industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Autonomous Navigation Algorithms for Mobile Robots based on LiDAR Technologies

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

Autor: **Leanne Rebecca Miller**
Director: Pedro Javier Navarro Lorente
Codirector: Carlos Fernández Andrés

Cartagena, Diciembre de 2017

Universidad Politécnica de Cartagena

Abstract

In recent years, the use of mobile robots for performing different tasks is increasing rapidly. It is now common to find robotic systems in industrial environments, military applications, agricultural systems and even in businesses carrying out increasingly sophisticated tasks. The development in the mobile robotics field has moved on from research in universities and businesses to use in everyday environments. The advances made in mobile robotics have been transferred to other fields, such as autonomous driving or space exploration. The obstacle avoidance algorithms, local and global path planning techniques and perception systems developed in mobile robotics are used for the new self-driving vehicles.

This dissertation presents a navigation system for a mobile robot with a differential steering system, which combines different path planning and obstacle avoidance algorithms. The algorithms implemented are combined with a localization system using different sensors installed on the robot (encoders, LiDAR, IMUs, etc). The main algorithms used in this project are: The A* algorithm, the Vector Field Histogram and dead reckoning.

Resumen

En los últimos años, el uso de los robots móviles para realizar diferentes tareas ha crecido de forma exponencial. Es habitual encontrar sistemas robotizados en ambientes industriales, aplicaciones militares, sistemas agrícolas e incluso en empresas realizado tareas cada vez más sofisticadas. Los desarrollos en robótica móvil han pasado del campo de la investigación en Universidades y empresas al entorno cotidiano. Los avances en robótica móvil han sido transferidos a otros campos como la conducción autónoma o la exploración espacial. Los algoritmos de evitación de obstáculos, los planificadores locales y globales de trayectorias y los sistemas de percepción desarrollados en la robótica móvil, son utilizados en los nuevos vehículos sin conductor.

Este trabajo final de grado presenta un sistema de navegación para un robot móvil de tipo diferencial, que combina distintos algoritmos de creación de trayectorias y evitación de obstáculos. Los algoritmos implementados se combinan con un sistema de localización utilizando diferentes sensores instalados en el robot (encoders, LiDAR, IMUs, etc.). Los principales algoritmos usados son: El algoritmo A*, el Vector Field Histogram y la navegación por estimación (dead reckoning).

Contents

List of Figures

CHAPTER 1

OBJECTIVES OF THE PROJECT

1.1 Motivation

In recent years, the number of automobile companies and universities researching into autonomous vehicles has increased rapidly. In order to completely achieve autonomous driving, various sub-systems have to be combined, such as perception systems, a localization system and navigation and trajectory planning systems. It is very important that an autonomous vehicle or robot knows where it is situated so it can carry out path planning more effectively with less error. In addition to the algorithms used, the accuracy of localization depends on the precision of the sensors used.

The motivation of this project is to acquire knowledge in the field of autonomous navigation systems and the algorithms and localization methods used in mobile robotics. In order to achieve this, problems of increasing difficulty related to mobile robots will be solved. These missions will be carried out in different environments designed on a simulator where a mobile robot will have to avoid obstacles, create trajectories, calculate its position and reach a target. In the future, the software developed could be adapted for use on an autonomous vehicle (Figure 1.1).

Figure 1.1 Google's self-driving car with LiDAR sensor

Autonomous vehicles and mobile robots can either be controlled by teleoperation or they can be programmed to perform operations autonomously. In this project, a navigation system has been designed for a small mobile robot so that it can operate autonomously in an environment, without the need for human intervention. The system involves driving from an initial position to a goal whilst avoiding obstacles and choosing the best trajectory.

The mobile robot navigation problem was defined in 1991 by Leonard and Durrant-Whyte [1] and consists in solving the following three questions: “Where am I?”, “Where am I going?” and “How do I get there?” from the robot’s point of view. The first question requires the mobile robot localization problem to be solved, while the second and third in addition are related to path planning and motion.

To solve the localization problem adequately, information from different sensors, such as encoders and inertial sensors, must be combined in what is known as sensor fusion. This information alone is not enough and to achieve accurate results must be analysed and corrected using a filter. The most common filter is called the Kalman filter which used a statistical approach.

The path planning and motion problem involves creating a trajectory from the starting position of the robot to a goal. This is known as global navigation and this must be combined with local navigation to be more effective. Local navigation is the avoidance of obstacles that were unknown when programming the original path. If unexpected obstacles appear in the way of the robot, the robot must avoid them and still head towards the final goal.

1.2 Objectives of the Project

The main objective of this project is the study and development of navigation, localization and obstacle avoidance algorithms for a mobile robot. To detect the environment and obstacles, a LiDAR (Light Detection and Ranging) sensor will be used as the main perception system. Other sensors that will be used include the robot encoders and an inertial measurement unit.

The project has been divided into six goals which are the following:

- Analysis of the state of the art in mobile robotics and navigation. Research about the current path planning and goal seeking algorithms and perception methods.

- Develop a navigation application including a user interface and obstacle avoidance algorithms that allow a robot to drive autonomously to a position defined by the user.

- Program a localization system using inertial sensors and encoders. In order to successfully carry out the path planning algorithms, it is important that the robot has a good localization system and knows where it is situated. The data obtained from the different sensors should be combined using a suitable method.

- Carry out tests using a mobile robotics simulator. Create an appropriate simulated environment with realistic features such as walls and doors.

- Perform tests and compare the results obtained with the mobile robot to those obtained with the robotics simulator. Check that the objectives have been met analyse the final results.

1.3 Structure of the Project

Chapter 1. Introduction.

In this chapter, the motivation for carrying out this project will be defined, along with the objectives of the project.

Chapter 2. State of the Art in Mobile Robotics.

A study of the current state of the art of the mobile robot industry will be carried out. The autonomous mobile robot navigation problem will be explained. The different subsystems that make up an autonomous navigation system will be studied: Perception, navigation and kinematics of a differential drive robot will be discussed.

Chapter 3. Navigation Algorithms.

The most common and well known global and local navigation algorithms currently used for autonomous mobile robots will be explained in detail.

Chapter 4. Implementation using a Robotics Simulator.

In this chapter, the software, the different simulated environments designed and the sensors used will be described. The functions and algorithms used to develop the software will be explained.

Chapter 5. Simulated Tests and Results.

The tests carried out with the robotics simulator will be described and the results obtained in different conditions and at different speeds will be compared. The software will also be compared to an existing navigation program.

Chapter 6. Conclusions and Future Projects.

The conclusions drawn from the project and the original objectives will be discussed. Future projects will be proposed.

CHAPTER 2

STATE OF THE ART IN MOBILE ROBOTICS

2.1 Mobile Robotics Industry

Since robots were introduced in factories in the early 1960s, they have evolved rapidly and are now capable of performing complicated and tedious tasks along the assembly line with a very high precision. However, this type of manipulator robots has one important disadvantage which is the lack of mobility. Unlike fixed base industrial robot arms, mobile robots can operate in large areas and explore environments that could be potentially dangerous to humans.

Mobile robots usually move by means of a set of wheels or articulated legs. Legged robots require greater mechanical complexity due to the fact higher degrees of freedom are needed compared to wheeled robots. In addition, on flat ground wheeled robots are generally more stable since all wheels are designed to be in contact with the ground [2].

Existing mobile robots can be separated into three main groups: Commercial robots, military robots and robots designed for research.

2.1.1 Autonomous Commercial Robots

Apart from working in hostile environments, it is also becoming more common to see robots working alongside humans, performing heavy and tedious tasks. On factory floors, Automated Guided Vehicles (AGVs) are being replaced by Autonomous Mobile Robots [3], with the latter capable of avoiding obstacles and redirecting their paths if necessary, unlike the AGVs which are stuck on a predetermined track.

An example of this type of commercial robot are Fetch Robotics' autonomous freight robots (Figure 2.1) which are designed to replace forklifts and trolleys in warehouses and can transport up to 1500kg of payload. These transporting robots navigate around the warehouses using LiDAR sensors and a RGB camera on the front [4].

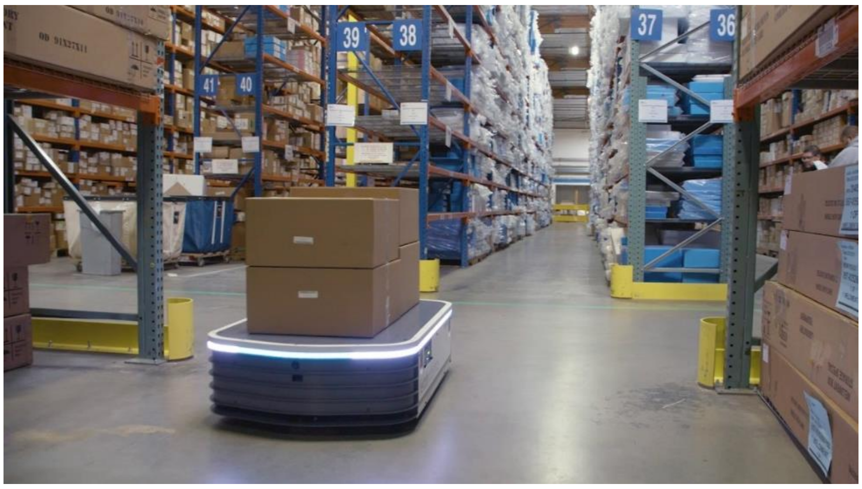


Figure 2.1 Fetch Robotics' Freight 500 Robot

Another example of an autonomous robot that works alongside humans is the RoboCourier from Swisslog (Figure 2.2). This robot is used for light weight hospital material transport and has automatic interfaces for doors and lifts and allows the safe transport of laboratory specimens and supplies [5]. The robot uses LiDAR laser sensors for navigation and navigates autonomously through busy hallways avoiding obstacles.

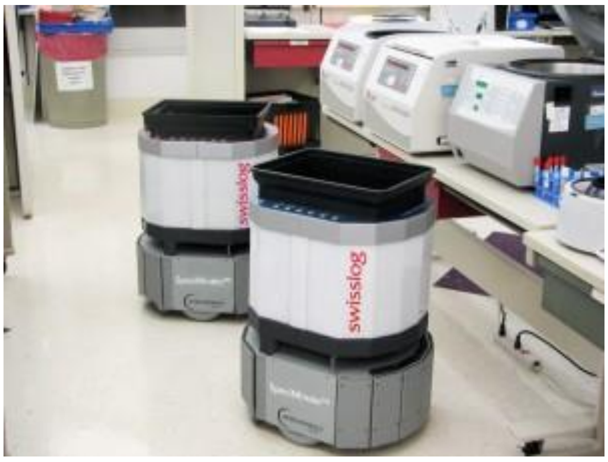


Figure 2.2 RoboCourier transporting robot

In recent years, a large variety of domestic robots have also become available to consumers, one of the most common is the autonomous vacuum cleaner. These robots vary greatly in cleaning capacity and price, with some having over five different cleaning modes. Some of the more complex vacuum robots can be programmed to clean in function of the size of the room or the amount or type of dirt to be cleaned and are even capable of mapping their environment and the surface area to be covered.

The iRobot Roomba (Figure 2.3 (Left)) is one of the most advanced models of vacuum robot and uses Simultaneous Localization and Mapping (SLAM) so it knows where it has already been and which areas still need to be cleaned [6]. These robots use infrared sensors to detect the surface underneath them to avoid falling off stairs, and have bumpers to detect obstacles. Infrared sensors are also used for close wall following, this way the robot reduces the amount of times it bumps into furniture, etc. There are many varieties of cleaning robot, including smaller desktop robots (Figure 2.3 (Right)).

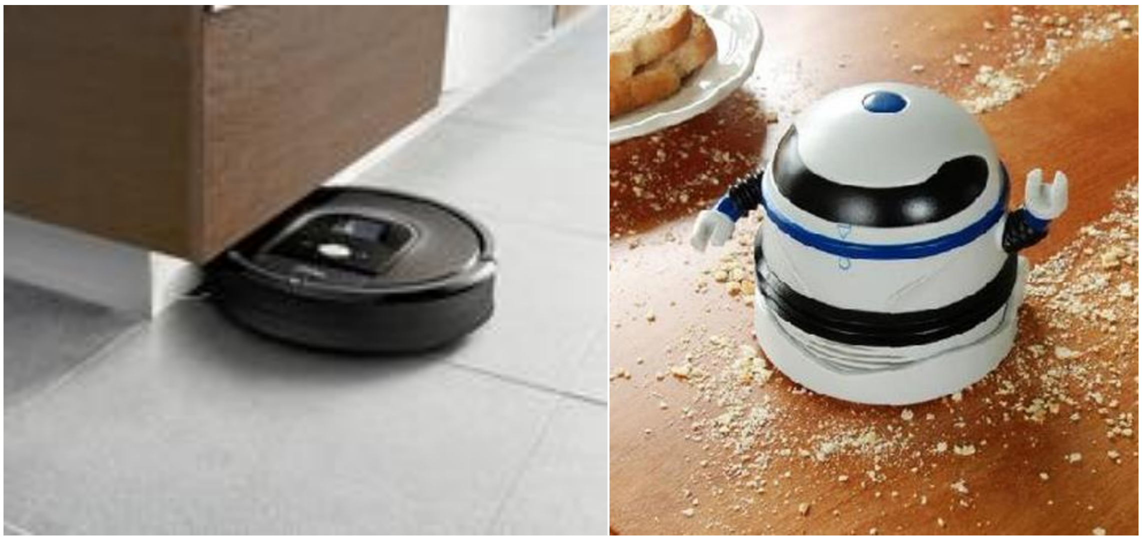


Figure 2.3. iRobot Roomba vacuum robot (Left), Desktop Robot cleaner (Right)

2.1.2 Military Mobile Robots

As robots are becoming more sophisticated and reliable, they are increasingly being used for military applications. Robots are starting to play an important role in patrolling and can perform dangerous missions and deal with explosives [7]. Military robots are different to industrial mobile robots as they are normally teleoperated and do not often carry out missions autonomously. This is usually for safety reasons and the robots are under the complete control of the operator.

Mobile military robots must also be able to operate on many types of terrain, which in many cases can be harsh and dangerous, and this will mean more restrictions when designing the robot. Data security is another vital requirement for military operations. The wireless data transmission between the robot and operator must be secured both ways so that there can be no interference with the data and images sent to and from the robot.

One of the most widely used military robots in the USA is TALON [8]. TALON is an unmanned, tracked military robot developed in the United States, designed to protect soldiers against explosive threats (Figure 2.4). The robot can transport heavy loads, perform explosive disposal missions, mine detection and rescue missions. TALON can be operated from up to one kilometre away and can work in contaminated areas for extended periods of time. The robot features a manipulator arm which can rotate 360˚ and has a microphone and loudspeaker. The robot can be equipped with a diverse range of sensors for gas, radiation and chemical detection as well as an extra rotating shoulder for heavy lifting. Additional firing circuits and a range of weapons such as shotguns can be installed.

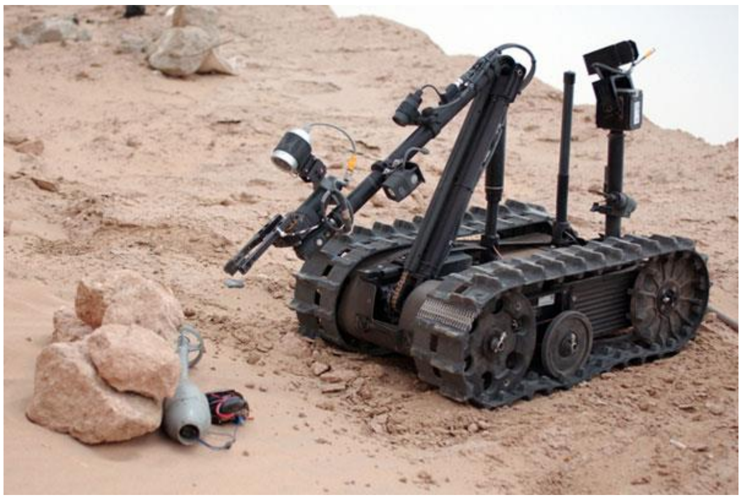


Figure 2.4 TALON Tracked Military Robot, USA

Another important military robot is General Robotics' DOGO [9]. The DOGO is a tactical combat robot armed with a 9mm pistol and provides live video footage using an omnidirectional vision system with eight video cameras (Figure 2.5). The robot has an interface which allows the operator to aim the weapon by simply touching the target on the screen. DOGO can operate on rough terrain and climb stairs, making it suitable for urban and rural warfare.

Figure 2.5 DOGO Tactical combat robot

2.1.3 Mobile Robots used for Research

Mobile robots have a broad range of applications, but to achieve the successful design of a mobile robot, many different areas of knowledge must be combined. To solve the localization and navigation problems a good understanding of electronics, sensors, programming and computer algorithms is necessary. For the locomotion, it is also important to have knowledge of kinematics, dynamics and mechanics.

Research on localization and navigation tasks for robots can be carried out using standard robot platforms designed for use in laboratory environments. These robots usually come with a few basic sensors such as ultrasonic sensors and wheel encoders and can be customized by adding LiDAR sensors, cameras or other mechanisms.

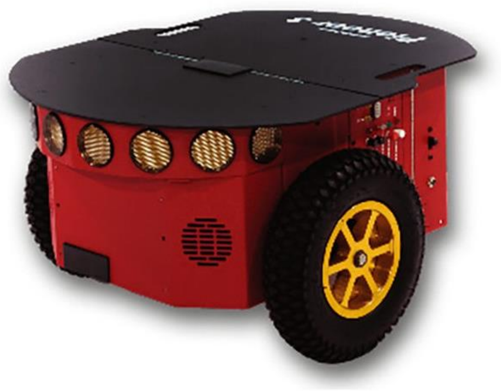


Figure 2.6. Pioneer P3-DX

One of the most popular research robots is the Pioneer P3-DX (Figure 2.6). Pioneer P3-DX is a differential drive robot with 16 ultrasonic sensors and an embedded motion controller. Many different sensors and actuators can be added, making this robot very versatile.

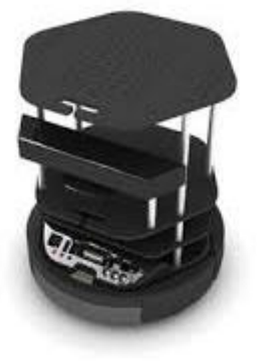


Figure 2.7 TurtleBot

TurtleBot is an open source robot designed for use in research and education [10]. With its basic components, users can carry out Simultaneous Localization and Mapping (SLAM) and follow a person's legs as they walk. TurtleBot uses the open source Robot Operating System platform and different sensors and actuators can be added to the basic platform which consists of a Kinect sensor (Figure 2.7).

2.2 Robotic Subsystems

Autonomous mobile robots combine different subsystems to perform the actions defined by the operator. Each subsystem has a certain task to carry out and the way these subsystems interact with each other defines the behaviour of an autonomous robot.

According to Siegwart [2], in order to accomplish autonomous navigation, four subsystems need to be successfully implemented:

- **Perception:** Consists in the robot extracting meaningful data about its environment from the sensors. Usually the information acquired from more than one type of sensor is used to reduce error, the readings from the different sensors can be combined using sensor fusion algorithms.

- **Localization:** The robot must know its relative position in the environment. This is one of the hardest parts of robot navigation due to sensor inaccuracies. The localization problem involves more than just determining the absolute and relative pose of the robot, map building and landmark extraction are also important components of this subsystem.

- **Cognition:** Also known as path planning, the robot must decide how to reach its goal, choosing a suitable trajectory and avoiding obstacles.

- **Motion Control:** The path calculated previously must be converted into velocities or the required motor input, so that the robot can carry out the planned action and reach the desired goal.

2.3 Perception and Localization

Perception is the ability of the robot to detect the environment that it is in and the obstacles in its path. The precision of the robot localization problem depends on the capacity of the sensors available on the robot and the accuracy of the algorithms used to process the raw data. The different sensors that can be used to obtain the posture of the robot are: wheel encoders, LiDAR and time of flight (TOF) sensors such as 2D and 3D laser scanners and ultrasonic sensors, inertial measurement sensors, artificial vision (TOF and artificial vision cameras), and GPS.

However, to obtain a precise information about the robot's environment it is necessary to combine these perception methods. Although GPS systems are good for outdoor robots, it is still necessary to have an alternative localization method, as GPS is no good in tunnels or indoor environments such as underground carparks. It is also necessary to combine artificial vision systems with other methods of positioning as cameras can be very sensitive to different light levels, and are especially sensitive to direct sunlight.

Robot localization is the ability of the robot to determine its position in the environment and is part of the perception problem. The position of the robot is very important as to carry out the assigned tasks correctly, the robot must know exactly where it is. The robot relies on the information obtained from the sensors to calculate its position, in outdoor environments, GPS is normally used, meanwhile for indoor situations other types of sensors must be available. The accuracy of localization does not only depend on the algorithms used, but also on the types of sensors and their precision, the type of surface that the robot is on, and the type of objects, as the sensor readings can vary depending on the material of the objects.

The sensors used on mobile robots can be classified into two groups:

1. Proprioceptive sensors, which measure the internal magnitudes of the robot, such as the motor speed or battery voltage.
2. Exteroceptive sensors, which acquire information about the robot's surroundings, such as the distance to obstacles.

2.3.1 Odometry

Encoders are proprioceptive sensors that measure the wheel rotation while the robot is moving. Optical encoders are the most commonly used type of odometry sensor and use a photodetector to sense the reflection or interruption of a beam of light on a spinning disk attached to a motor.

With the physical dimensions of the robot, it is possible to calculate the forward and angular velocities of the robot from the angular velocities of each wheel. The distance travelled by the robot, and therefore its position can be obtained by integrating the forward and angular velocities calculated previously.

2.3.2 Ranging Sensors

Ranging sensors are used to measure the proximity of features in the environment to the robot without physical contact with the objects. A signal (sound or light) is emitted by the sensor and the time taken for signal to be reflected back to the sensor is converted into a distance. This is known as time-of-flight ranging. The accuracy of the time-of-flight sensors depends mainly on the speed of the robot, the material of the objects in the environment, the variation in the propagation speed and inaccuracies in the arrival time of the return signal.

Early localization systems used sonar sensors, however 2D and 3D laser scanners are now more commonly used as they are faster and more reliable. For indoor localization, 2D scanners are normally used, due to the fact that 3D scanners are larger and more difficult to install on smaller indoor robots.

2.3.2.1 Ultrasonic Sensors

Ultrasonic sensors calculate the distance to objects by emitting a sound wave at a specific frequency and measuring the time it takes for the wave to bounce back. Ultrasonic sensors are one of the most affordable ranging sensors and were widely used before the 2D laser scanner became available [11], however they have some important drawbacks. Sound propagates in a cone-like manner, with an opening angle of between 20 and 40 degrees (Figure 2.8), which means that instead of getting precise directional information, the sensor only tells us that there is an object in this area.

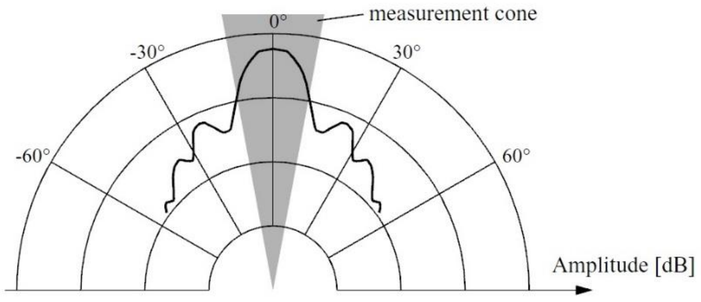


Figure 2.8 Ultrasonic sensor range (Siegwart)

Another important disadvantage of ultrasonic sensors is they can give bad readings with some materials if the sound wave is absorbed instead of reflected. The waves can also be reflected wide of the receiver leaving the object undetected.

2.3.2.2 Laser Scanners (LiDAR)

Laser scanners have significant improvements compared to ultrasonic sensors. These sensors emit beams of laser light instead of sound. In addition to giving very precise distance measurements, the directional information acquired is also much more reliable.

On mobile robots, the most common LiDAR sensors are 2D laser scanners, because 3D laser scanners are usually too large and heavy. The biggest problem with 2D scanners is that they only detect the environment on a flat plane at the level of the sensor, so for some applications this is not sufficient. However, there are some smaller models of 3D LiDAR scanners such as the Velodyne VLP-16 (Figure 2.9) which are suitable for smaller mobile robots and provide a complete view of the environment where the robot is operating.

Figure 2.9 Velodyne VLP-16 (Left) and Velodyne HDL-64E (Right)

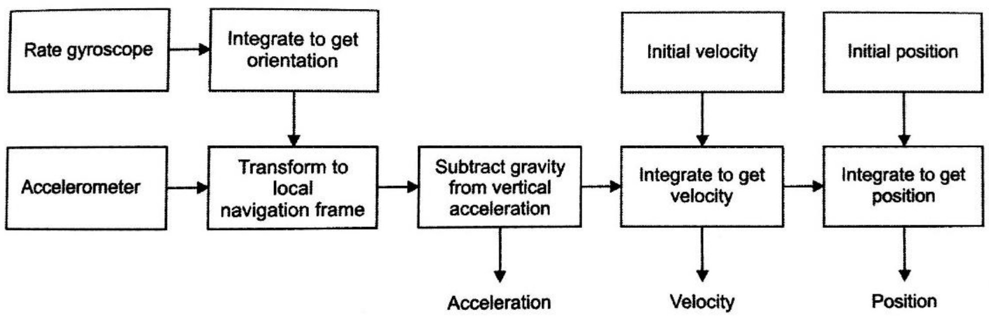
2.3.3 Inertial Navigation System

Inertial Navigation Systems use accelerometers, gyroscopes and compasses to estimate the position of the robot (Figure 2.10). Inertial Measurement Units (IMU) are often used for navigation on ships and aircraft as well as in robotics.

IMUs usually consist of a 3-axis accelerometer and a 3-axis gyroscope, although it is becoming increasingly common to include a heading sensor such as a compass and even thermometers and barometers.

Accelerometers measure the external forces acting upon them. These forces can be static such as gravity or dynamic. The static acceleration can be used to find out the angle the device is tilted at with respect to the Earth. The dynamic acceleration can be used to analyse the movement and calculate position.

Gyroscopes are devices that measure rotational motion. The angular velocity can be integrated to obtain the orientation. These sensors are often used on balancing robots to keep them upright.


Rate gyroscope
Integrate to get orientation
Initial velocity
Initial position
Accelerometer
Transform to local navigation frame
Subtract gravity from vertical acceleration
Integrate to get velocity
Integrate to get position
Acceleration
Velocity
Position
Figure 2.10 Calculating position with an IMU (Siegwart)

Compasses, unlike accelerometer and gyroscopes, are exteroceptive sensors and measure the magnetic field which is used for calculating the heading. The disadvantage of compasses is that they are easily affected by exterior magnetic fields which corrupts the data supplied by these sensors.

2.4 Path Planning

In order to operate autonomously, robots must be capable of finding their way from an initial position to a goal defined by the user. To operate autonomously, the robot must compute the most effective trajectory in a known environment whilst avoiding obstacles. In addition to creating a path around known obstacles, if there are any unexpected obstructions along the path, the robot must compute a new path so it can reach the final goal.

The path planning problem can be divided into two distinct parts:

- **Global navigation:** Involves planning a trajectory from the robot's current position to a goal. To do this, the robot needs a map, either complete or partial, especially in an environment that contains walls and corridors.

- **Local navigation:** Although the robot will usually have a map, this can be incomplete and there can be unpredicted obstacles in the robot's path. To reach the goal, the robot must avoid the obstacles in the most efficient manner possible and try to head as much towards the goal as possible.

To improve the performance of the robot, both techniques should be combined, allowing the robot to avoid unexpected obstacles whilst still heading towards the final goal. If only local navigation is used, although the robot will avoid obstacles and still head towards the goal, it will spend more time trying to find a clear path and could get lost. Global navigation is not sufficient alone either, as if there are any moving obstacles or the map given to the robot is incomplete, it will not be able to situate itself as precisely and could collide with the unknown obstacles.

