TECNOLÓGICO DE MONTERREY

Instituto Tecnológico y de Estudios Superiores de Monterrey
Maestría en Ciencias de la Ingeniería
Tesis

Multiple-Robot Motion Planning in an Unknown Environment

Rolando Bautista Montesano

Asesor:
Víctor de la Cueva Hernández

2013

Abstract

This Thesis proposes an implementation of a Multiple Robot System that handles a heavy object from an initial point to a final one. The System does not know where it is located inside a workspace, so it needs to find out its location to compute a path that connects the desired point.

The System is compound by three main elements: a Planner, a Transmitter, and the Robots. Each one of them executes different tasks. The Robots and the Transmitter are heavily bounded, they are in charge of controlling the robot as well as retrieving environment information. They are not in charge of any heavy computing operations because it is all done in the Planner. The Planner uses several Artificial Intelligence algorithms such as the Particle Filter, Kalman Filter and A* (A star) search. The environment is decomposed using an Approximate Cell Decomposition.

This work's main contribution is the usage of low-cost robots that work together using high complexity algorithms. All the employed sensors have a certain degree of uncertainty which is handled with non-parametric and Gaussian filters. The Follower Robots' task is reduced to track where the Transport platform goes, a low cost solution is proposed.

Throughout this book all of the mentioned concepts, implementations and problems will be explained.

Contents

Nomenclature

$\mathcal{O}$ An obstacle in $\mathcal{W}$ or in $\mathcal{Q}$

$\mathcal{Q}$ The configuration space of a robot is the space of all the configurations of the robot.

$\mathcal{W}$ Workspace is an Euclidean space represented as $\mathcal{R}^{\mathcal{N}}$ with $\mathcal{N} = 2$ or 3

$f(n)$ Estimated cost of the cheapest solution through n.

$g(n)$ Cost function of moving from node n to node m.

$h(n)$ Estimated cost of the cheapest path from the state at node n to a goal state.

P_f Final point in a Path Planning Algorithm

P_i Initial point in a Path Planning Algorithm

q An element of $\mathcal{Q}$

$r(q)$ The region of the workspace occupied by the robot r

Azimuth Angular measurement in a spherical coordinate system. The vector from an observer to a point of interest is projected perpendicularly onto a reference plane: the angle between the projected vector and a reference vector on the reference plane

BS2 BASIC Stamp 2 Microcontroller

Configuration It is a specification of the position of every point in an object relative to a fixed reference frame

Degrees of Freedom The dimension of the configuration space is equal to the number of independent variables in the representation of the configuration

GPS Global Positioning System

Latitude Geographic coordinate that specifies the north-south position of a point on the Earth's surface. Lines of constant latitude.

Longitude Geographic coordinate that specifies the east-west position of a point on the Earth's surface.

LRF Laser Range Finder

Simultaneous Localization and Mapping Algorithm employed by robots and autonomous vehicles to create maps in an unknown environment while it keeps track of its localization

List of Figures

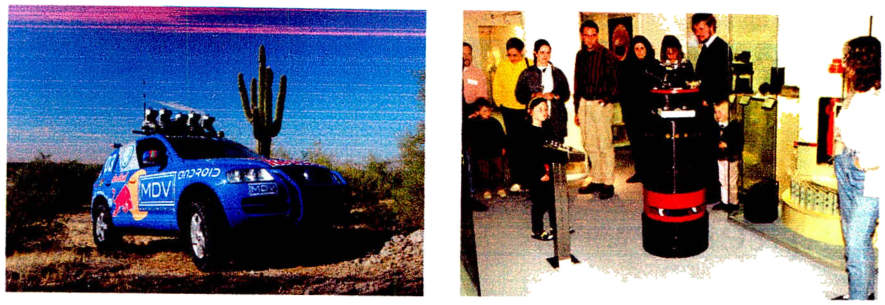
List of Algorithms

Chapter 1

Introduction

One ordinary day at all of 1997 a boy and his parents went to the cinema to see the *Star Wars* saga. After the films were over he realized how fascinating it could be to live day by day with robots such as *C3PO* and *R2D2* [21]. It seems that decades before many people had the same dream. They imagined and dreamed about it. What they did one day became what we know as Robotics. Creating machines that can perform several tasks has produced a technological revolution. The word *robot* comes from the Slavic Languages. It was used for the first time in *R. U. M.*, (*Rossum's Universal Robots*) , a book written by Karel Capek [7]. The exact term was *robotnik*. It described a breed of workers that were created from biological parts. They were capable of doing many chores that human beings could not. The word's meaning is "slave" or "worker". It perfectly describes what a robot is: an electromechanical or virtual device in charge of fulfilling certain activities. Robotics as a science was born in the *Foundation Series* from the legendary Isaac Asimov [6].

Far from Literature, Robotics is the combination of several areas such as Physics, Mathematics, Electronics, Mechanics and Computer Science. Robots are classified in several categories depending on its characteristics. Motion is one of the most important for this work. They can be fixed, mobile or hybrid. A fix robot has a part of it tied to a base and its movement is always bounded to that point. The mobile robot can move loosely through space. Finally the hybrid is a mixture of the first two. In a fixed robot, the task is performed by an end effector that will follow a path. For the mobile case, the entire robot performs the task after a path planning algorithm took place. A robot can also be classified by the type of motion and the number of Degrees of Freedom it has. It can be omnidirectional or holonomic when



(a) DARPA Challenge Stanford University's Stanley

(b) Deutsches Museum Bonn's RHINO

Figure 1.1: Autonomous Robots examples

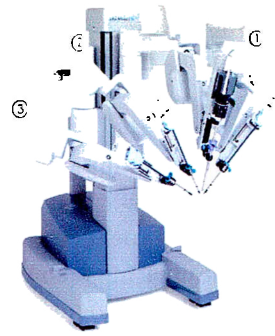
it can move at any direction, so a non holonomic one can only perform restrained movements, ergo just in one direction. Depending its DOF it can be non-redundant when it has as much as DOF as dimensions and redundant when there are more DOF than dimensions. Due its autonomy it can be directed, supervised or autonomous. It is directed when the user specifies movement by movement what the robot shall do. Supervised when a motion planning has been done given an environment for being executed later. Finally it is autonomous when it can take several decisions by itself under certain circumstances. There are lots of examples of robots with several movements characteristics and different degrees of autonomy.

One of the biggest challenges that Robotics faces is in autonomous path planning. The goal is to demand for a task in a high level programming language and the robot must transform the received instructions into a set of low level commands. While executing the task a path must be found so the mobile or fixed robot can follow it. This area's development has come through with applications such as digital animation, industrial processes verification and pharmaceutical design. Some well known examples can be:

- Transportation - An example of a transport for one or two persons is a Segway or the CyCab. They are a good environment-friendly alternative that use small space, are not noisy and its ecologic footprint is quite smaller than the car's one. A bigger scale example can be Stanford University's Stanley. Stanley can move freely in a highway by planning its own movements (Figure 1.1a).

- Museum Tour Guides - In 1997, a mobile robot named RHINO servved as a fully autonomous tour-guide at the Deutsches Museaum Bonn

(a) The Mars Rover



(b) The Da VInci Surgical System

Figure 1.2: Robotic applications used for Research

(Figure 1.1b). RHINO was able to lead museum visitors from one exhibit to the next by calculating a path using a stored map of the museum. Because the perfect execution model of the piano mover's problem is unrealistic in this setting RHINO had to be able to localize itself by comparing its sensor readings to its stored map.

- Planetary Exploration - The Mars Rover (Figure 1.2a) is a robot whose main task is to explore the Martian surface. It must take several samples from the ground and analyse them, it also has to take some photographs of the environment that will supply the scientists with valuable information about the Red Planet. As the robot does not know where it is, the SLAM algorithm is used.

- Medicine - The Da Vinci Surgical System (Figure 1.2b) is capable of performing surgeries with high precision actions and with the bonus that the Doctor can be in a remote location. Robots are used in invasive procedures. They enhance the surgeon's ability to perform technically precise maneuvers.

- Industry - Several robotic arms are employed for assembling, painting, welding or separating products. The complexity of their utilization is that in the workspace there is a great amount of movement and robots must have enough coordination to avoid collisions among them, the users or the product. Industrial robot installations are driven by economic factors, so there is a high priority on minimizing task execution time. This motivates motion planners that return *time-optimal* motion plans. Other kinds of tasks may benefit from other kinds of optimality, such as energy or fuel optimality for mobile robots. [9]

Robot path planning calculation is critical for all applications. The employed algorithm must be capable of finding a collision-free path, compute the required movements and assuring they are physically performed.

A path planner must consider the task, the robot and the algorithm. The task can be either navigation, localization or coverage. Navigation refers to the calculation of an obstacle free trajectory from an initial point (P_i) to a final point (P_f). When the robot needs to explore all the points in the workspace the task is a coverage task whereas in a localization task the robot needs to use information provided by some sensors. The robot must also be considered along with the environment. Together they determine the number of DOF of the system, the workspace and the configuration space. Finally the chosen algorithm must satisfy certain restraints such as the computational complexity and the solution it returns. Complexity deals with memory limitations and the required time for computing the calculations. The returned solution shows how complete the algorithm is. A complete algorithm guarantees to find a free path whenever one exits and to return failure otherwise. Approximate methods may not be complete; but, for most of them, the precision of the approximation can be tuned and made arbitrarily small, so that the methods are said to be *Resolution-complete.* [10]

This Thesis' objectives are:

- To develop a low cost Multiple Robot System capable of moving an object from an initial point P_i to a goal location P_f.
- Employ a Resolution Complete Path Planning algorithm to represent the chosen workspace.
- Implement an online centralized planner that uses sensor-based information.
- Deal with uncertainty using software solutions.
- Identify fixed and mobile obstacles.
- Obtain an optimal path.
- Overcome non-holonomic robot restraints.

In chapter 2 the State of the Art is presented. The most innovative, different and recent proposals are explained. Having special attention in topics such as Path Planning, Artificial Vision, Formations, Communications and Self-Localization. Chapter 3 explains all the algorithms that were used for developing this project: Motion Planning, Approximate Cell Decomposition, A*, Particle Filter and Kalman Filter. Chapter 4 gives a quick overview of the employed hardware used while developing the Robot System: the Boe-Bot Robot, Bluetooth Module, GPS, Compass, Laser Range Finder, Infrared Line Follower and structure modifications. After it, Chapter 5 describes how the algorithms explained in Chapter 3 were implemented in the Planner, as well as an example of how they work and what they take as inputs and return as outputs. The robot programs are also explained here. Chapter 6 shows the results of combining all the programs and some examples of the runs that were made. Finally in Chapter 7 the future work and modifications are presented as growth areas for the project.

