed by each algorithm some conclusions can be obtained:

- Algorithms that use the prediction of the dynamic obstacles are able to perform better path than the algorithms that do not use that information. Therefore, it is demonstrated that it is useful to have information about the moving obstacles for an obstacle avoidance algorithm, and take it into account in order to work in real scenarios.

- DCVM and DW4DO, which are the algorithms that use the information about the moving obstacles in the current iteration, and are able to reach the goal in all the experiments.

- DWA based algorithms usually perform in a smoother way for two reasons: the evaluated velocities on each iteration have a fixed step between them, while the evaluated velocities in CVM based algorithms depend on the environment perception. Also, the DWA based proposed algorithms have a term in the cost function to keep the same velocity and increases the stability of the algorithm, which is not available in CVM based ones.

- As the algorithms work in a local window with information obtained from the sensors, they can fall into local minima, so working along a global path planner is recommended.

- DW4DOT should have a better performance than DW4DO, however, due to its increasing time consumption (it evaluates 6 times more curvature arcs on each iteration, and with a bigger local mapping window) it can not perform with real time requirement scenarios.

Experiment	Algorithm	d	t	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
2 Obstacles Crossing	CVM	7.88	36.62	0.21	0.18	3.23	6.41
	PCVM	**7.79**	**25.07**	0.31	0.14	3.01	5.66
	DCVM	10.32	28.92	**0.35**	**0.09**	11.38	13.96
	DW4DO	14.24	45.53	0.3087	0.13	6.37	7.18
	DW4DOT	9.52	30.48	0.278	0.14	**2.77**	**3.53**
Corner and Two Obstacles	CVM	10.95	37.55	0.29	0.13	6.08	8.44
	PCVM	**9.42**	**33.12**	0.28	0.14	**5.27**	6.78
	DCVM	12.31	37.70	0.32	0.11	5.49	7.05
	DW4DO	15.303	41.66	**0.36**	**0.085**	5.328	**2.97**
	DW4DOT	19.323	56.20	0.33	0.11	3.516	4.05
Corner and Two Obstacles Approaching	CVM	**8.10**	58.42	0.13	0.17	**1.69**	**3.50**
	PCVM	8.57	26.47	0.32	0.11	5.15	5.42
	DCVM	8.33	28.20	0.29	0.12	3.70	4.64
	DW4DO	8.69	**23.57**	**0.36**	**0.08**	4.76	3.60
	DW4DOT	8.72	28.33	0.28	0.13	4.37	4.91
Multiple Obstacles	DCVM	**12.58**	41.37	0.30	0.12	6.26	7.95
	DW4DO	11.86	**32.38**	**0.36**	**0.074**	**5.36**	**3.95**
Crossroad	PCVM	8.17	**27.50**	0.29	0.14	4.00	6.14
	DCVM	10.81	44.42	0.24	0.18	4.14	6.57
	DW4DO	11.63	31.64	**0.35**	**0.08**	5.08	3.90
	DW4DOT	**7.25**	38.65	0.17	0.15	**1.9**	**2.68**
Lane Changing	CVM	**8.08**	44.42	0.18	0.17	**4.14**	6.60
	PCVM	26.23	97.20	0.26	0.17	4.22	7.13
	DCVM	8.93	47.70	0.18	0.18	2.76	**4.99**
	DW4DO	12.86	34.33	**0.36**	**0.07**	9.84	7.87
	DW4DOT	8.67	**32.57**	0.26	0.13	4.83	5.87

Table 5.7: Dynamic Obstacle Avoidance: numerical results

5.1.3 Navigation

In order to test the performance of the obstacle avoidance algorithms working in conjunction with global planners in a full navigation framework, several tests will be carried out.

Only the better working algorithms evaluated previously (DCVM and DW4DO) will be tested. The environment selected is a simulation in Gazebo of the Polytechnic School (west second floor). A simple Dijkstra global planner is executed and each algorithm must reach partial goals each 3m until the final goal. For these tests the DW4DO configuration will be changed to the 5 seconds prediction one. Three scenarios are tested:

- **First scenario**: In this scenario the ability of following a previously calculated path delimited by obstacles in the sides of the robot is evaluated. Figure 5.13 shows the path performed by

each algorithm (blue and red lines) and the local goals planned by the global planner (yellow circles).

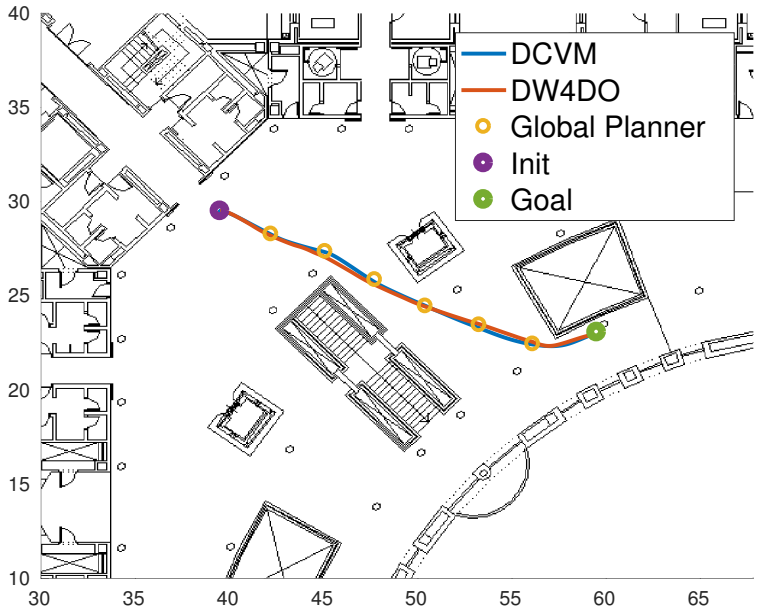


Figure 5.13: Navigation: Scenario 1

In this scenario both algorithms are able to reach the goal in a similar way because the intermediate goals are in the movement direction of the robot. DW4DO achieved the faster and shortest path to the goal, but DCVM achieved a smoother path with less angular velocities (numerical results are shown in table 5.8).

- **Second scenario**: this scenario tests how the algorithms react to not previously mapped obstacles that affect to the followed path. There are three boxes that affect to the planned path. The path without taking this obstacles into consideration, is an open curve between walls, which are narrowing to the end of the path. Paths followed by the robot are shown in figure 5.14, and not mapped obstacles are shown as blue boxes.

  Both algorithms are able to reach the goal, but the trajectory is slightly different. As DW4DO gets away from the partial goal, and finally reaches it in a direction that is not the straight line

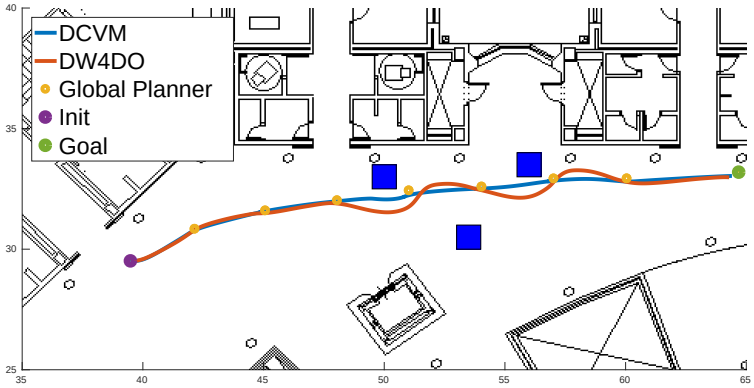


Figure 5.14: Navigation: Scenario 2

to the next goal, the robot tends to make wide open curves from one goal to the next, traversing more distance. DCVM gets closer to the obstacles reducing its velocity. DW4DO is able to reach the goal in less time and DCVM traversing less distance. (Numerical results are shown in table 5.8)

- **Third scenario:** this scenario is a more complex one and tests if the algorithm is able to reach goals that are not in the way of the robot (first goal is directly behind the robot) and if the robot is able to enter corridors, which is a common situation in real life scenarios. Paths followed by the robot are shown in Figure 5.15.

  Both algorithms are able to turn and face the first goal behind the robot. DCVM fails to reach the goal as it gets stuck in a local minima between a column and the corridor entrance. To get into a corridor is a hard task for algorithms based on curvature arcs, however, placing a goal just at the entrance helps the algorithm and makes the DW4DO able to reach the goal.

Table 5.8 shows a summary of the data collected during the navigation experiments, and it is clear that the smoothness of each test has similar values than the tests performed without the aid of a global path planner. It can be concluded that both algorithms are suitable

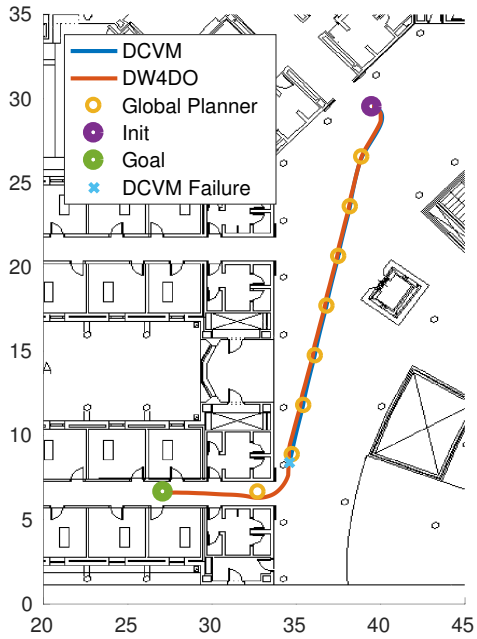


Figure 5.15: Navigation: Scenario 3

to work into a navigation framework, and the better the global path planner is the better both algorithms will perform.

Experiment	**Algorithm**	**d**	**t**	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Scenario 1	DCVM	58.36	21.29	0.36	0.08	**2.05**	**3.38**
	DW4DO	**54.97**	**21.18**	**0.38**	**0.03**	2.9	3.74
Scenario 2	DCVM	69.13	**25.45**	0.36	0.08	**2.25**	**4.07**
	DW4DO	**67.80**	26.21	**0.38**	**0.03**	5.72	8.60
Scenario 3	DW4DO	**80.56**	**30.67**	**0.38**	**0.05**	**4.82**	**6.88**

Table 5.8: Navigation: numerical results

5.1.4 Global Path Planning

In this section we are going to perform tests in order to check the behaviour of the DLP algorithm proposed.

As the proposed DLP works in two steps: first an A* algorithm for path planning is used and then the output is used to calculate the state lattice plan. Several tests are performed in order to test the better and faster cost function for the A* path planner. Two scenarios have been tested, with two different positions on each scenario. The multi resolution map has been limited to six depth levels.

For each scenario two cost functions have been tested: a function that adds the estimate cost to reach the goal and the cost to reach the cell (A* proposed function) and a function that only takes into account the estimate cost to reach the goal (guided Dijkstra).

The first scenario is a realistic map (parking at the Polytechnic School), with approximately 74 x 108 m, a higher resolution of 0.11 meters cell and a lower resolution of 3.52 meters cell. Figure 5.16 shows this scenario with planned paths using both cost functions.

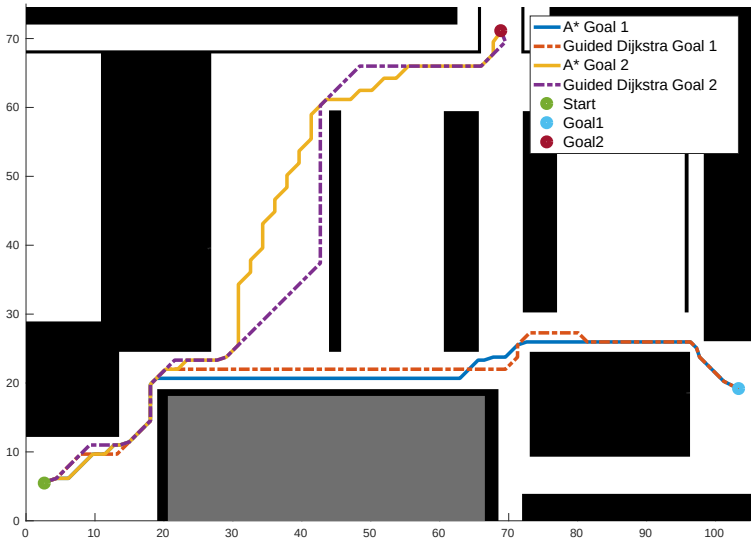


Figure 5.16: A* Weight Function Evaluation: Scenario 1

The second scenario is the map of the Polytechnic School (west second floor) that has been used for the navigation tests. It is a more complex map, with approximately 68 x 70 m, a higher resolution of 0.0625 meters cell and a lower resolution of 2 meters cell. Figure 5.17 shows this scenario with planned paths using both cost functions.

Figure 5.17: A* Weight Function Evaluation: Scenario 2

Table 5.9 shows the evaluated nodes and the expanded nodes not yet evaluated for each algorithm. It is clear that the Guided Dijkstra algorithm works in a faster way, expanding between 66 and 92 percent less nodes. Even when the A* algorithm produces more natural and short paths, as it takes into account the total length of the path, Guided Dijkstra algorithm will be used for DLP due to its low computational requirements.