The path planning algorithms most commonly used will be explained in detail in the following chapter.

2.5 Motion Control: Kinematics for a Differential Drive Robot

Before navigation and path planning can be carried out, it is important to understand the mechanical behaviour of the robot. This is known as kinematics and involves the study of the motion without considering the forces involved.

For a robot operating on a horizontal plane, there are three variables that define the pose of the robot: The x and y coordinates of the position on the plane and the heading which is the orientation with respect to the vertical axis. The possible movements that the robot can perform depend on the type and distribution of its wheels. A differential robot has two separately controlled motors which move wheels independently on the same axis on either side of the robot (Figure 2.11). If the forward velocity $\dot{X}$, the lateral velocity $\dot{Y}$ and the angular velocity ω are known, with the dimensions of the robot, the velocities v_L and v_R of each wheel can be calculated.

Differential drive robots are very sensitive to slight changes in the velocity of the wheels and a small error in the relative velocity between the wheels can cause great error in the trajectory.

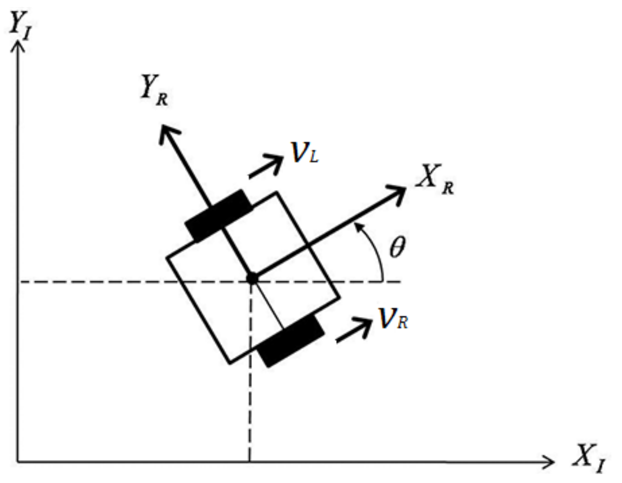


Figure 2.11 Differential Drive Robot

For this type of robot there are three simple types of motion:

1. If $v_L = v_R$ the robot will move forwards or backwards in a straight line, the angular velocity is 0 and there is no rotation.
2. If $v_L = -v_R$ then the robot will turn in place around the midpoint of the wheel axis. The forward velocity is zero.
3. If $v_R = 0$, the robot will turn in a circle around the right wheel. If $v_L = 0$, the rotation will be around the left wheel.

When one wheel rotates at a higher speed than the other, the robot will turn around a point which lies either to the right or the left along the wheel axis. This point is known as the Instantaneous Centre of Curvature (ICC) and is the centre of the circular path described by the robot.

For differential drive robot, there are two geometrical parameters that define the motion of the robot: The wheel radius and the length of the wheel axis. For these robots, there is no lateral velocity and the velocities for each individual wheel can be calculated from the rate of rotation around the ICC:

$$V_r = \omega \left(R + \frac{L}{2}\right)$$

$$V_l = \omega \left(R - \frac{L}{2}\right)$$

With R being the radius from the ICC to the midpoint of the wheel axis, L the distance between the two wheels and ω the angular velocity (Figure 2.12).

The distance travelled by a robot can be calculated with the radius of the wheel and the angle the wheel has rotated. If the robot is travelling forward, the distance s travelled with one complete turn of the wheel is:

$$s = 2\pi r$$

With r being the radius of the wheel.

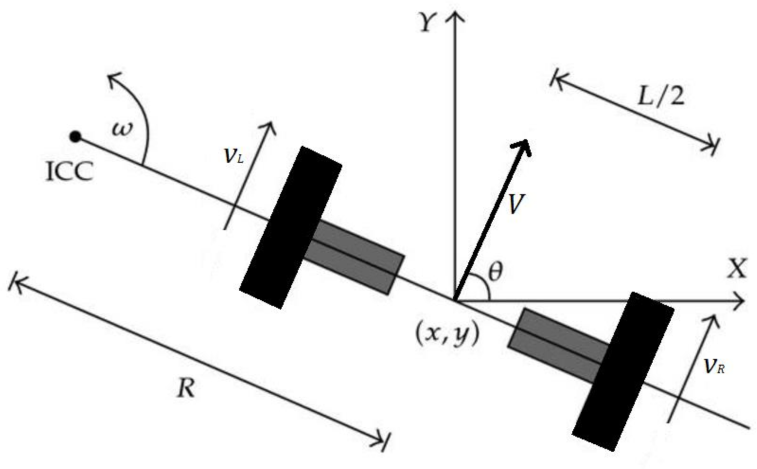


Figure 2.12 Instantaneous Centre of Curvature

Therefore, the distance travelled can be generally expressed as:

$$s = \alpha r$$

Where α is the angle rotated (in radians). The angle rotated or the angular velocity of the wheels of the robot is obtained from the encoders. The encoders can be used to obtain an estimation of the pose of the robot. With the velocities of each wheel, the overall angular velocity and forward velocity can be calculated. If these are then integrated, and added to the previous x, y and θ vales the current position is obtained. However, it is important to take into account that encoders alone are not sufficient for calculating the pose of the robot as they accumulate a lot of error.

CHAPTER 3

NAVIGATION ALGORITHMS

3.1 Environment Mapping Techniques

Before graph based searches can be performed, the environment must be divided into free and occupied space and represented on a graph. When choosing which mapping technique to use, three key factors must be considered:

- The precision with which the robot should reach its target.
- The types of features that need to be represented on the graph.
- The computational cost of the map, as this depends on the complexity of the graph.

Robot mapping methods can be continuous or discrete. Continuous mapping consists of creating a map with the data from the sensors on board the robot. As only the sensor readings are used, these maps are usually very precise. However, if there are many features the computation cost will be high and to store the map a lot of memory will be needed.

Discrete mapping techniques basically decompose the environment, distinguishing the features from open space. These types of maps are transformed into graphs with a much lower computational cost, making path planning easier to compute. The disadvantage is the precision, as the features are usually represented approximately. In this project, discrete mapping methods will be used.

3.1.1 Exact Cell Decomposition

One common technique is cell decomposition, which can either be exact or approximate. In exact cell decomposition, the cells can only be completely occupied or completely free (Figure 3.1). The position of the robot inside each cell is not important, but the robot must be able to move from its current cell to the adjacent cells. The disadvantage of this method is that the exact path of the robot is not defined, therefore the exact position of the robot in the cell is unknown.

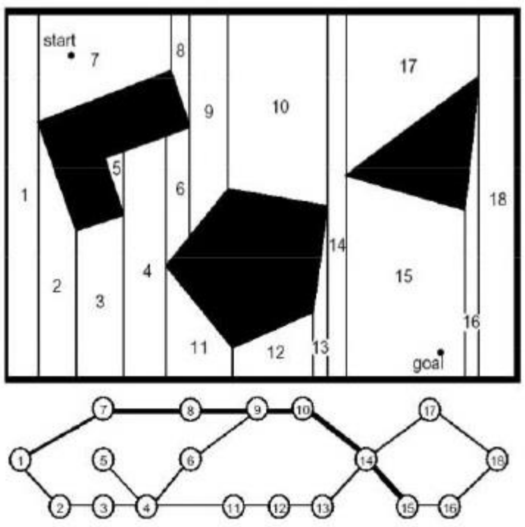


Figure 3.1 Exact cell decomposition

3.1.2 Approximate Cell Decomposition

Approximate cell decomposition is one of the most popular techniques due to the simplicity of grid representations. The most common method is the occupancy grid. The robot environment is divided into a grid with fixed sized squares, therefore obstacles are represented approximately. A drawback of this method is that a cell is classed as an obstacle even though it may not be completely occupied, and depending on the size of the grid squares, narrow passages can be lost [12].

It is also important when choosing the size of the grid squares to consider the size of the robot, as if the robot is bigger than the cell size, it is not possible to locate the robot in just one position. This reduces the accuracy of the grid when defining obstacles and for large robots this technique would not be suitable.

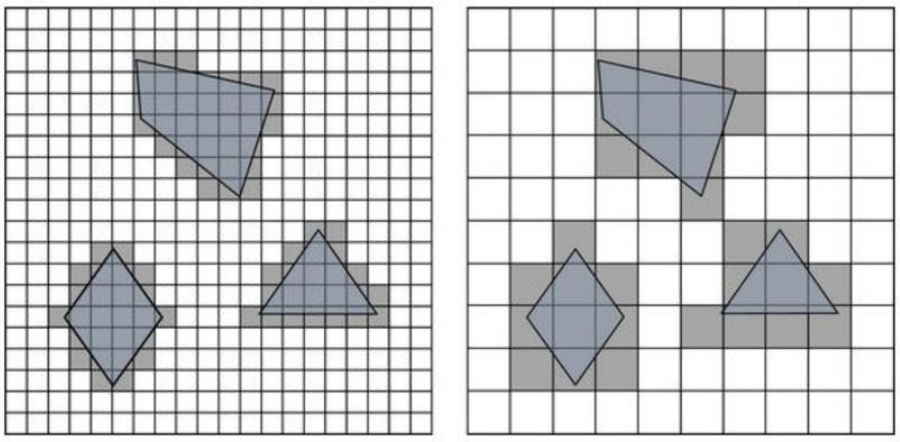


Figure 3.2 Effect of grid resolution for approximate cell decomposition

The size of the features in the environment must also be considered, as if there are small features, the resolution of the grid must be higher, but if the obstacles are large, then only a low resolution is needed. The resolution used also depends on the accuracy and time available for reaching the goal. If the robot needs to arrive quickly to the goal, a higher resolution can be used allowing the robot to pass through smaller gaps that would be classed as obstacles with a lower resolution (Figure 3.2).

3.1.3 Adaptive Cell Decomposition

This method is analogous to the approximate cell decomposition, as it also divides the environment into a grid. However, with this technique, the size of the squares depends on their proximity to an obstacle (Figure 3.3).

This method divides the environment into two squares, then uses a recursive decomposition procedure for each square. Each cell should either be free space or an obstacle, if both values are present in a cell, this cell is divided into four equal sized cells. This procedure is then applied to each of the new cells. Once a cell contains only free space or an obstacle, it remains the same size. This iterative process is carried out until the whole environment consists of cells of either obstacle or free space.

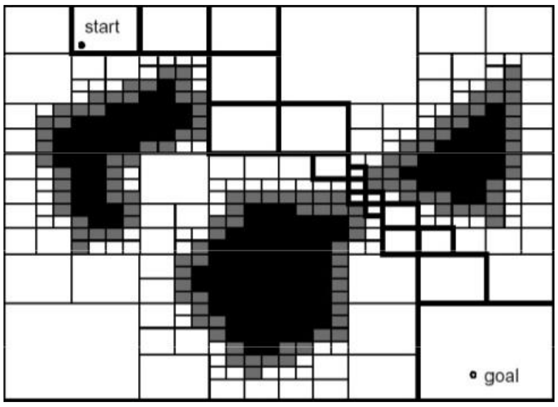


Figure 3.3 Adaptive cell decomposition

The precision of this method depends on the minimum cell size defined and the allowed computational cost.

3.1.4 Voronoi Diagram

A different approach for creating a map is the Voronoi diagram. This is a road map method that tries to maximize the distance between the robot and the obstacles [2]. This technique calculates divides the environment into Voronoi cells, with each one containing an obstacle or point. The cell boundaries are calculated as the midpoints between the obstacles. These points are joined together forming paths (Figure 3.4).

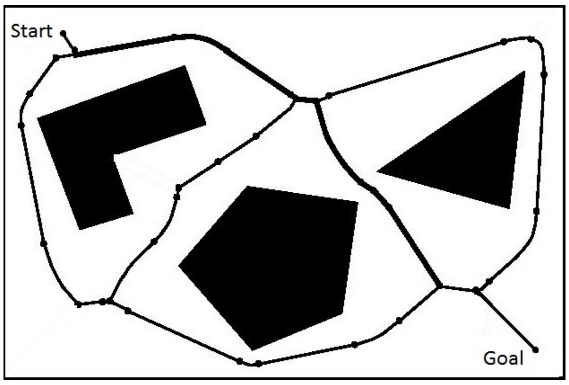


Figure 3.4 Voronoi diagram

The advantage of this method is that the robot will always be as far away as possible from the obstacles. Also, this type of diagram is very easy for the robot to follow using simple control rules. The disadvantage is that the computational cost is higher than for other methods because of the cost of calculating the equidistant points.

3.2 Path Planning Algorithms

The path planning problem, according to Dudek [13], consists in creating a path in an environment between the initial pose of the robot and the goal position in a such a way that the trajectory and planned motion is consistent with the kinematic constraints of the robot whist also avoiding collision. Path planning is the decision-making part of navigation, as it is responsible for finding the quickest and safest way to reach the goal.

Before carrying out any type of path planning, the robot environment must be transformed into a suitable discrete map. There are two general strategies according to Siegwart [2]:

- **Graph search:** First a graph of the robot world is built, then it is searched to find the best path.
- **Potential field planning:** Consists in imposing a mathematical function directly onto the space. The gradient of this function is then followed to the goal.

3.2.1 A* Path Planning Algorithm

The A* algorithm [14] is a graph search technique that finds the optimal path with the lowest cost from an initial node to any position in a previously defined graph. This method is similar to Dijkstra's algorithm [15], but includes a heuristic function to try and reduce the number of nodes explored.

Before the algorithm can be carried out, the robot world must be divided into a grid, which can be defined as a two-dimensional array. Every grid square must be classified as either a free space or an obstacle (Figure 3.5). All of the squares (also referred to as cells or nodes) originally have their status set as unvisited or obstacle. The shortest path is created by evaluating the adjacent cells and calculating a cost function to choose the best next move.

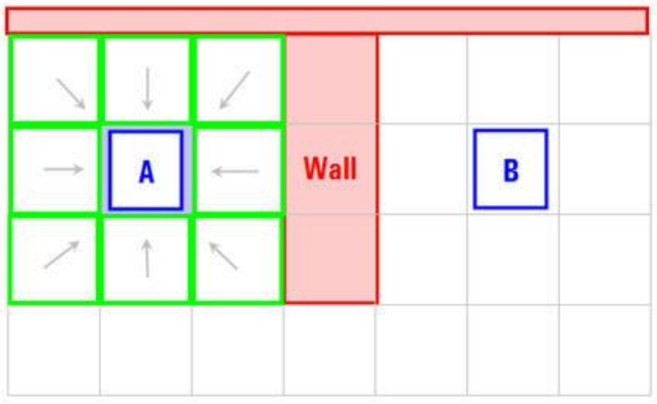


Figure 3.5 Robot world divided into a grid

The cost function f(n) is defined as:

$$f(n) = g(n) + h(n)$$

The function g(n) is the cost to move from the starting cell to any other position in the grid. The cost to move horizontally or vertically is given a value, and the cost to move diagonally is the square root of 2 times the cost of moving horizontally or vertically.

The function h(n) is the heuristic function which gives additional knowledge about the graph, making the algorithm more efficient. The heuristic is the estimated cost to move from the current position to the goal node. The performance of the algorithm depends on the heuristic used, but the efficiency on average is much better than that of the Dijkstra algorithm.

If the heuristic function h(n) is 0, the A* algorithm will behave the same as Dijkstra's and will be slower. If h(n) has a very high value, the algorithm will run very quickly, however the path may not be the shortest possible route. Some heuristics commonly used are the Manhattan distance (distance to target only moving horizontally and vertically), diagonal distance and the Euclidian distance. The heuristic used in this project is calculated estimating the Euclidian distance to target cell. This method estimates the absolute distance to the goal, ignoring any obstacles that might be in the way.

In the first step of the algorithm, all of the cells that are adjacent to the first cell are either added to an open list or a closed list. The open list contains all of the cells that are possible candidates for the next move. The closed list contains all of the cells that have already been evaluated or are obstacles. The value of the function f(n) is calculated for all the adjacent cells that are in the open list. Supposing that the cost for horizontal and vertical moves is 10, for diagonal moves 14 and with A being the parent cell, the values for the adjacent cells in Figure 3.5 would be as shown in Figure 3.6 [16]:

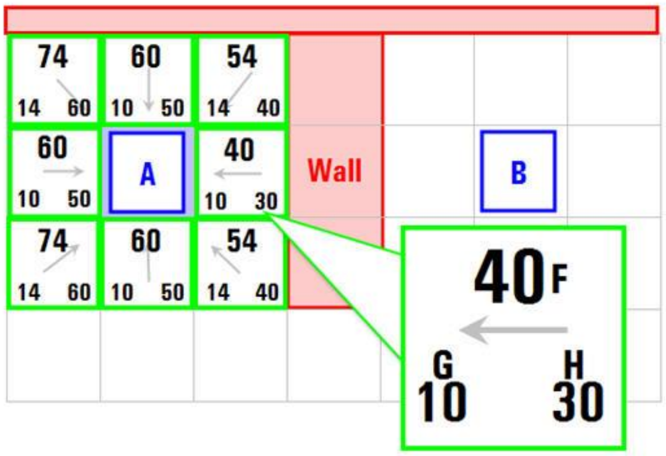


Figure 3.6 f(n), g(n) and h(n) scores for the previous configuration

Once the cost for each adjacent cell has been calculated, the cell with the lowest value is chosen (in this case the cell with a total f(n) score of 40) and the cells surrounding it that are not obstacles or the previous cell are added to the open list and their new cost scores are calculated (Figure 3.7).

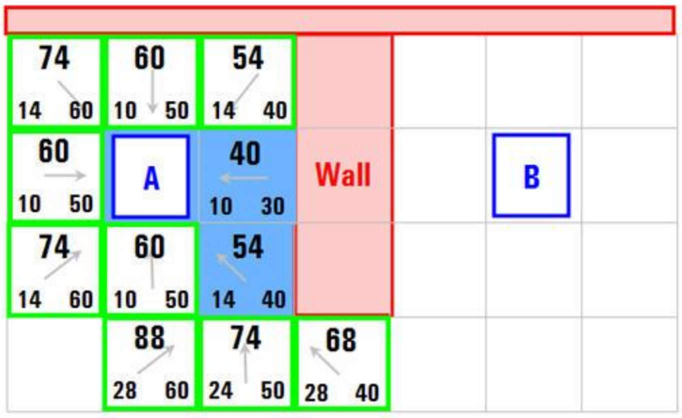


Figure 3.7 A* Algorithm after two iterations

This process is repeated until the goal position is reached (Figure 3.8). To determine the path once at the target, the algorithm works backwards moving to the parent cell of the current cell (shown by the orange arrows in Figure 3.8). This is the path with the lowest cost from the starting position to the target cell.

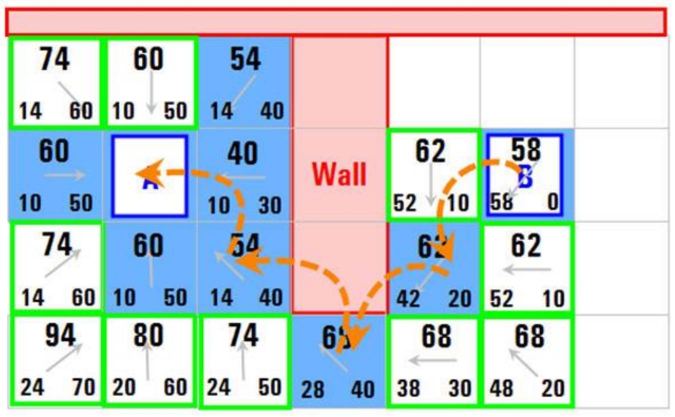


Figure 3.8 Final Path

In many cases, it is not necessary to obtain the shortest path, but find a path within a defined search time. The first solution can then be improved by reusing previous search efforts according to the available search time [17]. This is known as Anytime Replanning A* and if an accurate heuristic is used, far fewer states need to be explored than with the original A* algorithm.

3.2.2 D* Path Planning Algorithm

The D* algorithm (Dynamic A*) is a real-time planning version of the A* algorithm which allows the path to be recalculated if unexpected obstacles are encountered. If the environment map is incomplete or changes while the robot is moving, the path will need to be replanned [18]. With the A*, the robot would have to wait for the path to be completely recalculated, which is slow and inefficient. When the robot receives additional information about its environment, the D* algorithm can revise the path it calculated before and try to reduce the total cost of the trajectory.

The D* algorithm start by planning a path using the A* method. After a certain amount of time, the robot might observe some changes in the environment with its onboard sensors, and a new trajectory must be computed. The D* algorithm then generates a new path for the states that are affected by the new obstacle that was added or removed, without having to backtrack.

In addition to the open and closed lists used in the A* algorithm, this method has two more states for each node: Raise and lower. When a cell that was previously thought to be free space is actually an obstacle, the cost of that position increases, which is the Raise state. When a cell that contained an obstacle is free space, the state will be Lower because the cost of the path can be decreased.

Analogous to the A* algorithm, a dynamic version also exists for the D* algorithm, which is known as Anytime D*.

3.2.3 Potential Field Path Planning

Potential field planning is a method that resembles a charged particle moving through a magnetic field. This effect can be simulated in a robotics environment by creating an artificial potential field across the environment to attract the robot to the goal [19].

In a potential field, the goal position acts as an attractive force pulling the robot towards it and the obstacles generate a repulsive force, pushing the robot away from them. The potential field is defined across the whole of the robot world and calculated in the position of the robot (Figure 3.9).

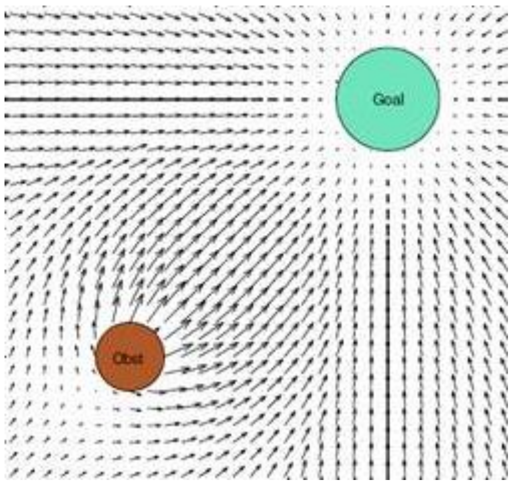


Figure 3.9 Potential field generated by an obstacle (repulsive) and a goal (attractive)

To calculate the resultant potential field in a certain position, all of the repulsive and attractive force vectors are added together (Figure 3.10). The force induced by the field is calculated in this position, and the robot should move according to this force.

The total potential field can be expressed mathematically as:

$$U(q) = U_{goal}(q) + \Sigma\, U_{obstacles}(q)$$

And the induced force is:

$$F = -\nabla U(q) = \left(\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right)$$

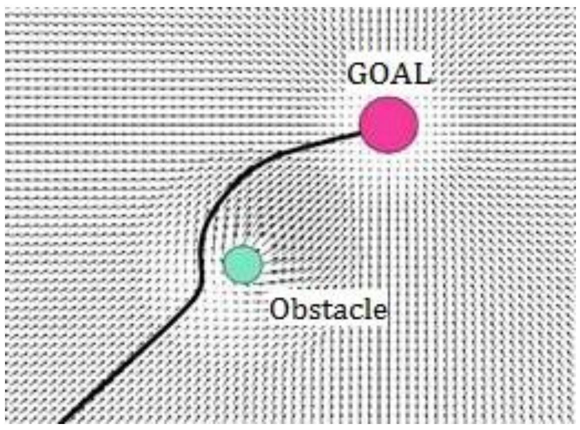


Figure 3.10 Path created by potential field forces

The potential field method was developed as a real-time obstacle avoidance approach that could be applied when the robot detects obstacles while moving, instead of having a pre-defined map. However, by only relying on local information, the robot can get stuck in a local minimum when the sum of the forces is zero, causing the robot stop in this position. This is a big problem that can occur frequently with the potential field method, as can be seen in Figure 3.11 with just a simple U-shaped obstacle, where the robot gets trapped inside the U.

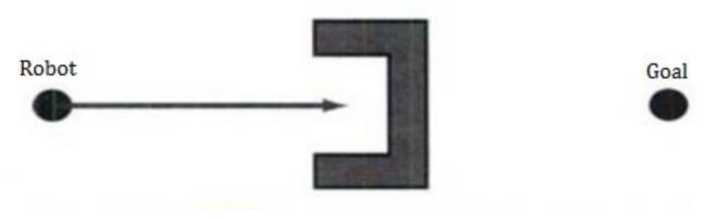


Figure 3.11 U-Shaped obstacle that causes a local minimum

3.3 Local Navigation Algorithms

3.3.1 Bug Algorithm

The simplest obstacle avoidance algorithm is the bug algorithm. This algorithm consists in heading towards the goal until encountering an obstacle, then following the obstacle until it's possible to head towards the goal again. This process is known as Bug0 and is repeated until successful, however some obstacles can confound the algorithm, causing it to fail.

The algorithm can be improved by adding memory of the past locations. When the robot comes across an obstacle, it circumnavigates the obstacle and maps it, then heads towards the goal from the closest point of approach. This is known as the Bug1 algorithm.

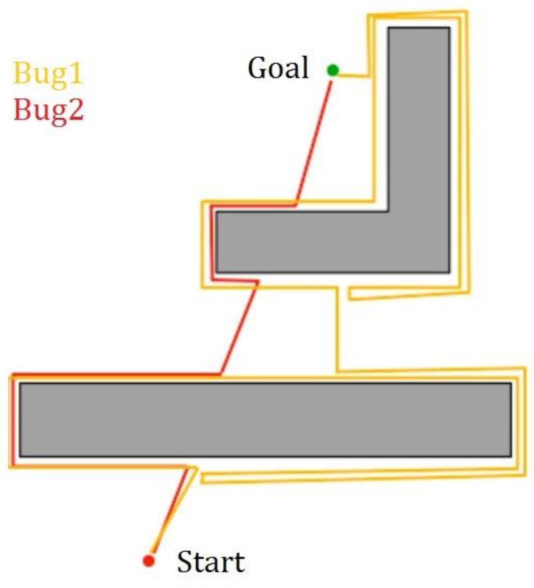


Figure 3.12 Comparison between Bug1 and Bug2 algorithms

The bug1 algorithm is tedious and if there are many obstacles it is very inefficient (Figure 3.12). The Bug2 algorithm consists in creating a line between the start position and goal and following this line until finding an obstacle. When an obstacle is encountered, the robot follows the perimeter until it comes across the line again closer to the goal.

3.3.2 Vector Field Histogram

The Vector Field Histogram (VFH) is a real-time motion planning algorithm proposed by Johann Borenstein and Yoran Koren in 1991 [20]. The VFH considers the size and velocity of the robot as well as the surroundings, unlike other obstacle avoidance algorithms.

The VFH creates a cartesian occupancy grid of the environment using the most recent sensor data, usually from a LiDAR or ultrasonic sensor and updates the grid in real-time. This grid is then converted into a one-dimensional polar histogram (Figure 3.13). The angle at which an obstacle is found is shown along the x-axis and the y-axis represents the probability that there is actually an obstacle in that direction based on the data from the occupancy grid. When the probability of an obstacle is below the threshold, the path in this direction is considered free.