Chapter 2

State of the Art

This chapter will explain the most important techniques and algorithms that are used in recent work. The covered topics will be:

- Path Planning
- Artificial Vision
- Formations and Communication
- Localization

2.1 Path Planning

2.1.1 Basic Problem

The goal of defining a basic path planning problem is to isolate some central issues and investigate them in depth before considering additional difficulties.

The basic problem defines that in $\mathcal{W}$ the robot is the only moving object, all its dynamic properties are ignored. Computed motions are contact-free so mechanical interaction between two objects is ignored. This way a physical path planning problem turns into a geometrical path planning problem. To simplify the problem even more, the robot will be a single rigid body whose movements is restrained by obstacles.

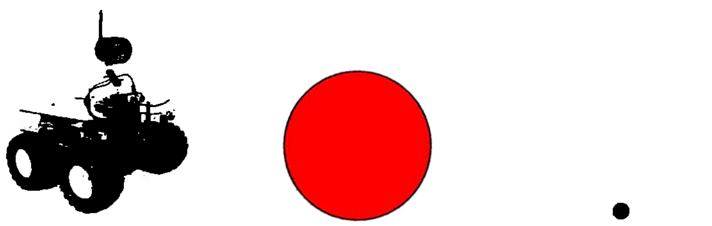
The **basic motion planning problem** resulting from these simplifications is the following (Figure 2.1c):

- The robot is a single point.

- The robot is the only object that moves in $\mathcal{W}$.
- No mechanical problems are considered.

J. C. Latombe defines the problem as:

> Given an initial position and orientation and a goal position and orientation of the robot r in $\mathcal{W}$, generate a path specifying a continuous sequence of positions and orientations of the robot avoiding contact with the obstacles, starting at the initial position and orientation, and terminating at the goal position and orientation. Report failure if no such path exists.



(a) Real Robot (b) 2D Model (c) Final Model

Figure 2.1: Robot representations

The following notation is consistent with the one proposed by Tomás Lozano Pérez. in [20], Jean Claude Latombe in [17] and Choset et. al in [9]. For more details it is recommended to consult the bibliography.

A Workspace ($\mathcal{W}$) is a $\mathbb{R}^2$ or $\mathbb{R}^3$ environment in which robots work. It can be decomposed in $\mathcal{WO}_{\backslash}$ and $\mathcal{W}_{free}$. The i-th obstacle can be represented as the first one, while the remaining free space that is not occupied by an obstacle is the second one.

$$\mathcal{W}_{free} = \mathcal{W} - \bigcup_i \mathcal{WO}_i \tag{2.1}$$

Path Planning algorithms are executed in the configuration space $\mathcal{Q}$ (Figure 2.2b) not in $\mathcal{W}$ (Figure 2.2a). $\mathcal{Q}$ can be defined as the set of all the possible robotic configurations. To the set of points in $\mathcal{W}$ occupied by the robot $\mathcal{R}$ in the configuration q will be denoted as $\mathcal{R}_q$. An obstacle in $\mathcal{Q}$ corresponds to the robot configurations that intersect an obstacle in $\mathcal{W}$ such that:

$$\mathcal{QO}_i = \{q|\mathcal{R}_q \bigcap \mathcal{WO}_i \neq 0\} \tag{2.2}$$

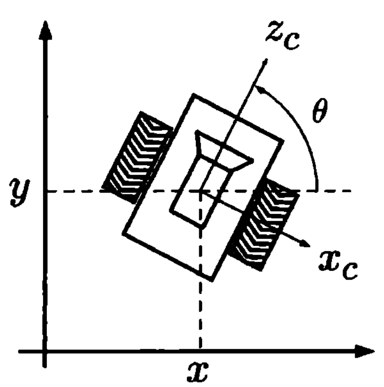
A path is a continuous curve in $\mathcal{Q}$ expressed by a continuous function defined in [0-1] for:

$$\mathcal{Q}_{free} = \mathcal{Q} - \bigcup_i \mathcal{Q}_i \tag{2.3}$$

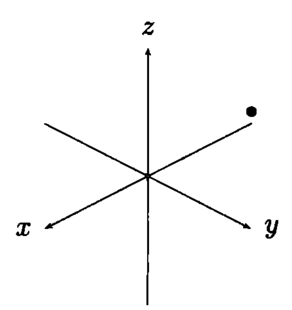
A continuous function $\mathcal{C}$ can be a solution to the Path Planning problem such that $\mathcal{C}[0-1] \rightarrow \mathcal{Q}$. The initial configuration is $q_i = \mathcal{C}(0)$ and the final one $q_f = \mathcal{C}(1)$. This can be generalized as:

$$\mathcal{C}(s) \in \mathcal{Q}_{free} \bigvee s \in [0,1] \tag{2.4}$$

A Robotic System Configuration is a specification of each point of a system. $\mathcal{Q}$ of a system is the space of all possible system configurations.



(a) Workspace



(b) Configuration Space

Figure 2.2: Space Representation

Lots of methods and solutions have been proposed an implemented to solve the basic path planning problem. Not all of them solve it completely. Some of the require that $\mathcal{W}$ is $\mathbb{R}^2$ and that the objects are represented as polygons. Most of the approaches can be generalized into: *roadmap*, *cell decomposition*, and *potential fields*.

2.1.2 Roadmap

Roadmaps are a network of 1D curves which represent the connectivity in $\mathcal{W}_{free}$ or $\mathcal{Q}_{free}$. After it the connectivity has been constructed it represents a set of standard paths that connect P_i with P_f via N point that bind them. The calculated path is the result of concatenating a subpath that connects P_i with the roadmap, a subpath that contains the roadmap and finally the one that connects the roadmap with P_f. [17]

Visibility Graph Method

This is one of the first path planning methods that were developed. It is applied in 2D $\mathcal{Q}$ with polygonal $\mathcal{QO}$. It is a non-directed graph whose nodes are q_i and q_f and all obstacle region. The links of the subgraph connect the obstacle vertices configurations and they determine the roadmap. The other links of the graph connect the initial and goal configurations with the roadmap. [17] (Figure 2.3)

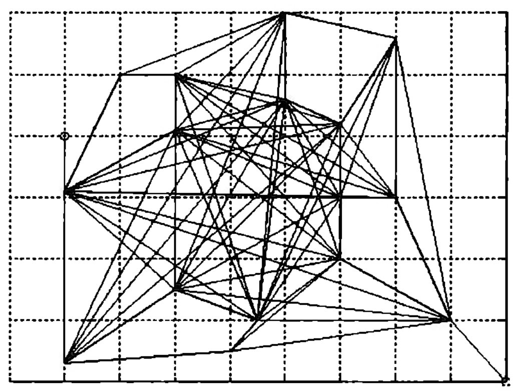


Figure 2.3: Visibility Graph Method

Probabilistic Roadmaps

A Probabilistic Roadmap algorithm constructs a map in a probabilistic way in $\mathcal{W}$. It divides planning into a learning and a query phase. During the first one a roadmap in Q_{free} is build, in the second one the configurations are connected in the roadmap. The roadmap is represented in an undirected graph $G = (V, E)$. Where the nodes V are a set of q chosen from Q_{free}. The edges in E are paths; an edge q_1, q_2 is a collision-free path that connects them. These paths are known as local paths, which are computed by a local planner.

During the query phase, the roadmap solves individual path-planning problems, where given q_i and q_f, it tries to connect them to the closest nodes q' and q'', respectively, in V. If it succeeds it reaches the graphs G for a set of edges E that connect q' to q''. It finally transforms the set of nodes into a path for the robot by recomputing each local path and concatenating them. These paths are stored in a global roadmap, but it implies more storage memory. The roadmap can be reused and augmented to capture connectivity in Q_{free}. Usually the learning phase runs before the query, but they can be interwoven. Sometimes it is feasible to spend lots of time during the learning phase if the roadmap will solve many queries. [9] (Figure 2.4)

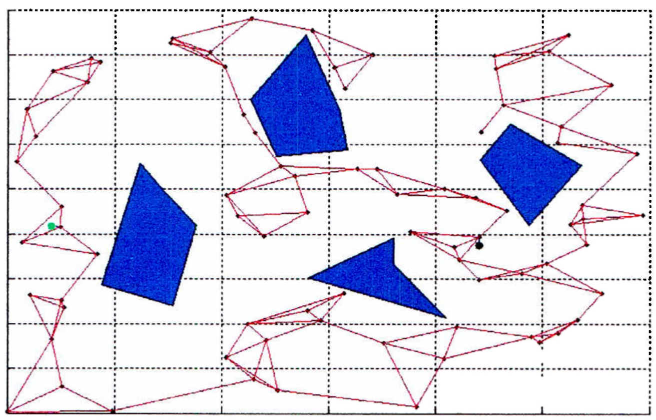


Figure 2.4: Probabilistic Roadmap

Voronoi Diagram

Another roadmap method is retraction. It defines a continuous function in Q_{free} to a 1D subset of itself such that the restriction of this function to this subset is the identity map. In a 2D $\mathcal{Q}_{free}$ the retraction is called Voronoi Diagram. It can be defined as the set of all $\mathcal{Q}_{free}$ whose minimal distance to a $\mathcal{QO}$ is achieved with at leas two points in the boundary of it. Its main advantage is the creation of free paths which usually maximize the gap between the robot and the surrounding obstacles. A free path between q_i and q_f is composed by three main subpaths: the first one is a straight line from q_i to q_i', a path from q_i' to q_f and finally a line from q_f' to q_f. [17] (Figure 2.5)