Once the Guided Dijkstra cost function has been selected, another set of tests have been carried out in order to check how the multi resolution map A* improves the path searching algorithm. Tests have been performed using the real map of the Polytechnic School, with approximate dimensions of 138 x 138 m. The higher resolution is set to 0.05 meter each cell, and it is doubled for each depth until the sixth one (0.1m/cell, 0.2m/cell, 0.4m/cell, 0.8m/cell and 1.6m/cell).

Paths obtained are represented in Figures 5.18 and 5.19 where

	A*		Guided Dijkstra	
Scenario	**Evaluated**	**Expanded**	**Evaluated**	**Expanded**
1	312	**42**	**86**	133
2	363	**37**	**32**	49
3	1550	**42**	**538**	294
4	1355	**107**	**117**	270

Table 5.9: Global Path Planning: A* Weight Function Evaluation

all the multi resolution A* path planners are able to reach the goal position, but the path is slightly different: as the path takes only into account the central point of the cells, increasing the size of the cells results in paths far away from the obstacles.

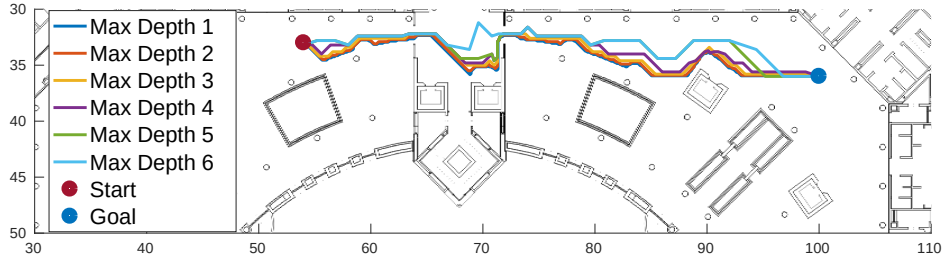


Figure 5.18: A* Multi Resolution Test: Scenario 1

Table 5.10 shows the nodes evaluated and the expanded ones not yet evaluated. It is clear that increasing the multi resolution depth significantly reduce the evaluated nodes, speeding up the algorithm. However, as the algorithm is a guided search, if there are only a few cells of the higher levels, it is possible to discretise in less levels (i.e. five levels instead of six) obtaining better results. With these experiments it has been demonstrated how the multi resolution path searching speeds up the algorithm.

In order to test if the guided Dijkstra algorithm improves the performance of the DLP, experiments are going to be carried out. The tests are performed in two different scenarios, one more complex than the other. The parameters of the DLP planner are: planning each 1.8 seconds, velocities limited to 0.5 m/s and 0.5 rad/s and velocities steps of 0.125 m/s and 0.125 rad/s, for linear and angular

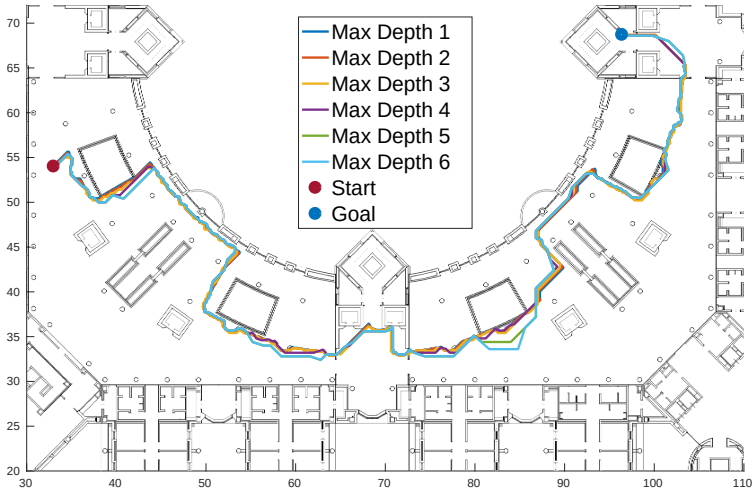


Figure 5.19: A* Multi Resolution Test: Scenario 2

	Scenario 1		Scenario 2	
Max Depth	**Evaluated**	**Expanded**	**Evaluated**	**Expanded**
6	**47**	**122**	1495	**387**
5	50	129	**1490**	398
4	66	169	1526	475
3	159	323	1949	697
2	272	661	3571	1117
1	510	923	10588	1968

Table 5.10: Global Path Planning: A* Multi Resolution Evaluation

respectively. Each scenario has been repeated five times with different aids obtained from the A* planner: no aid, and one goal each 10, 5, 2 and 0.1 meters. Figure 5.20 and 5.21 show the path planned for each execution.

Tables 5.11 and 5.12 show the results of the execution in both environments. In this table we can evaluate how is the path generated (less travelling time t, more linear velocity v and less angular velocity ω) and the processing time t_p (the algorithms were executed in the same machine, an i7 core with 16 Gb of RAM memory running Ubuntu 14.04). As the algorithm is executed with less guidance produces faster and more natural paths to the goal. As more guidance

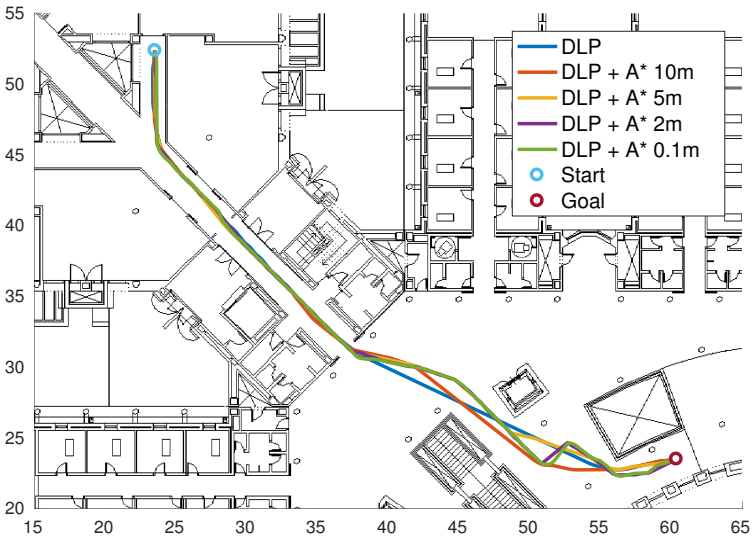


Figure 5.20: DLP + A* evaluation: Scenario 1

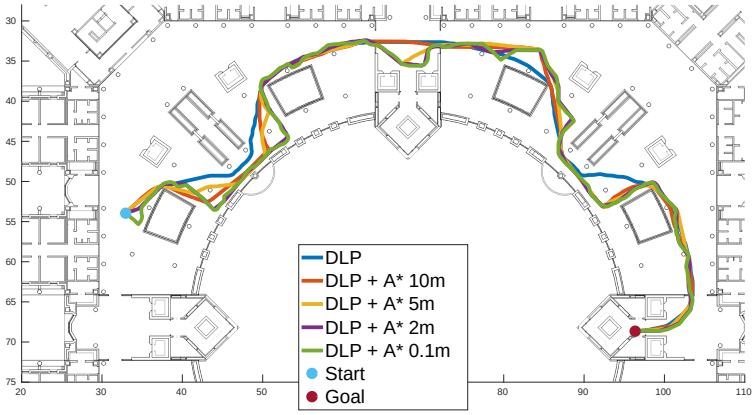


Figure 5.21: DLP + A* evaluation: Scenario 2

points are obtained, the resulting path makes some turns that are not desirable, almost fitting the guided Dijkstra planning (which is obtained in a fast way, but is not the best path). Nevertheless, as the size of the scenario increases and the goal is not reachable following a straight line, the computational cost of using DLP without guidance increases drastically, getting to more than half an hour for the evaluation in the second scenario. For this reason, a guidance with points

between 5m and 10m will be used (the costs increases very little with respect to the full guidance and the path improves considerably).

Algorithm	t_p (s)	**t** (s)	$\overline{v}$ (m/s)	$\overline{\omega}$ (rad/s)
DLP	50.41	**178.2**	**0.28**	**0.0512**
DLP + A* (10m)	4.8	217.8	0.24	0.063
DLP + A* (5m)	4.28	282.6	0.18	0.086
DLP + A* (2m)	4.31	244.8	0.22	0.094
DLP + A* (0.1m)	**4.25**	275.4	0.19	0.102

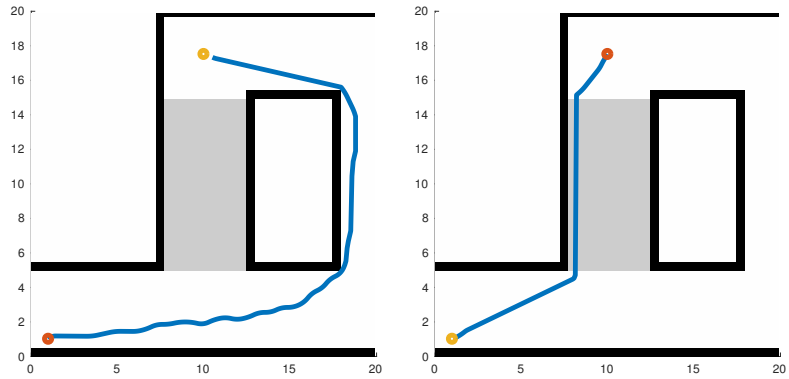
Table 5.11: DLP + A* evaluation: Scenario 1

Algorithm	t_p (s)	**t** (s)	$\overline{v}$ (m/s)	$\overline{\omega}$ (rad/s)
DLP	1880.28	369	**0.30**	0.0793
DLP + A* (10m)	17.65	624.6	0.190	**0.0605**
DLP + A* (5m)	4.57	**588.6**	0.203	0.0822
DLP + A* (2m)	4.17	667.8	0.191	0.0802
DLP + A* (0.1m)	**4.06**	844.2	0.158	0.119

Table 5.12: DLP + A* evaluation: Scenario 2

Several tests are performed using DLP algorithm in scenarios with some constraints given from adding semantic information. In Figure 5.22 there is a lane (shown in grey colour) that can be only traversed from north to south. In both sub figures the starting point is represented as red circle and the goal point as yellow circle. In the first test (Figure 5.22(a)) the lane can not be traversed and the algorithm plans a longer path through the right corridor. In the second test (Figure 5.22(b)) where the initial and final point are reversed, the algorithm is able to plan through the lane obtaining a shorter path. These tests show how DLP algorithm can be used to plan trajectories in city environments, where the lanes can be only traversed in one direction.

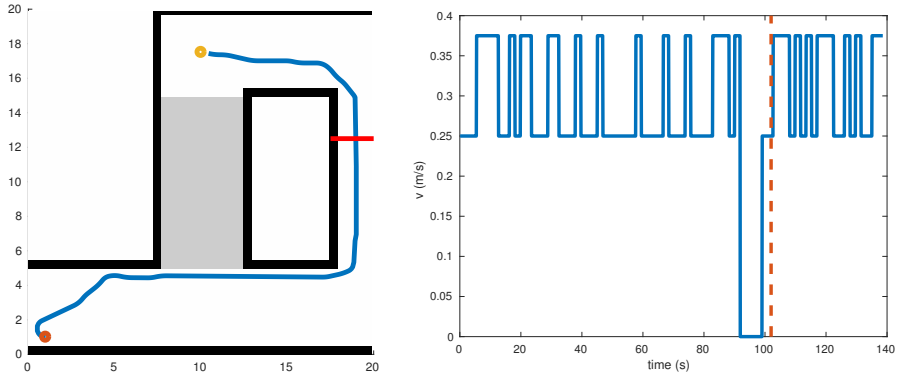
Several tests are simulated to check the performance of the DLP algorithm in the presence of dynamic obstacles. In the first test (Figure 5.23) an obstacle blocks the possible paths to the goal during a certain time, therefore it allows the robot pass like a semaphore



(a) Direction not Traversable (b) Direction Traversable

Figure 5.22: DLP Lane Example

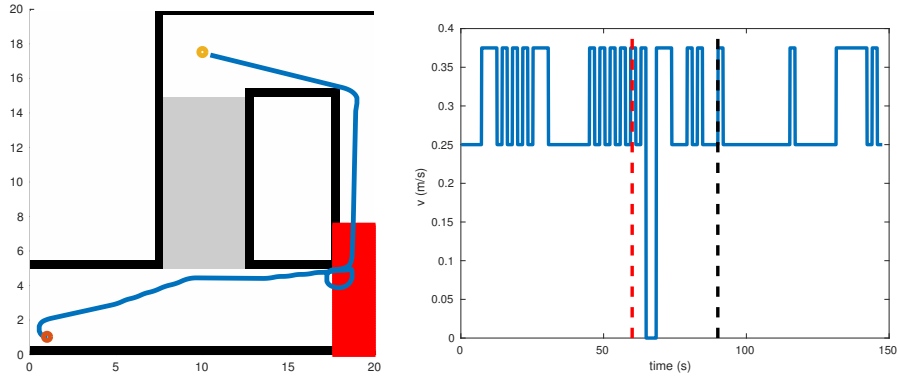
regulated cross. In Figure 5.23(a) the position of the obstacle is represented as a red line, and the path planned by the algorithm as blue line. Figure 5.23(b) shows the linear velocities that the robot should follow to achieve the path (blue line) and the time where the obstacle disappears (discontinued red line, at 102 seconds). The algorithm is able to plan paths to the goal even when an obstacle blocks all the possible solutions during a time. In the velocities figure is clear how the robot should stop before the obstacle till it disappears.