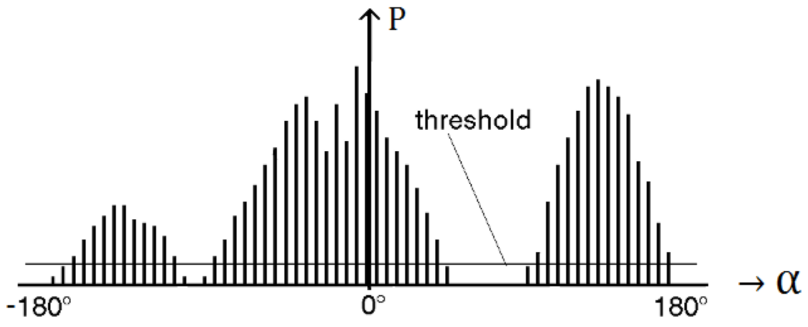


Figure 3.13 Polar Histogram

The spaces that are large enough for the robot to pass through are identified and a cost function is applied to find the gap with the lowest cost and this direction is chosen. Borenstein and Koren [21] defined a cost function G as:

$$G = a \cdot heading + b \cdot orientation + c \cdot previous\ direction$$

The heading is the target alinement of the robot with the goal. Orientation is the difference between the new heading and he current direction. Previous direction is the heading chosen previously. The values of the parameters a, b and c depend on the importance of each factor in the robot's behaviour. If the heading of the robot is the most important factor, the value of a will be higher than that of b and c.

When the robot approaches an obstacle, the velocity is reduced giving it more time to calculate a path. If no path can be found, the robot will reverse and turn looking for an alternative route.

CHAPTER 4

IMPLEMENTATION USING A ROBOTICS SIMULATOR

4.1 Software

The navigation software will be programmed using LabVIEW. LabVIEW is a programming environment which uses a graphical notation called "G" (for graphical), connecting nodes and wires through which the data flows. LabVIEW programs are called Vis (Virtual Instruments) and consist of three parts: a front panel, a block diagram and a connector panel (Figure 4.1).

The front panel of the program is the graphical user interface (GUI) and contains the input and output controls and indicators of the VI, allowing the user to supply and receive information from the system. When controls or indicators are placed on the front panel, a terminal is automatically placed on the block diagram.

The block diagram contains the graphical source code of the user interface. The objects on the front panel are represented as terminals on the block diagram. The terminals reflect the changes made to the front panel objects. Block diagrams include terminals, subVIs, functions, constants and structures which are connected by wires. SubVIs are smaller portions of code which can be called from within a VI. SubVIs are the same as Vis, consisting of a front panel and block diagram and are used for simplifying the main VI.

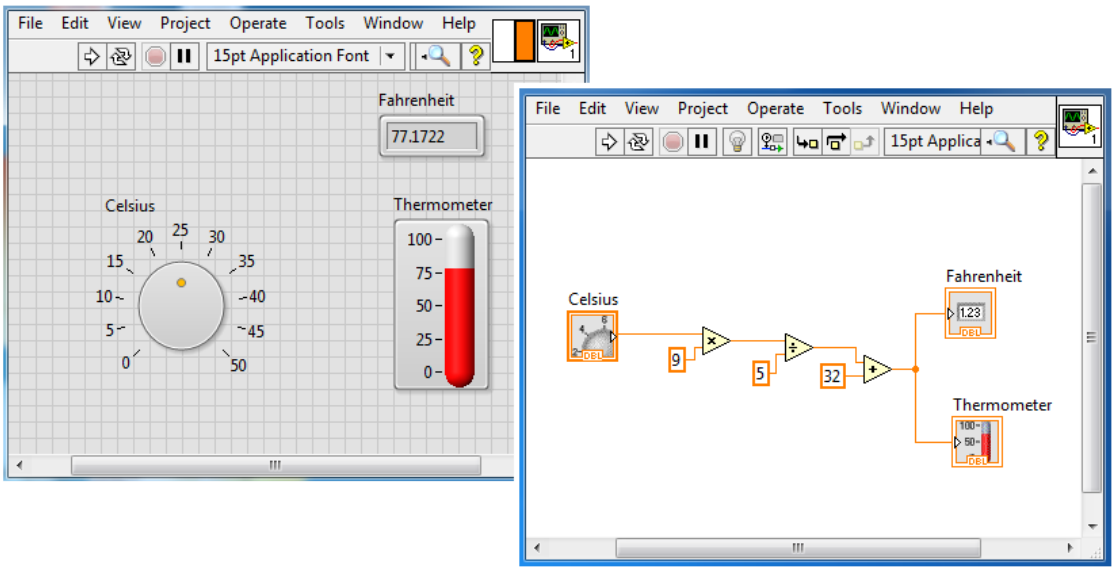


Figure 4.1 LabVIEW VI. Front panel (left), Block diagram (right)

The third art of a VI is the connector pane. The connector pane is needed when a VI is called as a subVI and appears in the top right-hand corner of every front panel next to the VI icon. The connector pane is a set of terminals that correspond to the controls and indicators of the VI (the inputs and outputs that are wired to the VI). The VI icon is the graphical representation of a VI and is what represents the VI on the block diagram when the VI is used as a subVI.

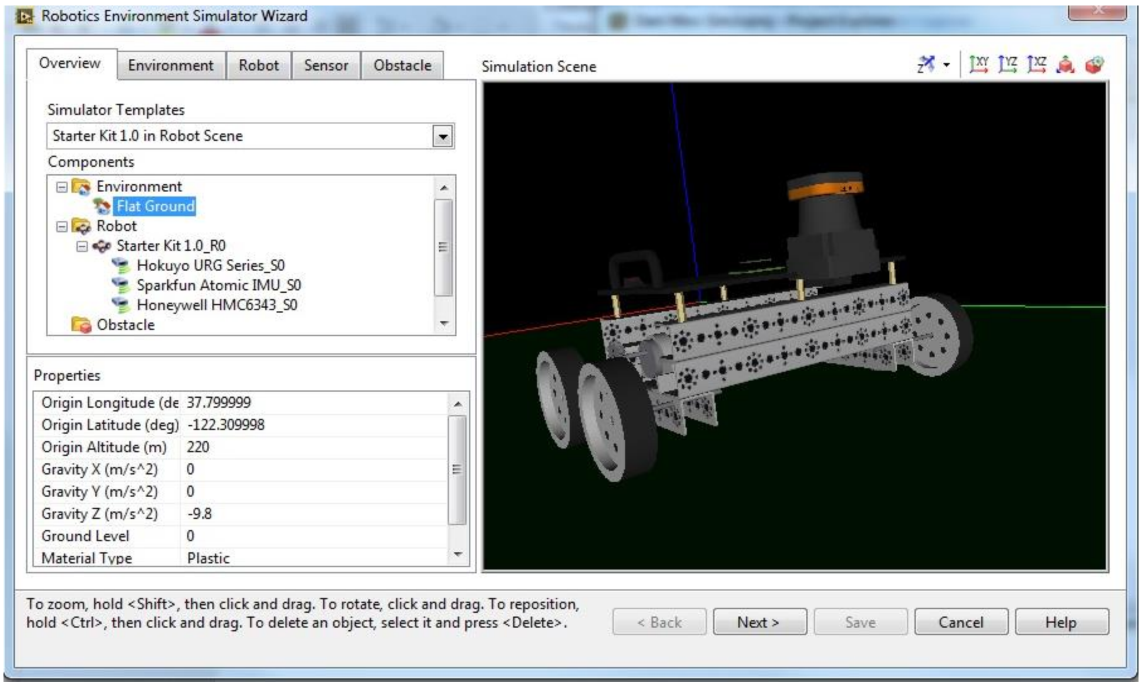
4.2 Robotics Simulator

The robotics simulator used for this project is the NI LabVIEW Robotics Module which is connected to the LabVIEW software. This module has a wide range of robots, sensors and environments, and there is also the option to design your own robot and environment with separate CAD software.

Before a new project is created, a window appears where the characteristics of the simulation must be defined. First an environment type is chosen, the type of environment must be suitable for the size and type of robot that will be used. Once a robot has also been chosen it usually has a sensor by default, but this can be removed and changed for a variety of other sensors.

4.2.1 Simulated Robot and Sensors

In this case the DaNI Starter Kit 1.0 has been used with a Hokuyo URG Series laser scanner, a Sparkfun Atomic 6DOF Inertial Measurement Unit (IMU) and a Honeywell HMC6343 compass module (Figure 4.2). Tests will be carried out in different simulation environments.



Simulator Templates: Starter Kit 1.0 in Robot Scene

Components
- Environment
  - Flat Ground
- Robot
  - Starter Kit 1.0_R0
    - Hokuyo URG Series_S0
    - Sparkfun Atomic IMU_S0
    - Honeywell HMC6343_S0
- Obstacle

Properties

Origin Longitude (de	37.799999
Origin Latitude (deg)	-122.309998
Origin Altitude (m)	220
Gravity X (m/s^2)	0
Gravity Y (m/s^2)	0
Gravity Z (m/s^2)	-9.8
Ground Level	0
Material Type	Plastic

To zoom, hold <Shift>, then click and drag. To rotate, click and drag. To reposition, hold <Ctrl>, then click and drag. To delete an object, select it and press <Delete>.

Figure 4.2 LabVIEW Robotics Simulator

The DaNI Starter Kit robot (Figure 4.3) is a differential drive robot based on a NI Single Board RIO controller. The NI Single Board RIO platform integrates a real-time processor, a reconfigurable FPGA and analogue and digital input/output. The robot moves by means of four wheels, controlled by two DC motors (right and left). Each motor has an optical encoder for odometry.

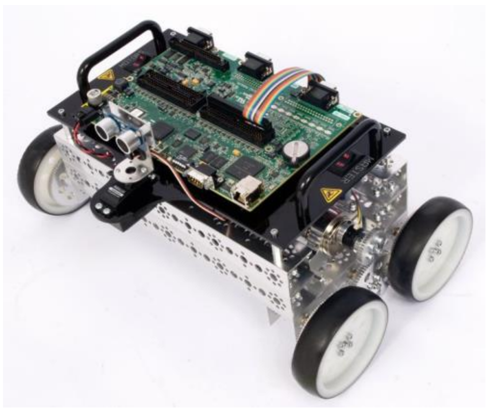


Figure 4.3 National Instruments DaNI Robot

The laser scanner used for the simulation is a Hokuyo URG series (Figure 4.4). This laser has a scanning range from 0.02m to 5.6m, covering 240 degrees. The scan rate is 10Hz with a 1mm resolution and ±10mm accuracy. Normally around 250-350 readings are obtained with each scan, however, with the robotics simulator just 27 points are obtained.

Figure 4.4 Hokuyo Laser Scanner

The only IMU sensor available on the simulator is the Sparkfun 6DOF (Degrees of Freedom) IMU with consists of a 3-axis accelerometer, a dual axis pitch and roll gyroscope and a yaw-rate gyroscope (Figure 4.5). The accelerometer has a ±3g range and the gyroscopes have a scale of 300˚/s.

Figure 4.5 6DOF IMU

For orientation, a 3-axis compass module is used (Figure 4.6). This compass measures the magnetic distortion in heading pitch and roll directions. The precision for heading calculation is 2˚.

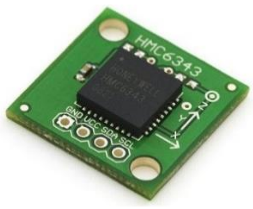
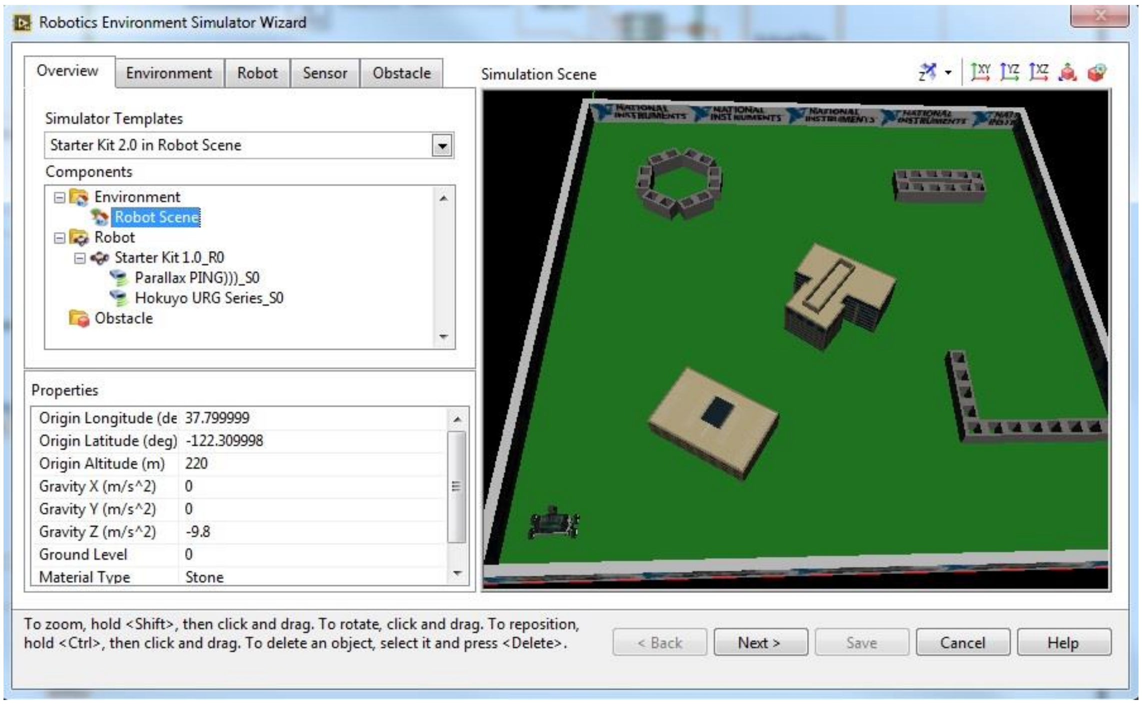


Figure 4.6 HMC6343 Compass Module

4.2.2 Simulator Environments

The algorithms implemented in this project will be tested in two different simulated environments. The first environment is designed especially for the DaNI Starter Kit robot (Figure 4.7).

The material of the ground and obstacles can be modified, in this case stone is used for the ground and the material for the obstacles is either stone, wood or plastic. The materials chosen are important as in real tests the robot will move differently on different surfaces and the precision of the sensors will vary slightly depending on the material of the obstacles.



Simulator Templates: Starter Kit 2.0 in Robot Scene

Components
- Environment
  - Robot Scene
- Robot
  - Starter Kit 1.0_R0
    - Parallax PING)))_S0
    - Hokuyo URG Series_S0
- Obstacle

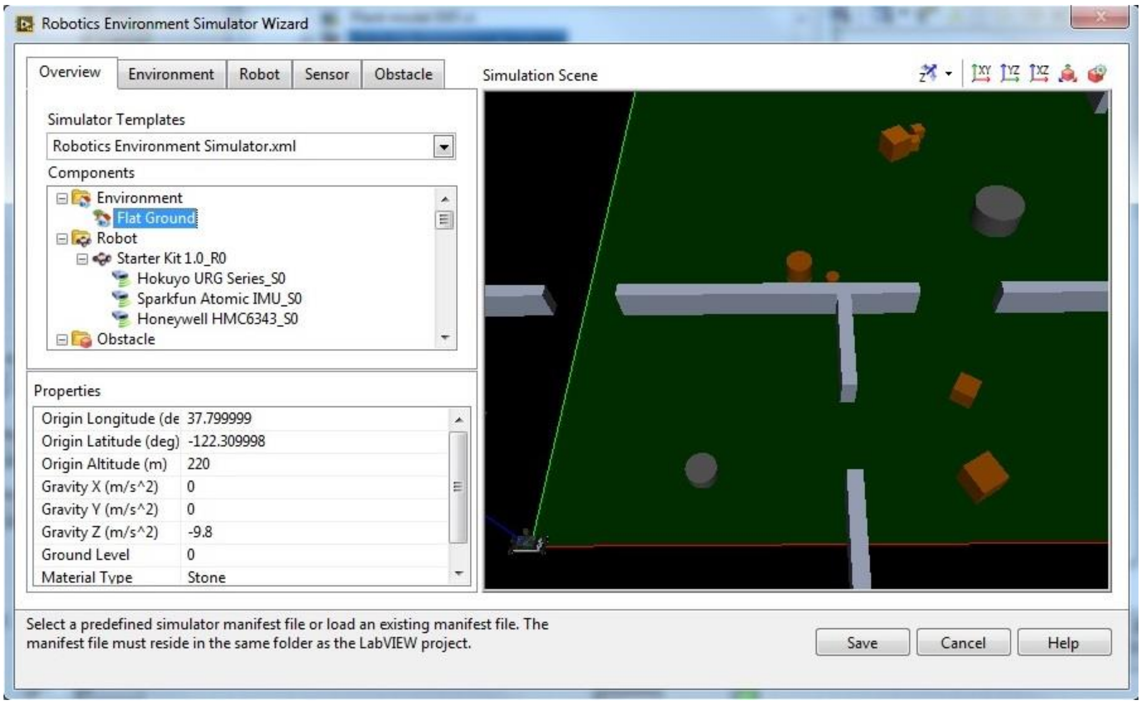
Properties

Origin Longitude (de	37.799999
Origin Latitude (deg)	-122.309998
Origin Altitude (m)	220
Gravity X (m/s^2)	0
Gravity Y (m/s^2)	0
Gravity Z (m/s^2)	-9.8
Ground Level	0
Material Type	Stone

To zoom, hold <Shift>, then click and drag. To rotate, click and drag. To reposition, hold <Ctrl>, then click and drag. To delete an object, select it and press <Delete>.

Figure 4.7 Robot Scene Environment

The second environment used has been built using obstacles and modifying their size and properties. This environment consists of a stone flat ground base and wooden walls and obstacles to resemble rooms and doorways (Figure 4.8).



Simulator Templates: Robotics Environment Simulator.xml

Components
- Environment
  - Flat Ground
- Robot
  - Starter Kit 1.0_R0
    - Hokuyo URG Series_S0
    - Sparkfun Atomic IMU_S0
    - Honeywell HMC6343_S0
- Obstacle

Properties

Origin Longitude (de	37.799999
Origin Latitude (deg)	-122.309998
Origin Altitude (m)	220
Gravity X (m/s^2)	0
Gravity Y (m/s^2)	0
Gravity Z (m/s^2)	-9.8
Ground Level	0
Material Type	Stone

Select a predefined simulator manifest file or load an existing manifest file. The manifest file must reside in the same folder as the LabVIEW project.

Figure 4.8 Customized Robot Environment

This could environment could also have been built by designing the environment using CAD software such as Solidworks and then importing the model to the simulator. For this project, the environment has been designed using the LabVIEW simulator as it is easier to modify the position and properties of the features.

4.3 Development of the Navigation Software

4.3.1 Initializing the Simulator, Robot and Sensors

The first part of the program is the initialization of the robot and sensors in the simulator. The simulation program consists of a manifest file, and ID List and the Vis. The manifest file is where the simulation environment and components are saved in an .xml file. The ID list is a text file which contains the ID names of the robots, components and obstacles in the environment.

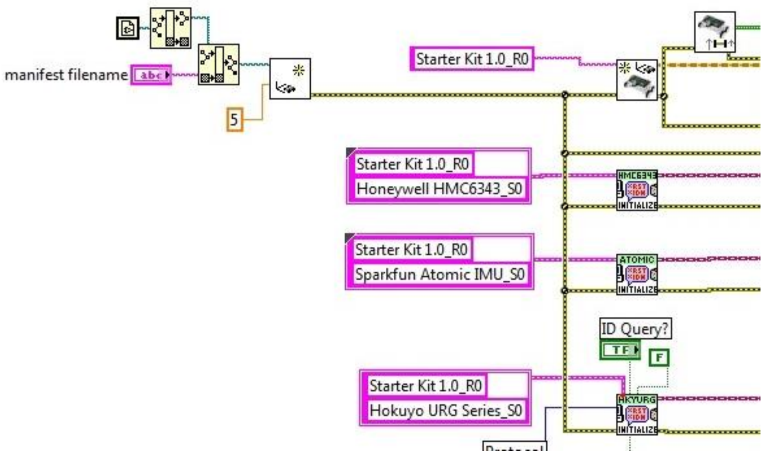


Figure 4.9 Simulator Initialization

To initialize a robot in the simulator, the initialize simulator block for the robot must be connected to the error in from the simulator and the robot ID from the ID list must be defined as an input (Figure 4.9). If more than one robot is used in the same program, the robot ID differentiates them. The same must be done for each sensor, the robot ID and sensor ID must be defined. The simulator must be loaded and initialized before any other part of the program, otherwise there will be an error.

4.3.2 Path Planning: A* Algorithm

To implement the A* algorithm, an occupancy grid of the robot environment is needed. The precision of the map provided will influence the results of the algorithm. In an occupancy grid map, the first cell (0,0) is located at the bottom left corner of the grid. In LabVIEW, a matrix is defined with the first element in the top left corner, so when an occupancy grid is defined, the LabVIEW Create Occupancy Grid Map VI automatically adjusts the input matrix or file relative to the new origin.

The first occupancy grid defined as a matrix (with the origin (0,0) in the top left corner) for the robot scene environment is:

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	100	1	1
1	1	100	100	1	1	1	1	100	100	1	1
1	100	100	100	1	1	1	1	1	100	1	1
1	100	100	1	1	1	1	1	1	1	1	1
1	1	1	1	100	100	100	1	1	1	1	1
1	1	1	1	100	100	1	1	1	1	1	1
1	1	1	1	1	100	1	1	1	1	1	1
1	1	100	100	1	100	1	1	1	100	1	1
1	1	100	1	1	1	1	1	1	100	1	1
1	1	100	1	1	1	1	1	1	100	1	1
1	1	100	1	1	1	1	1	1	1	1	1

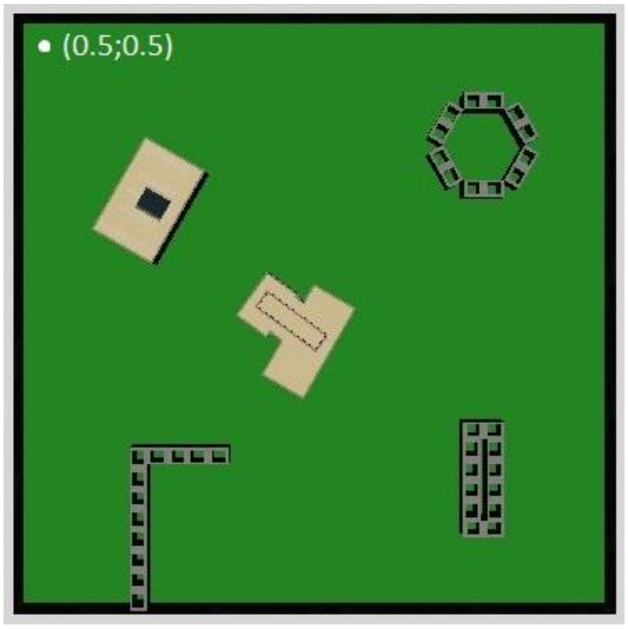


Figure 4.10 Robot scene rotated with origin in top left corner

The values in each cell are the cost for the robot to move to the cell. The obstacles have a cost of 100 and free space has a cost of 1. Other values could be used if a path is possible but not desired, for example, in the case of Google maps, backstreets would have a higher cost than motorways.

The total size of the environment (Figure 4.10) is 6 x 6 metres and the chosen size for each square in the grid is 0.5 x 0.5 metres. This size was chosen as the size of the robot is approximately 0.4m x 0.3m, if a smaller size were chosen the robot would occupy more than one square and the algorithm would not execute properly. The environment was divided into a grid of 12 x 12 cells.

For the second environment, the same sized squares were used and a grid of 18 x 20 cells was obtained, with the origin in the top left corner (Figure 4.11).

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	100	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
100	100	1	1	100	100	100	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	100	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
1	1	1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	100
100	100	100	100	1	1	100	100	100	100	100	100	100	100	1	1	100	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

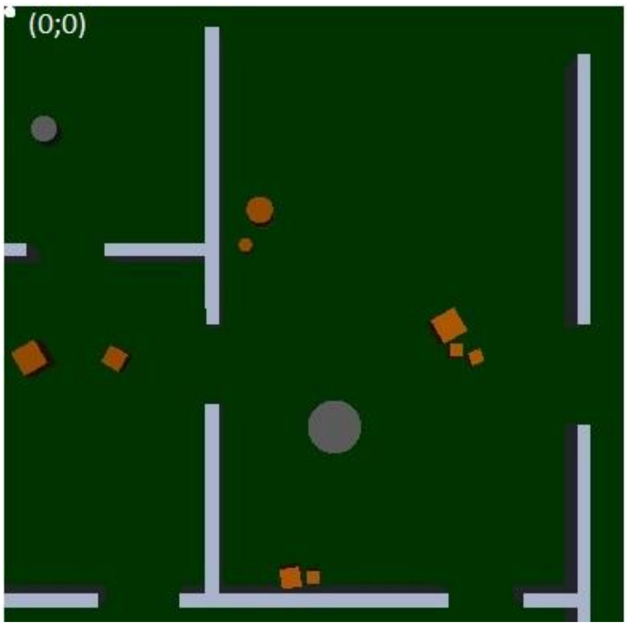


Figure 4.11 Second robot environment rotated with origin in top left corner

The occupancy grid can be defined directly as an array in LabVIEW or can be written in a text file and converted to an array. For this program, the map has been written in a text file as it is easier to manage when there are many cells.

Once the occupancy grid map has been defined, the start and goal cells must be defined. The square number is the position in the matrix that the goal occupies, not the position in metres of the robot. All the data is then processed by the A* VI which calculates the path with the lowest cost to reach the goal. This VI returns a set of path references which are then processed by the "Get Cells in Path VI". The output of this VI is a list of cell coordinates (Figure 4.12).

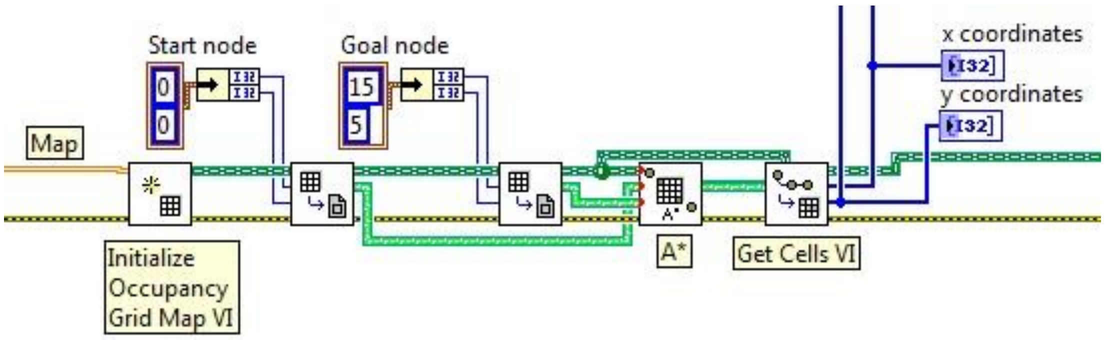


Figure 4.12 A* Algorithm in LabVIEW

4.3.3 Robot Motor Velocity Control

To control the movement of the robot a steering frame is used. The steering frame converts the centre robot velocity to the individual left and right wheel angular velocities. In the steering frame, the geometry of the robot and steering type is defined and the formulas explained in the previous chapter are used. For the DaNI robot, a steering frame is already available (Figure 4.13). The maximum forward and angular velocities and the direction the motor turns in are defined, as well as the geometrical parameters such as the distance between the wheels, wheel size and type, etc.