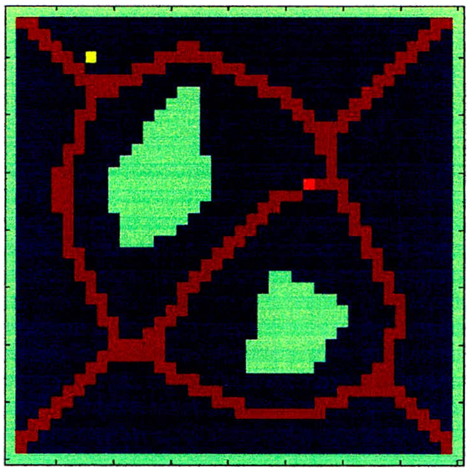


Figure 2.5: Voronoi Diagram

2.1.3 Cell Decomposition

One of the most popular planning methods is Cell Decomposition, it has been widely used and studied. It decomposes Q_{free} into small regions (cells) to generate a path from any q_i to q_f. A connectivity graph is used to search in the adjacency between the cells. Each node is a cell that belongs to $\mathcal{W}_{free}$ or $\mathcal{Q}_{free}$. Two nodes are linked if and only if they are adjacent. The resulting output of the search is called a *channel*, a continuous sequence of cells or path. [17]

Cell decomposition methods can be broken down further into *exact* and *approximate* methods:

- Exact cell decomposition methods decompose the free space into cells whose union is exactly the free space. The boundary of a cell corresponds to a criticality of some sort. (Figure 2.6a)

- Approximate cell decomposition methods produce cells of predefined shape whose union is strictly included in the free space. THe boundary of a cell does not characterize a discontinuity of some sort and has no physical meaning. (Figure 2.6b)

2.1.4 Potential Field

A straightforward approach for path planning is to discretize $\mathcal{Q}$ into a very small grid of q in which an informed search algorithm is employed to find

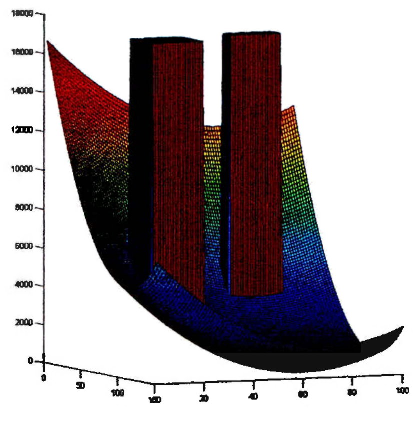
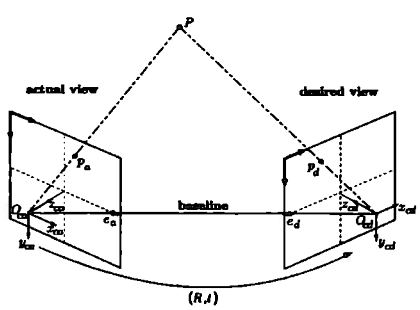
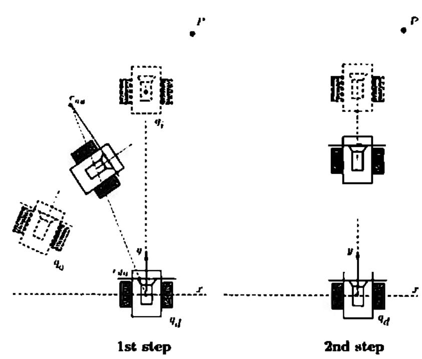


Figure 2.7: Potential Field

holonomic robot into a desired configuration. [12](Figure 2.8b) The robot is equipped with a camera that acquires an image $\mathcal{O}_c = \begin{bmatrix} x & y \end{bmatrix}^T$ which is compared wit a reference image, which was previously acquired in the desired configuration. The current position is analysed such that the robot moves to a configuration in which it only will need to perform a planar movement. This way it moves from $\mathcal{P}_i$ to $\mathcal{P}_f$.



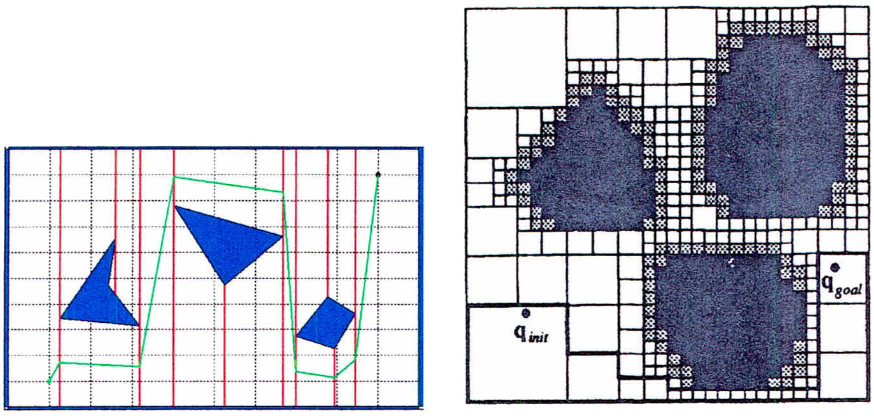
(a) Actual and desired configuration comparison



(b) Moving to a configuration and planar adjustment

Figure 2.8: Artificial Vision Strategies for Path Planning

Another approach proposes to equip robots with panoramic cameras that provide its exact location and distance from the other robots. These



(a) Exact cell decomposition (b) Approximate cell decomposition

Figure 2.6: Cell Decomposition

a path. As in any search algorithm, a good heuristic is required to avoid getting stuck in a certain q. Some heuristics can be described as functions known as *potential fields.*

Usually this metaphor is used to explain this approach: a robot (particle) moves influenced by an artificial potential produced by the goal P_f and the obstacles. The goal "attracts" the particle to it while any obstacle repulses it. The negated gradient of the potential is analogous to an artificial force that control the robot. At any q, the direction of the force is considered the direction of the motion. [17] (Figure 2.7)

This method can be very efficient but they have a main drawback, they can get stuck in a local minima of the potential function rather than the goal configuration.A way to solve it is to use potential functions that have only one local minima in the connected subset of Q_{free} which contains q_f. Another approach seeks to escape from the local minima using several mechanisms.

2.2 Artificial Vision

Pinhole cameras have been used for creating models in which the relationship between a 3D point an a image projection is found. This is achieved by using epipolar geometry. This type of geometry, also known as stereo vision geometry, relates the 3D points to 2D images. (Figure 2.8a) It uses the IBVS (Image Based Visual Servoing) algorithm for directing a mobile non-cameras can cover the complete workspace. The leader-robot follows a path while the trackers follow it. (Figure 2.9) The trackers have to keep a certain distance and orientation from the leader. This approach uses the Luenberger non-lineal observer as well as the Jacobian. [13]

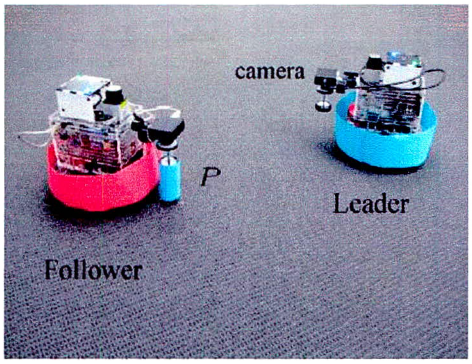


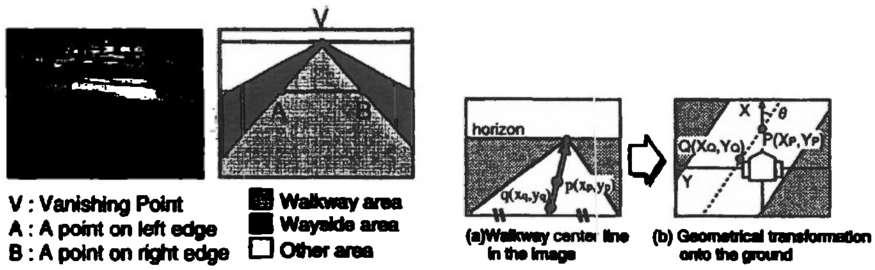
Figure 2.9: Leader - Follower Vision Tracking

After running a path planning algorithm the robot usually follows a route, but when the only information available is the intersections the robot shall pass it results necessary to check the actual location by Artificial Vision. Several works accomplished that a robot could follow the pavement's white line, so using this achievement as a basis a work was proposed for a Campus Walkway. An image is captured, the initial color attributes of the walkway must be acquired. (Figure 2.10a) The part corresponding to the trail shall be differenced from the extra elements. After it a line is drawn from the trail's horizon to the lower center part of the image. The generalization of it can be explained as follows: the path's form can be considered as a triangle, so the next step is to draw a line from the upper-most vertex of a triangle to the center of its base. The robot follows the computed line. (Figure 2.10b) While more environment samples are taken the robot moves faster and there are less possibilities of moving into a restricted area. [16]

2.3 Formations and communication

Control

There are two main approaches for controlling a multiple-robot. The centralized control takes all the decisions and assigns all actions in a single computer, the environment information is in it. Its main advantage is the



(a) Walkway Environment Model (b) Walkway Center Line

Figure 2.10: Campus Walkway

task high execution efficiency, whereas its low failure tolerance represents its disadvantage. If the main computer fails, everything fails. In a decentralized control each robot is capable of taking its own decisions and processing information. When this type of control is implemented local information builds up a global behaviour. The information comes from the robot's sensors and the communication among them. Even if individual actions are taken, each of them follows a cooperation strategy. Both approaches have the same problems: explicit communication and the behaviour in a dynamic environment.