(a) Path Planned (b) Linear Velocities

Figure 5.23: DLP Semaphore Test

Figures 5.24 shows a test where an obstacle blocks the possible paths to the goal and then moves and it makes the goal reachable. In Figure 5.24(a) the obstacle is represented as a red box, beginning its movement from the top border to the bottom border. In Figure 5.24(b) the linear velocities of the robot are shown, and the time where the obstacle begins to move (discontinued red line) and stops (discontinued black line). DLP algorithm is able to reach the goal, first the robot stops when it is close to the obstacle (same behaviour than the previous test) but when the obstacle moves towards it, the planner provides an avoidance manoeuvre and continues through the corridor.



(a) Path Planned (b) Linear Velocities

Figure 5.24: DLP Dynamic Obstacle Example

These tests demonstrate the ability of DLP algorithm to avoid situations where the obstacle avoidance algorithms fails, like paths that are blocked only during a time. However, the computation of the possible states of the robot increases, and the discretisation of possible paths variation should be reduced to work in real scenarios.

5.2 Real Robot Experiments

5.2.1 Dynamic Obstacle Avoidance

In order to test the behaviour of algorithms in real world scenarios, and also the performance of the RoboShop platform, several tests are carried out.

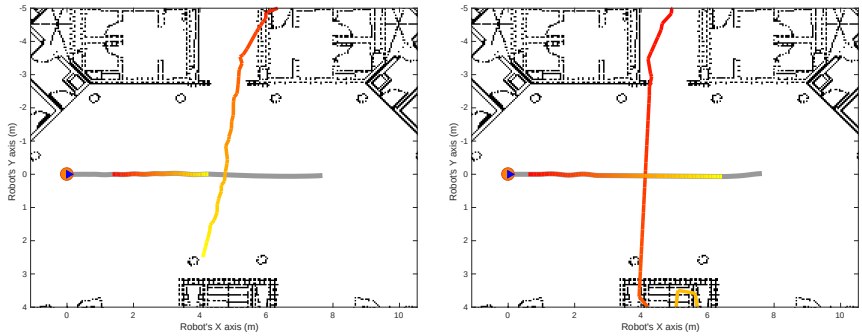
In these tests the RoboShop platform is equipped with a *Sick LMS-151* LIDAR to improve the perception range of the robot, allowing the algorithm to have more space where the dynamic obstacles can be tracked. The perception stage is achieved using the Dynamic Obstacles Map (DOMap) algorithm (explained in appendix A.2).

The localisation of the robot is based on the Odometry and IMU fusion (explained in appendix A.1). The tests are performed without the necessity of a map or a global localisation system, because the goals are relative to the robot start position and in reachable poses.

The two best performance obstacle avoidance algorithms have been selected for these tests: DCVM and DW4DO. Their configuration have been selected according to the best configuration selected in the simulated tests. The only change is that the possible accelerations has been increased in order to allow the robot plans with more possible directions. This is because in real scenarios the algorithm can be sped down and reduces its working frequency, and, as the acceleration limits are fixed in both algorithms, some feasible paths by the robot are not evaluated.

Due to the tests are performed with people moving around the robot, the scenarios can not be exactly repeated in the same way. For this reason, even when the numeric data collected from each test is shown in Table 5.13, it can not be compared between the both algorithms execution in the same way that is done in simulation. The paths performed by the robot are shown in figures. On each figure the robot starts in the orange circle and the path followed is shown in grey. When a dynamic obstacle is detected its path is shown in different colours, with the time of detection going from darker colours to lighter ones. The path of the robot is also shown in the same colour variation in order to show which part of the whole path is affected by the dynamic obstacles.

In the first scenario evaluated, there is a goal in front of the robot, which can be reached in a straight line bounded by walls (Figure 5.25). In this scenario a person crosses the natural path of the robot. The robot begins to detect the person when is coming out of a corridor. DCVM algorithm (Figure 5.25(a)) slightly varies its velocities and tries to avoid the person. As the person is moving faster than the robot the path to the goal is free of obstacles and the robot continues through it. DW4DO algorithm (Figure 5.25(b)) detects the person when appears and varies its trajectory trying to pass behind the predicted movement. Finally, as the path to the goal is free of obstacles the robot continues to the goal. Numerical results are shown in Table 5.13 where the performance of both algorithms are very similar (DCVM is slightly faster and DW4DO slightly smoother).

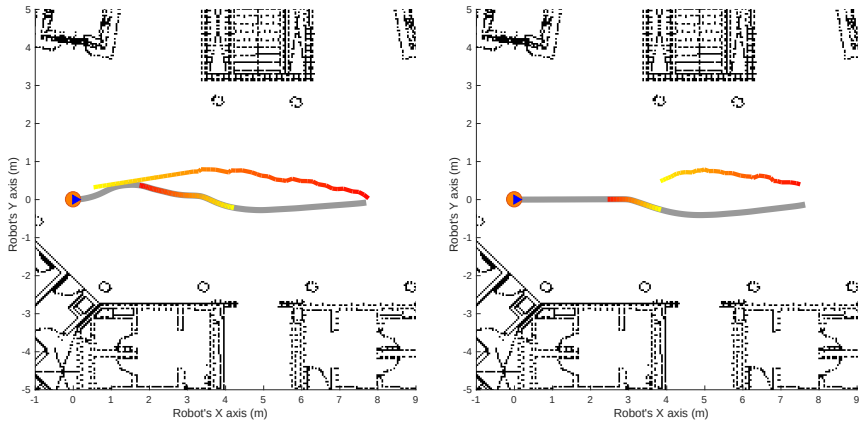


(a) DCVM (b) DW4DO

Figure 5.25: Dynamic Obstacle Avoidance: Real Scenario 1

The second scenario evaluated consists in the same goal with respect the robot, but it is situated in other position in the environment, where the path is bounded by different walls (Figure 5.26). In this scenario a person is moving parallel to the robot and towards it. In this case, both algorithms are able to detect the movement of the person approaching the robot, blocking its possible turn to the left. Both algorithms turn to the right in order to avoid the moving obstacle. DW4DO (Figure 5.26(b)) is able to go on before DCVM (Figure 5.26(a)) as it checks the whole footprint of the robot with the moving obstacles, and DCVM only checks the infinitesimal point paths that

crosses. This kind of obstacle avoidance is not feasible using algorithms that not takes into account the dynamics of the environment, as it considers that the moving person does not affect its trajectory until it is close to the robot. Numerical results are shown in table 5.13. In this case, DW4DO achieves and slightly better performance.

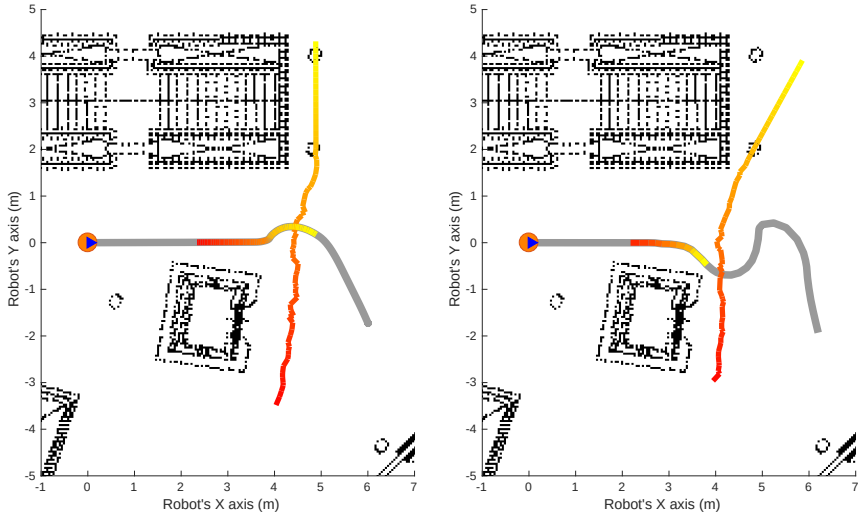


(a) DCVM (b) DW4DO

Figure 5.26: Dynamic Obstacle Avoidance: Real Scenario 2

In the third scenario evaluated there are two consecutive goals, one in front of the robot and one to its right side. The robot starts in a corridor formed by stairs and an elevator. When the robot is close to the first goal a person appears from behind the elevator and crosses its natural path close to the robot (Figure 5.27). When DCVM (Figure 5.27(a)) detects the person tries to avoid it traversing its path. As the person has been detected only with a few measurements, its velocities are not exact and the algorithm calculates that both trajectories does not collide. As time goes on, the person is closer to the robot and then it turns to avoid him. When the first goal is reached the robot turns and continues to the second goal. When DW4DO (Figure 5.27(b)) detects the person tries to avoid him from behind, in a safer manoeuvre. When the person passes the robot turns again to reach the goal and continues until the final goal. Numerical results are shown in Table 5.13, DCVM travels less distance, but as its avoidance manoeuvre does not get ahead of the obstacle, it results in

a slower trajectory than DW4DO performs.

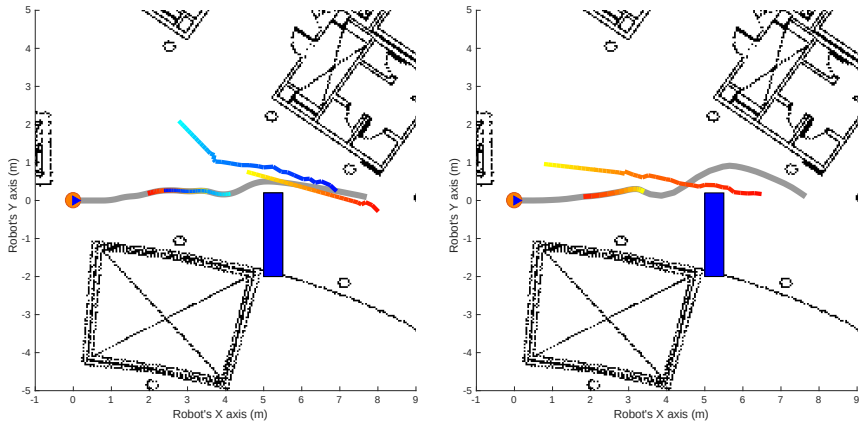


(a) DCVM (b) DW4DO

Figure 5.27: Dynamic Obstacle Avoidance: Real Scenario 3

In the fourth scenario evaluated there is an obstacle of big size between the robot and the goal, and it needs to be avoided by the robot by its left side. When the robot is close to the obstacle a person appears from behind it moving towards the robot (Figure 5.28). Both DCVM (Figure 5.28(a)) and DW4DO (Figure 5.28(b)) algorithms work in a similar way. The robot begins the avoidance manoeuvre turning to its left but when the person is detected, it turns to its right approaching to the static obstacle which is safer than intersect the dynamic trajectory. When the person passes, the robot retakes its avoidance manoeuvre and it is able to reach the goal. In the case of DCVM the perception stage was not able to detect the person correctly and label it as two different detections (red and blue lines in Figure) making the avoidance more unstable. Numerical results are shown in table 5.13 where can be seen that DCVM achieves a better performance, even when the path followed is riskier.

In the fifth scenario the robot must reach a goal in front of it, and as it moves two persons crosses the path of the robot and they stop in

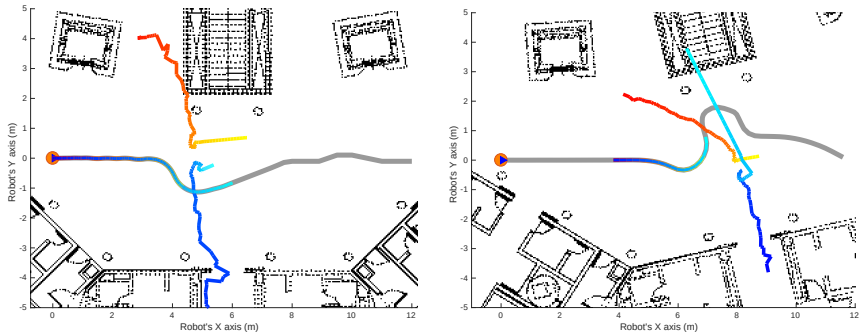


(a) DCVM (b) DW4DO

Figure 5.28: Dynamic Obstacle Avoidance: Real Scenario 4

the middle of it, creating a difficult situation as two dynamic obstacles transform into a big static one (seen in Figure 5.29). When DCVM algorithm detects the moving obstacles they block the path to the goal and the robot begins turning with high curvature values in order to avoid them. When the two persons stop, the perception algorithm is able to detect that situation and the robot begins an avoidance manoeuvre around the formed obstacle (Figure 5.29(a)). DW4DO algorithm performs a similar manoeuvre, but, as the perception is always different (the test can not be exactly replicated) the robot begins turning to its right trying to avoid the first person detected from behind. Once the perception of the other one affects the robot path, the robot turns with high curvature values in order to avoid both crossing obstacles trajectories. When the two persons keep still the robot moves around them and reaches the goal (Figure 5.29(b)). Numerical results are shown in table 5.13 where DCVM obtains best results as DW4DO performs a longer and slower trajectory.