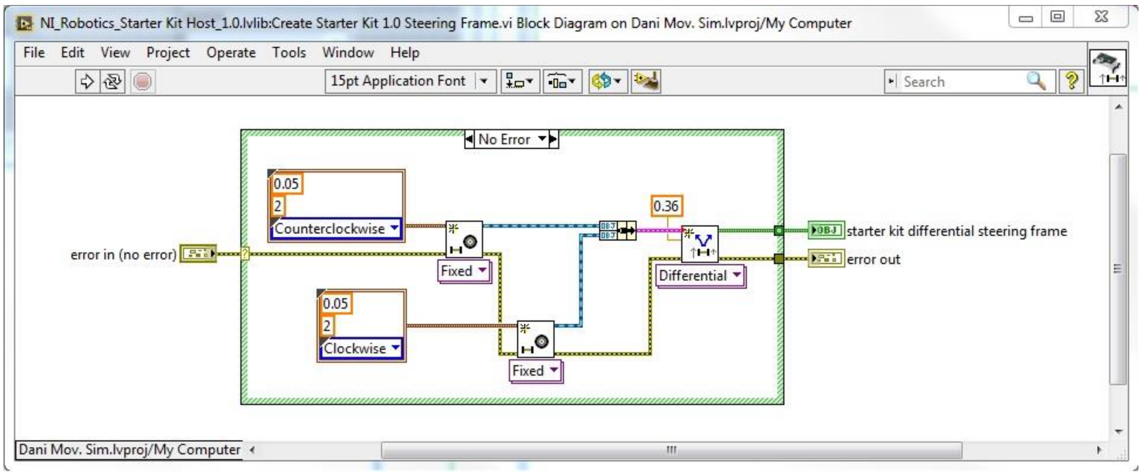

NI_Robotics_Starter Kit Host_1.0.lvlib:Create Starter Kit 1.0 Steering Frame.vi Block Diagram on Dani Mov. Sim.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
No Error
0.05
2
Counterclockwise
Fixed
0.05
2
Clockwise
Fixed
0.36
Differential
error in (no error)
starter kit differential steering frame
error out
Dani Mov. Sim.lvproj/My Computer

Figure 4.13 Steering frame for DaNI robot

The steering frame input is a 1D array of the forward velocity, lateral velocity and angular velocity. For a differential robot, the lateral velocity will always be 0. The output of the steering frame is an array of the left and right wheel velocities. These velocities are then sent to the "Write Motor Velocity Setpoints VI".

In the same way that the velocities are written, the estimated wheel velocities can be obtained from the wheel encoders. These can be converted back to the forward and angular centre velocities of the robot by the steering frame "Get Velocity from Motors VI".

4.3.4 Localization: Dead Reckoning

Dead Reckoning consists in determining the position of the robot from the encoders by measuring the distance travelled. The uncertainty of dead reckoning increases over time and distance.

The change in heading is obtained by integrating the angular velocity, ω. The sampling interval dt is 20ms which is the time between each iteration of the while loop. This difference in heading is added to the initial angle to obtain an estimation of the current heading.

$$\theta = \theta_0 + \int \omega \, dt$$

To obtain the x and y coordinates, the velocities in the forward and lateral direction are rotated by the difference in angle. These rotated velocities V_x' and V_y' are then integrated to obtain the new position. The time interval dt is the time between each iteration of the while loop. The new position is the sum of the initial position and the change in x and y coordinates.

$$x = x_0 + \int V_x' \, dt$$

$$y = y_0 + \int V_y' \, dt$$

The estimated position of the robot is shown on the user interface and is compared to the actual position of the robot.

The actual position of the robot can be obtained by the "Get Simulator Reference VI" when using the robot simulator. With this VI it is possible to ask the robot its exact position in Cartesian coordinates, the velocity and angular velocity among other variables (Figure 4.14).

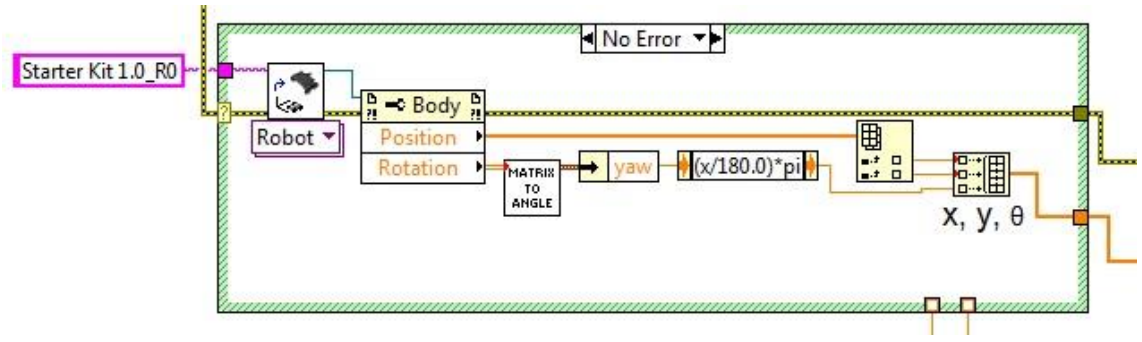


Figure 4.14 Simulated robot real position and heading

4.3.5 Obstacle Avoidance: Vector Field Histogram Algorithm

To obtain better results from the A* algorithm, it has been combined with a local navigation algorithm. If the robot environment is only partially mapped, there could be obstacles in the original planned path which are unknown to the robot.

There are two vector field histogram (VFH) Vis available in LabVIEW, a basic VI for obstacle avoidance, and an Advanced VFH VI for when the robot must also reach a goal. The Advanced VFH has been used with the coordinates previously obtained from the A* algorithm as sub-goals.

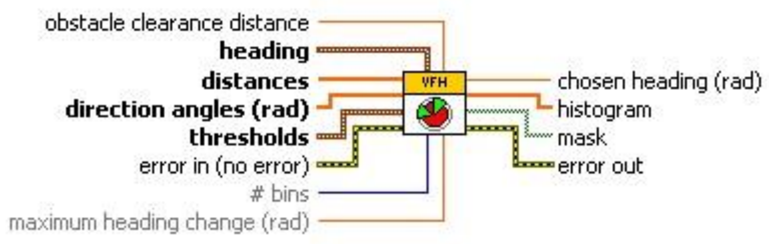


Figure 4.15 Advanced VFH VI

The inputs to the Advanced VFH VI are:

- The obstacle clearance distance is the minimum distance by which the centre of the robot can approach an obstacle. The obstacle clearance distance and threshold distances are controls on the front panel and are defined by the user.

- The heading is an input of two angles: The current heading of the robot and the direction at which the goal position lies (target heading). The current heading is obtained directly from the compass sensor. The target heading is calculated from the position obtained by dead reckoning with the encoders. The heading is calculated by subtracting the current x and y coordinates from the goal coordinates and applying the 2-input inverse tangent to obtain the angle to the goal position.

- Distances are the measurements obtained from the LiDAR sensor and are the distances to obstacles.

- Direction angles are the angles in radians at which objects are located and correspond to the distances. These are also given by the LiDAR sensor.

- Thresholds specify how close an object must be to be considered an obstacle and when an obstacle is considered too close to clear. This cluster contains the inner threshold distance (distance from obstacle at which a direction is considered blocked) and the outer threshold distance (distance to an obstacle at which the direction is no longer considered blocked). These parameters are defined by the user on the user interface.

- # bins are the number of headings the VI considers for travelling. To get the best results, a smaller number of headings than is possible should be used. The best results have been obtained on the simulator with a #bins of 24 with 27 LiDAR readings. This variable is also defied by the user on the user interface.

- The maximum heading change is the greatest angle that the robot can change direction by when a path is blocked. Although it is possible for the robot to turn 90 degrees or more, the best results were attained with this parameter set to 85 degrees.

The outputs from the VFH VI are:

- The chosen heading is the heading calculated by the vector field histogram as the best direction of travel with respect to the goal position and the obstacles in the environment.
- Histogram returns the histogram data arranged by angle and direction.

- The mask contains Boolean data about whether a direction in the environment is blocked.

The sub-goals calculated by the A* algorithm are sent to a for loop. Inside the for loop, a while loop is executed until each sub-goal is reached. In the while loop all the data from the sensors and LiDAR is obtained and processed.

Another while loop is used within these loops for the VFH and it is here that the command velocities (forward and angular velocities) are defined, depending on the distance to obstacles and to the goal. When the robot approaches the goal, the forward velocity is reduced in order to reach the goal with more precision. If an obstacle is close, the velocity is also reduced while a new heading is chosen. If there is no possible direction, an error is given and the robot turns back to try to find a space.

Outside the while loop containing the VFH and command forward velocity calculation, the command heading is converted to angular velocity and this along with the forward velocity is sent to the “Apply Velocity to Motors VI” where it is converted into right and left wheel angular velocities.

The estimated velocities from the encoders are obtained and used for the dead reckoning calculation. This estimated position is subtracted from the sub-goal coordinates and once the distance left is considered close enough to the sub-goal, the next iteration of the for loop can be executed.

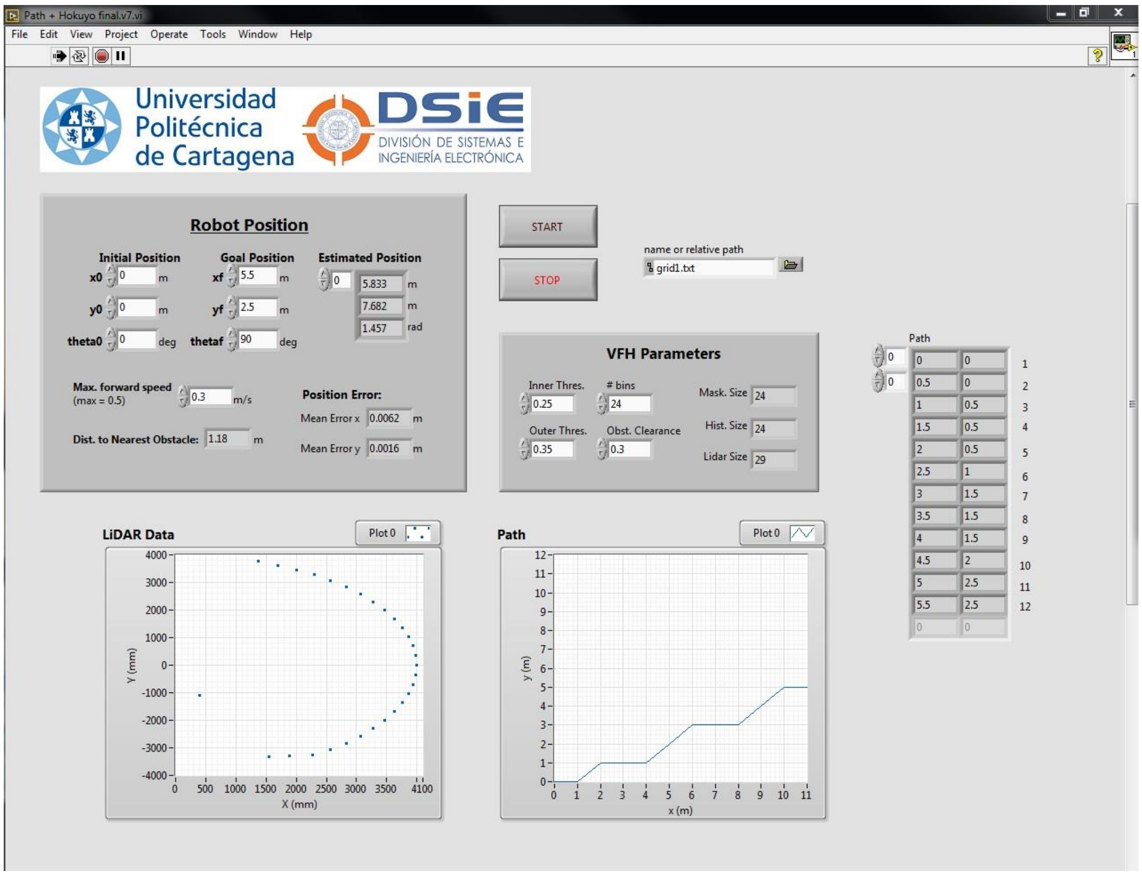
4.3.6 Graphical User Interface (Front Panel)

The user interface designed consists of a control panel where the pose of the robot can be seen and the start and goal positions can be defined (Figure 4.16). Both the position estimated using the encoders and the exact position from the simulator are shown. The maximum forward speed is also a control (the maximum speed of the robot is 0.5m/s).

The parameters for the VFH can also be defined by the user and the LiDAR and mask size are shown. The VFH parameters have been included as controls because the thresholds for one environment might not be the most suitable in a different environment. For an environment with narrow paths the thresholds could be lowered allowing the robot to pass more closely to the obstacles.

The data obtained from the LiDAR sensor is converted to Cartesian coordinates and shown on a graph. The path coordinates in metres are shown in array format and in the cell coordinates are displayed on a graph.

As the objective of the software is for the robot to create a path and move to a goal, the robot centre velocities, wheel velocities and data from the IMU sensors has not been displayed on the user interface. This data is processed within the software and was not considered necessary on the front panel.



Robot Position

Initial Position		Goal Position		Estimated Position	
x0	0 m	xf	5.5 m	0	5.833 m
y0	0 m	yf	2.5 m		7.682 m
theta0	0 deg	thetaf	90 deg		1.457 rad

Max. forward speed (max = 0.5) 0.3 m/s

Dist. to Nearest Obstacle: 1.18 m

Position Error:

Mean Error x 0.0062 m

Mean Error y 0.0016 m

name or relative path: grid1.txt

VFH Parameters

Inner Thres.	0.25
# bins	24
Mask. Size	24
Outer Thres.	0.35
Obst. Clearance	0.3
Hist. Size	24
Lidar Size	29

Path (index 0, 0)

0	0	1
0.5	0	2
1	0.5	3
1.5	0.5	4
2	0.5	5
2.5	1	6
3	1.5	7
3.5	1.5	8
4	1.5	9
4.5	2	10
5	2.5	11
5.5	2.5	12
0	0	

Figure 4.16 User Interface

4.4 Flowcharts of the Software

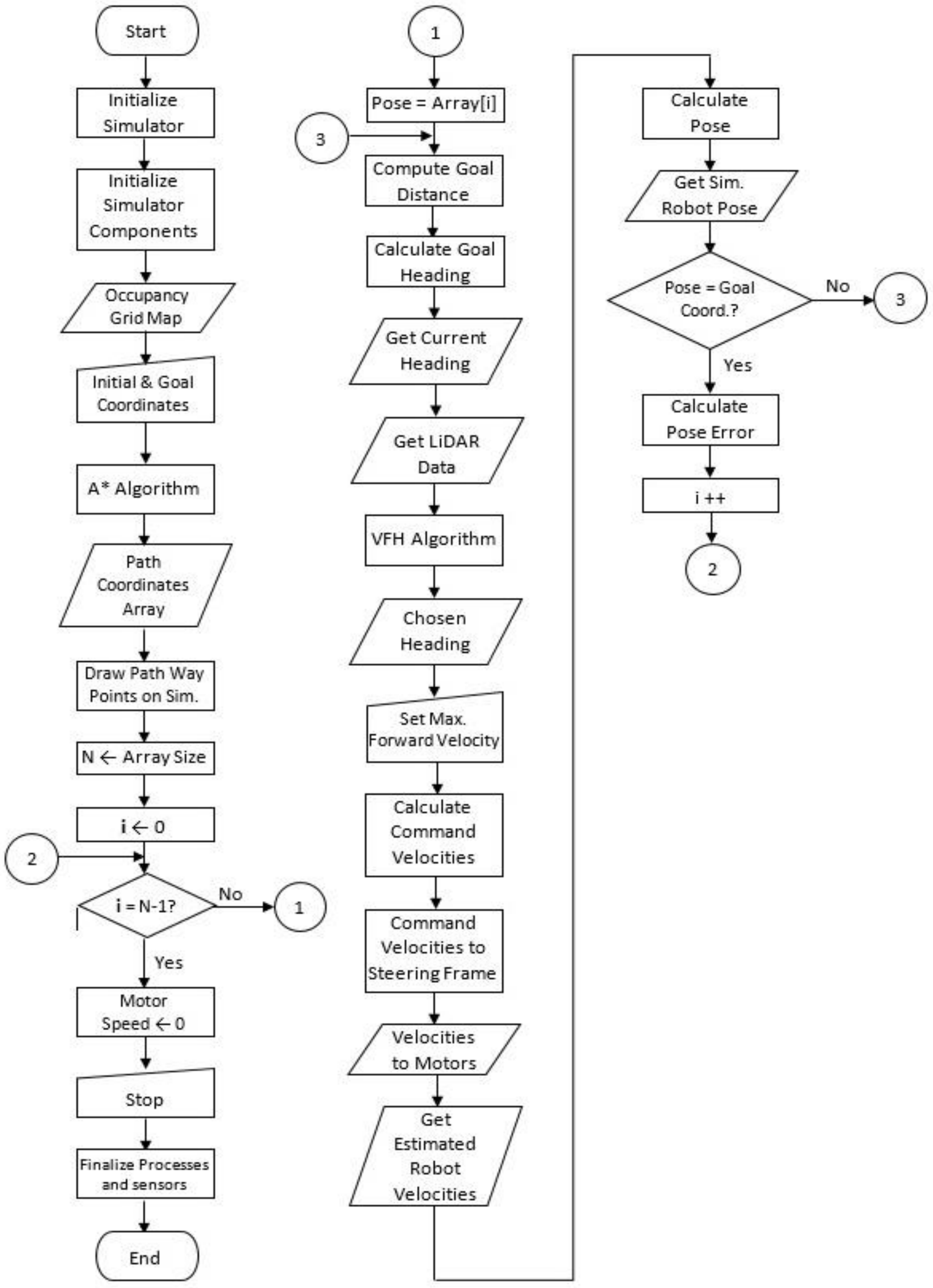
The flow chart of the software developed represents the different processes carried out in order for the robot to reach the goal.

The algorithms that have been developed for the robot can be shown in flowcharts. The flowcharts consist of two main parts:

- The main program containing the initialization and global path planning.
- The navigation to goal and obstacle avoidance algorithms.

Once the components of the program have been initialized and the user has defined the initial and final coordinates, the A* algorithm is performed, obtaining an array of path coordinates. These coordinates are used as sub-goals inside a for loop which is executed once for each pair of x and y coordinates. Once the final goal has been reached, the motor speeds are set to 0 and the processes for each sensor, the robot and the simulator are finalized.

The second part of the program starts by calculating the distance and heading to the goal and obtaining all the necessary inputs for the VFH algorithm. Once the chosen heading has been calculated, the command velocities for the robot are sent to the steering frame to be converted into the left and right wheel velocities. These are then sent to the motors. The encoders read the estimated velocities and calculate the current position of the robot. This is all performed in a while loop that executes until the sub-goal has been reached.


Start
Initialize Simulator
Initialize Simulator Components
Occupancy Grid Map
Initial & Goal Coordinates
A* Algorithm
Path Coordinates Array
Draw Path Way Points on Sim.
N ← Array Size
i ← 0
2
i = N-1?
No
1
Yes
Motor Speed ← 0
Stop
Finalize Processes and sensors
End
1
Pose = Array[i]
3
Compute Goal Distance
Calculate Goal Heading
Get Current Heading
Get LiDAR Data
VFH Algorithm
Chosen Heading
Set Max. Forward Velocity
Calculate Command Velocities
Command Velocities to Steering Frame
Velocities to Motors
Get Estimated Robot Velocities
Calculate Pose
Get Sim. Robot Pose
Pose = Goal Coord.?
No
3
Yes
Calculate Pose Error
i ++
2

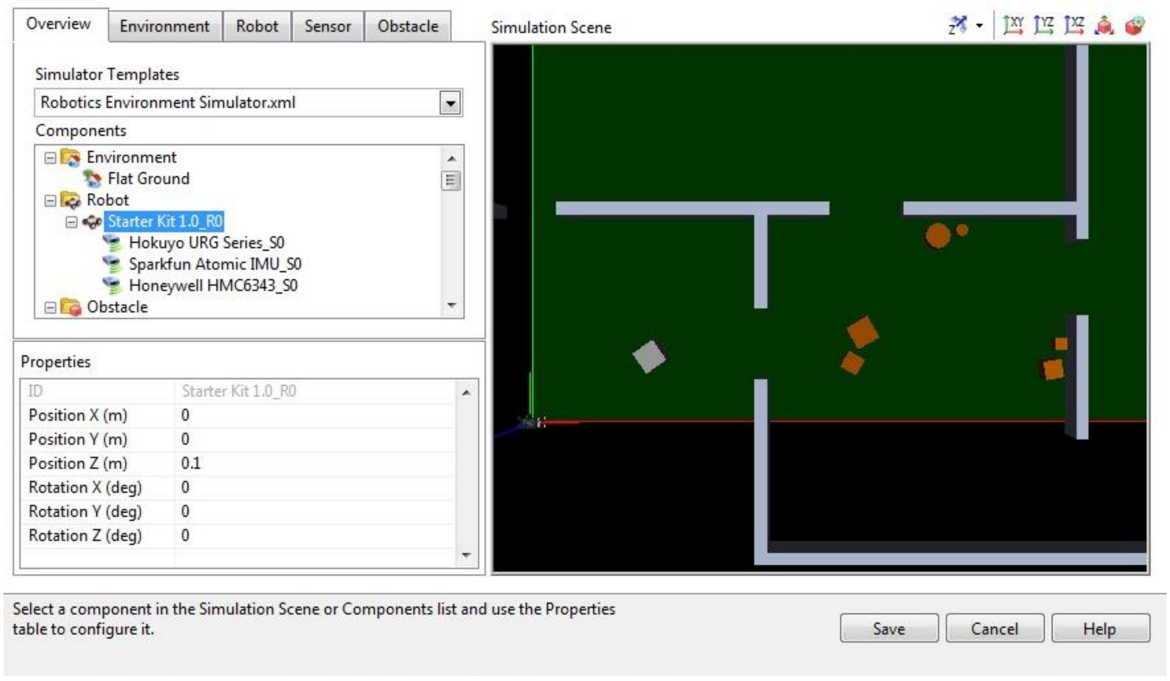
CHAPTER 5

TESTS AND RESULTS

5.1 Algorithms Tested in the Customized Simulated Environment

The first test carried out consists of the robot simply arriving to the goal position using the A* algorithm combined with the Vector Field Histogram. For this test, unknown objects have been added to the environment which are not represented in the occupancy grid map. As the aim of this experiment is to test the path planning and obstacle avoidance algorithms, the exact position of the robot is obtained directly from the simulator.

The simulator environment for this experiment is shown in Figure 5.1.



Simulator Templates

Robotics Environment Simulator.xml

Components

- Environment
  - Flat Ground
- Robot
  - Starter Kit 1.0_R0
    - Hokuyo URG Series_S0
    - Sparkfun Atomic IMU_S0
    - Honeywell HMC6343_S0
- Obstacle

Properties

ID	Starter Kit 1.0_R0
Position X (m)	0
Position Y (m)	0
Position Z (m)	0.1
Rotation X (deg)	0
Rotation Y (deg)	0
Rotation Z (deg)	0

Select a component in the Simulation Scene or Components list and use the Properties table to configure it.

Figure 5.1 Robot environment for first test

For this test, the starting pose is at the origin with a heading of 0 degrees (0, 0, 0). The target is situated at (10, 0.5, 180) (Figure 5.2).

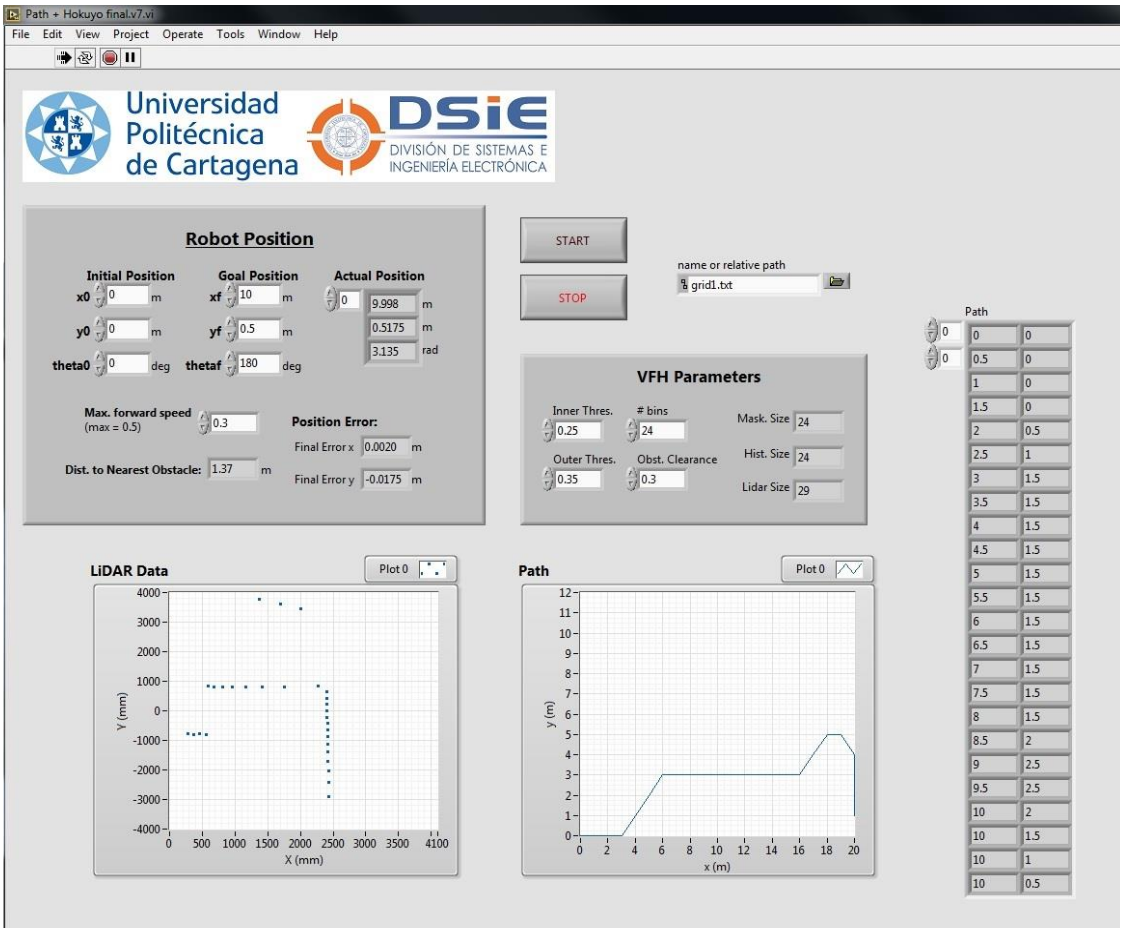
The thresholds chosen for this case are:

- Inner threshold: 0.25m
- Outer threshold: 0.35m
- Obstacle clearance needed: 0.3m

The maximum forward speed chosen is 0.3m/s.

The robot has been programmed so that when an object is detected closer than 0.4 metres, and the robot is within 0.3 metres of the next path waypoint, this point is bypassed and the robot continues to the next sub-goal (Figure 5.5) using the vector field histogram. This avoids the robot from continuously circling an object that lies on or too near to the path originally calculated.

The final error with respect to the goal is also calculated. The error for each waypoint has not been calculated in this case, as the robot will leave the path to avoid colliding with the obstacles (Figure 5.4), therefore to calculate the error at each point along the path would not be appropriate.


Path + Hokuyo final.v7.vi
File Edit View Project Operate Tools Window Help
Universidad Politécnica de Cartagena
DSIE DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA
START
STOP
LiDAR Data
Plot 0
Y (mm)
X (mm)
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
0
500
1000
1500
2000
2500
3000
3500
4100
Path
Plot 0
y (m)
x (m)
12
11
10
9
8
7
6
5
4
3
2
1
0
0
2
4
6
8
10
12
14
16
18
20

Robot Position

Initial Position		Goal Position		Actual Position	
x0	0 m	xf	10 m	0	9.998 m
y0	0 m	yf	0.5 m		0.5175 m
theta0	0 deg	thetaf	180 deg		3.135 rad

Max. forward speed (max = 0.5): 0.3

Dist. to Nearest Obstacle: 1.37 m

Position Error:
Final Error x 0.0020 m
Final Error y -0.0175 m

name or relative path: grid1.txt

VFH Parameters

Inner Thres.	0.25	# bins	24
Outer Thres.	0.35	Obst. Clearance	0.3
Mask. Size	24		
Hist. Size	24		
Lidar Size	29		

Path (index 0, 0)

0	0
0.5	0
1	0
1.5	0
2	0.5
2.5	1
3	1.5
3.5	1.5
4	1.5
4.5	1.5
5	1.5
5.5	1.5
6	1.5
6.5	1.5
7	1.5
7.5	1.5
8	1.5
8.5	2
9	2.5
9.5	2.5
10	2
10	1.5
10	1
10	0.5

Figure 5.2 User interface

The path chosen by the robot to reach the goal can be seen in Figure 5.3. The objects marked 1 and 2 are not represented in the occupancy grid map, so when the A-star algorithm is performed these are not considered.