Leader-follower

The leader-follower scheme can be implemented when the number of robots goes from to to n. A robot must be the leader. It receives precise instructions from a path planner of how to move. The followers' only task is to track and follow the leader's movements. Not all the robots' skills and abilities are employed which can be seen as a disadvantage. In the other hand the minimum communication between the robots is an advantage. When the number of followers is greater than two, it is almost impossible for the i-th follower to esteem the desired path because there is an accumulated error generated by all the other robots. Because of that the virtual leader concept was implemented. In it the i-th follower esteems the path of the i-th leader. (Figure 2.11)

Communication

For big teams in big, dynamic and unknown environments where classic communication mechanisms are not liable it is necessary to reduce the load

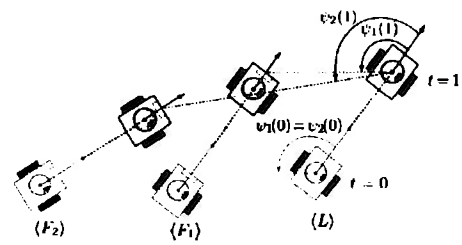


Figure 2.11: Leader - Follower Scheme

in data transmission. Several formations can be implemented as restrictions between each robot position respect to another team member to maintain a certain shape in the formation. A formation can be represented as a set of nodes and edges in a connectivity graph. Each node represents the localization of each agent and the edges represent the communication links among them. The information flows in a separate way in slow and fast time-scale. The fast one is used in critical decision taking which handle time constraints situations such as collisions or formation adjustments. It controls movements and paths. The slow one just takes place among non-adjacent robots. Short term information occurs between neighbours, this reduces a communication in complex formations. When the desired path is computed the communication is weakly connected because individual decisions are propagated to all the agents. [8]

Kinematic Model

In teams that employ formations the main problem resides in controlling position and orientation of a group of robots as a whole. Several kinematic models have been used for the leader where x and y describe position and θ orientation.

$$\left[\dot{x} = v\cos\theta \quad \dot{y} = v\sin\theta \quad \dot{\theta} = \omega\right]^T \tag{2.5}$$

Multiple-Robot Systems

In some circumstances it is necessary to work with a set of robots where each one of them is part of a global task. It is necessary to take care interaction among them and the environment to prevent collisions, look after all the agents, the environment and the task. The biggest challenge that a robot group faces is the wheel's steering error or communication breakdowns. The first works that were made sought to reduce the computing load of mobile decentralized non-holonomic robots. It used a leader-follower scheme, where the leader knew the movement instruction and the followers esteemed the path depending of the object's movement in a coordinated way. [15]

Restraints

When a team is deployed with a specific task such as cleaning, environment recognition, survivor rescue or just following a path it is necessary to know the robot's limitations and restraints of energy and time. Sometimes the number of robots for a certain task must be minimum and the result maximum. It results quite important to find a balance between performance while executing the task and handling energy. While robots move and work faster the goal is achieved in less time but the power consumption is high. In the other hand if the task is done while taking extreme care of the energy consumption the due time may be surpassed. That is why a velocity variation plan must be done so energy consumption can be optimized. Multiple-robot systems are deployed in these situations so the load can be divided though it is important to maintain communication among them so there are no repeated actions. [31]

Object Manipulation

A problem that has been greatly investigated is object manipulation by multiple-robots. Certain tasks cannot be performed by just one robot because of its size, weight or the lack of sensor information. It is quite important to consider that any action will affect the other members of the team, that is why coordination is critical. If the robots are equipped with a gripper or a robotic arm other factors as manipulation speed or movement delay the problem grows bigger. [22]

Lots of approaches have been presented for a group of agents manipulating an object. One of them proposes to displace an object with multiple-robots wielding contact forces. These are modelled like non-linear potential gradients that describe the load deformation. They also work as an implicit communication way: physical interaction between the load and the agents feedbacks the agent with information of what the other robots are doing. [14].

There is another approach called *Object Enclosure* in which the transported object is caged by a team. (Figure 2.12) It creates a bounded and mobile area for transporting the object as well as manipulating it. Its great advantage is that there is no need for continuous contact so path planning is simpler and more robust. The algorithm works this way: The robots approach the object independently using the Potential Field Algorithm. An optimal formation is computed. In this formation the object will not *escape* from them while transporting it. A robot is designed as a leader and the other members change their position in the formation. [32]

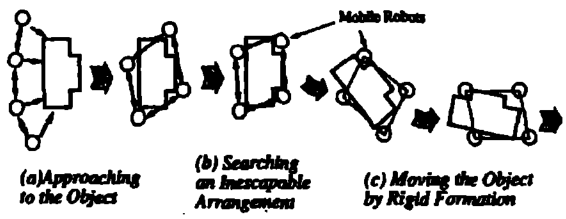


Figure 2.12: Object Enclosure Algorithm

Using intelligent agents is also another alternative. The agents could be able to learn from a complex and unknown environment using Q-Learning. When the agent moves the object, it identifies the environment's state and autonomously it computes the optimal form to move to the goal. Its pushing points are predefined to ease the robot's computation. [30]

Algorithm development for the problem has returned like SBS (Situated Behaviour Set), where some behaviours were developed for certain situations in which a robot can find itself. The task complexity is evaluated, as well as looking for a partner, checking the object and determining if it is possible to move it. Depending on its parameters the outputs are: change direction, push the object, rotate the object or look for help. [24]

Until now handling an object has just been done by pushing it, the following solution pulls the object with a flexible tool. A rope allows to simplify the computing process as well as having more options for moving the object. When the object is surrounded by a rope, the contact surface, mass centre, and rotation are changed. (Figure 2.13) The geometry does not matter so it is easier to move because there is more stability and the controllability grows, so the number of robots decreases. [28]

Coupled or decoupled path planning algorithms for multiple robots are also a solution. The coupled ones create paths for all the robots by combining the states of the robots in a space-state. It uses a centralized architecture. Its complexity augments exponentially with the number of robots so it is not handy to implement. Decoupled algorithms compute the paths for each robot and its main appeal is the decentralized architecture. It simplifies

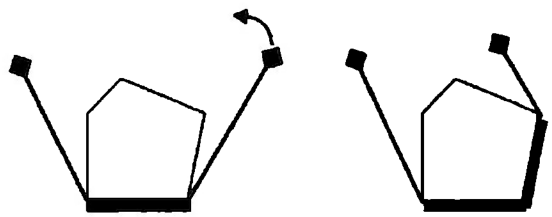
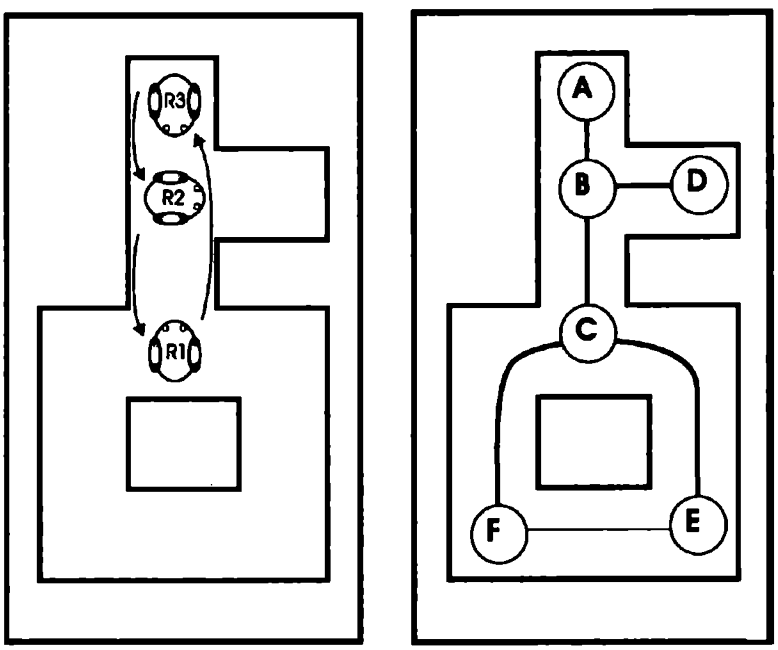


Figure 2.13: Nonprehensile Pulling by Multiple Robots

the implementation because it is all computed in several phases. Before the movement is performed a decision tree must be done with all the possible robot configurations. (Figure 2.14a) During the first phase the robots move to determined leafs of the tree, where a robot r moves to a leaf $\mathcal{L}$. In the second step a collision-free path for the robots is computed. If there is a robot without a path then it is computed. (Figure 2.14b) Finally, only one robot moves at the time, this way there will be no collisions. [23]



(a) Planning Problem (b) Graph-based map

Figure 2.14: Strategy for Coordinating Multiple Robots Within Roadmaps

Finally a hunting system was developed. A group of mobile robots have the main task of capturing a mobile object. Reinforcement Learning was used for providing the hunters certain intelligence. Each of them has a visibility of 2π. Using the available information it can take its own decisions. The hunting algorithm is composed by several states. During the tracking state the robot moves around randomly to explore the environment. The prosecution state activates when the robot *sees* the pray and immediately it begins the chase. During the capture state the hunter asks if any other robot is chasing it, else it hunts it. The last state is prediction, this states appears when the target is lost and an estimation shall be done. [33]

Communication

It results quite important to have a liable and adaptable communication system in a navigation system. The LOCISS (Locally Communicable Infrarred Sensory System) is used for inter-robot identification and IDC (Intelligent Data Carrier) for localization. LOCISS was developed to identify if a mobile object is an obstacle or another team member. With ordinary sensors it turns to be almost impossible to differentiate one from another, so an ID, location and speed information are transmitted. If received data is the same as the one sent then the object is an obstacle. IDC is a mobile device composed by writer and a reader where environment information is stored. The combination of both algorithms outcomes in a local path planning problem. [29]

Chapter 3

Theoretical framework

Robot motion planning, as it has been explained, uses lots of areas. This work employs Navigation concepts, Artificial Intelligence and Path Planning algorithms as well Gaussian and Nonparametric Filters. The previous chapter gave a general overview of some of the concepts that will now be explained deeply. All the algorithms employed for developing this Thesis will be explained in this chapter.

3.1 Motion Planning

3.1.1 Task

A motion planner most important characteristic is according to the problem it is designed to be solved. There are four tasks it must accomplish, they are: navigation, coverage, localization and mapping. In *Navigation* a collision-free motion is calculated between two q or states for the robot. *Coverage* has to do with using a sensor or an actuator interact with the space. *Localization* deals with the problem of interpreting and using a map to interpret sensor data to determine the current q. Finally, *Mapping* has to do exploring and sensing an unknown environment to construct a good enough representation for using it in in navigation, coverage and localization. [17]

3.1.2 Properties of the Robot

A motion planner is strongly bounded to the robot properties while solving the task. This is, the robot and the environment determine the number of DOF and the form of Q. Once it is defined, the robot motion must be known, if it can move instantaneously into any direction in Q it is considered to be omnidirectional, else if it has velocity constraints, such as a car, it is called nonholonomic. Another way to model a robot is by employing kynematic equations with velocities as control, or by the employment of dynamic motion equations controlled by forces. [9]

Kinematic Constraints

In the basic problem we assumed that the robot was a free-flying object, the only constraints on its motions were due to the obstacles. In some problems we may want to impose additional kinematic constraints to the robot's motions.