The last scenario (Figure 5.30) tested is similar to the previous one, but in this case the two persons comes from behind the robot in a diagonal movement, blocking all the possible paths to the goal until they stop in the direct path from the robot to the goal. After the robot passes persons start moving again. When DCVM algo-

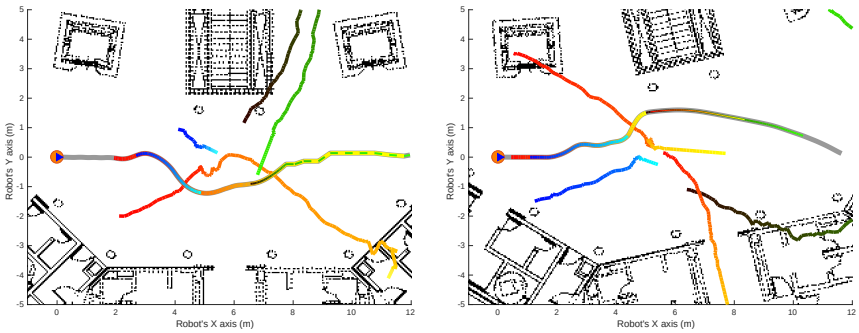


(a) DCVM (b) DW4DO

Figure 5.29: Dynamic Obstacle Avoidance: Real Scenario 5

rithm (Figure 5.30(a)) detects the moving obstacles the robot begins an escape manoeuvre turning in high curvature turns. When the perception algorithm is able to detect that both persons stops the robot continues the normal avoidance manoeuvre and it is able to reach the goal. When DW4DO (Figure 5.30(b)) detects the persons it begins to turn trying to avoid them from behind. As both person moves faster than the robot it returns to the original trajectory and it avoids the obstacle formed as a static one, being able to reach the goal. Numerical results are shown in table 5.13 where it can be seen that DW4DO performs a shorter, faster and smoother trajectory to the goal.

With these tests performed, it can be concluded that both algorithms are able to work in real scenarios with similar performance than in the simulated ones. The main difference between the real and simulated behaviour is that the perception stage introduces some error which are not present in simulation, but the algorithms are responsive enough to deal with them in the majority of the cases. However, these errors can lead the robot to some unavoidable collision scenarios where the robot should stop. It is also proven that our RoboShop platform performance is good enough to navigate autonomously and smoothly.



(a) DCVM (b) DW4DO

Figure 5.30: Dynamic Obstacle Avoidance: Real Scenario 6

Experiment	Algorithm	d	t	$\overline{v}$	σ_v^2	$\overline{\lvert\omega\rvert}$	σ_ω^2
Scenario 1	DCVM	7.69	**19.70**	**0.39**	**0.07**	4.93	7.72
	DW4DO	**7.64**	19.76	0.38	0.07	**3.69**	**6.55**
Scenario 2	DCVM	7.86	20.70	**0.38**	**0.07**	6.87	9.74
	DW4DO	**7.71**	**19.93**	**0.38**	**0.07**	**2.75**	**5.72**
Scenario 3	DCVM	**7.26**	32.60	0.22	0.19	**5.09**	**12.44**
	DW4DO	8.87	**28.43**	**0.31**	**0.13**	13.70	20.22
Scenario 4	DCVM	**7.84**	**20.10**	**0.39**	**0.06**	**8.58**	**12.29**
	DW4DO	8.07	23.38	0.33	0.11	9.15	13.38
Scenario 5	DCVM	**12.70**	**31.40**	**0.39**	**0.06**	**10.43**	**14.52**
	DW4DO	13.93	37.50	0.37	0.09	11.82	18.74
Scenario 6	DCVM	12.84	31.31	**0.38**	**0.07**	12.12	14.64
	DW4DO	**12.45**	**32.76**	**0.38**	**0.07**	**8.38**	**13.90**

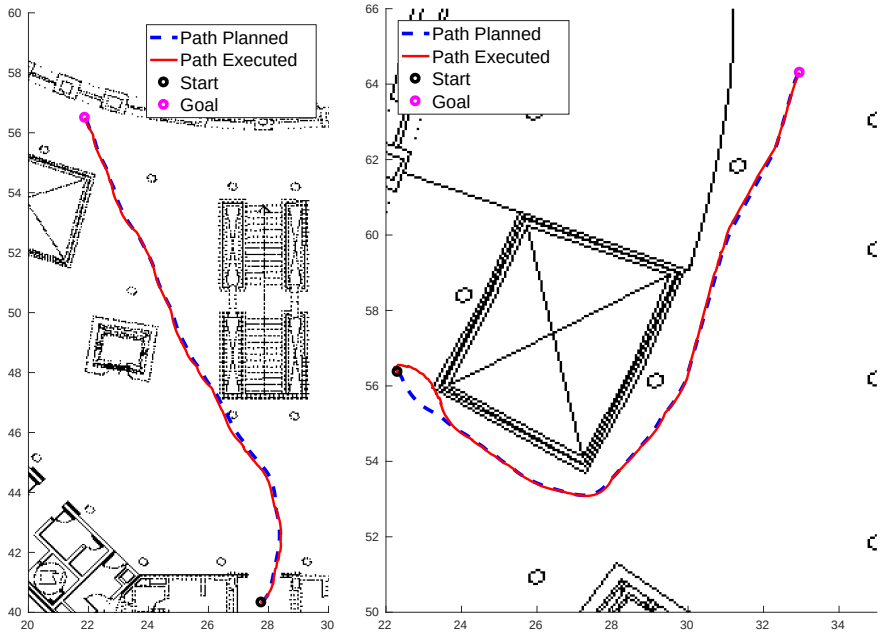
Table 5.13: Real Robot: Dynamic Obstacle Avoidance: numerical results

5.2.2 Global Path Planning

Several tests are carried out in order to test the behaviour of the DLP global planner in real environments. In these tests the robot is localised using an Adaptive Monte Carlo Localization (AMCL) and odometry fusion explained in appendix A.1.

Two tests are performed in the Polytechnic School second floor. In the Figures 5.31(a) and 5.31(b) the route planned by DLP algorithm

is shown as dashed blue lines, and the route followed by the robot is shown as red lines. In order to follow the route, a control algorithm (explained in Appendix A.3) is used. Numeric results are shown in 5.14, where the mean errors in position ($\overline{\epsilon_d}$ and $\overline{\epsilon_\theta}$) and velocities ($\overline{\epsilon_v}$ and $\overline{\epsilon_\omega}$) have similar values to the ones obtained in simulation (shown in Table A.3 in Appendix A.3), even when the localisation system introduces some errors (especially when the AMCL localisation varies into different particles, and introduces some discontinuities in the localisation).



(a) Scenario 1 (b) Scenario 2

Figure 5.31: DLP: Real Robot Tests

DLP has proven to work in real scenarios, however, the localisation and control system induces some errors (especially when the route planned involves turns without movement) that makes risky its use without a local obstacle avoidance system.

Experiment	$\overline{\epsilon_d}$ (m)	$\overline{\epsilon_\theta}$ (rad)	$\overline{\epsilon_v}$ (m/s)	$\overline{\epsilon_\omega}$ (rad/s)
Scenario 1	0.0655	0.0710	0.082	0.138
Scenario 2	0.0852	0.2230	0.012	0.133

Table 5.14: DLP: Numerical Errors

5.2.3 ABSYNTHE Project

This thesis has been developed into the ABSYNTHE project, where a communication and effective cooperation between humans and robot teams is intended.

To test this project, a Cicerone application has been proposed. In this application several agents are involved: human, data stored in remote machines and heterogeneous robots.

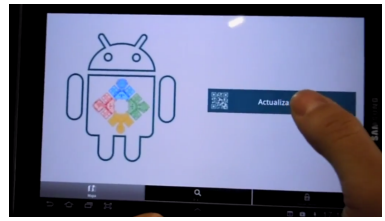
The Cicerone application communicates with the human users via Quick Response (QR) codes that are set in interest points inside a building, and it can be read by a mobile device application (Figure 5.32(a)). Each code is linked to a webpage where the current position of the user is stored. In this application the user can search how to go to an interesting point in the building, selecting the goal (for example, a room into the building) and some settings (for example, use the elevators or the stairs). The path to goal is calculated in a remote computer, where discrete positions of the building representing the topological relevant information are stored, and a Dijkstra algorithm is executed to obtain the shortest path that satisfies the restrictions (Figure 5.32(b)).

Once the path has been received by the user application, there is a possibility to call a Cicerone to guide him to the goal. This Cicerone is an autonomous robot. Each robot in the environment is connected to a central database, and a task assignment algorithm decides which robot needs to move to help user, based on different criteria like distance to the user, capabilities of the robot (not all robots can reach all points in the building) and if the robot is now free or occupied.

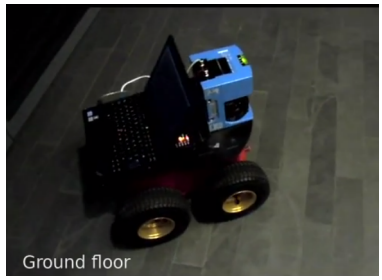
This system works with heterogeneous robots, each one with different capacities. In the demonstration two different robots are used:

a Pioneer 3-AT with off-road capabilities and a *Sick LMS200* LIDAR with 18 m coverage (Figure 5.32(c)) placed on the basement floor of the building and a Pioneer 3-DX without off road capabilities, with a *Hokuyo URG-04LX* LIDAR (Figure 5.32(d)) with 5 m coverage placed on the third floor. Both robots execute the same localisation and navigation algorithms adapted to their capabilities.

(a) Augmented Reality Codes



(b) ABSYNTHE App



(c) Robot 1

(d) Robot 2

Figure 5.32: ABSYNTHE project demonstration

As the project finalised before the end of this thesis, the demonstration has not been made with the final proposed system. The localisation system is based on AMCL filter, navigation and perception are achieved using preliminary versions of DCVM and DOMap algorithms and global path planning has been achieve using a simple Dijkstra algorithm. In order to limit the movement of each robot on the building, the maps that robots use for path planning are different and they are stored in the remote server. Even if more than one robot shares the same environment, its maps can be different in order to adapt to its capacities (for example, going through sand parts of the environment if the robot has off-road capabilities or not). In the

demonstration, each robot is limited to move in one floor (basement and third floor).

A video of this demonstration can be watched in the RobeSafe research group channel [1]. In this example, the ABSYNTHE system was tested in the European Centre For Soft Computing in Mieres (Asturias). An user enters the building and reads an QR code, setting the goal in a room in the third floor. The path is calculated using the elevator and a Cicerone has been called. As there are one robot on each floor, the task dispatch system order the basement robot to move towards the user and the third floor robot moves to the elevator waiting for the user to reach that point. When the basement robot reaches the user position, it plays an audio message welcoming the user, and goes to the elevator slowly so the user can follows it. When the robot reaches the elevator another audio message tells the user the destiny floor. The messages and the situations when each message must be executed are also stored in the remote database. Once the user reaches the third floor, the robot is waiting in front of the elevator with a message in the tactile screen. When the user touches the screen the robot goes to the final destination. Finally, when the room is reached the robot plays another message thanking the user.

This demonstration shows the possibilities of the collaboration between human and robots, and how a full navigation framework, that takes into account the movement of the dynamic obstacles (like the user following the robot) is needed.

5.2.4 RoboShop Project

This thesis was also developed inside the RoboShop project. In this project, our built platform PROPINA was improved to work as a shop assistant. To demonstrate the feasibility of the proposal, tests under real conditions in the Juguetrónica shop basement in Madrid were performed.

The Juguetrónica scenario is a real challenging one for so many reasons: it has very few empty space for moving with the robot, it is

[1] https://www.youtube.com/watch?v=lD44C1WyR58&t=9s

usually crowded of people, the objects in the environment have very irregular forms, there are bright lights and shelves and doors made of glass are present.

In order to communicate with the customers, the robot is equipped with a tactile interface where an HMI helps the interaction (it is shown in Figure 5.33), playing audio messages and waiting for inputs in the screen. As a future work it is proposed to be also responsive for audio commands. In this screen a database of the sections and products of the store is present. When the customer selects a product to buy, the robot will go along the user in order to show where the product is and giving additional information about it. All the positions within the map where the products are and its information are stored in a database. The navigation of the robot is fully autonomous.

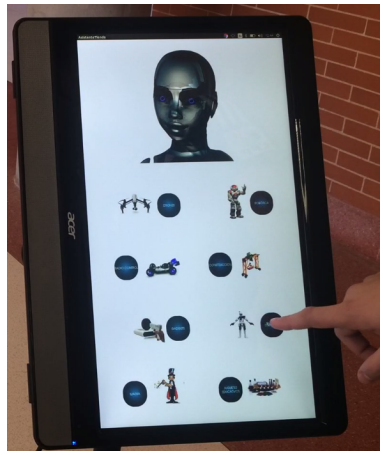
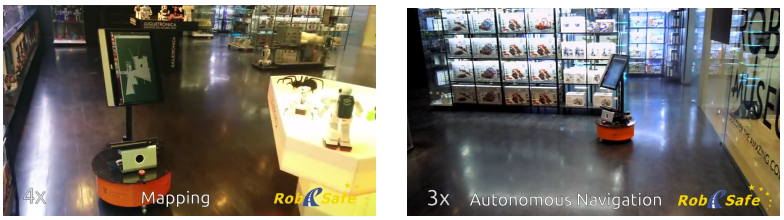


Figure 5.33: RoboShop HMI

As the system can be deployed in different scenarios, the first step to achieve is building a map for localisation and path planning. This process can be seen in Figure 5.34(a). The localisation is achieved using a fusion with AMCL (built using only the LIDAR and not the other sensors), dead reckoning odometry and IMU measurements, fused using an Extended Kalman Filter (EKF) filter as it is explained in Appendix A.1. Even with this localisation system, in that challenging, small and dynamic scenario the localisation can fail. In order

to correct this possible failures Augmented Reality (AR) codes are situated in the environment in order to be detected by the two cameras of the robot (one pointed forward and one pointed to the ceiling). When an AR code is detected the robot can correct its position if it was erroneous.