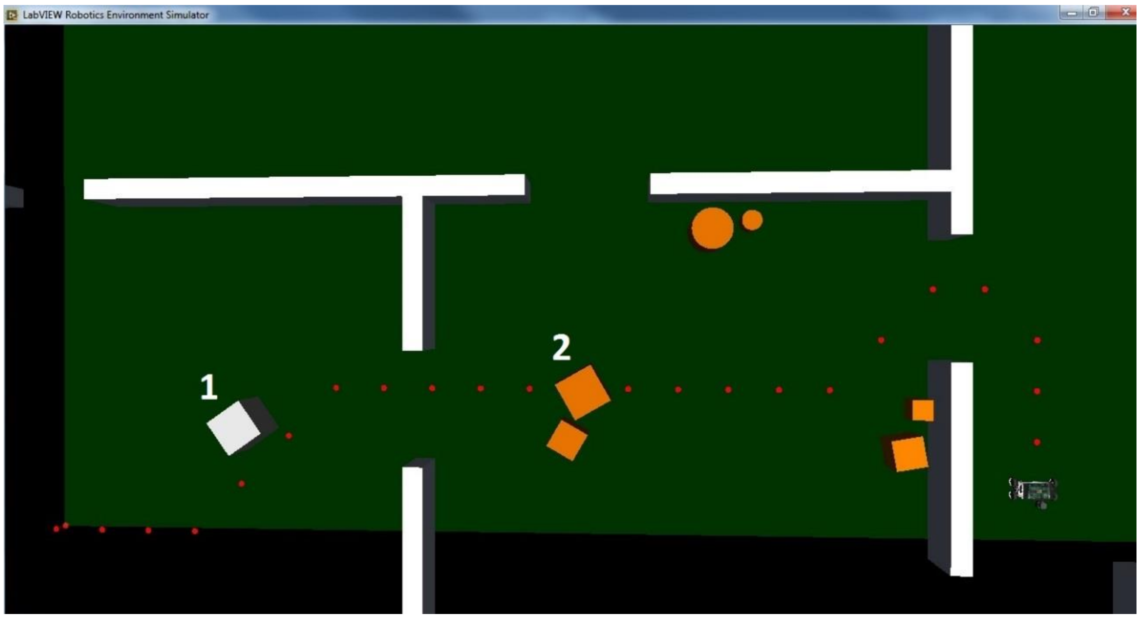


Figure 5.3 Path chosen by the robot in the first simulation

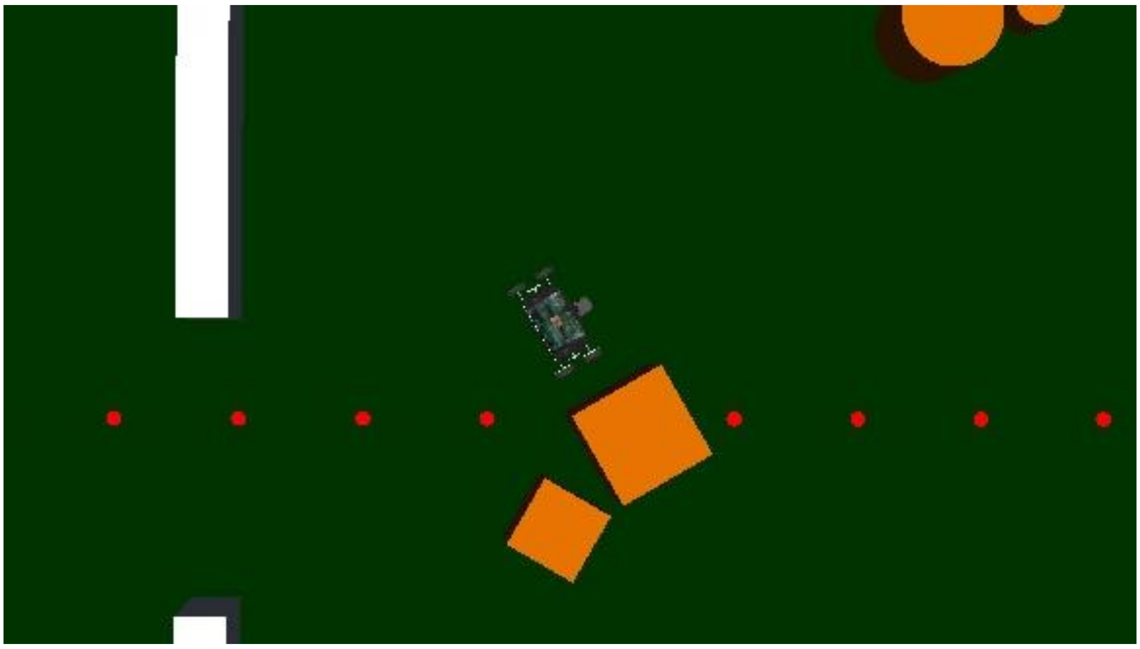


Figure 5.4 Obstacle avoidance

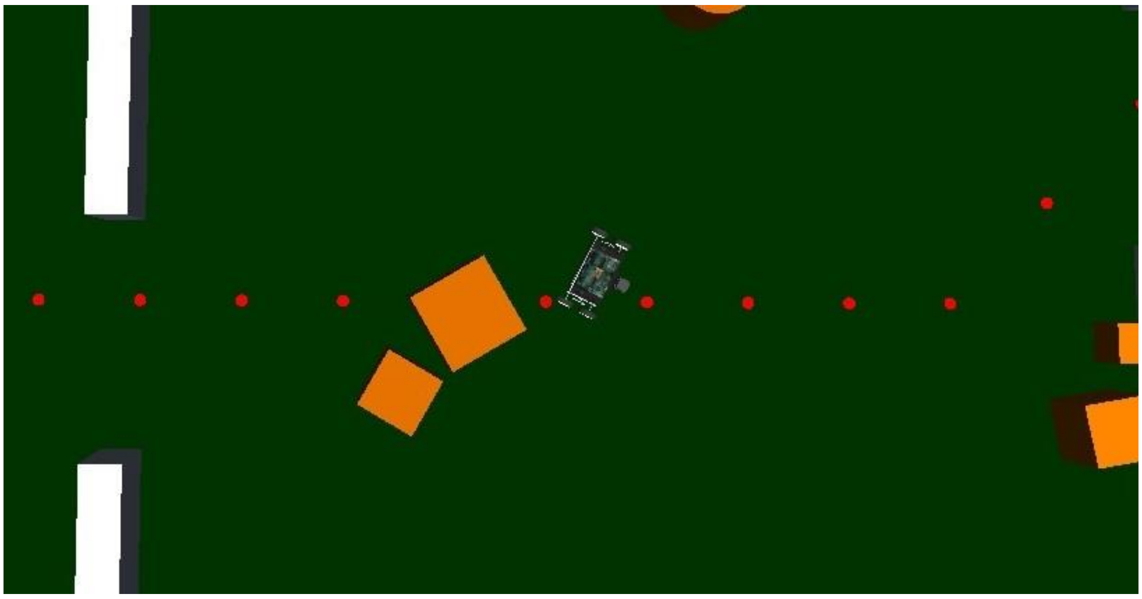


Figure 5.5 Robot re-joins the planned path once past the obstacle

The simulation was repeated five times and the final error in metres has been calculated for the x and y coordinates. The results are shown in Table 5.1.

Test	*Position X*	*Position Y*	*Abs. Error X*	*Abs. Error Y*	*% Error X*	*% Error Y*
1	9,9990	0,4834	0,0010	0,0166	0,010	3,213
2	9,9994	0,4820	0,0006	0,0180	0,006	3,475
3	10,0008	0,4830	0,0008	0,0170	0,008	3,288
4	10,0006	0,4807	0,0006	0,0193	0,006	3,717
5	10,0007	0,4829	0,0007	0,0171	0,007	3,307
Mean	**10,0001**	**0,4824**	**0,0007**	**0,0176**	**0,007**	**3,400**

Table 5.1 Final positions and errors in metres

The results obtained in each simulation are very consistent and the errors obtained were minimal (Figure 5.6), for this reason the simulation was only repeated five times. The errors in x and y were of 0.007% and 3.4% respectively.

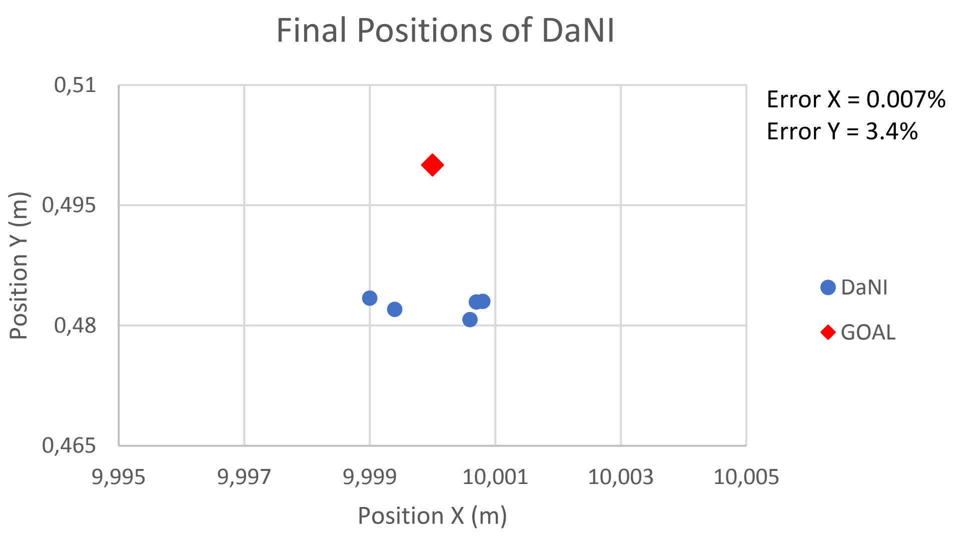


Figure 5.6 Final Positions of DaNI after the first simulation

The mean absolute errors for this experiment are of 0.7 mm in the horizontal direction and 17.6 mm in the vertical direction. These results show that the combination of the algorithms was effective.

5.2 Test with Dead Reckoning in the Customized Simulated Environment

The second test consists of the robot performing a simple trajectory, with a previously unknown obstacle and using dead reckoning. This test will be carried out at different speeds to compare the errors at different speeds.

For this test, instead of wiring the real position to the VFH and path calculation algorithms, the position calculated from the encoder data is used. The starting pose is at the origin with a heading of 0 degrees (0, 0, 0). The target is situated at (5.5, 2.5, 90). The path chosen is shown in Figure 5.7.

The thresholds chosen for the following tests were the following:

- Inner threshold: 0.25m
- Outer threshold: 0.35m
- Obstacle clearance needed: 0.3m

The obstacle marked “1” is unknown to the robot and lies within the inner threshold range, meaning that the robot will not be able to reach the waypoint marked “2” without a collision, even though the obstacle seems to lie behind and to the side of this point.

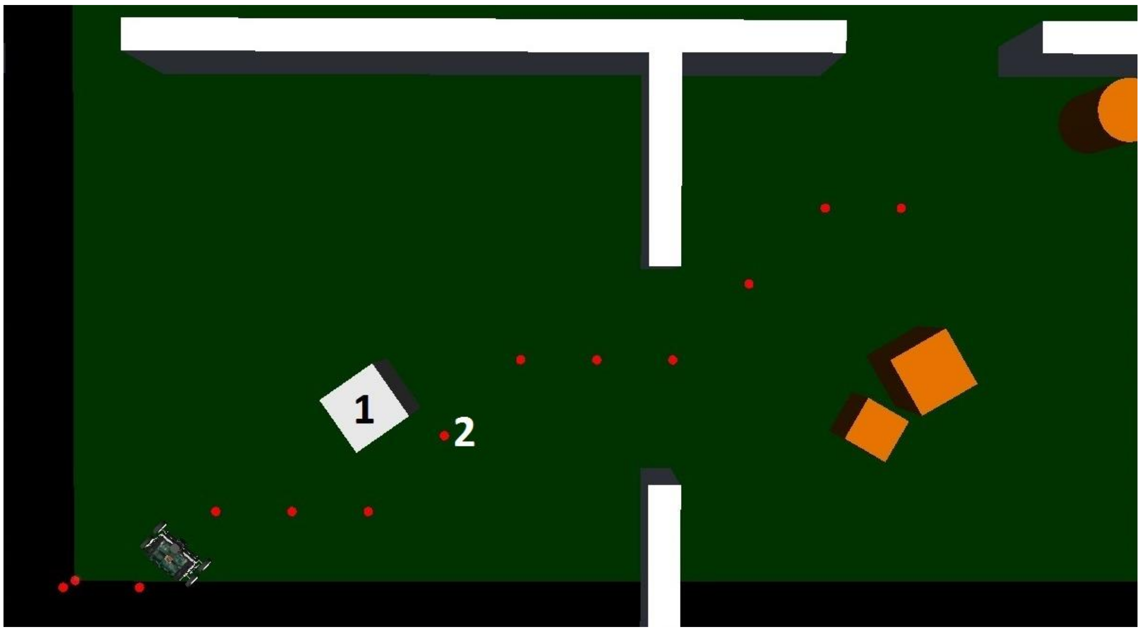
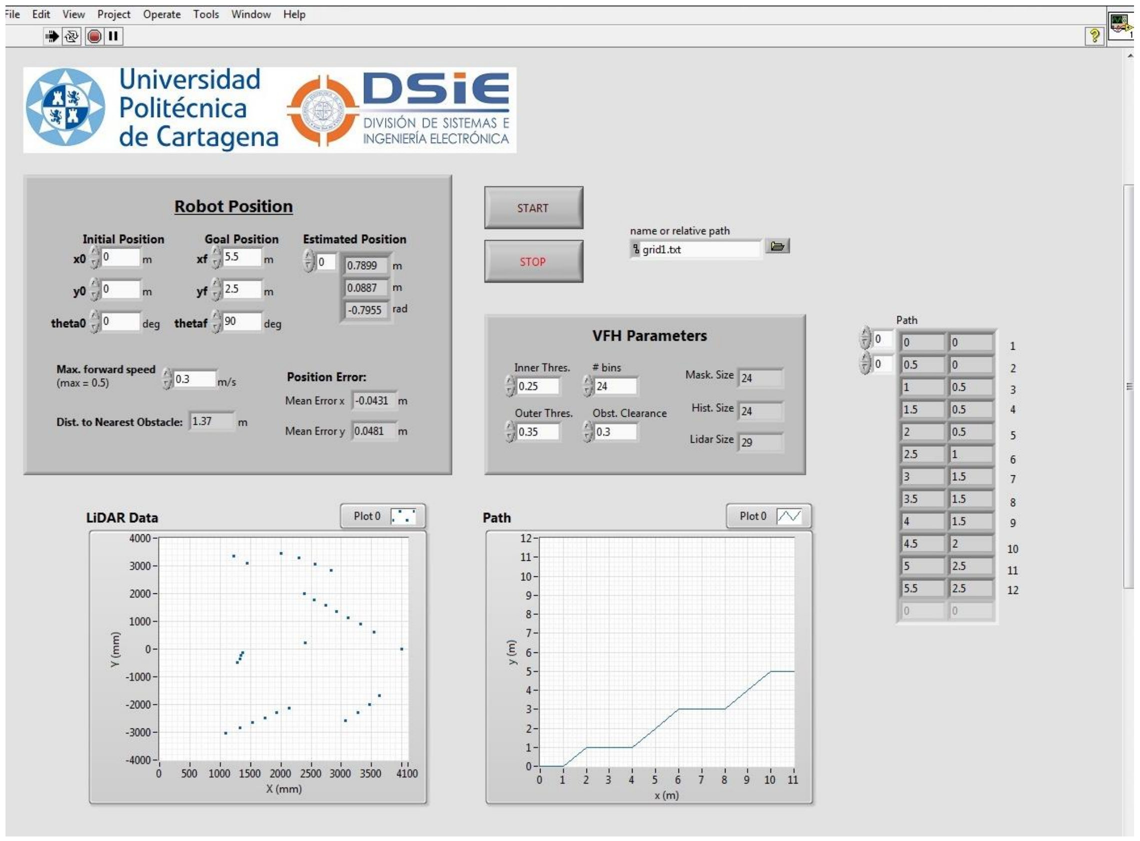


Figure 5.7 Path chosen for the second set of tests

5.2.1 Test 1: Maximum forward speed of 0.3m/s

The first test was performed with a maximum speed of 0.3m/s and a maximum angular velocity of 2 rad/s. The path calculated for this test is shown on the user interface (Figure 5.8) and marked in the robot simulation environment (Figure 5.9).


File Edit View Project Operate Tools Window Help
Universidad Politécnica de Cartagena
DSIE DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA
START
STOP
LiDAR Data
Plot 0
Y (mm)
X (mm)
Path
Plot 0
y (m)
x (m)

Robot Position

Initial Position		Goal Position		Estimated Position
x0	0 m	xf	5.5 m	0.7899 m
y0	0 m	yf	2.5 m	0.0887 m
theta0	0 deg	thetaf	90 deg	-0.7955 rad

Max. forward speed (max = 0.5): 0.3 m/s

Dist. to Nearest Obstacle: 1.37 m

Position Error:
Mean Error x: -0.0431 m
Mean Error y: 0.0481 m

name or relative path: grid1.txt

VFH Parameters

Inner Thres.: 0.25
# bins: 24
Outer Thres.: 0.35
Obst. Clearance: 0.3
Mask. Size: 24
Hist. Size: 24
Lidar Size: 29

Path

0	0	1
0.5	0	2
1	0.5	3
1.5	0.5	4
2	0.5	5
2.5	1	6
3	1.5	7
3.5	1.5	8
4	1.5	9
4.5	2	10
5	2.5	11
5.5	2.5	12
0	0	

Figure 5.8 User Interface with estimated position

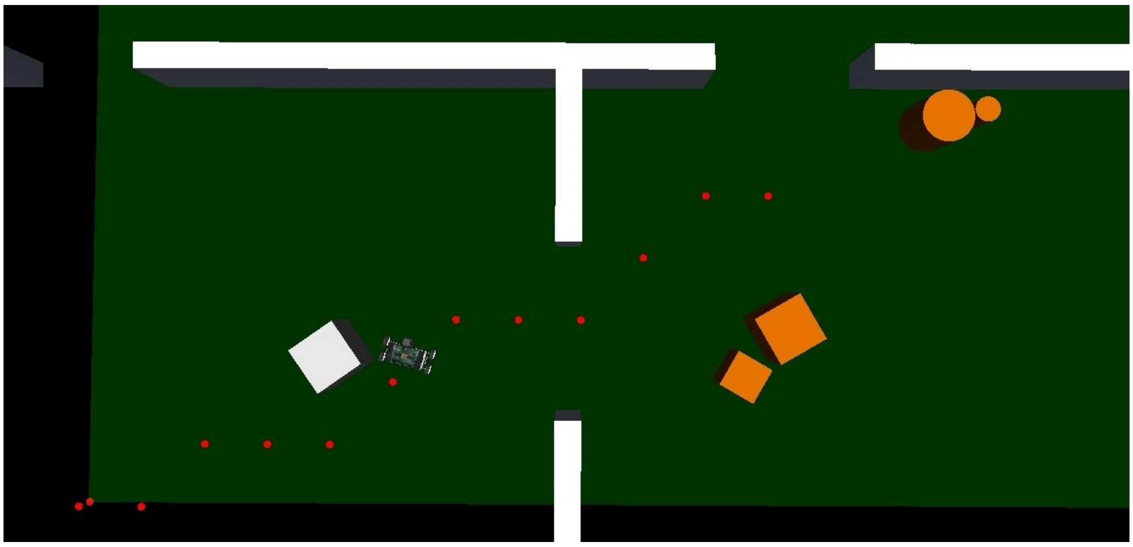


Figure 5.9 The robot leaves the path enough to avoid the obstacle

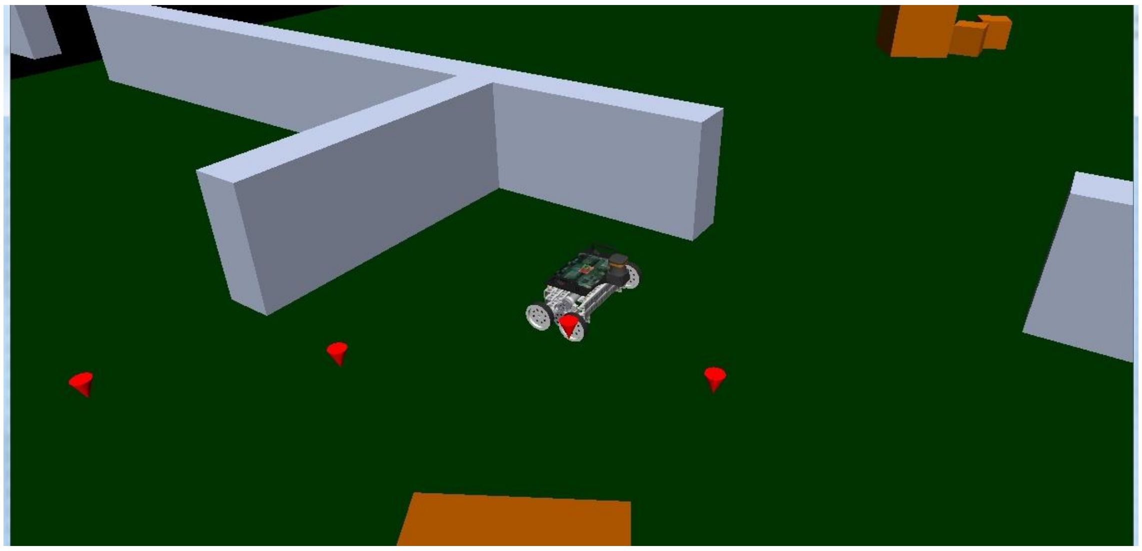


Figure 5.10 Final position obtained in test with dead reckoning

The simulation was performed ten times, with the final errors in metres shown in Table 5.2.

Test	*Abs. Error X*	*Abs. Error Y*	*% Error X*	*% Error Y*
1	0,327	0,183	5,61	6,82
2	0,242	0,055	4,21	2,15
3	0,313	0,145	5,38	5,48
4	0,293	0,049	5,06	1,92
5	0,332	0,029	5,69	1,15
6	0,295	0,092	5,09	3,55
7	0,312	0,012	5,37	0,48
9	0,392	0,065	6,65	2,53
10	0,214	0,034	3,75	1,34
Mean	**0,302**	**0,081**	**5,20**	**2,83**

Table 5.2 Errors in metres obtained at 0.3m/s

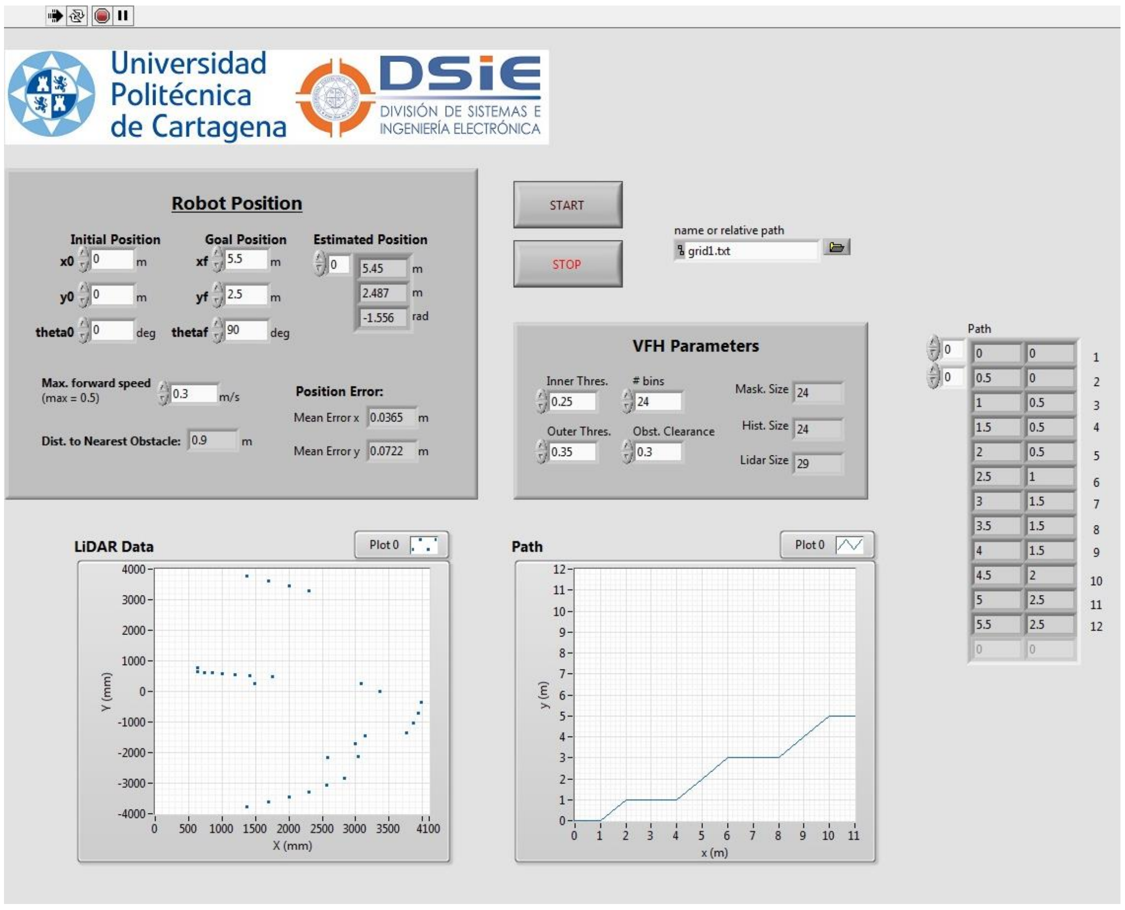
In the eighth simulation, whilst trying to avoid the wall, the robot went off course inside the first room and never reached the goal. This test was not included in the calculation of the mean absolute error.

The absolute errors obtained for the 10 tests were between 0.21m and 0.39m in x and between 0.012m and 0.183m in y. The mean absolute error in x (0.302m) was over three times greater than in y (0.081) (Figure 5.10). This is most likely due to the fact the robot travels almost twice as far in x than in y, therefore as more distance is covered, more error is made.

5.2.2 Test 2: Dead reckoning correction using constants

In order to try and reduce the error in both directions, in the position estimation calculations, the x and y coordinates have been multiplied by constants to improve the precision. The initial and final poses are the same as in the previous test. The maximum forward speed is 0.3 m/s and the maximum angular velocity 2 rad/s. The VFH parameters and thresholds are the same as those used in the previous test (

Figure 5.11).


Universidad Politécnica de Cartagena
DSIE DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA

Robot Position

Initial Position		Goal Position		Estimated Position	
x0	0 m	xf	5.5 m	0	5.45 m
y0	0 m	yf	2.5 m		2.487 m
theta0	0 deg	thetaf	90 deg		-1.556 rad

Max. forward speed (max = 0.5) 0.3 m/s

Dist. to Nearest Obstacle: 0.9 m

Position Error:
Mean Error x 0.0365 m
Mean Error y 0.0722 m

START
STOP

name or relative path
grid1.txt

VFH Parameters

Inner Thres. 0.25
# bins 24
Mask. Size 24
Outer Thres. 0.35
Obst. Clearance 0.3
Hist. Size 24
Lidar Size 29

Path

0
0

0	0	1
0.5	0	2
1	0.5	3
1.5	0.5	4
2	0.5	5
2.5	1	6
3	1.5	7
3.5	1.5	8
4	1.5	9
4.5	2	10
5	2.5	11
5.5	2.5	12
0	0	

Figure 5.11 GUI for the second test

The constants used were 0.9 for the horizontal direction and 1.15 for the vertical direction. These constants were multiplied as shown in Figure 5.12. As these constants are multiplied, the position calculation error at the start of the trajectory is greater than without the constants. However as the robot travels further, these errors are reduced noticeably.