Holonomic Constraints Let us assume that a configuration is represented by a list of parameters of minimal cardinality. A holonomic equality constraint relates these parameters and can be solved for one of them, this way, the relation reduces the dimension of the current $\mathcal{Q}$ by one.

Suppose a 3D object $\mathcal{A}$ translates freely but it is constrained to rotate in a fixed axis. $\mathcal{A}$'s orientation can be represented by three angles, but it can be expressed as two independent equations. $\mathcal{W}$'s dimension is of 6 while $\mathcal{Q}$'s is of 4. The particular case when $\mathcal{A}$ can translate freely at its current location is considered a holonomic constraint problem, however as this problem is totally equivalent to a motion problem for a point in $\mathbb{R}^N$ it is a particular case of the basic motion planning problem.

Holonomic constraints certainly affect the definition of the robot's configuration space and may even change its global connectedness. Nonetheless, holonomic constraints do not raise new fundamental issues. [17]

Nonholonomic Constraints A non holonomic equality constraint is a non-integrable equation involving the configuration parameters and their derivatives (velocity parameters). Such a constraint does not reduce the dimension of the space of configurations attainable by the robot, but reduces the dimension of the space of possible differential motions at any given configuration. [17]

Consider a car-like robot $\mathcal{A}$ rolling on a flat ground as in Figure 3.1. The car can be modelled as a rectangular object that moves in $\mathcal{W} = \mathbb{R}^2$. In an empty space the robot can be driven at any position with any orientation, its $\mathcal{Q}$ is 3D, two of translation and one of rotation such that $q = \begin{bmatrix} x & y & \theta \end{bmatrix}$. x and y are Cartesian coordinates in a axis $\mathcal{F}_{\mathcal{W}}$, they represent the midpoint R just in the middle R of the robot. $\theta \in [0, 2\pi)$ is the angle between the x axis and the robot $\mathcal{A}$'s orientation. Assuming that there is no slipping and all movements are deterministic, the velocity of R points along $\mathcal{A}$'s axis. Its motion constraint is given by:

$$-\sin\theta dx + \cos\theta dy = 0 \tag{3.1}$$

As equation 3.1 is non-integrable it represents a nonholonomic equality constraint. Due to it, the differential motions $[\delta x \quad \delta y \delta \quad \theta]$ of the robot at any q is a 2D space. If the robot was a free-flying object the space would be 3D. The car's instantaneous motion is determined by two parameters: the linear velocity along its main axis and the steering angle. However, when the steering angle different from zero it affects its orientation, therefore its linear velocity, so the robots's q span in a 3D space.

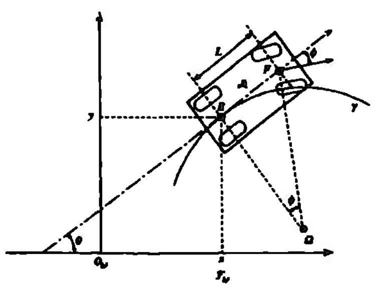


Figure 3.1: Car-like Robot Model

Nonholonomic constraints restrict the geometry of the feasible free paths between two configurations. They are much harder to deal with a planner than holonomic constraints. [17]

Uncertainty The basic problem assumes that the robot can follow exactly paths generated by the planner. It also assumes that the geometry of the robot, the geometry of the obstacles, and the location of the obstacles are accurately known. Currently there are no robot settings that satisfies these assumptions, and both robot control and geometric models are imperfect. In many cases these imperfections can be ignored because the task may allow certain tolerance, but that is not always the case.

In the other hand, the robot may have a small or no knowledge of $\mathcal{W}$, so it would have to trust completely in its sensors at execution time to get enough information of the environment so the task can be accomplished. In this particular case, the robot needs to explore $\mathcal{W}$ and usually this approach is outside motion planning, although it is possible to interweave planning, execution and monitoring activities, but if there is no a priori knowledge path planning has almost no relevance. [18]

A middlepoint situation can be where there is a small error in robot control and in all geometric models but the errors are just found in certain bounded regions: an obstacle is expected to be in a certain location but it is displaced, a robot moves in a different direction than the commanded one but it displaces in a narrow cone centered along the desired direction. To deal with that kind of error the robot shall be equipped with sensors that it can employ during execution so it can acquire additional information. However, sensors are not deterministic either, a position sensor does not always return the exact q in which the robot is. It results that sensors also contain certain error in uncertainty region. If that error can be controlled a motion plan can be generated so it can be tolerant to the overall error. [25]

The motion planning problem with bounded uncertainty can be stated as follows:

> Given an initial region $\mathcal{I}$ and a goal region $\mathcal{G}$ in the robot's configuration space, generate a motion plan whose execution guarantees the robot to reach a configuration in $\mathcal{G}$ if it starts from any (unknown) configuration in $\mathcal{I}$, despite bounded uncertainty in control, sensing and model. A solution to this problem is a plan that combine motion commands and sensor readings that interact at execution time in order to reduce uncertainty and guide the robot toward the goal. [17]

Planning in bounded uncertainty comes up with new issues that are not covered in the basic problem or in its extensions. Due to uncertainty in control a motion command may produce any path among the infinitely many ones which are consistent with both the command and the uncertainty, all paths must reach the goal so the planner can guarantee success. The plan must also finish in the goal, but due to uncertainty in sensing, it may be a big problem to know if the goal has been reached. The planner must also retrieve enough information that will allow the controller to choose the correct actions.

Uncertainty leads to the usage of sensor-based motion commands whose behaviour is less sensitive to errors than purely position-controlled motion commands. *Force-compliant* motion commands are one example of such commands. When used, the robot may touch obstacle surfaces and slide on them, rather than just stop. Planning such sensory-based motion commands may require the physics of the workspace to be taken into consideration. [17]

3.1.3 Properties of the Algorithm

Once the robot and its task have been defined, the algorithms to be used shall be chosen according how they will solve the problem. This is, the planner may come with optimal solutions in a certain criteria such as length, execution time or energy consumption, or just satisfy the constraints. Besides that, computational complexity, memory requirements, running time (constant, polynomial or exponential) must also be considered. The size of the problem description could be the number of DOF of the robot system, the amount of memory needed to describe the robot and the obstacles in the environment, etc., and the complexity can be defined in terms of the works case or the average case. [9]

Some planners are complete, they will always find a solution to the problem if one exists or return failure. This last property is really desirable. In a motion planning problem, as the number of DOF grows, complete solutions require lots of computational resources and may not be feasible to use them. That is why some weaker forms of completeness are sought. One form of that is resolution completeness, if a solution exists in certain discretization resolution, the planner will find a solution. Another form, but weaker, is probabilistic completeness, the probability of finding a solution converges to 1 in an infinite time.

Optimality, completeness and computational complexity naturally trade off with each other. We must be willing to accept increased computational complexity if we demand optimal motion plans or completeness from our planner.

We say a planner is offline if it constructs the plan in advance, based on a known model of the environment, and then hands the plan of to an executor. The planner is online if it incrementally constructs the plan while the robot is executing. In this case, the planner can be sensor-based, meaning that it interleaves sensing, computation and action. The distinction between offline algorithms and online sensor-based algorithms can be somewhat murky; if an offline planner runs quickly enough, for example, then it can be used in a feedback look to continually replan when new sensor data updates the environment model. The primary distinction is computation time, and practically speaking, algorithms are often designed and discussed with this distinction in mind. A similar issue arises in control theory when attempting to distinguish between feedforward control and feedback control, as techniques like model predictive control essentially use fast feedforward control generation in a closed loop. [17]

3.2 Approximate Cell Decomposition

This path planning approach consists of representing the robot's free space $\mathcal{Q}_{free}$ as a collection of cells. Cells are required to have a simple prespecified shape, like a rectangular shape. Cells do not represent exactly the free space, instead they approximate in a conservative way, that is why the name it receives. A connectivity graph representing the adjacency relation among the cells is built and searched for a path. The rules for using a cell shape are:

1. Achieve space decomposition by iterating the same simple computation.

2. To be relatively insensitive to numerically approximate computations.

In this planning method the amount of free space can be controlled for the generated path by establishing a minimal size for the cells. This is important when the error in geometric models and/or robot control is not despicable.

The boundaries of the generated cell are kind of arbitrary, they do not characterize discontinuities in motion constraints. As they conservatively represent the free space they may fail to find a free path, even if one exists. This drawback can be attacked by augmenting the time it can employ to find a solution. [18]

Most approximate cell decomposition methods allow the size of the cells to be locally adapted to the geometry of the obstacle region. Presetting the size of the cells could result in significant difficulties: a large cell size would prevent free paths from being found, while a small size would require increased computation times. So most methods operate in a hierarchical way, they generate an initial coarse decomposition and then locally refining this decomposition until a free path is found or the decomposition becomes too small.

The principle of the approximate cell decomposition approach can be applied to the basic motion planning problem in its full generality, as well to most of its extensions. However, the time and space complexity of the methods based on this approach grows quickly with the dimension m of the configuration space. These methods are applied only when this dimension is small enough.

Description

A rectangle is defined as a closed region of the following form in a Cartesian space $\mathbf{R}^n$:

$$\{(x_1, \dots, x_n) | x_1 \in [x'_1, \ x''_1], \dots, x_n \in [x'_n, \ x''_n]\} \tag{3.2}$$

The differences $x''_i - x'_i$, $i = 1, \dots, n$ are called the dimensions of the rectangle. None of this is zero. Let R be a robot whose configuration space $\mathcal{Q}$ is $\mathbf{R}^N$ with $N = 2$ or 3. A configuration q is represented by the coordinates of R's reference point $\mathcal{P}_{\mathcal{R}}$ in the frame $\mathcal{F}_{\mathcal{W}}$ attached to the workspace.

We assume that the set of possible positions of R is contained in a rectangle $D \subset \mathbf{R}^N$. We represent $\mathcal{Q}_{free}$ as:

$$\mathcal{Q}_{free} = \frac{\mathcal{R}}{\mathcal{QO}} \tag{3.3}$$

Where $\mathcal{R} = int(D)$ if $\mathcal{Q} = \mathbf{R}^N$

Let $\Omega = cl(R)$. It is a rectangle of $\mathbf{R}^m$, where m is the dimension of the configuration space $\mathcal{Q}$.