(a) Mapping (b) Navigation

Figure 5.34: RoboShop Project demonstration

As this project finalises before the ending of this thesis, the navigation framework used is not the final one. One previous version of DW4DO algorithm along a simple Dijkstra global planner is used. For perception, sonar measurement along LIDAR are used, because of the existence of glass objects in the environment.

The maps used for localisation and global path planning are different. There are obstacles where the robot can collide but are not detected by the LIDAR and sonar sensors due to its range limitations and height limitation. These obstacles are inserted in the planning map avoiding the robot goes near them. Also, areas where the robot must not go are neglected in the map.

These tests demonstrates the feasibility of this autonomous shop assistant platform in a real challenging scenario (Figure 5.34(b)). Full video of this tests can be watched on the RobeSafe research group channel [2].

[2]https://www.youtube.com/watch?v=H2ZsfAPaDr0

Chapter 6

Conclusions and Future Work

The main goal of this thesis was the development of an autonomous navigation system for unmanned ground vehicles which could deal with dynamic obstacles.

This has become an increasing necessity in the last years, because autonomous UGVs need to work in uncontrolled environments, assuring its security and the environment.

In this thesis an study of the state of the art navigation algorithms (from local to global ones) has been done. As classic algorithms only take into account static obstacles, an algorithm (CVM) has been extended to work with dynamic obstacles (into two different proposals PCVM, and DCVM). These algorithms have proven its efficiency in real scenarios, increasing the safety of an autonomous robot.

Based on the results obtained from these algorithms, another algorithm (DW4DO) has been proposed in order to increase the robot performance, improving the stability in the robot movement. This algorithm has been extended to a medium time planner algorithm (DW4DOT) in order to solve situations where obstacle avoidance algorithms do not achieve the best solution.

In order to integrate these obstacle avoidance algorithms into a navigation platform, the dynamic obstacle avoidance techniques have been extended to a global route planner (DLP) which can solve scenarios where global information about the whole path of the robot is

needed.

To complete the autonomous movement of a robot, perception and localisation algorithms are needed. Implementation of techniques into the navigation framework has been done. Also, our own robotic platform has been developed from scratch in order to improve the performance of the commercial ones.

This navigation framework has been tested into two real projects: ABSYNTHE and RoboShop, where different robots were used. In these projects the navigation framework has demonstrated is efficiency improving the security of the robot and allowing these platforms to work along humans.

6.1 Main Contributions

- A robotic platform has been developed in order to work as a shop assistant in challenging crowded environments with moving obstacles. The platform has been tested in a real scenario inside the Juguetrónica shop.

- A local mapping algorithm has been proposed in order to store information about the static and dynamic obstacles on the surroundings of the robots, saving memory with respect to the classic approaches. This local mapping allows the development of algorithms (local and global) that can deal with dynamic obstacles.

- An implementation of the static obstacle avoidance algorithm CVM has been done into the ROS framework. This algorithm has been tested, and two extensions have been proposed: PCVM and DCVM which take dynamic obstacles into account, and are able to solve situations where CVM collides with moving obstacles and improving its velocity and stability.

- A dynamic obstacle avoidance algorithm DW4DO has been proposed and implemented in order to improve the flaws of the previous ones. This algorithm increases the stability of the

avoidance manoeuvres, and subsequently, reduces the power consumption of the robot.

- A medium planner algorithm DW4DOT has been proposed in order to solve scenarios where a reactive obstacle avoidance algorithm can not find the best solution, taking into account the dynamic obstacles at the same time.

- Based on the results obtained by the dynamic obstacle avoidance algorithms, a global path planner that can deal with dynamic environments (DLP) has been developed. It has been tested in simulation and real scenarios, where it has been proven that this algorithm can solve situations where obstacle avoidance algorithms fail, especially paths that are blocked for a certain time and the robot needs to perform a stop maneouvre to reach the goal.

- All these implementations have been tested in simulated and real scenarios (Polytechnic School, European Centre for Soft Computing and Juguetrónica Shop). The tests performed show how a navigation that deal with dynamic obstacles improves the human robot collaboration and the security of the robot and environment.

6.2 Future Work

- Publish the proposed navigation system under the ROS platform.

- Extension of the algorithms based on movements primitives (DW4DO, DW4DOT and DLP) to work with robots that have another motion systems different to a differential one, like omnidirectional or humanoid robots.

- Extension of the localisation system proposed, fusing IMU and odometry measures, to achieve the localisation in full 3D environments, allowing the robot to be localised in the presence of ramps or uneven terrains.

- Extension of the obstacle avoidance algorithms for working in 3D environments, in unmanned ground vehicles or even in unmanned aerial ones.
- Improve the route recalculation stage of DLP in order to work with real robots time constraints.
- Extension and adaptation of the full navigation framework to work in autonomous car scenarios, where semantic information of the environment needs to be integrated.
- Extensive test of the proposed navigation system, using the RoboShop shop assistant, in the Juguetrónica shop. Also the system will be tested using the Softbank Robotics Pepper robot.

Appendices

Appendix A

Localisation, Perception and Control Algorithms

A.1 Localisation System

As has been explained in the chapter 3 using only the odometry for long term navigation is not enough due to its accumulative errors. For this reason, using more sensors to improve the tracking of the robot is needed.

An algorithm to fuse odometry data with information obtained from commercial IMU has been presented in our previous work [Pintor et al., 2016].

Following the scheme of Figure A.1 our proposal is to merge the IMU filtered orientation with the robot's odometry in order to improve it.

The IMU orientation can be obtained by the integration of the rotational speeds provided by the gyroscopes (with respect to the origin). But, due to the accumulative error produced by the integration it is necessary to compensate it. A way of correct that orientation measurement is adding information from the accelerometers and/or magnetometers. Some of the most commonly used systems are the complementary filters [Mahony et al., 2005], the Mahony filter [Mahony et al., 2008] and the Madgwick filter [Madgwick, 2010]. In the proposed system a Madwick filter has been used, because it includes

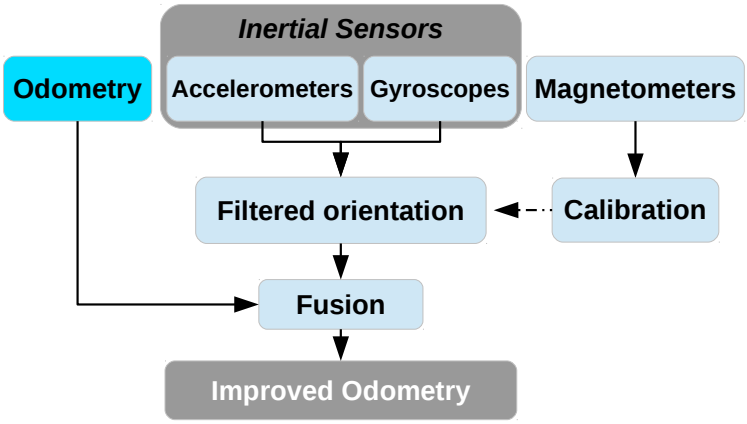


Figure A.1: Localisation System

compensation of the gyroscope bias and magnetic distortion. In case of the addition of the magnetometers the absolute orientation of the IMU can be obtained.

In order to fuse both sensors information, an EKF has been used because the odometry calculus is a non linear system.

The current state of the filter can be calculated using the equation A.1 where $\hat{x}_t$ is the estimation of the current state, x_{t-1} is the state in the previous previous, u_{t-1} is the input of the system, w_{t-1} is the process noise and f is a non linear state transfer function.

$$\hat{x}_t = f(x_{t-1}, u_{t-1}) + w_{t-1} \tag{A.1}$$

Equation A.2 represents the observation model $\hat{z}_t$, where h is the sensor model and v_t is the measurement noise.

$$\hat{z}_t = h(x_t) + v_t \tag{A.2}$$

Assuming that indoor differential robots moves on planar environments, its state can be represented with the equation A.3, where $posx_t$ and $posy_t$ are the coordinates of the robot in the world frame, θ_t is its orientation, V_t is its linear velocity and Ω_t is its angular velocity in the time instant t.

$$\hat{x}_t = [posx_t \;\; posy_t \;\; \theta_t \;\; V_t \;\; \Omega_t]^T \tag{A.3}$$

Robots kinematic can be obtained by the equation A.6, where Δd is the distance increment (obtained by equation A.4 and $\Delta\theta$ is the angular increment (obtained by equation A.5). In these equations Δt represents the time increment between states of the filter.

$$\Delta d = V_{t-1} * \Delta t \tag{A.4}$$

$$\Delta\theta = \Omega_{t-1} * \Delta t \tag{A.5}$$

$$\begin{bmatrix} posx_t \\ posy_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} posx_{t-1} + \Delta d * cos(\theta_{t-1}) \\ posy_{t-1} + \Delta d * sin(\theta_{t-1}) \\ \theta_{t-1} + \Delta\theta \end{bmatrix} \tag{A.6}$$

In order to linearise the system, the state vector Jacobian is used, represented by the transition matrix F (equation A.7).

$$F = \begin{bmatrix} 1 & 0 & -\Delta d * sin(\theta_{t-1}) & \Delta t * cos(\theta_{t-1}) & 0 \\ 0 & 1 & \Delta d * cos(\theta_{t-1}) & \Delta t * \sin(\theta_{t-1}) & 0 \\ 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.7}$$

In this proposal, the EKF implementation available in ROS [Moore and Stouch, 2016] has been used. This implementation used as function h a identity matrix which allows to update partially the filter with data input from different sensors (and even at different frequency).

Even if this filter can represent positions in 3D space, in this proposal only variables to represent the robot in 2D cartesian space are used, neglecting the other variables (cartesian axis z, roll and pitch orientations) to avoid introducing errors in the filter and increasing its complexity. The inputs of the filter are the robot's odometry and the data obtained from the IMU.

Due to the difference of the data type from both sensors, the inputs of the filter are the linear and angular velocities (V_{odom} and Ω_{odom}) obtained from the odometry and the angular velocity (Ω_{imu})

from the IMU. Using only the velocities, it allows the filter to integrate the positions (final odometry) of the robot without the needing of calibration and alignment of the position measurements of both sensors.

In order to test this implementation with commercial platforms, experiments has been carried out in and indoor environment around the second floor of the University of Alcalá Polytechnic School using a *Pioneer 3-AT* robot platform and two different IMU units: one designed for general purpose (*Trivisio Colibri*) and one designed for use in mobile devices (*Invensense MPU6500* accelerometer and gyroscope and a *Yamaha YAS537 magnetometer* embedded in a *Samsung Galaxy S6* mobile phone).

The system has been evaluated in two scenarios, with the robot moving in teleoperation mode, at a maximum velocity of 0.5m/s and 0.3rad/s.

- **Short path travel**: around elevators (which influence the magnetometers measurements) making several turns in a approximately 20m x 15 m surface and a travelled distance of 78 meters.

- **Long path travel**: in a square of 70m x 70m approximately, with a travelled distance around 260 m. The robot passes through several metallic doors that affect to the magnetometer measurements.

To obtain a quality measure of each test, the widely used AMCL as Ground Truth [Thrun et al., 2001]. This system obtains the position of the robot within a map comparing the measures from a LIDAR, the movement model of the robot, and a metrical map of the building.

On each scenario, four different combinations are evaluated (two IMUs and two different algorithms):

- *Od + IMU + EKF*: Fusing the odometry data and all the IMU sensors (accelerometers, gyroscopes and magnetometers).

- *Od + IMU (Mad) + EKF*: Fusing the odometry data with the filtered IMU data, using only accelerometers and gyroscopes to avoid the influence of the magnetometers.