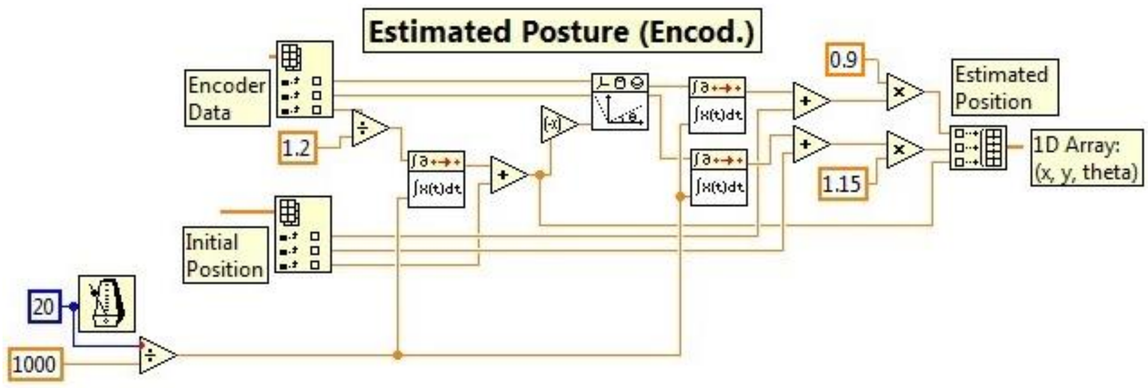


Figure 5.12 Modification of the pose calculations

The simulation was performed 10 times and the results can be seen in Table 5.3.

Test	*Abs. Error X*	*Abs. Error Y*	*% Error X*	*% Error Y*
1	0,036	0,072	0,65	2,80
2	0,071	0,051	1,27	2,00
3	0,029	0,027	0,52	1,07
4	0,025	0,031	0,45	1,22
5	0,033	0,095	0,60	3,66
6	0,039	0,113	0,70	4,32
7	0,004	0,078	0,07	3,03
8	0,022	0,116	0,40	4,43
9	0,028	0,056	0,51	2,19
10	0,035	0,041	0,63	1,61
Mean	**0,032**	**0,068**	**0,58**	**2,63**

Table 5.3 Errors in metres obtained with the constants

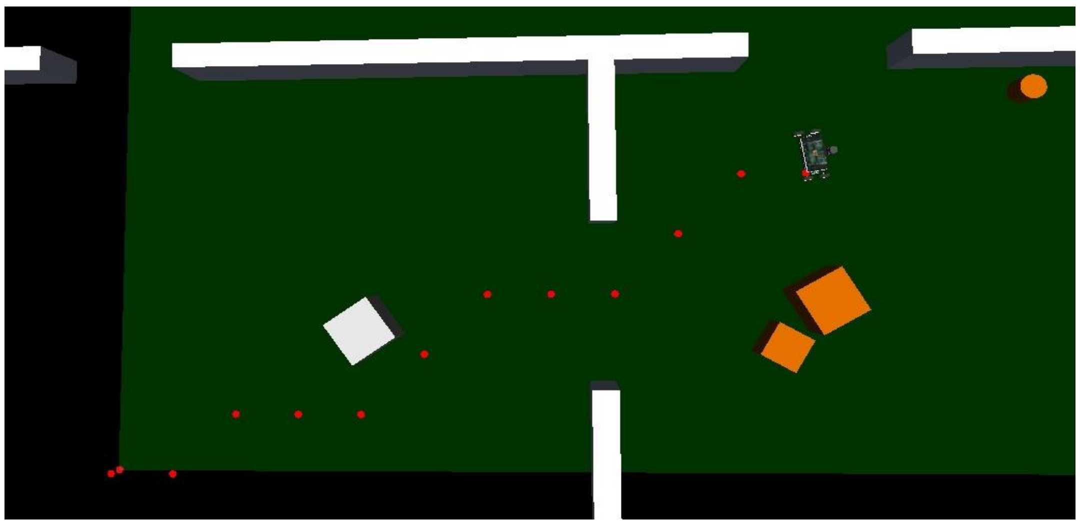


Figure 5.13 Final position after improving dead reckoning

After adding the constants, the mean absolute error in x was 0.042m and 0.068m in y. By adding these constants, the results were improved greatly: in x the mean absolute error decreased by 0.26m and in y the mean absolute error decreased by 0.013m. The robot remained on track in all tests and reached the goal with higher accuracy (Figure 5.14). However, with the horizontal constant, the vertical error was worse and had to be adjusted using a second constant.

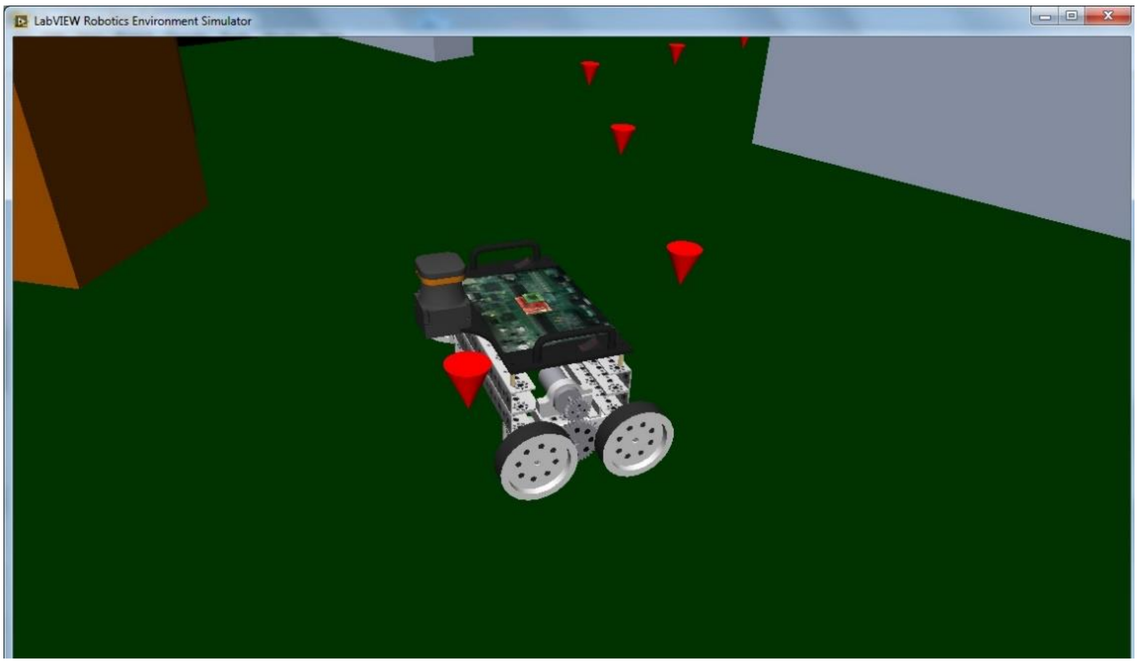


Figure 5.14 Final position after corrections

5.2.3 Test 3: Maximum forward speed of 0.25m/s

The second test was carried with a maximum speed of 0.25m/s and a maximum angular velocity of 2 rad/s. The initial and target positions are the same as in the previous test, therefore the path calculated is also the same, with the unknown object lying close to the path (Figure 5.15).

The results obtained are shown in Table 5.4:

Test	*Abs. Error X*	*Abs. Error Y*	*% Error X*	*% Error Y*
1	0,014	0,024	0,25	0,95
2	0,018	0,005	0,33	0,20
3	0,041	0,082	0,74	3,18
4	0,092	0,118	1,65	4,51
5	0,052	0,087	0,94	3,36
6	0,041	0,069	0,74	2,69
7	0,046	0,088	0,83	3,40
8	0,063	0,079	1,13	3,06
9	0,059	0,087	1,06	3,36
10	0,075	0,084	1,35	3,25
Mean	**0,050**	**0,072**	**0,90**	**2,80**

Table 5.4 Errors in metres obtained at 0.25m/s

The constants used were 0.9 for the horizontal direction and 1.2 for the vertical direction.

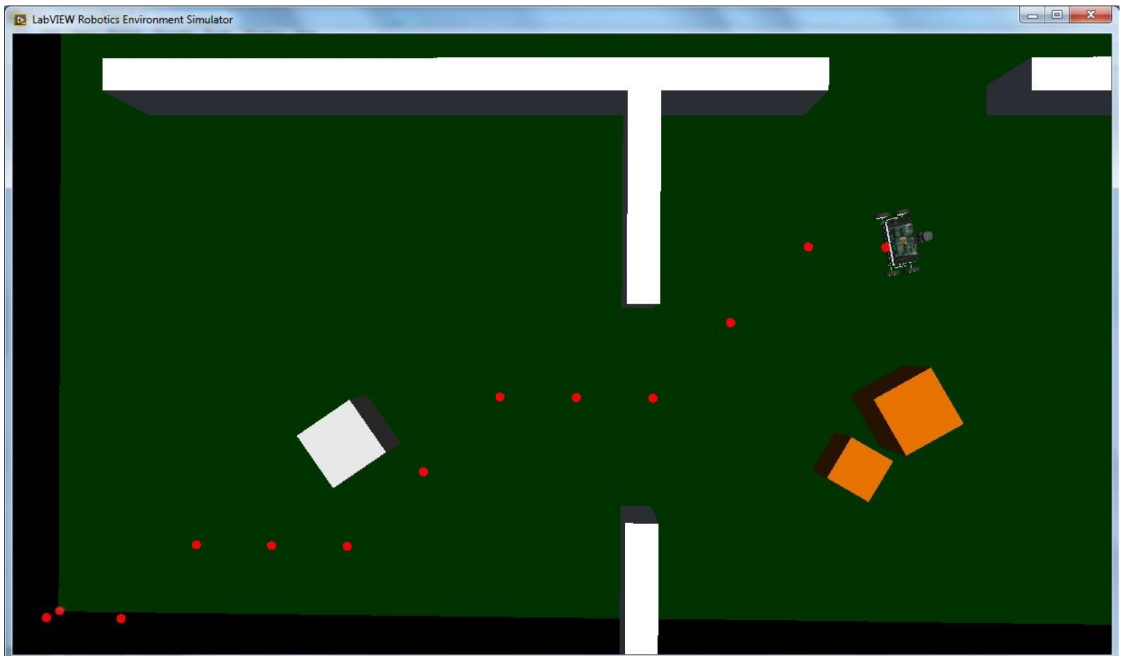


Figure 5.15 Third test simulation

At this speed the mean absolute error in the x is 0.05m and in y is 0.072m. The error in y for this test is higher than that obtained in x. The results obtained for a speed of 0.25 m/s were of the same magnitude as those obtained at a speed of 0.3 m/s.

5.2.4 Test 4: Maximum forward speed of 0.35m/s

The final test was carried out in the same conditions as the previous test, but at a speed of 0.35m/s (Figure 5.16)

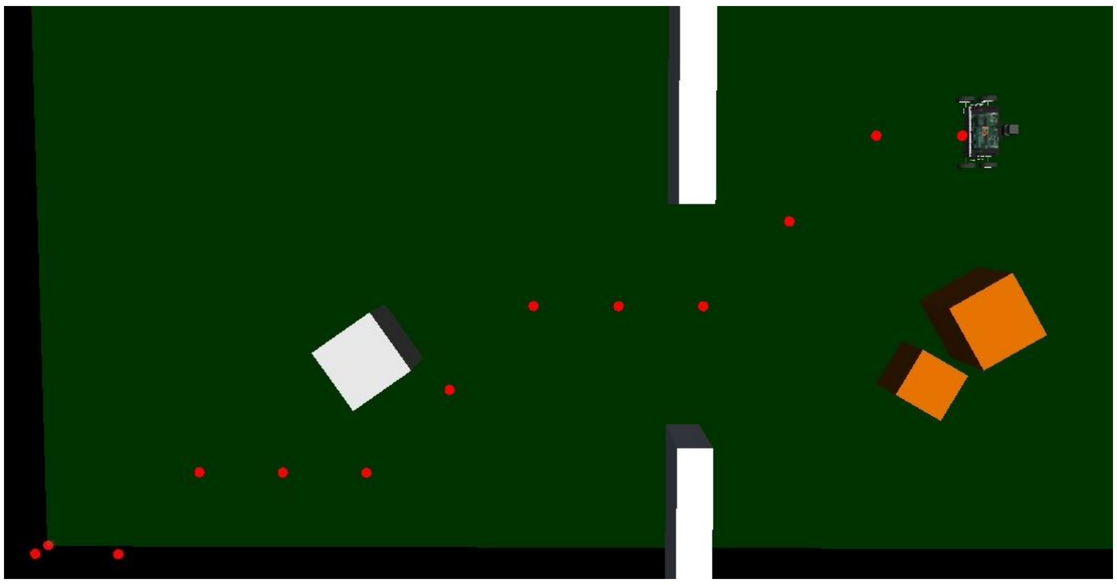


Figure 5.16 Simulation at 0.35m/s

The simulation was repeated ten times. The constants used were 0.9 for the horizontal direction and 1.15 for the vertical direction.

The results obtained are shown in Table 5.5:

Test	*Abs. Error X*	*Abs. Error Y*	*% Error X*	*% Error Y*
1	0,035	0,106	0,63	4,07
2	0,052	0,148	0,94	5,59
3	0,078	0,139	1,40	5,27
4	0,063	0,058	1,13	2,27
5	0,048	0,026	0,87	1,03
6	0,047	0,003	0,85	0,12
7	0,072	0,013	1,29	0,52
8	0,103	0,034	1,84	1,34
9	0,010	0,091	0,18	3,51
10	0,094	0,067	1,68	2,61
Mean	**0,060**	**0,069**	**1,08**	**2,63**

Table 5.5 Errors in metres obtained at 0.35m/s

The mean absolute error obtained in x is 0.06m and that in y is 0.069m.

5.2.5 Comparison of the results from the different tests

The final positions of DaNI at a velocity of 0.3 m/s with and without the error correction are shown in Table 5.6:

Speed 0,3m/s	**Without Correction**		**Corrected**	
Test	**Position X**	**Position Y**	**Position X**	**Position Y**
1	5,173	2,683	5,536	2,572
2	5,258	2,555	5,429	2,551
3	5,187	2,645	5,529	2,473
4	5,207	2,549	5,525	2,531
5	5,168	2,529	5,533	2,595
6	5,205	2,592	5,461	2,613
7	5,188	2,488	5,504	2,578
8	-	-	5,478	2,616
9	5,108	2,565	5,528	2,556
10	5,286	2,466	5,535	2,541
Mean	**5,198**	**2,577**	**5,506**	**2,562**

Table 5.6 Final positions of DaNI

The final position obtained from each test has been represented graphically (Figure 5.17)

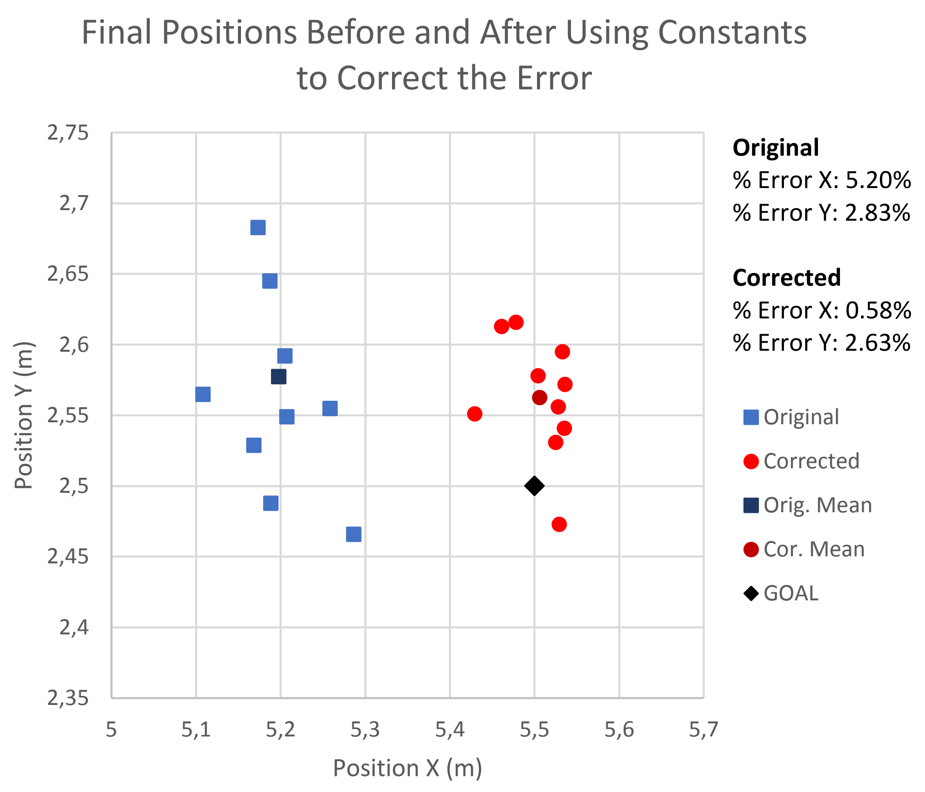


Figure 5.17 Final Positions of DaNI at 0.30m/s

A summary of the mean errors and constants used for correcting the position estimation are shown in Table 5.7:

Test	Mean Abs. Error X	Mean Abs. Error Y	% Error X	% Error Y	Constant X	Constant Y
Original dead reckoning:						
0,3 m/s	0,302	0,081	5,2	2,83	-	-
With corrected dead reckoning:						
0,25 m/s	0,050	0,072	0,90	2,80	0,9	1,2
0,30 m/s	0,032	0,068	0,58	2,63	0,9	1,15
0,35 m/s	0,060	0,069	1,08	2,63	0,9	1,15

Table 5.7 Comparison of mean absolute errors

The mean error obtained at 0.30m/s with the original dead reckoning calculations is 6 times greater in the x direction than that obtained after applying the constants. The errors generated in y are of the same magnitude (0.081 without the constant, 0.072 with the correction).

For forward speeds of 0.25m/s and 0.35m/s the mean absolute error obtained in x (0.05m and 0.06 respectively) was almost double that for a speed of 0.30m/s (0.032). For the y coordinate, the mean absolute errors for each velocity only varied 4mm (0.072 at 0.25m/s, 0.068 at 0.30m/s and 0.069m/s)

The final positions reached by DaNI after testing the software for different forward velocities are shown in Table 5.8.

Speed	**0,25m/s**		**0,30m/s**		**0,35m/s**	
Test	**Position x**	**Position y**	**Position x**	**Position y**	**Position x**	**Position y**
1	5,514	2,476	5,536	2,572	5,535	2,606
2	5,518	2,505	5,429	2,551	5,448	2,648
3	5,459	2,418	5,529	2,473	5,422	2,639
4	5,408	2,618	5,525	2,531	5,563	2,442
5	5,448	2,587	5,533	2,595	5,548	2,526
6	5,541	2,431	5,461	2,613	5,547	2,503
7	5,454	2,588	5,504	2,578	5,572	2,487
8	5,563	2,421	5,478	2,616	5,397	2,534
9	5,559	2,587	5,528	2,556	5,510	2,591
10	5,425	2,584	5,535	2,541	5,406	2,567
Mean	**5,489**	**2,522**	**5,505**	**2,562**	**5,494**	**2,554**

Table 5.8 Final positions of DaNI with different speeds

The positions from the previous table are shown in Figure 5.18. The results obtained for each speed are evenly distributed, with the results for 0.30m/s being less dispersed than those at other speeds. The best position was reached for a forward velocity of 0.30m/s, with only one simulation with an absolute error greater than 0.039m in x.

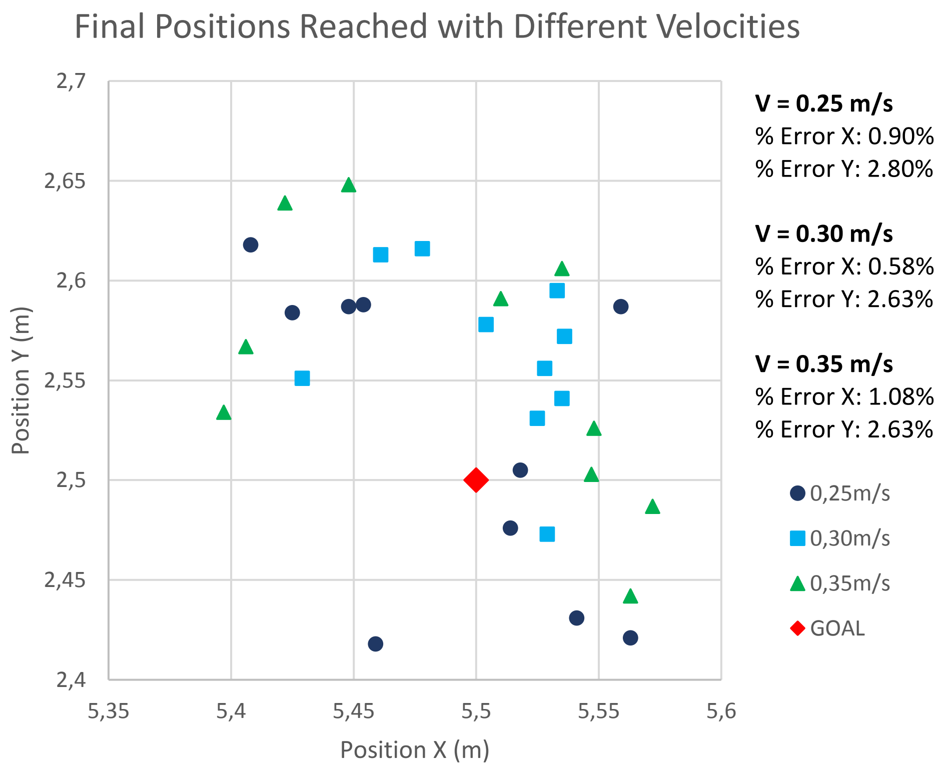


Figure 5.18 Final positions after changing the maximum speed

Table 5.9 shows the average final coordinates that the robot believes are the goal coordinates.

Test	Position x	Position y
Original dead reckoning		
0,3 m/s	5,198	2,577
With corrected Position estimation		
0,25 m/s	5,489	2,522
0,30 m/s	5,505	2,562
0,35 m/s	5,495	2,554

Table 5.9 Average final positions reached by the robot in each test

At 0.3m/s the average x coordinate is greater than the target, whilst at a higher or lower speed, the average position is less than the target. In all cases the y coordinate is greater. The most efficient test carried out was Test 2, at a forward speed of 0.3m/s and using constants to correct the dead reckoning calculations. These coordinates have been represented graphically in Figure 5.19.

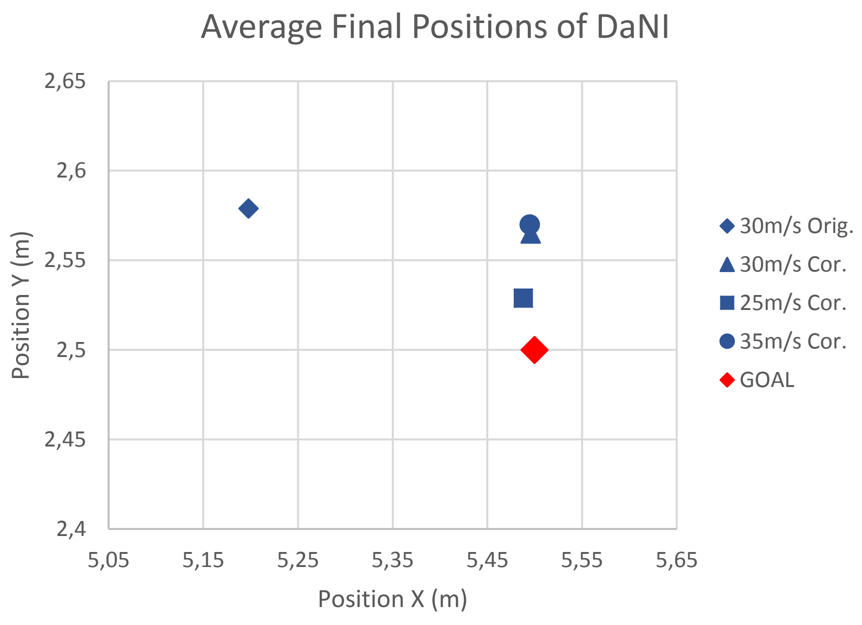


Figure 5.19 Average final positions of the robot

5.3 Comparison of the Software with the Artificial Neural Network Algorithm

5.3.1 Artificial Neural Network

An existing autonomous navigation algorithm for the Starter Kit robot DaNI is the Artificial Neural Network (ANN). The ANN is a machine learning technique which is capable of learning on its own and adapting to changing conditions.

The ANN paradigm is based on the biological nervous system, such as the information processing mechanism of the human brain. The information processing system is composed of many interconnected processing elements (neurons), working together to solve a problem [22].

An artificial neuron is a device with many inputs and a single output. The neuron has two modes of operation, a learning mode and a working mode. In the learning mode, the neuron can be trained to act or not to act depending on the input pattern. In the working mode, when an input pattern that has been learnt previously is detected, the output associated with that input becomes the current output. When an input that has not been learnt is given, the neural network gives the output which corresponds to a taught input pattern that is the most similar to the current input.

5.3.2 ANN Software

The ANN example software is programmed in LabVIEW to be used with the robotics simulator. The ANN control algorithm is applied to the NI Starter Kit 1.0 DaNI robot and is used for trajectory tracking.

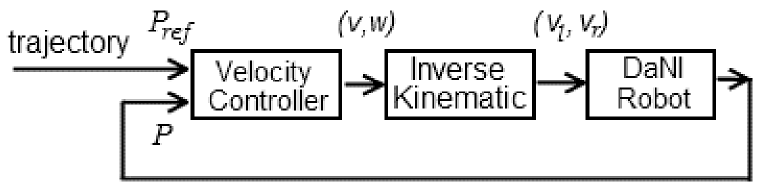
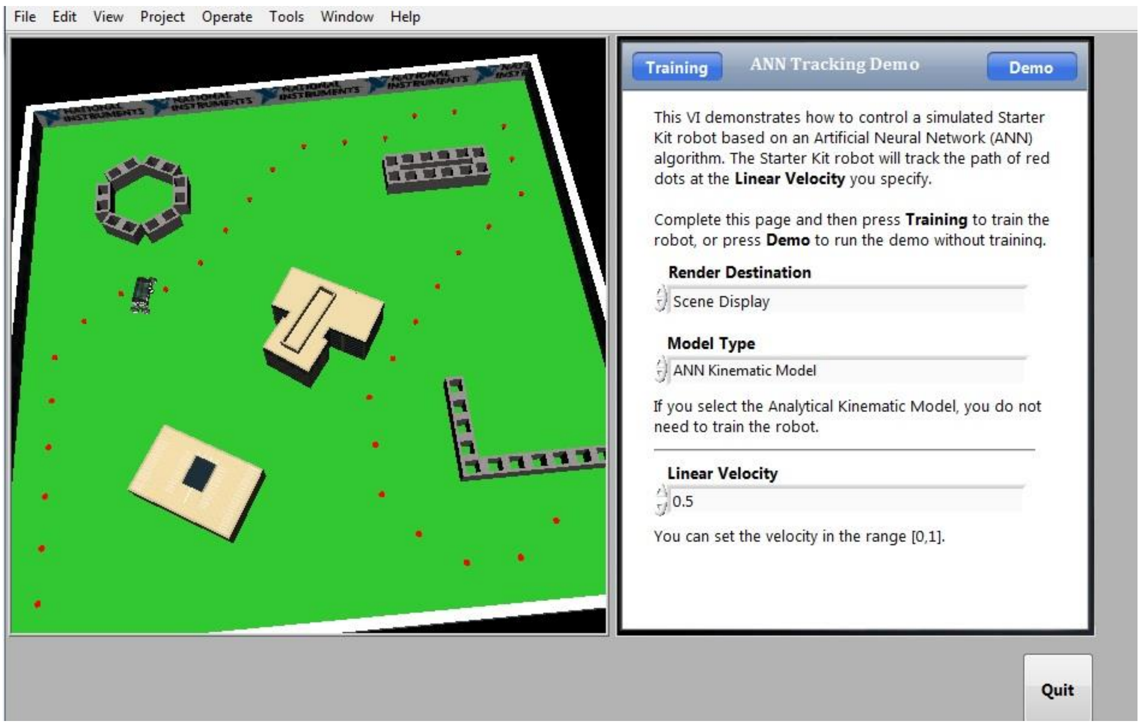


Figure 5.20 Trajectory tracking control

In the control loop diagram (Figure 5.20), P_{ref} are the coordinates of the desired position (x, y, theta) and P is the current position of the robot. The output of the velocity controller is the desired linear and angular velocity (v,w). These velocities are calculated according to the difference between the target position and the current position of the robot. The inverse kinematic model converts the linear and angular velocities to the right and left wheel velocities (v_r,v_l). The right and left wheel velocities are the input velocities applied to the robot.