A rectangle decomposition $\mathcal{P}$ of Ω is a finite collection of rectangles $\{\kappa_i\}_{i=1,\dots,r}$ such that:

- Ω is equal to the union of κ_i:

$$\Omega = \bigcup_{i=1}^{r} \kappa_i \tag{3.4}$$

- The interiors of the κ_i's do not intersect

$$\forall i_1, i_2 \in [1, r], i_1 \neq i_2 : int(\kappa_{i_1}) \cap int(\kappa_{i_2}) = \emptyset \tag{3.5}$$

Each rectangle κ_i is called a cell of the decomposition $\mathcal{P}$ of Ω.

Two cells are adjacent if and only if their intersection is a set of non-zero measure in $\mathbf{R}^{m-1}$. The intersection is computed by taking into account that $\mathcal{Q} = \mathbf{R}^2 \times S^1, (x, y, 2\pi)$ is identified with $(x, y, 0)$.

A cell κ_i is classified as:

- EMPTY - if and only if its interior does not intersect an obstacle region. $\kappa_i) \cap \mathcal{O} = \emptyset$.

- FULL - if and only if κ_i), is entirely contained in the obstacle region, $\kappa_i) \subseteq \mathcal{O}$.
- MIXED - otherwise.

The connectivity graph associated with a decomposition $\mathcal{P}$ of $\Omega$is the non-directed graph G defined as follows:

- The nodes of G are the EMPTY and MIXED cells of $\mathcal{P}$.
- Two nodes of G are connected by a link if and only if the corresponding cells are adjacent.

Given a rectangle decomposition $\mathcal{P}$ of Ω, a channel is defined as a sequence $(\kappa_{\alpha_j})_{j=1,\ldots,p}$ of EMPTY and/or MIXED cells such that any two consecutive cells κ_{α_j} and $\kappa_{\alpha_{j+1}}$, $j \in [1, p-1]$, are adjacent. A channel that only contains EMPTY Cells is called an E-channel. A channel that contains at least one MIXED cell is called an M channel. If $(\kappa_{\alpha_j})_{j=1,\ldots,p}$ is an E-channel, then any path connecting any configuration in (κ_{α_1}) to any configuration (κ_{α_p}) and lying in $int(\cup_{j=1}^{p} \kappa_{\alpha_j})$ is a free path. If $(\kappa_{\alpha_j})_{j=1,\ldots,p}$ is an M-channel, there may exist a free path connecting two configurations (κ_{α_1}) and (κ_{α_p}), and lying in $int(\cup_{j=1}^{p} \kappa_{\alpha_j})$, but there is no guarantee that this is the case. [18]

Given an initial configuration $q_i \in \mathcal{Q}_{free}$ and a goal configuration $q_f \in \mathcal{Q}_{free}$, the problem is to generate an E-channel $(\kappa_{\alpha_j})_{j=1,\ldots,p}$, such that $q_i \in \kappa_{\alpha_1}$, and $q_f \in \kappa_{\alpha_p}$. If such a channel is generated, let $\beta_j = \partial\kappa_{\alpha_j} \cap \kappa_{\alpha_{j+1}}, j = 1, \ldots, p-1$, be the intersection of the boundaries of two successive cells. A free path joining the initial to the goal configuration can be extracted from the E-channel by linking q_i to q_f by a polygonal line whose vertices are points $Q_J \in int(\beta_j)$. For every j such that β_{j-1} and β_j are subsets of the same face of κ_α, an additional point Q'_{j-1} located in the interior of κ_{α_j} should be included among the path's vertices, since in this case the line segment $Q_{j-1}Q_j$ is not guaranteed to lie entirely in the robot's free space. If necessary, the polygonal path can be smoothed. [17]

Hierarchical path planning consists of generating an E-channel by constructing successive rectangle decompositions of Ω and searching the associated connectivity graphs. Let $\mathcal{P}_i$, $i = 1, 2, \ldots$, denote the successive decompositions of Ω. Each decomposition $\mathcal{P}_i$ is obtained from the previous one, $\mathcal{P}_{i-1}$ (with $\mathcal{P}_0 = \{\Omega\}$), by decomposing one or several MIXED cells, the other cells being unchanged. Whenever a decomposition $\mathcal{P}_i$, is computed, the associated connectivity graph, denoted by G_i, is searched for a channel connecting q_i to q_f.

input : Connectivity graph G_i
κ: MIXED cell;
Π_i: M-channel;
Compute a rectangle decomposition $\mathcal{P}$ of Ω;
$i \leftarrow 0$;
repeat
    channel $\leftarrow$ `SearchForChannel(`G_i`)`;
    if channel $==$ *E-channel* **then**
        return Success;
    else if channel $==$ *M-channel* **then**
        $\mathcal{P}_{i+1} \leftarrow \mathcal{P}_i$;
        foreach κ *in* Π_i **do**
            `RectangleDecomposition(`κ`)`;
            $\mathcal{P}_{i+1} \leftarrow [\mathcal{P}_{i+1} \backslash \{\kappa\}] \cup \mathcal{P}^{\kappa}$;
        end
    end
until *Success* ;

Algorithm 1: Approximate Cell Decomposition

The search of G_i can be guided by an heuristic. In particular, one could search for an E-channel before searching for an M-channel. But, although the heuristic function should put an extra cost on MIXED cells in order to generate an E-channel quicker, it may also be appropriate to prefer short channels over long ones. Thus, although an E-channel may exist in a graph G_i, it may nevertheless be preferable to generate a significantly shorter M-channel instead, and refine G_i accordingly. Notice that any E-channel existing in G_i will continue to exists in all the graphs G_j, $j > 1$. [18]

Let us assume that the region $cl(\mathcal{Q}_{free})$ is a manifold with boundary. Then the algorithm can be made complete - guaranteed to terminate and return an E-channel whenever q_i and q_f lie in the same connected component of $\mathcal{Q}_{free}$ - by working out some details appropriately, for instance:

- The search of the connectivity graph should be complete and it should output an E-channel whenever one exits.

- All the dimensions of every MIXED cell in $\mathcal{P}_i$ should tend toward 0 when $i \rightarrow \infty$.

However, for an unknown region of $\mathcal{Q}_{free}$, there is no upper bound on the worst-case computation time.

The computing time can be bounded at the expense of completeness by imposing constraints on the minimal dimensions of the cell. For example, one possible constraint is that the total volume of the EMPTY and FULL Cells in the decomposition $\mathcal{P}^{\kappa}$ of κ be greater than a predefined ratio $\lambda \in (0, 1)$ of the volume of κ; in addition, every MIXED cell whose volume is smaller than a prespecified value ϵ is re-labelled as FULL. If such a constraint is imposed, the algorithm is no longer guaranteed to output an E-channel whenever one exists. However, if one exists, the algorithm will find one provided that both λ and ϵ are selected small enough. For this reason, the planning method is said to be resolution-complete. [17]

3.3 A* (A star)

Informed search algorithms employ problem-specific knowledge beyond the definition of itself. It finds solutions in a more efficient way than uninformed search algorithms. The general form of A* is called best-first search, which is a generalization of the Tree or Graph Search algorithm. In it, a node is selected for expanding it based on an evaluation function $f(n)$. That function is calculated as a cost estimated, so the node with the lowest evaluation is expanded first. The choice made by f will determine the whole strategy. Best-first algorithms use an heuristic function $h(n)$ to be included as a component of f. Heuristic functions are a way of providing additional knowledge of the problem to the search algorithm.

The most popular variation of the Best-first Search is A* search. Its nodes evaluation system consists of the combination of the cost to reach the node $g(n)$ and $h(n)$, the cost to get from the node to the goal:

$$f(n) = g(n) + h(n) \tag{3.6}$$

$g(n)$ gives is the cost function of moving from the start node a node n, in the other hand, $h(n)$ gives the estimated cost of the path with the lowest combined heuristic. By combining them the estimated cost $f(n)$ is calculated.

If the cheapest solution is being searched, what is usually done is to try the lowest value node of $g(n) + h(n)$. Using this strategy, it can be demonstrated that A* search is both complete and optimal.

```
input  : problem
output: A solution, or failure
node: a node with STATE = problem. INITIAL-STATE, PATH-COST =
0;
frontier: a priority queue ordered by PATH-COST, with node as the
only element;
explored: an empty set
repeat
    if Empty?(frontier) then return failure;
    node: Pop(frontier);
    if problem.GoalTest(node.STATE) then return Solution(node);
    add node.STATE to explored;
    foreach action in problem.Actions(node.STATE) do
        child ← ChildNode(problem, node, action);
        if child.STATE is not in explored or frontier then
            frontier ← Insert(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
            replace that frontier node with child
        end
    end
until Solution or failure ;
```

Algorithm 2: A* Search

Conditions for optimality: Admissibility and consistency

$h(n)$ must be an admissible heuristic so optimality can be reached. To be admissible it must not overestimate the cost to reach the goal. As $g(n)$ represents the actual cost to reach n through the path, and $f(n) = g(n) + h(n)$, it will never overestimate the true cost of a solution in the current path through n.

Admissible heuristics are usually optimistic, this is, they estimate the cost of solving the problem is smaller than it really is. Monotocity or consistency is another required condition for using A* in a graph search. A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n, a, n') + h(n') \tag{3.7}$$

The previous equation is a form of the general triangle inequality, it stipulates that each triangle's edge cannot be longer than the sum of the other two. The triangle is formed by n, n' and the goal G_n closest to n. In an admissible heuristic, the inequality can be interpreted this way: if a route from n to G_n exists via n' and it is cheaper than $h(n)$, it will violate the property that $h(n)$ is a lower bound on the cost to reach the goal node.

A* is complete, optimal and optimally efficient, but it is not always the best solution for all search needs. This is, in some problems the number of states that surround the goal is exponential in the length of the solution. The complexity of A* can provoke that it is not feasible to find an optimal solution. To solve it, A* modifications have been proposed, they find sub-optimal solutions more rapidly, or a good search heuristics can be designed, still it will give enormous savings compared to an uninformed search..

Its main drawback is computational time, it saves all generated nodes in memory. It is not rare that it runs out of memory before a timeout occurs. That is why A* is not used in large-scale problems, yet some algorithms overcome the space problem while taking care of not sacrificing optimality or completeness, however execution time is slightly affected. [27]

3.4 Kalman Filter

Gaussian techniques all share the basic idea that beliefs are represented by multivariate normal distributions:

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{3.8}$$

The density over x is characterized by the mean μ and the covariance Σ. μ is a vector that has the same dimensionality as x. Σ is a quadratic matrix that is symmetric and positive-semidefinite. The dimension is the dimensionality of the state x squared.