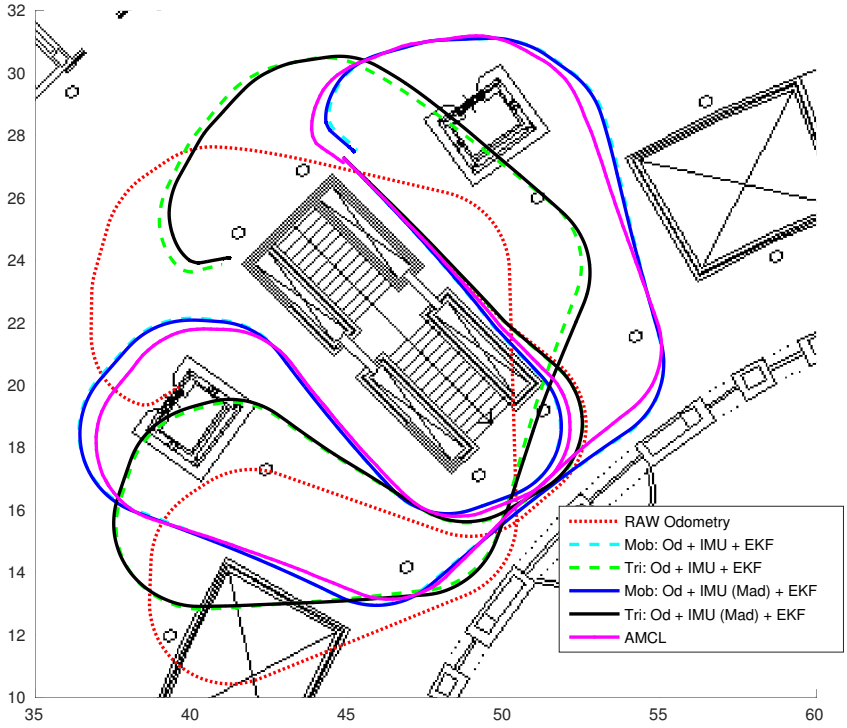


Figure A.2: Localisation: Scenario 1

Figures A.2 and A.3 show the odometry obtained on each scenario and the Tables A.1 and A.2 show the measurement error of each test where *Dist* and *Ang* are the angular error between the final position of each algorithm and the one obtained by the AMCL. Also, the average error with respect the AMCL localisation system is shown during the entire travel, since a better final position does not always means that the path calculated is correct.

Based on the results, it is proven that fusing both sensors improves the odometry of the robot. Comparing the different tests the best location is obtained using the IMU integrated on the Smartphone, which is not intended for this type of use.

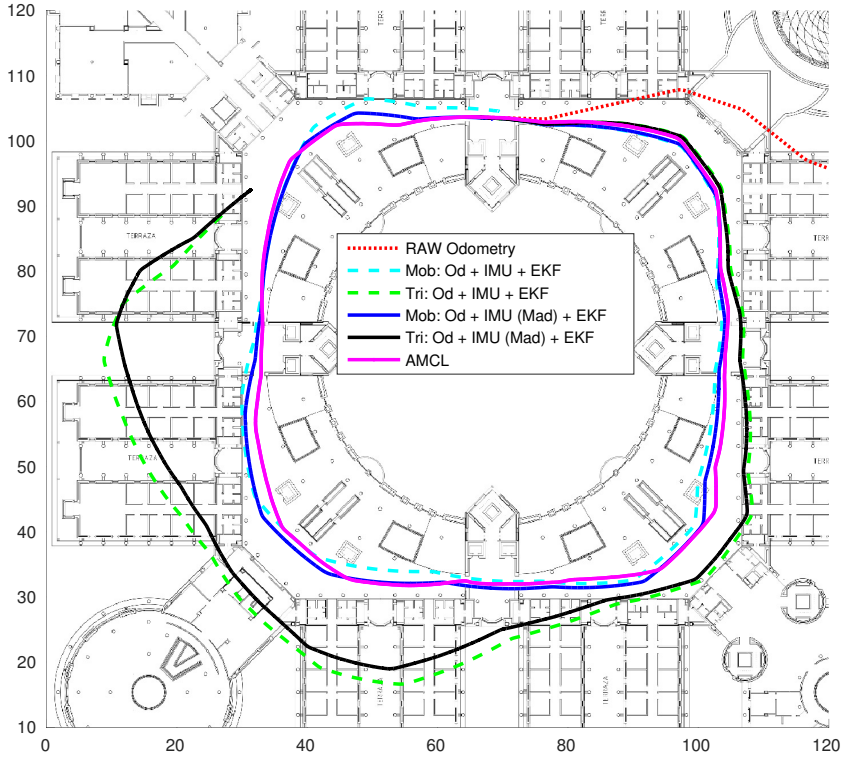


Figure A.3: Localisation: Scenario 2

Algorithm	**Dist(m)**	**Ang(°)**	**Avg(m)**
RAW Odometry	9.01	68	5.22
Mob: Od+IMU+EKF	0.49	-7.08	0.611
Tri: Od+IMU+EKF	5.18	52.42	3.06
Mob: Od+IMU(Mad)+EKF	**0.37**	**-7.02**	**0.54**
Tri: Od+IMU(Mad)+EKF	4.86	48.31	2.85

Table A.1: Localisation: Scenario 1 Results

Comparison between the use of the magnetometer or not, it is proven than in uncontrolled indoor environments, the using of magnetometers induces errors due the drastic magnetic field changes produces by metallic doors or motors (like the elevators engines).

However, using only a tracking localisation system is not enough for global path planning purposes (and therefore, full navigation framework systems) and a localisation technique like AMCL need to

Algorithm	Dist(m)	Ang(°)	Avg(m)
RAW Odometry	222.26	-90.0	111.08
Mob: Od+IMU+EKF	4.87	-8.16	3.95
Tri: Od+IMU+EKF	40.61	43.97	16.08
Mob: Od+IMU(Mad)+EKF	**3.26**	**-4.66**	**2.39**
Tri: Od+IMU(Mad)+EKF	35.61	38.08	13.85

Table A.2: Localisation: Scenario 2 Results

be used. But this algorithm can have several problems: limited field of view from the sensors, symmetric or near symmetric maps or wide open spaces. This kind of scenarios can cause big errors or "jumps" in the localisation system. In order to mitigate that localisation errors in the whole system, in our work presented in [Cano et al., 2015] we proposed using an EKF filter in a similar way than the explained above, using the AMCL position as another input, in a way that the localisation "jumps" of the AMCL are filtered by the odometry. In this work also a migration task between heterogeneous machines is proposed in order to improve the performance of the algorithms.

Figure A.4 shows an experiment using this system. A *Kuka Youbot* runs autonomously to four intermediate goals, performing a square path of 115 meters, and using the filtered AMCL localisation to do that. The localisation algorithm can be executed in two different machines: a robot embedded less powerful computer and a more powerful computer connected via WiFi. In the figure, green lines represent the execution of the localisation system in the powerful machine and red lines represent the execution of the localisation system in the less powerful machine when the more powerful one is not available. With that experiment it is proven that the filtered AMCL localisation is a valid approach and, also, how the migration of tasks to more powerful computers where they are available improves the navigation.

Figure A.4: Localisation: AMCL + EKF

A.2 Perception System

In real scenarios, obtaining the velocities and dynamics of the environment is a hard challenge. In controlled scenarios these velocities can be measured with sensors situated in the environment, especially cameras. But the cost to deploy a system of these characteristics (in money and time) makes it not possible for all the situations.

For that reason, in the real experiments the system used is the DOMap, developed in our previous work [Llamazares et al., 2013] and in Ángel Llamazares' thesis [Llamazares, 2017]. This method is able to detect the dynamics of the environment using a LIDAR sensor that is mounted in the robot.

The scheme of DOMap algorithm is shown in Figure A.5. In order to estimate the movement of the environment the laser measures need to be discretised. In the approach a cell size according with the error of the LIDAR measures is used in order to filter these errors.

Once the laser grid is built, it is treated as an image and computational vision techniques are used to measure the displacement between two consecutive frames of the image. In the implementation, a pyramidal optical flow algorithm is used [Bouguet, 2000] which use an iterative process with different cell sizes to obtain the movement

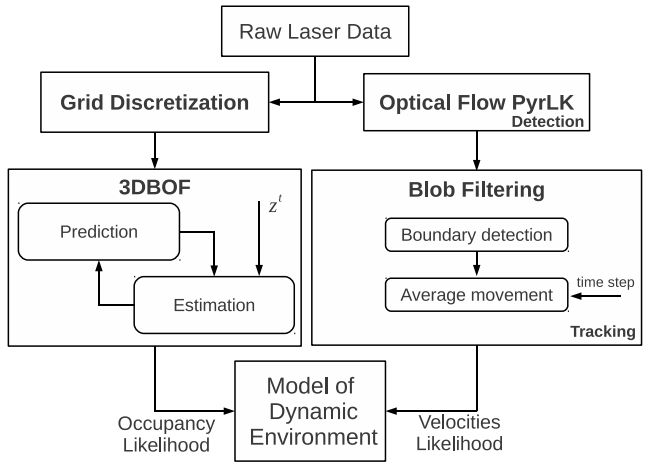


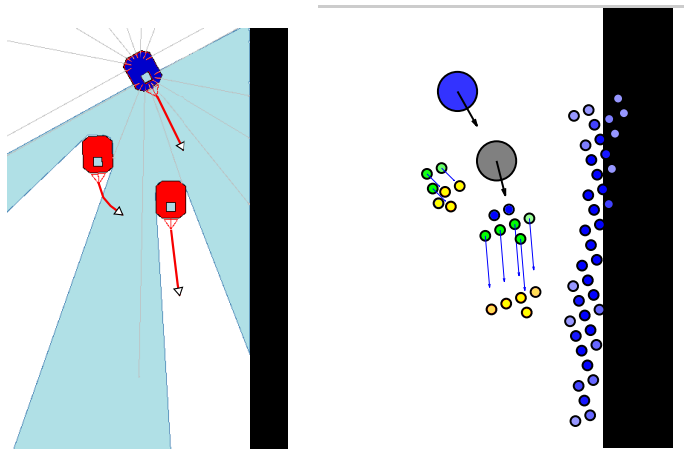
Figure A.5: Perception: DOMap Scheme

of the points detected.

In order to filter the output of the optical flow, a blob filtering stage is performed using opening and close operators. The cells that are joined are considered a unique obstacle and its velocity is the average motion of the laser cells inside each boundary.

Finally the motion calculated for each blob is transferred to each individual point, generating a pointcloud with the LIDAR measures on each instant, and the linear velocities calculated. The algorithms that use DOMap as an input need to consider the velocities constant between iterations.

Figure A.6 shows an example of the velocities prediction. In the Figure A.6(a) the robot (blue polygon) is moving and detect one static obstacle (black wall in the left) and two moving obstacles (movement is represented with red arrows). After executing the algorithm (shown in Figure A.6(b)) each measure of the sensor (small circles) has a linear velocity assigned. Also, the prediction of the robot (grey circle) and the obstacles (yellow circles) are obtained integrating the predicted measures.



(a) Simulated Environment

(b) Sensing Results

Figure A.6: Perception: DOMap example

A.3 Robot's Control

In order to follow a route, a control stage must be developed which take into accounts the time, positions and velocities of each route point. A route is generated by a set of velocity commands than can be directly commanded to a robot. However, if these commands are applied in an open loop control model, any perturbation that occurs may deviate the robot from the calculated route. These perturbations can be timing errors, dynamic error from the motors, computational time error, simulation errors, etc.

The close loop control used is a proportional controller to follow non linear trajectories. The inputs to the control are the next desired position and velocities of the robot on each iteration, and a control law to reach a position with a desired velocity is applied. This control law assumes that the velocity between intervals is constant, which is the same assumption that the DLP does ([Amoozgar and Zhang, 2012]).

The control law is calculated using the following equations:

Three different errors as taken as input to the control law: equation A.8 calculates the distance between the goal point at each iter-

ation (x_g, y_g) and the robot position (x, y). Equation A.9 calculates the angular difference to the goal, where θ is the robot orientation. Equation A.10 calculates the angular difference with respect to the goal angle θ_g.

$$d = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{A.8}$$

$$\alpha = \begin{cases} 0 & if \quad d = 0 \\ atan(\frac{y_g - y}{x_g - x}) - \theta & if \quad d \neq 0 \end{cases} \tag{A.9}$$

$$\epsilon_\theta = \theta_g - \theta \tag{A.10}$$

The velocities that need to be applied to the robot are given by equations A.11 and A.12, where v and ω are the commanded linear and angular velocities respectively, v_g is the linear velocity of the interval, v_{ls} is the linear velocity dependent of the distance to the desired point, v_{md} is the linear velocity desired depending on the goal velocity and orientation and θ_{md} is the angular desired acceleration. $K_{\omega 2}$ is a proportional constant that need to be tuned.

$$v = v_{ls} * cos(\alpha) + v_g * cos(\epsilon_{theta}) \tag{A.11}$$

$$\omega = \dot{\theta}_{md} + v_{md} * (K_{\omega 2} * (v_{ls} * sin(\alpha) + v_g * sin(\epsilon_\theta)) + d * sin(\alpha)) \tag{A.12}$$

The linear velocity dependant of the distance is calculated using the equation A.13, where $kv2$ is a proportional constant that has to be tuned.

$$v_{ls} = kv2 * d \tag{A.13}$$

The linear desired velocity is obtained using the equation A.14 where the linear velocity of the goal, the linear velocity of the proportional controller (v_{ls}) and the angular difference to the goal position and orientation are involved.

$$v_{md} = \sqrt{v_g^2 + v_{ls}^2 + 2 * v_g * vls * cos(\alpha - \epsilon_\theta)} \tag{A.14}$$

The angular desired velocity is obtained using the equation A.15 and the angular desired acceleration is obtained deriving that equation.

$$\theta_{md} = atan(\frac{v_{ls} * sin(\alpha - \epsilon_\theta)}{v_g + v_{ls} * cos(\alpha - \epsilon_\theta)}) + \theta_g \tag{A.15}$$

In order to assure the Lyapunov stability, both constants need to be greater than zero.