This VI demonstrates how to control a simulated Starter Kit robot based on an Artificial Neural Network (ANN) algorithm. The Starter Kit robot will track the path of red dots at the **Linear Velocity** you specify.

Complete this page and then press **Training** to train the robot, or press **Demo** to run the demo without training.

Render Destination
Scene Display

Model Type
ANN Kinematic Model

If you select the Analytical Kinematic Model, you do not need to train the robot.

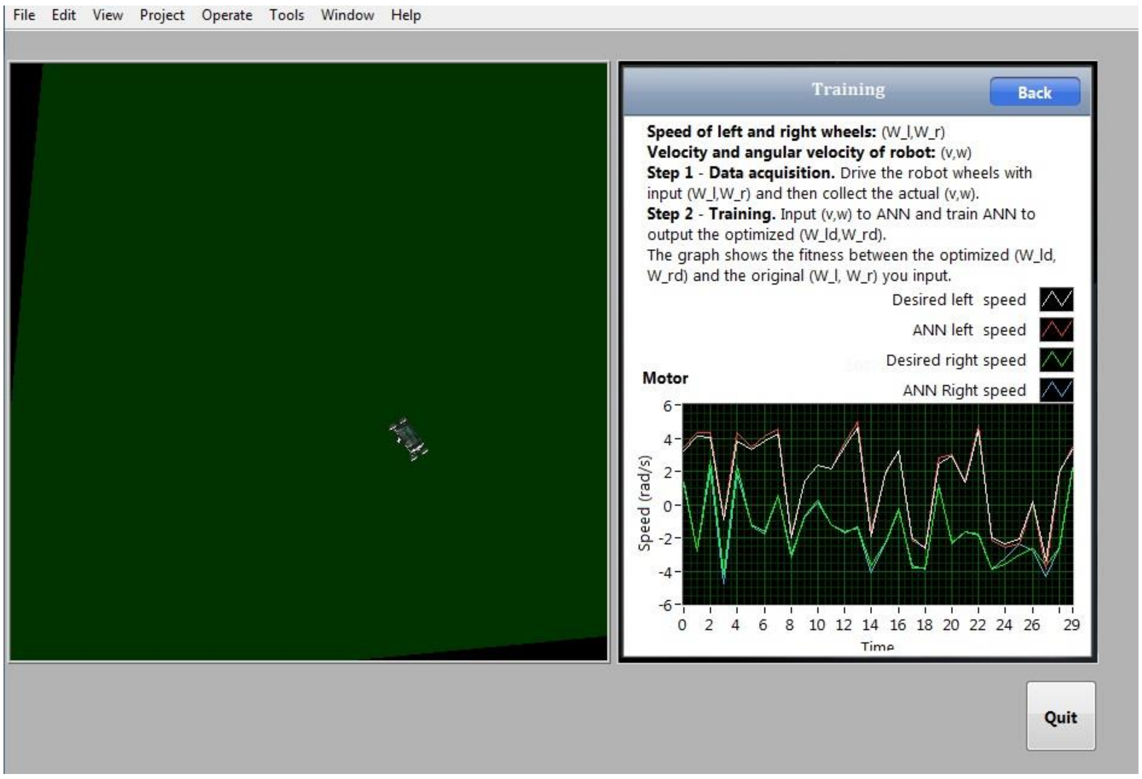
Linear Velocity
0.5

You can set the velocity in the range [0,1].

Figure 5.21 ANN Software user interface

On the front panel of this VI (Figure 5.21) the user can choose the type of model (for this test the ANN Kinematic Model has been chosen. Different linear velocities can also be chosen, within a range of 0 to 1 m/s.

When the VI is executed, for the ANN Kinetic model, the robot must be trained before performing the navigation algorithm (Figure 5.22). The forward and angular velocities are used as the inputs to train the ANN to output the desired right and left wheel velocities. The input layer consists of two neurons (v,w). The internal layer contains 100 neurons and the out layer has 2 neurons (v_r,v_l).



Speed of left and right wheels: (W_l,W_r)
Velocity and angular velocity of robot: (v,w)
Step 1 - Data acquisition. Drive the robot wheels with input (W_l,W_r) and then collect the actual (v,w).
Step 2 - Training. Input (v,w) to ANN and train ANN to output the optimized (W_ld,W_rd).
The graph shows the fitness between the optimized (W_ld, W_rd) and the original (W_l, W_r) you input.

Figure 5.22 ANN Training

Once the simulation and the robot have been initialized, the VI creates a path to the target position. For this a subVI is used (Figure 5.23). This subVI calculates a trajectory by interpolating between positions in a path made up of (x,y) coordinates. The default interpolation method used is the cubic Hermite method. Path is an array of x and y positions and ntimes specifies the interpolation locations between each (x,y) element. The output of this subVI, trajectory, is an array of coordinates with the desired heading and distance from the previous point to the current point (x, y, theta, ds).

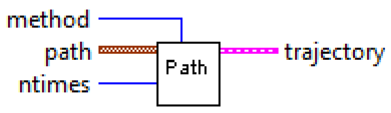


Figure 5.23 Trajectory planning subVI

This trajectory is input to the virtual vehicle posture subVI (Figure 5.24), along with the desired linear velocity of the robot and the time interval at which to calculate the position. This VI then calculates the real velocity and the angular velocity of the robot.

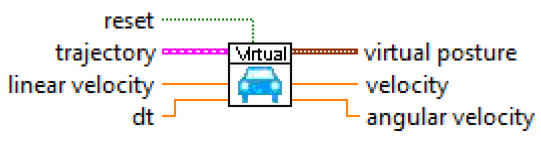


Figure 5.24 Virtual Vehicle Posture subVI

The position of the robot is obtained directly from the simulator by the ANN Get posture subVI (Figure 5.25). This VI returns the current posture information (x, y, theta) of the robot. To use the ANN algorithm on a real robot instead of a simulated robot, the posture of the robot would have to be obtained using the information from sensors.

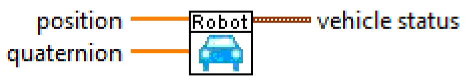


Figure 5.25 ANN Get Posture subVI

The right and left wheel velocities are calculated by the ANN tracking subVI (Figure 5.26). ESN trained specifies the weight of the input, output and feedback and internal weight. Model type is the inverse kinematic model to use. The current and desired positions are obtained from the get posture and virtual vehicle posture subVis. The inputs vref (nan) and wref (nan) specify the reference linear and angular velocity respectively. The calculated wheel velocities obtained are then applied to the motors.

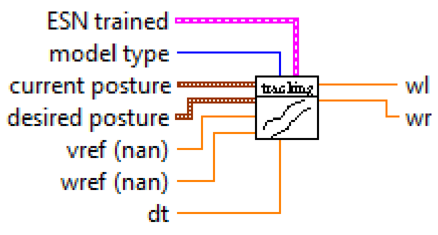


Figure 5.26 ANN Tracking wheel controller subVI

5.3.3 A-Star and Vector Field Histogram Software

To compare the programmed software with the ANN algorithm some modifications were made to the original program to make the test as fair as possible. The biggest difference with the ANN algorithm is that the trajectory chosen is not the shortest as occurs with the A-star algorithm. In order to obtain a longer and more complicated path, extra obstacles were added to the original robot scene environment (Figure 5.27) for the A-star algorithm software. If no additional obstacles were used, the A-star algorithm would complete the path much more efficiently than the ANN algorithm as it would choose the path with the lowest cost.

As the ANN algorithm obtains the robot position directly from the simulator, the program has been modified, so that in both cases the posture of the robot is obtained directly from the simulator. The maximum robot speed in both cases was set to 0.3m/s. However, when the robot is trained by the ANN algorithm, the robot wheel velocities are automatically set according to the distance to the next way point in the trajectory.

The final errors have been calculated comparing the final position of the robot with the goal position. The mean error of the ten simulations is then calculated.

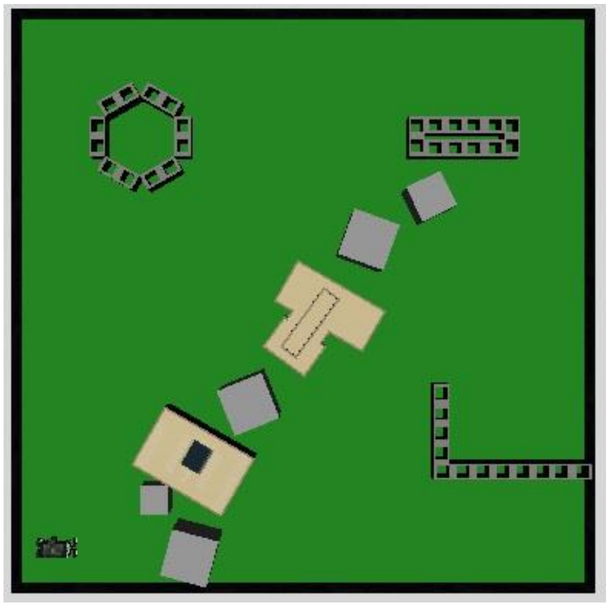


Figure 5.27 Robot scene modified for ANN comparison

5.3.4 Software Comparison Results

To compare the programs, the mean error has been calculated and each program has been run 10 times in the same conditions. In each case the trajectory way points are shown in the simulation as red cone-shaped markers.

The trajectory planned by the A-star algorithm software is shown in Figure 5.28.

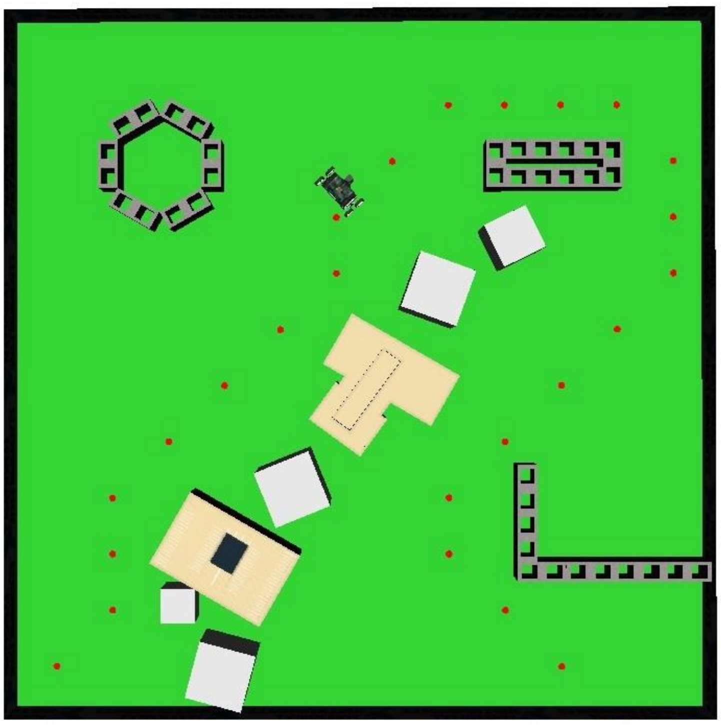
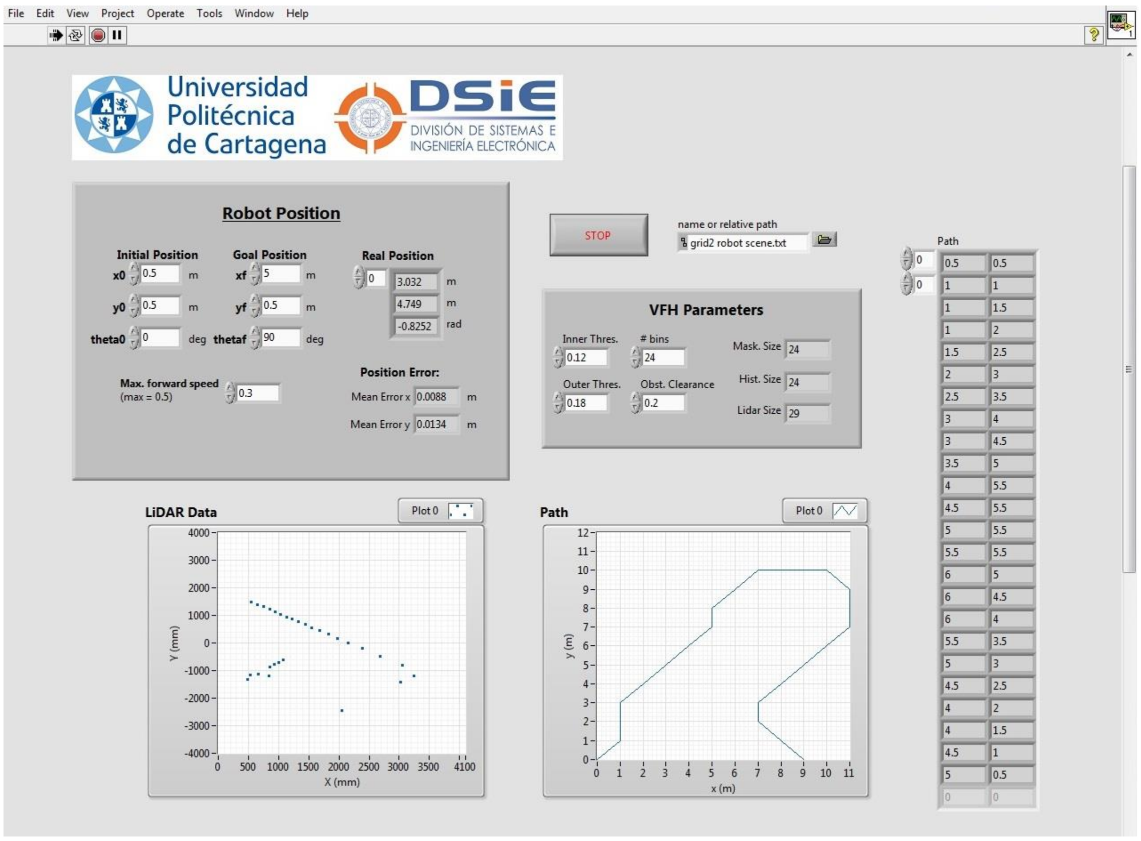


Figure 5.28 Trajectory planned by A-Star algorithm

The mean error calculated has been added to the user interface and the exact position of the robot is shown, as well as the initial and goal position coordinates, in metres (Figure 5.29).


File Edit View Project Operate Tools Window Help
Universidad Politécnica de Cartagena
DSIE DIVISIÓN DE SISTEMAS E INGENIERÍA ELECTRÓNICA
STOP
LiDAR Data
Plot 0
Y (mm)
X (mm)
Path
Plot 0
y (m)
x (m)

Robot Position

Initial Position		Goal Position		Real Position	
x0	0.5 m	xf	5 m	3.032	m
y0	0.5 m	yf	0.5 m	4.749	m
theta0	0 deg	thetaf	90 deg	-0.8252	rad

Max. forward speed (max = 0.5): 0.3

Position Error:
Mean Error x 0.0088 m
Mean Error y 0.0134 m

name or relative path: grid2 robot scene.txt

VFH Parameters

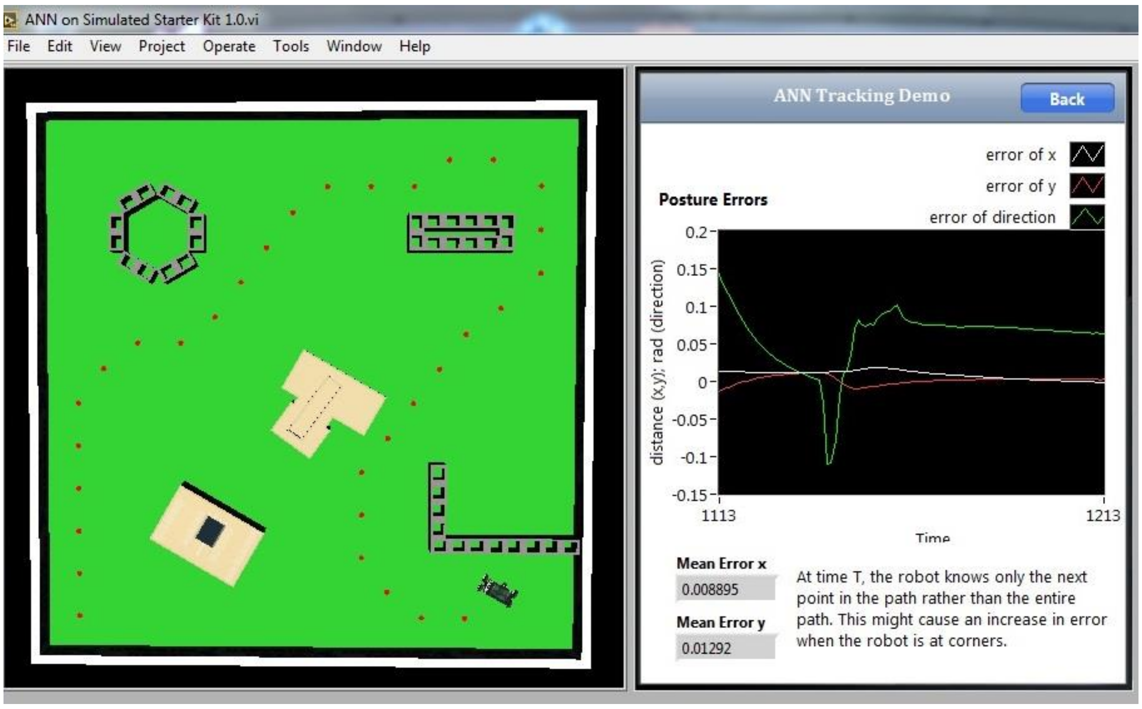
Inner Thres.	0.12
# bins	24
Outer Thres.	0.18
Obst. Clearance	0.2
Mask. Size	24
Hist. Size	24
Lidar Size	29

Path

0.5	0.5
1	1
1	1.5
1	2
1.5	2.5
2	3
2.5	3.5
3	4
3	4.5
3.5	5
4	5.5
4.5	5.5
5	5.5
5.5	5.5
6	5
6	4.5
6	4
5.5	3.5
5	3
4.5	2.5
4	2
4	1.5
4.5	1
5	0.5
0	0

Figure 5.29 User interface used for the comparison

The path chosen by the ANN algorithm software is similar (Figure 5.30):


ANN on Simulated Starter Kit 1.0.vi
File Edit View Project Operate Tools Window Help
ANN Tracking Demo
Back
Posture Errors
error of x
error of y
error of direction
distance (x,y); rad (direction)
Time
1113
1213

Mean Error x 0.008895

Mean Error y 0.01292

At time T, the robot knows only the next point in the path rather than the entire path. This might cause an increase in error when the robot is at corners.

Figure 5.30 ANN Trajectory and user interface

The results obtained for the error for the A-star and VFH algorithm are shown in Table 5.10**:**

Test	***Position X***	***Position Y***	***Abs. Error X***	***Abs. Error Y***	***% Error X***	***% Error Y***
1	5,0062	0,4984	0,0062	0,0016	0,12	0,32
2	5,0056	0,4988	0,0056	0,0012	0,11	0,24
3	5,0051	0,5015	0,0051	0,0015	0,10	0,30
4	5,0059	0,4982	0,0059	0,0018	0,12	0,36
5	5,0055	0,4986	0,0055	0,0014	0,11	0,28
6	5,0065	0,5011	0,0065	0,0011	0,13	0,22
7	5,0052	0,4989	0,0052	0,0011	0,10	0,22
8	5,0053	0,5009	0,0053	0,0009	0,11	0,18
9	5,0065	0,4987	0,0065	0,0013	0,13	0,26
10	5,0065	0,4987	0,0065	0,0013	0,13	0,26
Mean	**5,0058**	**0,4994**	**0,0058**	**0,0013**	**0,12**	**0,26**

Table 5.10 Mean trajectory errors for the developed software

In all ten tests, the robot successfully reached the goal without colliding with any obstacles. The error in x is consistently about three times greater than in y in each test. The mean absolute error for the A* and VFH program in x is 5.8mm and in y 1.3mm.

The trajectory errors obtained with the ANN algorithm are shown in Table 5.11:

Test	***Position X***	***Position Y***	***Abs. Error X***	***Abs. Error Y***	***% Error X***	***% Error Y***
1	5,0088	0,5129	0,0088	0,0129	0,18	2,52
2	5,0069	0,5092	0,0069	0,0092	0,14	1,81
3	5,0102	0,5134	0,0102	0,0134	0,20	2,61
4	5,0097	0,5108	0,0097	0,0108	0,19	2,11
5	5,0094	0,5125	0,0094	0,0125	0,19	2,44
6	5,0073	0,5098	0,0073	0,0098	0,15	1,92
7	5,0074	0,5101	0,0074	0,0101	0,15	1,98
8	5,0097	0,5141	0,0097	0,0141	0,19	2,74
9	5,0051	0,5072	0,0051	0,0072	0,10	1,42
10	5,0061	0,5086	0,0061	0,0086	0,12	1,69
Mean	**5,0081**	**0,5109**	**0,0081**	**0,0109**	**0,16**	**2,12**

Table 5.11 Mean trajectory errors for the ANN algorithm

In all tests the robot performed the planned trajectory successfully and reached the target avoiding all obstacles. The mean absolute error in x is 8.1mm and in y is 10.9mm.

In the horizontal direction, the error obtained for the ANN algorithm is slightly higher than for the software developed using the A-star algorithm. However, in the vertical direction, the error obtained after training the ANN and performing a similar trajectory is around ten times greater than that of the A-star algorithm software.

The final positions that DaNI reached in each part of the experiment have been represented in Figure 5.31. The results obtained with the ANN are more dispersed than the results for the developed software in this project. For the A* and VFH software, the final position reached by the robot was more consistent than that reached with the ANN software. The mean absolute errors in x and y for the A* and VFH software were 5.8mm and 1.3mm respectively. The mean absolute errors for the ANN software were 8.1mm and 10.9mm for x and y respectively. The error in y was around ten times greater with the ANN software (2.12%) than that for the A* and VFH software (0.26%)

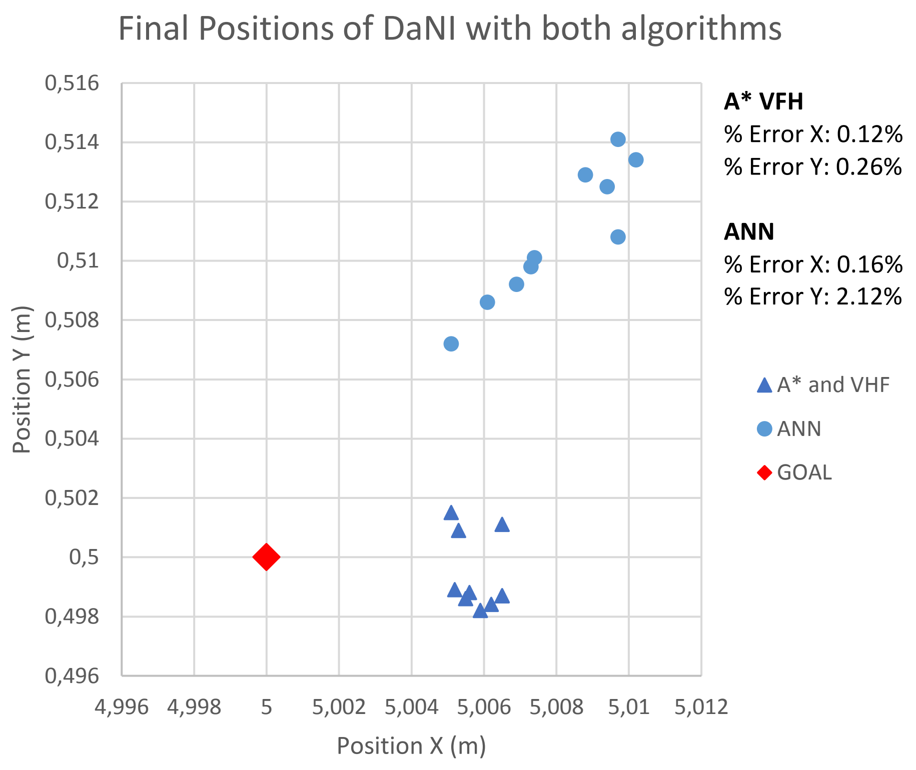


Figure 5.31 Final Positions of DaNI in the third simulation

CHAPTER 6

CONCLUSIONS AND FUTURE PROJECTS

6.1 Conclusions

The general objective of this project was to study different navigation algorithms for mobile robots and develop trajectory planning and obstacle avoidance software using a LiDAR sensor. In this chapter the specific objectives of the project will be discussed.

- A state of the art study was carried out, about the current research situation in mobile robotics. The robots currently available and their applications in real world scenarios were studied along with the sensors used for perception systems in different environments and the most common global and local navigation algorithms.

- Autonomous navigation software was developed in LabVIEW for a small mobile robot platform: DaNI. The software consisted of global navigation that consisted in path planning using the A* algorithm and local navigation for obstacle avoidance using the Vector Field Histogram algorithm. With the software designed, the robot created a trajectory which consisted of an array of x and y coordinates (waypoints). When the robot encountered unknown obstacles, using the vector field histogram algorithm and a 2D laser scanner it was able to avoid collision and head towards the next waypoint.

- Using the encoders on the robots wheels and an IMU, the current pose and heading of the robot was calculated. The x and y coordinates of the robot were determined using dead reckoning. The heading of the robot was calculated using the gyroscope and compass on the IMU.

- The software was tested using the LabVIEW Robotics Simulator with the NI Starter Kit 1.0 robot DaNI. The perception sensor used was a Hokuyo URG 2D laser scanner. For localization the robot optical encoders, a Sparkfun 6DOF IMU and a HMC6343 electronic compass were used.

- Three different tests were carried out using the robotics simulator:

    1. The first experiment consisted in testing the navigation algorithms in an environment with unknown obstacles without dead reckoning. Using the exact robot position given by the simulator for localization, the robot reached the target with minimal errors and successfully avoided all obstacles.
    2. In the second test, the dead reckoning localization was used. The errors obtained in the first experiment were corrected using constants. After applying these constants, the software was executed for three different speeds and the results obtained were compared.
    3. The last simulated test was a comparison of the software designed in this project with an existing navigation program: The Artificial Neural Network (ANN). A new simulation environment was designed in order to make the comparison as fair as possible. The results obtained were very similar, with the developed software being slightly more accurate than the existing ANN software.

6.2 Future Projects

The study of robotics combines many complex systems, therefore this work could be continued by improving each of these different systems, as well as adapting the algorithms in order to use them for autonomous driving. Some of these ideas are discussed in the following sections.