The commitment to represent the posterior by a Gaussian has important ramifications. This is, Gaussians are unimodal: they have a single maximum, this characteristic is widely used for tracking problems in robotics: the posterior is focused around the true state with a small margin of uncertainty. Gaussians posteriors are a poor match for global estimation problems where many hypotheses exists because each of them forms its own mode in the posterior. [18]

Linear Gaussian Systems

The moments of parametrization is the parametrization of a Gaussian by its mean and covariance. Each of them are the first and second moments of a probability distribution; the rest of them are zero for a normal distribution. The Kalman filter is the best studied technique for implementing Bayes filters. It was invented by Swerling and Kalman as a way for filtering and predicting the behaviour of a Linear Gaussian System. It implements a belief computation for continuous states, so it cannot be applied in discrete or hybrid space states. It represents beliefs by the moments parametrization. At time t, the current belief is represented by the mean μ_t and covariance Σ_t. Posteriors are Gaussian if the following three properties hold, in addition to the Markov assumptions of the Bayes filter. [27]

1. The state transition probability $p(x_t|u_t, x_{t-1})$ must be a linear function in its arguments with added Gaussian noise. This is expressed as:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{3.9}$$

Here x_t and x_{t-1} are state vectors, and u_t is the control vector at time t. Both of these vectors are vertical vectors. They are of the form:

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \tag{3.10}$$

and

$$u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{n,t} \end{pmatrix} \tag{3.11}$$

A_t and B_t are matrices. A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t. B_t is of size $n \times m$, with m being the dimension of the control vector u_t. By multiplying the state and control vector with the matrices A_t and B_t, respectively, the state transition function becomes linear in its arguments. Thus, Kalman filters assume linear system dynamics.

The random variable ϵ_t is a Gaussian random vector that models the uncertainty introduced by the state transition. It is of the same dimension as the state vector. Its mean is zero, and its covariance will be denoted R_t. A state transition probability as seen in (3.9) is called linear Gaussian, to reflect the fact that it is linear in its arguments with additive Gaussian noise. [18]

Equation (3.9) defines the state transition probability $p(x_t|u_t, xt-1)$. This probability is obtained by plugging (3.9) into the definition of the multivariate normal distribution. The mean of the posterior state is given by $A_t x_{t-1} + B_t u_t$ and the covariance by R_t.

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}}$$
$$\exp\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\} \tag{3.12}$$

2. The measurement probability $p(z_t|x_t)$ must also be linear in its arguments, with added Gaussian noise:

$$z_t = C_t x_t + \delta_t \tag{3.13}$$

Here C_t is a matrix of size $k \times n$, where k is the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise. The distribution of δ_t is a multivariate Gaussian with zero mean and covariance Q_t. The measurement probability is thus given by the following multivariate normal distribution:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\} \tag{3.14}$$

3. Finally, the initial belief $bel(x_0)$ must be normally distributed. This belief will have mean μ_0 and covariance by Σ_0.

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\} \tag{3.15}$$

These three assumptions are sufficient to ensure that the posterior $bel(x_t$ is always a Gaussian, for any point in time t.

Algorithm

Kalman filters represent the belief $bel(x_t)$ at time t by the mean μ_t and the covariance Σ_t. The input of the Kalman filter is the belief at time $t-1$, represented by μ_{t-1} and Σ_{t-1}. To update these parameter, Kalman filters require the control u_t and the measurement z_t. The output is the belief at time t represented by μ_t and Σ_t.

In lines 1 and 2, the predicted belief $\overline{\mu}$ and $\overline{\Sigma}$ is calculated representing the belief $\overline{bel}(x_t)$ one step later, but before incorporating the measurement z_t. This belief is obtained by incorporating the control u_t. The mean is updated using the deterministic version of the state transition function (3.9), with the mean μ_{t-1} substituted for the state x_{t-1}. The update of the covariance considers the fact that states depend on previous states through the linear matrix A_t. This matrix is multiplied twice into the covariance, since the covariance is quadratic matrix.

input: $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
1 $\overline{\mu}_t = A_t\mu_{t-1} + B_t u_t$;
2 $\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$;
3 $K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$;
4 $\mu_t = \overline{\mu}_t + K_t(z_t - C_t\overline{\mu}_t)$;
5 $\Sigma_t = (I - k_t C_t)\overline{\Sigma}_t$;
6 **return** μ_{t-1}, Σ_{t-1}

Algorithm 3: The Kalman Filter Algorithm

The belief $\overline{bel}(x_t)$ is subsequently transformed into the desired belief $bel(x_t)$ in lines 3 through 5, by incorporating the measurement z_t. The variable K_t, computed in line 3 is called Kalman gain. It specifies the degree to which the measurement is incorporated into the new state estimate. Line 4 manipulates the mean, by adjusting it in proportion to the Kalman gain

K_t and the deviation of the actual measurement, z_t, and the measurement predicted according to the measurement probability (3.13). The key concept here is the innovation, which is the difference between the actual measurement z_t and the expected measurement $C_t\bar{\mu}_t$ in line 4. Finally, the new covariance of the posterior belief is calculated in line 5, adjusting for the information gain resulting from the measurement. [25]

The Kalman filter is computationally quite efficient. Each iteration of the Kalman filter is lower bounded by $O(k^{2.4})$, where k, where k is the dimension of the measurement vector z_t. This approximate cubic complexity stems from the matrix inversion in line 3. [18]

3.5 Particle Filter

Nonparametric filters are an alternative to Gaussian techniques. The posterior they use is represented by a finite number of values which correspond to a region in the space state. Some nonparametric Bayes filters decompose the state space, each of the correspond to a probability of the posterior density in a region of the space state. Some others approximate the space state by sampling the posterior distribution. The number of parameters can be varied such that the quality of the approximation depends on it. If that number goes to infinity, nonparametric techniques converge uniformly to a correct posterior under smoothness assumptions. [18]

Basic Algorithm

The particle filter is a nonparametric implementation of the Bayes filter. It approximates the posterior using a finite number of parameters. It represents the posterior $bel(x_t)$ by a set of random state samples taken from the posterior. Rather than representing the distribution in a parametric form - like a normal distribution - particle filters represent a distribution by a set of samples drawn from this distribution. This distribution is approximate and nonparametric so it can represent much better a space of distributions. Another advantage is that it can model nonlinear transformations of random variables. [25]

In particle filters, the samples of a posterior distribution are called particles and are denoted

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \tag{3.16}$$

Each particle $x_t^{[m]}$ (with $1 \leq m \leq M$) is a concrete instantiation of the state at time t. Put differently, a particle is hypothesis as to what the true world state may be at time t. Here M denotes the number of particles in the particle set $\mathcal{X}_t$. In practice, the number of particles M is often a large number. In some implementations M is a function of t or other quantities related to the belief $bel(x_t)$.

The intuition behind particle filters is to approximate the belief $bel(x_t)$ by the set of particles $\mathcal{X}_{\sqcup}$. Ideally, the likehood for a state hypothesis x_t to be included in the particle set $\mathcal{X}_t$ shall be proportional to its Bayes filter posterior $bel(x_t)$:

$$x_t^{[m]} \sim p(x|z_{1:t}, u_{1:t}) \tag{3.17}$$

As a consequence of it, the denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region. As it will discussed, the property (3.17) holds only asymptotically for $M \uparrow \infty$ for the standard particle filter algorithm. For finite M, particles are drawn from a slightly different distribution. In practice,, this difference is negligible as long as the number of particles is not too small.

The particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that particle filters construct the particle set $\mathcal{X}_t$ recursively from the set $\mathcal{X}_{t-1}$.

input: $\mathcal{X}_t, u_t, z_t$
$\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$;
for m = *1 to M* **do**
  sample $x_t^m \sim p(x_t|u_t, x_{t-1}^{[m]})$;
  $w_t^{[m]} = p(z_t|x_t^{[m]})$;
  $\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$;
end
for m = *1 to M* **do**
  draw i with probability $\propto w_t^{[i]}$;
  add $x_t^{[i]}$ to $\mathcal{X}_t$;
end
return $\mathcal{X}_t$

Algorithm 4: The Particle Filter Algorithm

The input of this algorithm is the particle set $\mathcal{X}_{t-1}$, along with the most recent control u_t and the most recent measurement z_t. The algorithm then first constructs a temporary particle set $\overline{\mathcal{X}}$ that represented the belief $\overline{bel}(x_t)$. It does this by systematically processing each particle $x_{t-1}^{[m]}$ in the input particle set $\mathcal{X}_{\sqcup-\infty}$. Subsequently, it transforms these particles into the set $\mathcal{X}_t$, which approximates the posterior distribution $bel(x_t)$. In detail:

1. Line 3 generates a hypothetical state $x_t^{[m]}$ for time t based on the particle $x_{t-1}^{[m]}$ and the control u_t. The resulting sample is indexed by m, indicating that it is generated from the m-th particle in $\mathcal{X}_{t-1}$. This step involves sampling from the state transition distribution $p(x_t|u_t, x_{t-1})$. To implement this step, one needs to be able to sample from this distribution. The set of particles obtained after M iterations is the filter's representation of $\overline{bel}(x_t)$.

2. Line 4 calculates for each particle $x_t^{[m]}$ the so-called importance factor, denoted $w_t^{[m]}$. Importance factors are used to incorporate the measurement z_t into the particle set. The importance, thus, is the probability of the measurement z_t under particle x_t^m, given by $w_t^{[m]} = p(z_t|x_t^{[m]})$. If we interpret $w_t^{[m]}$ as the weight of a particle, the set of weighted particles represents (in approximation) the Bayes filter posterior $bel(x_t)$.

3. During the second *for*, resampling or importance sampling is implemented. The algorithm draws with replacement M particles from the temporary set $\overline{\mathcal{X}}_t$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the importance weights into the resampling process, the distribution of the particles change: Whereas before the resampling step, they were distributed according to $\overline{bel}(x_t)$, after the resampling they are distributed (approximately according to the posterior $bel(x_t) = \eta p(z_t|x_t^{[m]})\overline{bel}(x_t)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement. More important are the particles not contained in $\mathcal{X}_t$: Those tend to be the particles with lower importance weights.