Several tests are carried out in order to check the behaviour of the path following control algorithm. The constants of the proportional controller has been adjusted experimentally ($kv2 = 0.8$, $kw2 = 0.1$). Figures A.7 and A.8 show two executions of this algorithm in simulation, where the paths planned by DLP are shown in dashed blue lines and the paths followed by the robot are shown in red lines.

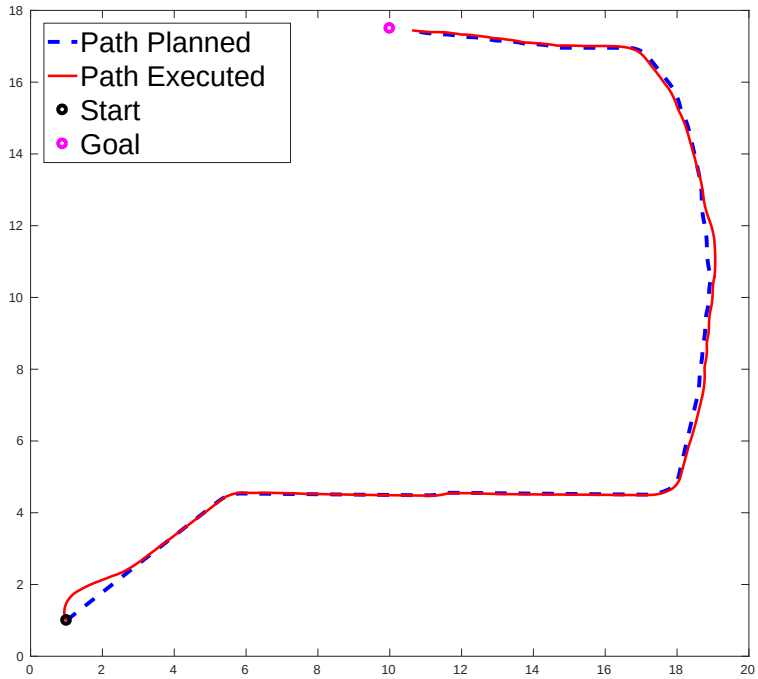


Figure A.7: DLP Path Following Test 1

Table A.3 shows the numerical errors of the path followed. The proposed algorithm is able to follow the path adjusting to it, however

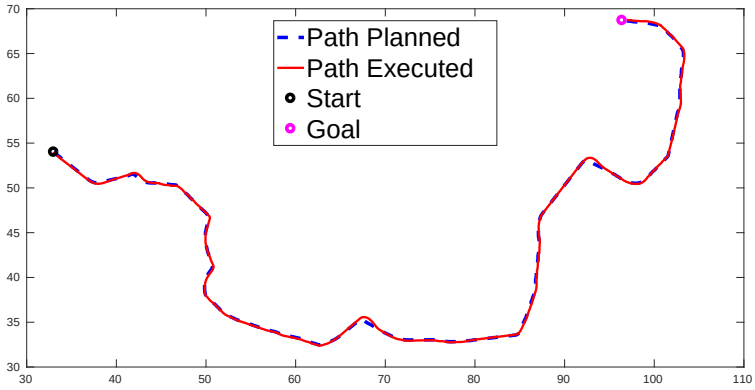


Figure A.8: DLP Path Following Test 2

it have two big issues: as it is a proportional controller its error with respect to the reference is not zero, making the robot usually go near but parallel to the path. Another big issue is when the route planned has a zero velocity (the robot should stop before a blocking obstacle or turns without displacement). As the angular difference with respect the reference is involved in the control equations, when the distance error to the point is near zero (but not zero) the robot moves when it should be stopped.

Experiment	$\overline{\epsilon_d}$ (m)	$\overline{\epsilon_\theta}$ (rad)	$\overline{\epsilon_v}$ (m/s)	$\overline{\epsilon_\omega}$ (rad/s)
Scenario 1	0.066	0.108	0.009	0.124
Scenario 2	0.064	0.065	0.010	0.138

Table A.3: Robot's Control: Numerical Errors

The control algorithm has proven to be able to follow a route planned, where the reference changes constantly during time. However, in real scenarios a simple collision checking algorithm is needed: if the controller gets away from the path and towards an obstacle the robot should stop and plans a new path to the goal.

Appendix B

Publications Derived from this PhD Dissertation

B.1 Journal Publications

2014 **Perception and Navigation in Unknown Environments: The DARPA Robotics Challenge**, *E. Molinos, Á. Llamazares, N. Hernández, R. Arroyo, A. Cela, J.J. Yebes, M. Ocaña, L.M. Bergasa*, Advances in Intelligent Systems and Computing, Vol. 253, Pages 321-329.

2014 **Competing in the DARPA Virtual Robotics Challenge as the SARBOT Team**, *E. Garcia, M. Ocaña, L.M. Bergasa, M. Ferre, M. Abderrahim, J. Arevalo, D. San-Merodio, E. Molinos, N. Hernández, Á. Llamazares, F. Suarez, S. Rodriguez*, Advances in Intelligent Systems and Computing, Vol. 253, Pages 381-396.

2013 **Dynamic Obstacle Avoidance Using Bayesian Occupancy Filter and Approximate Inference**, *Á. Llamazares, V. Ivan, E. Molinos, M. Ocaña, S. vijayakumar*, Sensors (ISSN: 1424-8220), Vol. 13(3), pages 2929-2944.

2013 **Comparison of Local Obstacle Avoidance Algorithms**, *E. Molinos, J. Pozuelo, Á. Llamazares, M. Ocaña, J. López*, Lecture Notes in Computer Science (ISSN: 0302-9743), Vol. 8112.

B.2 Conference Publications

2016 **Mejora de la Odometría de un robot móvil aplicando medidas inerciales**, *R. Pintor, Á. Llamazares, E. Molinos, M. Ocaña*, Seminario anual de automática, electrónica industrial e instrumentación (SAAEI 16).

2015 **Dynamic Process Migration in Heterogeneous ROS-based Environments**, *J. Cano, E. Molinos, V. Nagarajan, S. Vijayakumar*, 17th International Conference on Advanced Robotics (ICAR 2015).

2014 **Dynamic Obstacle Avoidance based on Curvature Arcs**, *E. Molinos, Á. Llamazares, M. Ocaña, F. Herranz*, 2014 IEEE/SICE International Sympoisum on System Integration (IEEE/SICE SII 2014).

2014 **Development of a Navigation System for a Robotic Shop Guide**, *M. Ocaña, Á. Llamazares, E. Molinos, N. Hernández, F. Herranz, P. Revenga, E. López*, XV Workshop of Physical Agents (WAF2014).

2014 **Integrating ABSYNTHE Autonomous Navigation System Into ROS**, *Á. Llamazares, E. Molinos, M. Ocaña, F. Herranz*, Workshop on Modelling, Estimation, Perception and Control of All Terrain Mobile Robots In 2014 IEEE International Conference on Robotics and Automation (IEEE ICRA 2014).

2013 **Proyecto SARBOT: Introducción de robots humanoides en tareas de búsqueda y rescate en entornos urbanos degradados**, *E. Garcia, J. Arévalo, D. Sanz-Merodio, L.M. Bergasa, M. Ocaña, E. Molinos, N. Hernández, Á. Llamazares, M. Abderrahim, S. Rodriguez, M. Ferre, F. Suárez*, Congreso en Defensa y Seguridad (DESEi+d 2013).

B.3 Partially Related Publications

B.3.1 Journal Publications

2017 **Multi-Sensorial SLAM System for Low-Cost Micro Aerial Vehicles in GPS-denied Environments**, *E. López, S. García, R. Barea, L.M. Bergasa, E. Molinos, R. Arroyo, E. Romera, S. Pardo*, Sensors (ISSN: 1424-8220), Vol. 17(4), pages 1-27.

2014 **WiFi SLAM algorithms: an experimental comparison**, *F. Herranz, Á. Llamazares, E. Molinos, M. Ocaña, M. Sotelo*, Robotica (ISSN: 1469-8668).

B.3.2 Conference Publications

2016 **Indoor SLAM for Micro Aerial Vehicles Using Monocular Camera and Sensor Fusion**, *S. García, E. López, R. Barea, L.M. Bergasa, A. Gómez, E. Molinos*, IEEE International Conference on Autonomous Robot Systems and Competitions (IEEE ICARSC 2016).

2015 **Indoor SLAM for Micro Aerial Vehicles Using Visual and Laser Sensor Fusion**, *E. López, R. Barea, A. Gómez, Á. Saltos, L.M. Bergasa, A. Nemra, E. Molinos*, Second Iberian Robotics Conference ROBOT 2015.

2015 **Cooperative Adaptive Cruise Control for a Convoy of Three Pioneer Robots**, *F. Martin, V. Milanés, M. Ocaña, E. Molinos, Á. Llamazares*, Second Iberian Robotics Conference ROBOT 2015.

2014 **A Comparison of SLAM Algorithms with Range Only Sensors**, *F. Herranz, Á. Llamazares, E. Molinos, M. Ocaña*, IEEE International Conference on Robotics and Automation (IEEE ICRA 2014).

Bibliography

[Alonso et al., 2012] J. M. Alonso, K. LeBlanc, M. Ocaña and E. Ruspini, "ABSYNTHE: Abstraction, synthesis, and integration of information for human-robot teams". In *Proceedings of the International Workshop on Perception in Robotics, IEEE Intelligent Vehicles Symposium*, pages P14.1–P14.6 (2012).

[Alsaab and Bicker, 2014] A. Alsaab and R. Bicker, "Improving velocity obstacle approach for obstacle avoidance in indoor environments". In *2014 UKACC International Conference on Control (CONTROL)*, pages 325–330 (2014).

[Amoozgar and Zhang, 2012] M. Amoozgar and Y. Zhang, "Trajectory tracking of wheeled mobile robots: A kinematical approach". In *Mechatronics and Embedded Systems and Applications (MESA), 2012 IEEE/ASME International Conference on*, pages 275–280, IEEE (2012).

[Babinec et al., 2014] A. Babinec, F. Duchoň, M. Dekan, P. Pásztó and M. Kelemen, "Vfh*tdt (vfh* with time dependent tree): A new laser rangefinder based obstacle avoidance method designed for environment with non-static obstacles". *Robotics and Autonomous Systems*, volume 62(8), pages 1098–1115 (2014).

[Bekris et al., 2006] K. E. Bekris, M. Glick and L. E. Kavraki, "Evaluation of algorithms for bearing-only slam". In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1937–1943, IEEE (2006).

[Benzerrouk et al., 2012] A. Benzerrouk, L. Adouane and P. Martinet, "Dynamic obstacle avoidance strategies using limit cycle for the navigation of multi-robot system". In *IEEE/RSJ International Conference on Intelligent Robots and Systems. 4th Workshop on Planning, Perception and Navigation for Intelligent Vehicles* (2012).

[Biswas and Veloso, 2012] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation". In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702, IEEE (2012).

[Borenstein and Feng, 1996] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots". *IEEE Transactions on robotics and automation*, volume 12(6), pages 869–880 (1996).

[Borenstein and Koren, 1989] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots". *Systems, Man and Cybernetics, IEEE Transactions on*, volume 19(5), pages 1179–1187 (1989).

[Borenstein and Koren, 1991] J. Borenstein and Y. Koren, "The vector field histogram and fast obstacle-avoidance for mobile robots". *IEEE Journal of Robotics and Automation*, volume 7, pages 278–288 (1991).

[Bouguet, 2000] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker". *Intel Corporation, Microprocessor Research Labs* (2000).

[Cano et al., 2015] J. Cano, E. Molinos, V. Nagarajan and S. Vijayakumar, "Dynamic process migration in heterogeneous ros-based environments". In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 518–523, IEEE (2015).

[Chakravarthy and Ghose, 1998] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: a collision cone

approach". *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, volume 28(5), pages 562–574 (1998).

[Choi, 2014] J. Choi, "Kinodynamic motion planning for autonomous vehicles". *International Journal of Advanced Robotic Systems*, volume 11(6), page 90 (2014).

[Choset and Burdick, 1996] H. Choset and J. Burdick, "Sensor based motion planning: The hierarchical generalized voronoi graph" (1996).

[Chou et al., 2011] C. C. Chou, F. L. Lian and C. C. Wang, "Characterizing indoor environment for robot navigation using velocity space approach with region analysis and look-ahead verification". *IEEE Transactions on Instrumentation and Measurement*, volume 60(2), pages 442–451 (2011).

[Coué et al., 2006] C. Coué, C. Pradalier, C. Laugier, T. Fraichard and P. Bessiere, "Bayesian Occupancy Filtering for Multitarget Tracking: an Automotive Application". *Int. Journal of Robotics Research*, volume 25(1), pages 19–30 (2006).

[Daniel et al., 2014] K. Daniel, A. Nash, S. Koenig and A. Felner, "Theta*: Any-angle path planning on grids". *CoRR*, volume abs/1401.3843 (2014).

[Dijkstra, 1959] E. W. Dijkstra, "A note on two problems in connexion with graphs". *Numerische Mathematik*, volume 1(1), pages 269–271 (1959).

[Dongshu et al., 2011] W. Dongshu, Z. Yusheng and S. Wenjie, "Behavior-based hierarchical fuzzy control for mobile robot navigation in dynamic environment". In *2011 Chinese Control and Decision Conference (CCDC)*, pages 2419–2424 (2011).