6.2.1 Implementation of the path planning algorithms on an autonomous vehicle

The main objective of this project was to develop navigation algorithms for a mobile robot which could later be implemented on an autonomous vehicle. To achieve this, a good understanding of the mechanics of the vehicle used would be necessary. Unlike the mobile robot used in this project which uses differential steering, four wheeled vehicles use the Ackerman steering geometry which is more complex. Also, for mobile robots, the control variables are the forward linear velocity and the angular velocity, meanwhile for a vehicle, the elements to control would be the acceleration, steering, the brakes and the gears.

Another important change that would have to be made to the software developed in the occupancy map used. For a vehicle navigating in an urban environment, instead of having to define a grid manually, a map can be obtained from a maps source (such as Google). This map can converted into a binary array in the same way that the occupancy map was, with roads having a low cost and the non-navigable areas having a high cost (Figure 6.1).

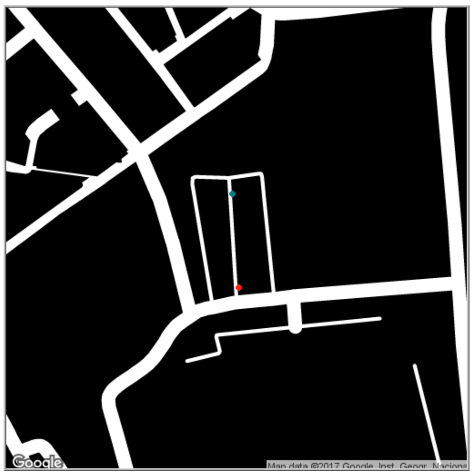


Figure 6.1 Example of a 2D occupancy map for a vehicle (P.J. Navarro, 2017)

The most complex part of adapting the algorithms to a vehicle would be combining them with the data obtained from the LiDAR sensors. In this project a 2D laser scanner was used. However, for a vehicle the data provided by these sensors alone would not be sufficient. On intelligent vehicles, the most common type of perception sensors are 3D LiDAR laser scanner which obtain a 3D cloud a points, in order to detect obstacles and features at different levels.

6.2.2 The use of Bezier curves for dynamic local path planning

In this work the local navigation was performed using the Vector Field Histogram algorithm to detect unknown obstructions. Another type of algorithm consists of calculating Bezier curves between the path waypoints. This method generates a number of Bezier curves between two points (Figure 6.2) and the optimal trajectory can be established according to the criteria chosen (such as maximum distance, maximum distance to obstacle, etc).

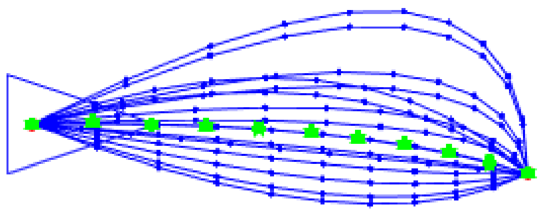


Figure 6.2 Example of Bezier curves between two points (P.J. Navarro, 2017)

Once tested on a mobile robot, this method could also be applied to an autonomous vehicle.

6.2.3 Improvement of the localization system

One of the biggest problems encountered during the development of this project was that of localization. There are different approaches to the localization problem. Some possible methods for more accurate localization are the Kalman filter, map matching, GPS systems, vision systems (beacons and landmarks), etc.

The Kalman filter is statistical method used to obtain predictions of the future state of a system by filtering out noise and other inaccuracies from sensor readings. The data obtained from the IMU sensors (accelerometer, gyroscope and compass) can be combined in a state-space system with the readings given by the wheel encoders to obtain a more accurate position of the robot.

Another technique is map matching. This consists of comparing maps provided by the perception sensors and overlapping the features (corners or doorways, for example). The distance travelled by the robot can be obtained by calculating the difference in angle and distance to the features extracted from the previous map.

Once map matching is accomplished, the next step would be to perform Simultaneous Localization and Mapping (SLAM). SLAM consists of constructing a map of the environment whilst simultaneously keeping track of the pose of the robot and navigation towards a target. When performing SLAM, the robot extracts landmarks from the maps that it creates to determine its position. This method is very accurate as the errors obtained from the perception sensors are usually lower than those of the inertial sensors and encoders.

BIBLIOGRAPHY

[1] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 376–382, Jun. 1991.

[2] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, Second Edi. 2011.

[3] C. Shea, "Self-Driving Vehicles Quick to Replace AGVs," 2016. [Online]. Available: http://blog.robotiq.com/self-driving-vehicles-quick-to-replace-agvs. [Accessed: 11-Apr-2017].

[4] E. Ackerman, "Fetch Robotics Introduces Burly New Freight Robots - IEEE Spectrum," 2017. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/industrial-robots/fetch-robotics-introduces-burly-new-freight-robots. [Accessed: 11-Apr-2017].

[5] Prweb, "Swisslog Introduces Next-Generation RoboCourier^{TM} Autonomous Mobile Robot for Secure, Hospital-Wide Delivery of Light Payloads," 2013. [Online]. Available: http://www.prweb.com/releases/2013/7/prweb10969033.htm. [Accessed: 11-Apr-2017].

[6] "Roomba Robot Vacuum | iRobot," 2017. [Online]. Available: http://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx. [Accessed: 11-Apr-2017].

[7] B. Brumson, "Robotics in Security and Military Applications," 2011. [Online]. Available: https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Robotics-in-Security-and-Military-Applications/content_id/3112.

[8] Army Technology, "TALON Tracked Military Robot, United States of America," 2017. [Online]. Available: http://www.army-technology.com/projects/talon-tracked-military-robot/.

[9] "GENERAL ROBOTICS LTD. Dogo." [Online]. Available: http://www.glrobotics.com/dogo-members. [Accessed: 23-Aug-2017].

[10] "Robot móvil TURTLEBOT 2 ROS | Robotnik." [Online]. Available: http://www.robotnik.es/robots-moviles/turtlebot-2/.

[11] P. Skrzypczyński, "Mobile Robot Localization: Where We Are and What Are the Challenges?," Springer, Cham, 2017, pp. 249–267.

[12] Ifg, "Roadmap Methods vs. Cell Decomposition in Robot Motion Planning."

[13] G. Dudek and M. Jenkin, *Computational principles of mobile robotics*. Cambridge University Press, 2010.

[14] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[16] "An Introduction to A* Path Planning (using LabVIEW) - Discussion Forums - National Instruments." [Online]. Available: http://forums.ni.com/t5/LabVIEW-Robotics-Documents/An-Introduction-to-A-Path-Planning-using-LabVIEW/ta-p/3521668. [Accessed: 21-Feb-2017].

[17] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm."

[18] A. Stentz, “The Focussed D* Algorithm for Real-Time Replanning,” 1995.

[19] H. Safadi, “Local Path Planning Using Virtual Potencial Field,” 2007.

[20] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, Jun. 1991.

[21] J. Borenstein and Y. Koren, “High-speed obstacle avoidance for mobile robots,” in *Proceedings IEEE International Symposium on Intelligent Control 1988*, pp. 382–384.

[22] Jagreet, “Overview of Artificial Neural Networks and its Applications,” 2017. [Online]. Available: https://www.xenonstack.com/blog/overview-of-artificial-neural-networks-and-its-applications.

APPENDIX I

PROJECT SUMMARY IN SPANISH

Resumen

En los últimos años, el uso de los robots móviles para realizar diferentes tareas ha crecido de forma exponencial. Es habitual encontrar sistemas robotizados en ambientes industriales, aplicaciones militares, sistemas agrícolas e incluso en empresas realizado tareas cada vez más sofisticadas. Los desarrollos en robótica móvil han pasado del campo de la investigación en Universidades y empresas al entorno cotidiano. Los avances en robótica móvil han sido transferidos a otros campos como la conducción autónoma o la exploración espacial. Los algoritmos de evitación de obstáculos, los planificadores locales y globales de trayectorias y los sistemas de percepción desarrollados en la robótica móvil, son utilizados en los nuevos vehículos sin conductor.

Este trabajo final de grado presenta un sistema de navegación para un robot móvil de tipo diferencial, que combina distintos algoritmos de creación de trayectorias y evitación de obstáculos. Los algoritmos implementados se combinan con un sistema de localización utilizando diferentes sensores instalados en el robot (encoders, LiDAR, IMUs, ec.). Los principales algoritmos usados son: El algoritmo A*, el vector field histogram y la navegación por estimación (dead reckoning).

I.I Capítulo 1. Objetivos del Proyecto

La motivación de este proyecto es adquirir los conocimientos en el ámbito de los sistemas de navegación autónoma y los algoritmos de navegación y localización empleados en la robótica móvil. Para ello se resolverán problemas de complejidad creciente relacionados con la robótica móvil. Estos problemas se materializarán en escenarios implementados en un simulador donde un robot móvil tendrá que evitar obstáculos, crear trayectorias, localizarse, y alcanzar un objetivo.

El problema de navegación de los robots móviles fue definido en 1991 por Durrant-Whyte et al [1]. Para su resolución, debía responderse a las siguientes preguntas: ¿dónde estoy?, ¿a dónde voy? Y ¿cómo puedo llegar?. La primera pregunta o localización depende de los sensores y equipos empleados en el robot. Mientras que las dos últimas están relacionados con la generación de trayectorias.

Este proyecto tiene como objetivo general el estudio y desarrollo de algoritmos de navegación para robótica móvil. Para la detección del entorno y de los obstáculos se utilizará como sensor principal un LiDAR (Light Detection and Ranging).

Los objetivos particulares del proyecto son:

- Realizar un estado del arte sobre los algoritmos de navegación en robótica móvil.
- Desarrollar algoritmos de navegación autónoma y evitación de obstáculos.
- La programación de un sistema de localización usando sensores inerciales y encoders.
- Realizar ensayos del software usando un simulador de robótica móvil.
- Implementar los algoritmos desarrollados en un pequeño robot móvil
- Realizar pruebas y verificar que se hayan cumplido los objetivos del proyecto.

I.II Capítulo 2. Estado del Arte de la Robótica Móvil

I.II.I La Industria de la Robótica Móvil

Los robots fueron introducidos en las líneas de producción en los años sesenta y desde entonces han evolucionado rápidamente. Sin embargo, los robots de manipulación tienen una desventaja: la movilidad. Los robots móviles pueden operar en áreas extensas y en situaciones peligrosas para las personas. Los robots móviles existentes se pueden separar en tres grupos principales: Los robots comerciales, robots para aplicaciones militares y los robots diseñados para la investigación y aprendizaje.

Robots Comerciales

Además de trabajar en ambientes hostiles, los robots trabajan al lado de los trabajadores en las fábricas, realizando tareas pesadas. Los vehículos guiados están siendo remplazados por los robots móviles autónomos. La ventaja de los robots móviles autónomos es que no tienen que seguir un camino fijado por unos raíles, y pueden evitar obstáculos y recalcular el camino a seguir para llegar a su destino. Ejemplos de robots comerciales son robots de transportación usados en fábricas y almacenes, robots para transportar materiales en hospitales y los robots domésticos de limpieza.

Robots para Aplicaciones Militares

Como los robots móviles pueden realizar tareas cada vez más complicadas y de forma más fiable, están siendo empleados para llevar a cabo misiones militares. Se usan principalmente para patrullar y realizar misiones con explosivos. Se diferencian de los robots industriales porque se suelen controlar mediante la teleoperación y no trabajan de forma autónoma.

Robots Diseñados para la Investigación

Para diseñar un robot móvil con éxito, hay que combinar muchos sistemas diferentes. Para resolver los problemas de navegación y localización, es necesario tener buenos conocimientos de la electrónica, de sensores, de programación y algoritmos. Para la locomoción, hace falta saber de la cinemática, la dinámica y la mecánica.

I.II.II Los Subsistemas de la Robótica Móvil

Los robots móviles autónomos combinan diferentes sistemas para realizar las tareas definidas por el usuario. Según Siegwart [2], para lograr la navegación autónoma se deben implementar cuatro subsistemas:

- Percepción: Consiste en extraer información útil del entorno del robot.
- Localización: El robot debe conocer su posición relativa en el entorno.
- Cognición: El robot tiene que crear una trayectoria para llegar a su meta.
- Control de movimiento: La trayectoria definida se debe convertir en movimiento, por ejemplo se puede calcular las velocidades de los motores necesarias.

I.II.III Percepción

La percepción es la capacidad del robot para detectar su entorno mediante sensores. La precisión del problema de localización depende de la calidad de la información obtenida por los sensores. Existen diferentes tipos de sensores para coger información del alrededor del robot: encoders en las ruedas, sensores de LiDAR (2D y 3D), sensores ultrasónicos, sensores inerciales, cámaras de visión artificial y GPS. Para obtener el mejor resultado para la localización, es necesario combinar sensores de varios tipos.

I.II.IV Navegación

El problema de navegación se puede dividir en dos partes:

- **Navegación global:** Consiste en construir una trayectoria del punto de inicio hasta un punto final. Para poder realizar esto el robot necesita un mapa, que puede ser completo o parcial.
- **Navegación local:** Aunque el robot tendrá un mapa, este puede estar incompleto, o la posición de los obstáculos puede cambiar. Para evitar una colisión, el robot debe calcular una nueva trayectoria para llegar a la meta de manera eficiente.

I.II.V Cinemática para un robot diferencial

Para localizar un robot en un plano, es necesario tres variables: las coordenadas *x* e *y*, y la orientación respeto al eje vertical del robot. El tipo de movimiento que puede realizar el robot depende de la distribución de las ruedas. Un robot diferencial tiene dos motores en el mismo eje, uno para controlar las ruedas de un lado del robot. Cuando el motor de un lado gira más rápido que el otro, el robot girará alrededor de un punto que se encuentra a un lado del robot. Este punto es el centro de la circunferencia descrito por el robot.

I.III Capítulo 3. Algoritmos de Navegación

I.III.I Técnicas de mapeo del entorno

Antes de poder calcular una trayectoria, el entorno del robot debe de ser dividido en espacio libre y obstáculos y representado gráficamente. Antes de elegir que técnica de mapeo emplear, se debe tomar en cuenta tres factores: La precisión necesaria, tipos de características que hay que representar y el coste de computación del mapa.

Existen varios métodos de mapeo:

- Descomposición exacta: En este tipo de mapa, las celdas pueden estar ocupadas o no. La posición del robot en la celda no es importante.

- Descomposición aproximada: Es una técnica muy simple. El entorno es dividido en celdas del mismo tamaño. Aunque una celda no esté completamente ocupado por un obstáculo, se clasifica como tal.

- Descomposición adaptada: En esta técnica, el tamaño de las celdas depende de su proximidad a un obstáculo. Una celda que contiene espacio libre y obstáculo se divide en dos, hasta que dentro de la ella solo hay espacio libre u obstáculo, no las dos cosas.

- Diagrama de Voronoi: Este método crea caminos entre los obstáculos, intentando maximizar la distancia entre el robot y el obstáculo. Este método tiene un coste computacional más alto que los otros métodos.

I.III.II Algoritmos de planificación de trayectorias

Los métodos actuales más comunes para la creación de trayectorias entre dos puntos son los siguientes:

- **Algoritmo A***

El algoritmos A* es una técnica que busca el camino de menos coste, es decir, el más fácil. Es derivado del algoritmo de Dijkstra [15], pero incluye un una función heurística para reducir el número de celdas exploradas. El coste se calcula como la suma del coste para mover a cualquier posición desde el punto de inicio y el coste estimado para mover desde la posición actual hasta la meta.

- **Algoritmo D***

El algoritmos D* es una versión del algoritmo A* de tiempo real. Si los obstáculos se mueven, este algoritmo recalcula el camino hasta la meta. Comienza como el algoritmo A*, y después de cada cierto tiempo, actualiza la trayectoria.

- Método de los Campos Potenciales

Este método se parece a una partícula cargada moviéndose en un campo magnético. La meta se parece a una fuerza atractiva, atrayendo a la partícula. Los obstáculos son como fuerzas repulsivas, empujando la partícula en la otra dirección. La fuerza resultante en la partícula (robot) se calcula como la suma de todas las fuerzas que actúan sobre él.

I.III.III Algoritmos de navegación local

Los algoritmos de navegación local se emplean principalmente para la evitación de obstáculos.

- Algoritmo "Bug"

Este algoritmo es el más simple, sin embargo el menos eficiente. Consiste en ir hacia la meta hasta que se detecte un obstáculo. Cuando encuentra el obstáculo, lo rodea completamente y vuelve al punto que se encuentre más cercana a la meta. Existe una mejora de este algoritmo que se denomina "Bug2", que consiste en crear una línea entre los puntos de inicio y el final, y al encontrar un obstáculo, lo rodea hasta que encuentre la línea principal.

- Algoritmo "Vector Field Histogram"

El Vector Field Histogram (VFH) es un método de tiempo real, que además del entorno, toma en cuenta el tamaño y la velocidad del robot. El VFH crea una cuadricula del estado del entorno; libre u ocupado. Este mapa se convierte en un histograma polar, con el ángulo en el eje horizontal, y la probabilidad de que haya un obstáculo en el eje y. Se define un límite, y si la probabilidad de que hay un obstáculo está por debajo del valor definido, en esa dirección el camino se considera libre.

Se buscan los huecos más grandes por donde puede pasar el robot, y después se aplica una función de coste y se elige la dirección con menor coste. En esta función se toma en cuenta la orientación actual, la orientación deseada y la dirección anterior.

I.IV Capítulo 4. Implementación con un Simulador de Robótica

I.IV.I Software

El entorno de programación elegido para realizar este proyecto es LabVIEW. LabVIEW emplea un lenguaje grafico denominado "G", basado en el flujo de datos, conectando nodos y cables.

Un programa de LabVIEW tiene tres partes: una interfaz gráfica, un diagrama de bloques y un panel conector. Un programa en LabVIEW se denomina VI.

La interfaz gráfica es lo que ve el usuario y contiene las entradas y salidas del sistema. El diagrama de bloques es el código del programa e incluye las funciones, variables y terminales de los objetos de la interfaz gráfica. El panel conector se usa para los subVIs, y contiene los terminales para las entradas y salidas que hay conectar al bloque del subVI cuando es llamado en otro diagrama de bloques.

I.IV.II Simulador de Robótica

Para realizar las pruebas se emplea el simulador de robótica de LabVIEW. Este simulador contiene una variedad de robots, sensores y entornos, además existe la posibilidad de diseñar tu propio robot, sensor o entorno.

En este proyecto, para las pruebas de simulación, se utilizó el robot DaNI (Starter Kit 1.0 de National Instruments), con un LiDAR Hokuyo URG, un IMU de 6DOF, un compás electrónico HMC6343 y los encoders del robot.

Se diseñaron dos entornos para realizar pruebas. El primero es prediseñado especialmente para el robot DaNI. El segundo fue diseñado en el propio simulador, partiendo de un entorno vacío, colocando obstáculos para crear habitaciones.

I.IV.III Desarrollo del Software de Navegación

La primera parte del software consiste en inicializar la simulación con el robot y sus sensores.

- Implementación del algoritmo A*

Una vez inicializado, se carga un mapa incompleto del entorno, para realizar el algoritmo A*. El mapa usado es un array de dos dimensiones con valores de 1 o 100. Los 1s representación el espacio libre y los 100 son obstáculos. El entorno fue dividido en cuadros de 0,5 por 0,5 metros, y cada cuadro corresponde con un valor en el mapa. Los cuadros iniciales y finales después deben de ser definidos para que el algoritmo calcule el camino óptimo.

- Control de las velocidades de los motores

Para controlar la velocidad de los motores del robot, se emplea un “steering frame”. El objetivo de esta función es convertir las velocidades lineal y angular deseadas del robot en velocidades angulares para los motores.

- Localización mediante “Dead reckoning”

El dead reckoning consiste en integrar las velocidades lineal y angular obtenidas de los encoders para obtener la posición en x y en y. Para calcular el error de la estimación, también se obtiene la posición exacta del robot a través del simulador.

- Evitación de obstáculos: Vector Field Histogram (VFH)

Para mejorar los resultados del algoritmo A*, este se ha combinado con el VFH. El VFH detecta obstáculos en tiempo real que no están definidos en el mapa original o que han movido de su posición inicial. Para aplicar esta función, el usuario debe definir los límites y la distancia requerida por el robot para evitar un obstáculo sin colisión. El VFH también necesita las lecturas del LiDAR. Las salidas obtenidas del VFH son el histograma y la nueva orientación para evitar los obstáculos.

- Interfaz Gráfica

La interfaz gráfica consiste de un panel de control donde el usuario puede definir la posición inicial y final del robot. También se pueden definir la velocidad lineal máxima y los parámetros para el algoritmo VFH. Los datos obtenidos del LiDAR y los puntos medios de la trayectoria calculada se muestran en gráficas.

I.IV.IV Flujogramas del Software

El flujograma del programa representa los distintos procesos que se llevan a cabo en el programa para que el robot alcance la meta. El programa se puede dividir en dos partes principales:

- El programa principal que contiene la inicialización de los elementos simulados y el algoritmo de navegación global.
- Un bucle for para cada punto intermedio donde se ejecuten los algoritmos de navegación local y localización.

I.V Capítulo 5. Pruebas y Resultados con el Simulador

I.V.I Prueba de los Algoritmos en el Entorno Diseñado

La primera prueba consistió en el robot creando una trayectoria (en forma de un array de coordenadas x-y) y después mediante la navegación local evitando los obstáculos desconocidos que encuentre por el camino. La posición inicial es el (0,0,0) y la de la meta es (10, 0.5, 180).

Para esta prueba, la localización se hizo sacando la posición y orientación directamente del simulador, ya que para la primera prueba el objetivo era probar el funcionamiento de los algoritmos de navegación.

En este prueba habían dos obstáculos desconocidos, y que se encuentran en algunos de los puntos intermedios de la trayectoria calculada por el algoritmo A*. El robot fue programado para que cuando detecte un obstáculo dentro de 0,4 metros, si no puede llegar más cerca que 0,3 metros, sigue ya al punto siguiente de la trayectoria. Así se evite que el robot circule infinitamente al obstáculo buscando al punto.

Se ha calculado el error final con respecto a la meta. No se ha calculado el error medio del camino, ya que el robot se desviará de la trayectoria planeada para evitar a los obstáculos.

Los resultados de esta simulación obtenidos fueron muy precisos y consistentes. Se pueden ver en detalle en la sección 5.1 del proyecto.

I.V.II Pruebas con Dead Reckoning

Para la segunda prueba con el robot, el método de localización empleado fue el de "Dead reckoning". El punto inicia fue el mismo que el anterior, y la meta fue (5.5, 2.5, 90). Para esta simulación, se realizaron cuatro experimentos diferentes.

1. El primer experimento fue llevado a cabo a una velocidad máxima de 0.3m/s. No se aplicaron ningún constante o medio de corrección para la estimación de la posición del robot. Los errores obtenidos era del orden de 30cm en el eje x y de 8cm en el eje y.
2. Para el segundo experimento, se emplearon constantes para corregir la posición calculada mediante "Dead reckoning". La velocidad máxima fue de 0.3m/s. La posición

final alcanzada corrigiendo el error de localización fue bastante más precisa, con errores del orden de 0.5 - 6.5 cm.

3. La tercera prueba consistió en cambiar la velocidad lineal máxima a 0.25m/s. También se usaron constantes para corregir la posición obtenida de la información de los encoders. Los resultados obtenidos fueron más dispersos que en el experimento a 0.3m/s, aunque el error medio fue del mismo orden.
4. Para el último experimento, empleó una velocidad lineal de 0.35m/s. Los resultados obtenidos se parecían también a los anteriores, del mismo orden pero más dispersos que para una velocidad de 0.3m/s.

En la sección 5.2.5 se realiza una comparación detallada de los resultados obtenidos en los cuatro experimentos. Se concluye que los mejores resultados fueron obtenidos para una velocidad lineal de 0.3m/s usando constantes para corregir la estimación de la posición.

I.V.III Comparación del software desarrollado con software existente

Para la tercera simulación se realizó una comparación del software desarrollado en este proyecto con un algoritmo existe programado para el mismo robot: El Artificial Neural Network (ANN). Un ANN es un sistema de aprendizaje que puede adaptarse a la situación actual en el entorno. El algoritmo de ANN determina las velocidades del robot dependiendo de su posición actual y la posición del siguiente punto del camino. Antes de realizar la trayectoria, el robot debe de ser entrenado. Esto consiste en calcular las velocidades angulares de las ruedas a partir de las velocidades lineal y angular necesarias.

El software desarrollado en este proyecto fue adaptado para que funcione en las condiciones más parecidas posibles a las de software de ANN. Para la localización, ambos programas usaron la posición suministrada por el simulador. Para que el A* siguiera una camino parecido, se usaron obstáculos adicionales y el mapa fue modificado.

En esta comparación, los errores finales han sido calculados y comparados en la sección 5.3.4 del proyecto. Los errores obtenidos con el ANN fueron mayores que los del software del proyecto. Además los errores obtenidos con el A* y VFH fueron más consistentes.

I.VII Capítulo 7. Conclusiones y Futuros Trabajos

I.VII.I Conclusiones

El objetivo general de este proyecto fue de estudiar diferentes algoritmos de navegación para los robots móviles y desarrollar un programa que permite al robot crear una trayectoria y seguirla evitando obstáculos usando con sensor principal un LiDAR

- Se realizó un estudio del estado del arte sobre la situación actual en la robótica móvil. Se estudiaron los robots disponibles actualmente en el mercado y para la investigación, además de los sensores más usados.
- Se desarrolló software de navegación autónoma en el entorno LabVIEW para un pequeño robot móvil: DaNI. El software consistió en la combinación de un algoritmo de navegación global (Algoritmo A*) y otro de navegación local (VFH)
- Se calculó la posición relativa del robot en el entorno mediante un sistema de localización basado en el “Dead reckoning”, usando los encoders del robot. La orientación también fue obtenido, mediante un IMU.

- Se realizaron pruebas con el simulador con el robot DaNI (Starter Kit 1.0 de National Instruments). Para estas pruebas, los sensores usados fueron un Hokuyo URG LiDAR, los encoders del robot, un IMU de 6DOF de Sparkfun y un compás electrónico HMC6343.

- En el simulador se realizaron tres pruebas distintos:
  1. El primero se realizó para probar la combinación de los algoritmos de navegación. Para la localización, la posición exacta fue obtenida del simulador. En esta prueba, el robot llegó a la meta con errores mínimos.
  2. En la segunda prueba, el sistema de localización mediante dead reckoning fue probado. Después de la primera simulación, se corrigieron los errores y el software fue probado para distintas velocidades del robot.
  3. La última prueba con el simulador fue una comparación del software desarrollado en el proyecto con un sistema de navegación autónoma ya existente.

I.VII.II Futuros Trabajos

El estudio de la robótica móvil combina muchos sistemas complejos, por lo tanto hay gran cantidad de posibles trabajos futuros. Algunas de las posibilidades más interesantes son los siguientes:

- Implementación del Sistema de navegación en un vehículo autónomo

La motivación de este proyecto fue el desarrollo de un sistema de navegación autónomo, con el fin de acabar adaptándolo a un vehículo. Para lograr esto, sería necesario un buen conocimiento de la mecánica de un vehículo. Las variables de control a calcular, no serían velocidades, sino la aceleración. Los sensores para la percepción en un vehículo también serían más complejas (LiDAR 3D) y habría que tomar en cuenta el eje vertical.

- Sustituir el VFH por las curvas de Bezier

Otro método para la navegación local sería el uso de curva de Bezier. Este método genera una serie de curvas entre dos puntos, de orden definido por el usuario. Después se elige la trayectoria óptima.

- Mejora del sistema de localización.

El problema más grave que se encontró durante el desarrollo del proyecto fue el de localización. Hay varias opciones para mejorar la localización, algunos siendo: Filtro de Kalman, emparejamiento de mapas, sistemas de GPS, sistemas de visión artificial, etc.