[Durham and Bullo, 2008] J. W. Durham and F. Bullo, "Smooth nearness-diagram navigation". In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 22-26,*

2008, Acropolis Convention Center, Nice, France, pages 690–695, IEEE (2008).

[Faisal et al., 2013] M. Faisal, R. Hedjar, M. A. Sulaiman and K. Al-Mutib, "Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment". *International Journal of Advanced Robotic Systems*, volume 10(1), page 37 (2013).

[Fernández et al., 2004] J. L. Fernández, R. Sanz, J. A. Benayas and A. R. Diéguez, "Improving collision avoidance for mobile robots in partially known environments: the beam curvature method". *Robotics and Autonomous Systems*, volume 46(4), pages 205–219 (2004).

[Fiorini and Shiller, 1993] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm". In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 560–565, IEEE (1993).

[Foley et al., 1994] J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes and R. L. Phillips, *Introduction to computer graphics*, volume 55. Addison-Wesley Reading (1994).

[Fox et al., 1997] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance". *Robotics Automation Magazine, IEEE*, volume 4(1), pages 23 –33 (1997).

[Hansen and Zhou, 2011] E. A. Hansen and R. Zhou, "Anytime heuristic search". *CoRR*, volume abs/1110.2737 (2011).

[Hart et al., 1968] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths". *IEEE Transactions on Systems, Science, and Cybernetics*, volume SSC-4(2), pages 100–107 (1968).

[Hwang and Ahuja, 1992] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning". *Robotics and Automation, IEEE Transactions on*, volume 8(1), pages 23–32 (1992).

[Kamon et al., 1996] I. Kamon, E. Rivlin and E. Rimon, "New range-sensor based globally convergent navigation algorithm for mobile robots". In *Proceedings - IEEE International Conference on Robotics and Automation.*, volume 1, pages 429–435 (1996).

[Kavraki et al., 1996] L. Kavraki, P. Svestka, J.-C. Latombe and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In *IEEE International Conference on Robotics and Automation*, pages 566–580 (1996).

[Khatib, 1986] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots". In *Autonomous robot vehicles*, pages 396–404, Springer (1986).

[Ko et al., 1998] N. Y. Ko, R. Simmons, K. Reid and G. Simmons, "The lane-curvature method for local obstacle avoidance". In *Proceedings 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications, October 13-17, 1998, Victoria, BC, Canada*, pages 1615–1621 (1998).

[Koenig and Likhachev, 2002] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain". In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 968–975, IEEE (2002).

[Korf, 1985] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search". *Artif. Intell.*, volume 27(1), pages 97–109 (1985).

[Kummerle et al., 2013] R. Kummerle, M. Ruhnke, B. Steder, C. Stachniss and W. Burgard, "A navigation system for robots operating in crowded urban environments". In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 3225–3232 (2013).

[Kushleyev and Likhachev, 2009] A. Kushleyev and M. Likhachev, "Time-bounded lattice for efficient planning in dynamic envi-

ronments". In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1662–1668, IEEE (2009).

[Latombe, 1991a] J.-C. Latombe, *Approximate Cell Decomposition*, pages 248–294. Springer US, Boston, MA (1991a).

[Latombe, 1991b] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA (1991b).

[LaValle and James J. Kuffner, 2001] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning". *The International Journal of Robotics Research*, volume 20(5), pages 378–400 (2001).

[Lee et al., 2012] D. Y. Lee, Y. F. Lu, T. K. Kang, I. H. Choi and M. T. Lim, "3d vision based local obstacle avoidance method for humanoid robot". In *2012 12th International Conference on Control, Automation and Systems*, pages 473–475 (2012).

[Lengyel et al., 1990] J. Lengyel, M. Reichert, B. R. Donald and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware". *SIGGRAPH Comput. Graph.*, volume 24(4), pages 327–335 (1990).

[Liang et al., 2011] Y. Liang, L. Xu, R. Wei, B. Zhu and H. Hu, "Behavior-based fuzzy control for indoor cleaning robot obstacle avoidance under dynamic environment". In *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 637–640 (2011).

[Llamazares, 2017] A. Llamazares, *Laser-Based Detection and Tracking of Moving Obstacles to Improve Perception of Unmanned Ground Vehicles*. Ph.D. thesis, University of Alcalá (2017).

[Llamazares et al., 2013] A. Llamazares, V. Ivan, E. Molinos, M. Ocaña and S. Vijayakumar, "Dynamic obstacle avoidance using bayesian occupancy filter and approximate inference". *Sensors*, volume 13(3), pages 2929–2944 (2013).

[Llamazares et al., 2014] Á. Llamazares, E. Molinos, M. Ocaña and F. Herranz, "Integrating absynthe autonomous navigation system into ros". In *Internacional Conference on Robotics and Automation (ICRA 2014), IEEE* (2014).

[López, 2004] E. López, *Sistemas de navegación global basado en procesos de decisión de Markov parcialmente observables. Aplicación a un robot de asistencia personal*. Ph.D. thesis, University of Alcalá (2004).

[López et al., 2005] E. López, L. M. Bergasa, R. Barea and M. S. Escudero, "A navigation system for assistant robots using visually augmented pomdps". *Autonomous Robots*, volume 19(1), pages 67–87 (2005).

[Lumelsky and Stepanov, 1987] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape". volume 2(4), pages 403–430 (1987), special issue on robotics.

[Madgwick, 2010] S. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays". *Report x-io and University of Bristol (UK)*, volume 25 (2010).

[Mahony et al., 2005] R. Mahony, T. Hamel and J. M. Pflimlin, "Complementary filter design on the special orthogonal group so(3)". In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1477–1484 (2005).

[Mahony et al., 2008] R. Mahony, T. Hamel and J. M. Pflimlin, "Nonlinear complementary filters on the special orthogonal group". *IEEE Transactions on Automatic Control*, volume 53(5), pages 1203–1218 (2008).

[Masehian and Katebi, 2014] E. Masehian and Y. Katebi, "Sensor-based motion planning of wheeled mobile robots in unknown dynamic environments". *Journal of Intelligent & Robotic Systems*, volume 74(3), pages 893–914 (2014).

[Maxim Likhachev and Thrun, 2004] G. G. J. Maxim Likhachev and S. Thrun, "Ara : Anytime a with provable bounds on sub-optimality". In *Advances in Neural Information Processing Systems 16*, pages 767–774, MIT Press (2004).

[Mbede et al., 2012] J. B. Mbede, A. Melingui, B. E. Zobo, R. Merzouki and B. O. Bouamama, "zslices based type-2 fuzzy motion control for autonomous robotino mobile robot". In *Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pages 63–68 (2012).

[Mínguez and Montano, 2004] J. Mínguez and L. Montano, "Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios". *IEEE Transactions on Robotics and Automation 20(1)*, pages 45–59 (2004).

[Mohanty and Parhi, 2012] P. K. Mohanty and D. R. Parhi, "Path generation and obstacle avoidance of an autonomous mobile robot using intelligent hybrid controller". In *Proceedings of the Third International Conference on Swarm, Evolutionary, and Memetic Computing*, SEMCCO'12, pages 240–247, Springer-Verlag, Berlin, Heidelberg (2012).

[Molinos, 2013] E. Molinos, "Comparativa de algoritmos de evitación de obstáculos". In *Master Bachellor Thesis, Universidad de Alcalá* (2013).

[Molinos et al., 2014] E. Molinos, Á. Llamazares, M. Ocaña and F. Herranz, "Dynamic obstacle avoidance based on curvature arcs". In *2014 IEEE/SICE International Symposium on System Integration*, pages 186–191 (2014).

[Molinos et al., 2013] E. Molinos, J. Pozuelo, A. Llamazares, M. Ocaña and J. López, *Comparison of Local Obstacle Avoidance Algorithms*, pages 39–46. Springer Berlin Heidelberg, Berlin, Heidelberg (2013).

[Montemerlo et al., 2009] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Hähnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt and S. Thrun, "Junior: The stanford entry in the urban challenge." In M. Buehler, K. Iagnemma and S. Singh (Editors), *The DARPA Urban Challenge*, volume 56 of *Springer Tracts in Advanced Robotics*, pages 91–123, Springer (2009).

[Moore and Stouch, 2016] T. Moore and D. Stouch, *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*, pages 335–348. Springer International Publishing, Cham (2016).

[Murphy et al., 1999] R. R. Murphy, K. Hughes, A. Marzilli and E. Noll, "Integrating explicit path planning with reactive control of mobile robots using trulla". *Robotics and Autonomous Systems*, volume 27(4), pages 225–245 (1999).

[Newman et al., 2006] P. Newman, D. Cole and K. Ho, "Outdoor slam using visual appearance and laser ranging". In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1180–1187, IEEE (2006).

[Nilsson, 1984] N. J. Nilsson, "Shakey the robot". Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025 (1984).

[Ocaña, 2005] M. Ocaña, *Sistema de localización global WiFi aplicado a la navegación de un robot semiautónomo*. Ph.D. thesis, University of Alcalá (2005).

[Ocaña et al., 2005] M. Ocaña, L. M. Bergasa, M. A. Sotelo and R. Flores, "Indoor robot navigation using a pomdp based on wifi and ultrasound observations". In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2592–2597 (2005).

[Ocaña et al., 2014] M. Ocaña, Á. Llamazares, E. Molinos, N. Hernández, F. Herranz, P. Revenga and E. López, "Development of a Navigation System for a Robotic Shop Guide". In U. de León (Editor), *XV Workshop of Physical Agents*, pages 169–177 (2014).

[Pepy et al., 2006] R. Pepy, A. Lambert and H. Mounier, "Path planning using a dynamic vehicle model". In *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, volume 1, pages 781–786, IEEE (2006).

[Phillips and Likhachev, 2011] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments". In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5628–5635, IEEE (2011).

[Pintor et al., 2016] R. Pintor, Á. Llamazares, E. Molinos and M. Ocaña, "Mejora de la odometría de un robot móvil aplicando medidas inerciales". In *Seminario anual de automática, electrónica industrial e instrumentación SAAEI 16, Libro de Resumenes* (2016).

[Pivtoraiko and Kelly, 2005] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces". In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, pages 3231 – 3237 (2005).

[Quigley et al., 2009] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, "Ros: an open-source robot operating system". In *ICRA workshop on open source software*, volume 3, page 5, Kobe (2009).

[Samet, 1988] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures". In *Theoretical Foundations of Computer Graphics and CAD*, pages 51–68, Springer (1988).

[Seder and Petrovic, 2007] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the

presence of moving obstacles". In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1986–1991, IEEE (2007).

[Simmons, 1996] R. Simmons, "The curvature-velocity method for local obstacle avoidance". In *International Conference on Robotics and Automation* (1996).

[Stentz , 1994] A. T. Stentz , "Optimal and efficient path planning for partially-known environments". In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 4, pages 3310 – 3317 (1994).

[Stentz, 1995] A. Stentz, "The focussed d* algorithm for real-time replanning". In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1652–1659, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995).

[Stentz, 2002] A. Stentz, "Constrained dynamic route planning for unmanned ground vehicles". In *Proceedings of the 23rd Army Science Conference* (2002).

[Tang et al., 2013] S. H. Tang, C. Ang, D. Nakhaeinia, B. Karasfi and O. Motlagh, "A reactive collision avoidance approach for mobile robot in dynamic environments". *Journal of Automation and Control Engineering*, volume 1(1), pages 16–20 (2013).

[Thrun, 1998] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation". *Artificial Intelligence*, volume 99(1), pages 21–71 (1998).

[Thrun et al., 2001] S. Thrun, D. Fox, W. Burgard and F. Dellaert, "Robust monte carlo localization for mobile robots". *Artificial intelligence*, volume 128(1-2), pages 99–141 (2001).

[Thrun et al., 2006] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann et al., "Stanley: The robot that won the darpa

grand challenge". *Journal of field Robotics*, volume 23(9), pages 661–692 (2006).

[Ulrich and Borenstein, 1998] I. Ulrich and J. Borenstein, "VFH+: reliable obstacle avoidance for fast mobile robots". In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1572 –1577 vol.2 (1998).

[Ulrich and Borenstein, 2000] I. Ulrich and J. Borenstein, "Vfh*: Local obstacle avoidance with look-ahead verification". In *ICRA*, pages 2505–2511 (2000).

[Winston, 1992] P. H. Winston, *Artificial Intelligence (3rd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1992).

[Yaghmaie et al., 2013] F. A. Yaghmaie, A. Mobarhani and H. D. Taghirad, "A new method for mobile robot navigation in dynamic environment: Escaping algorithm". In *2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, pages 212–217 (2013).

[Yahja et al., 1998] A. Yahja, A. T. Stentz , S. Singh and B. Brummit, "Framed-quadtree path planning for mobile robots operating in sparse environments". In *In Proceedings, IEEE Conference on Robotics and Automation, (ICRA)*, Leuven, Belgium (1998).

[Yap et al., 2011] P. Yap, N. Burch, R. C. Holte and J. Schaeffer, "Block a*: Database-driven search with applications in any-angle path-planning." In *AAAI* (2011).

[Zhong et al., 2011] X. Zhong, X. Peng and J. Zhou, "Dynamic collision avoidance of mobile robot based on velocity obstacles". In *Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*, pages 2410–2413 (2011).