The resampling step has the important function to force particles back to the posterior $bel(x_t)$. In fact, an alternative (and usually inferior) version of the particle filter would never resample, but instead would maintain for each particle an importance weight that is initialized by 1 and updated multiplicatively:

$$w_t^{[w]} = p(z_t|x_t^m)w_{t-1}^{[w]} \tag{3.18}$$

Such a particle filter algorithm would still approximate the posterior, but many of its particles would end up in regions of low posterior probability. As a result, it would require many more particles; how many depends on the shape of the posterior. The resampling step is a probabilistic implementation of the Darwinian idea of survival of the fittest: It refocuses the particle set to regions in state space with high posterior probability. By doing so, it focuses the computational resources of the fiilter algorithm to regions in the state space where they matter the most. [25]

Chapter 4

Employed Hardware

This chapter includes a description of the employed and developed hardware for this Thesis. This work's main goal is to implement several algorithms and make them interact so the task of transporting a heavy object is achieved, but it would not be accomplished without the correct hardware.

4.1 Boe-Bot Robot

The Parallax Boe-Bot robot (Figure 4.1) is the focus of the activities and projects for various type of students, from a beginner student to an advanced designer. Its popularity comes from how versatile it is for being modified. It can be used as an standard mobile robot with wheels that uses servo motors. [3] Another possibility is using those motors to turn it into a crawler robot that can interact in other type of environments. [4] A final modification provided by the manufacturer is adding the capability of carrying objects by using a gripper that can handle up to 14 pounds. [5] Its chassis can accept several modifications into it so its functionality can only grow.

4.2 BASIC Stamp 2

A Microcontroller is a programmable device that is designed into watches, cellphones, calculator, etc. In robot applications, the microcontroller is programmed to sense when a button has been pressed, to communicate, read sensors or move. [19] The BASIC Stamp is widely used in educational, hobby, and industrial applications. (Figure 4.2) Its main capabilities are:

1. Processor Speed: 20 MHz.

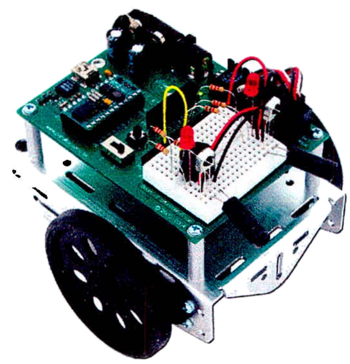


Figure 4.1: Boe-Bot Robot

2. Program Execution Speed: ~ 4000 PBASIC instructions/sec.
3. RAM Size: 32 Bytes.
4. EEPROM Size: 2 KBytes ~ 500 PBASIC instructions.
5. I/O pins: 16 + 2 dedicated serial.
6. Current Draw: @5 VDC: 3 mA Run, 50 μA Sleep.
7. Source/Sink Current per I/O: 20 mA / 25 mA
8. Source/Sink Current per unit: 40 mA / 50 mA per 7 I/O.
9. PBASIC Commands: 42.
10. Package: 24-pin DIP
11. Industrial-Rated since Rev J

This microcontroller is designed to interact in a friendly way with the Boe-Bot robot and with most of the sensors that will be described in this Chapter, that is why it was chosen. It is a low cost high capability microcontroller which I/O ports can perform the typical functions of a pin, but its main advantage is that they can work as PWM outputs, Serial TX/RX pins and with some software modifications as an I^2C. Most of the products in the market have one or two serial ports and limited PWM dedicated pins so if several sensors need serial communications those microcontrollers would not be a good choice. It was also included with the Boe-Bots that were used, that was an extra reason of why to use it.

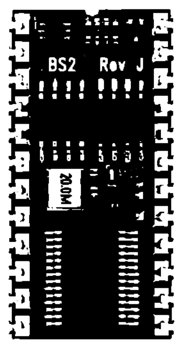


Figure 4.2: BASIC Stamp 2 Microcontroller

4.3 Bluetooth Module

A Robot needs to communicate with other agents or with a computer depending if it works in a centralized or decentralized way. When working in a prototype, usually wired communication is used, usually serial or USB. But when a robot needs to work in a remote location, wireless communication becomes quite important. The Embedded Blue 500 module will be used, it is shown in Figure 4.3.

Bluetooth is a technology standard for electronic devices to communicate with each other using a short-range radio. It is ofter referred to as a "cable replacement" technology, because it is commonly used to connect things that have traditionally been connected by wires. It is based on a frequency hopping spread spectrum modulation (FHSS) technique. The term spread spectrum describes a number of methods for spreading a radio signal over multiple frequencies, either simultaneously or in series. Bluetooth utilizes FHSS to reduce interference and increase security. The signal is rapidly switched from channel to channel many times per second in a pseudo-random pattern that is known by the sender and the receiver. This provides robust recovery of packet errors caused by interference from another radio source at a particular frequency. Also, data is generally more secure because it is not possible to receive more than a fraction of the data unless the hopping patter is known. Bluetooth utilizes frequency hopping in the 2.4 GHz radio band and hops at a relatively fast pace with a raw data rate of about 1 Mbps. This translates to about 700 kbps of actual useful data transfer. The eb500 module supports a maximum sustained bidirectional data rate of 230.4 kbps.

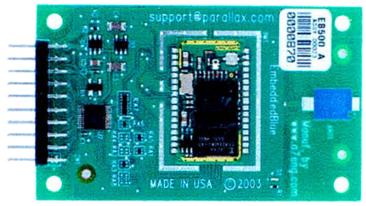


Figure 4.3: EmbeddedBlue 500

The eb500 supports two main operating modes: command mode and data mode. Upon power up, the eb500 enters command mode and is ready to accept serial commands. In this mode baudrate can be changed, also locate other devices and check firmware version. Once the eb500 radio is connected to another Bluetooth device, the eb500 automatically switches into data mode. All data transmitted while in this mode will be sent to the remote the remote device and no further commands can be sent until the eb500 radio is disconnected or switched back to command mode. [11]

4.4 GPS

A Global Positioning System is a space-based navigation system that can provides several information such as time, location, speed, course, date, satellites in view, elevation, azimuth, signal strength, and local zone time among others. It is a program of the U.S. Department of Defence. It was developed to overcome limitations of previous systems. The receiver uses the messages it receives to determine the transit time of each message and computes the distance to each satellite using the speed of light. Each of these distances and satellites' locations define a sphere. The receiver is on the surface of each of these spheres when the distances and the satellites' locations are correct. Many GPS units show derived information such as direction and speed, calculated from position changes. In typical GPS operation, four or more satellites must be visible to obtain an accurate result.

The PMB-648 GPS (Figure 4.4) has several operation modes where different data is returned. The selected mode is the Recommended Minimum Specific GNSS Data (RMC). [26] The output message has the form of:

$$\texttt{\$GPRMC}, 161229.487, \texttt{A}, 3723.2475, \texttt{N}, 12158.3416, \texttt{W}, 0.13, 309.62, 120598, , *10 \tag{4.1}$$

Figure 4.4: PMB-648 GPS SiRF Internal Antenna

The data contained in the messages is separated by a comma, the values are:

1. Message ID - RMC protocol header.
2. UTC Time - Coordinated Universal Time in the `hhmmss.sss` format.
3. Status - Valid or non valid data.
4. Latitude - Latitude location in the `ddmm.mmmm` format.
5. N/S Indicator - Indicates if the location is at the North or at the South.
6. Longitude - Longitude location in the `dddmm.mmmm` format.
7. E/W Indicator - Indicates if the location is at the East or at the West.
8. Speed Over Ground - Speed in knots.
9. Course Over Ground - Course in degrees.
10. Date - Date in the `ddmmyy` format.
11. Magnetic Variation - It can be East or West.

4.5 Compass

The Compass Module 3-Axis HMC5883L (Figure 4.5) is a low-field magnetic sensing device with a digital interface. The compass module converts any magnetic field to a differential voltage output on 3 axes. This voltage shift is the raw digital output value, which can then be used to calculate headings or sense magnetic fields coming from different directions. The module is designed for use with a large variety of microcontrollers with different voltage requirements.

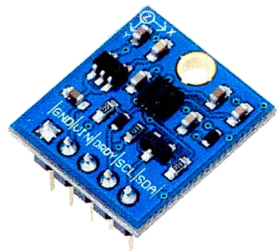


Figure 4.5: Compass Module 3-Axis HMC5883L

The compass module communicates via a two-wire I^2C bus system as a slave device. It supports standard and fast modes, 100 kHz and 400 kHz, but does not support a high-speed mode. No external pull-up resistors are required to support these standard and fast speed modes. The compass returns the value of three magnetic fields in a Cartesian environment, so to turn it into degrees or radians a Cartesian to Spherical conversion must be performed. [1]

4.6 Laser Range Finder

The Parallax Laser Range Finder (LRF) Module is a distance-measuring instrument that uses laser technology to calculate the distance to a targeted object. The design uses a Propeller processor, CMOS camera, and laser diode to create a low-cost laser range finder as shown in FIgure 4.6. Distance to a targeted object is calculated by optical triangulation using simple trigonometry between the centroid of laser light, camera, and object.

Its optimal measurement range of 15 - 122 cm with an accuracy error ¡5%, average 3%. The maximum object detection distance is 2.4 meters. It is communicated with asynchronous serial 300 - 115,200 baud with automatic baud rate detection.

4.7 Infrared Line Follower

The Infrared Line Follower Kit from Parallax provides eight infrared emitter and receiver pairs for high - precision line-following applications (Figure 4.7). Upon connecting power, the onboard ICM7555 chip begins sending a 38 - 43 kHz signal through all 8 IR LEDs. If the IR LED is over a white surface, light is reflected to the IR receiver, and its output is low. When the IR

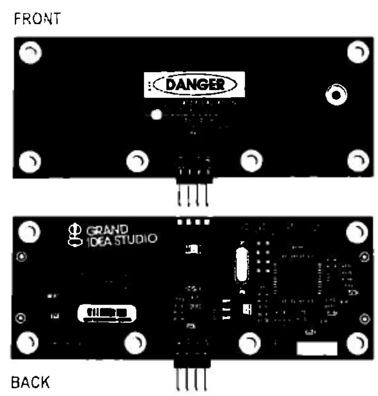


Figure 4.6: Laser Range Finder

LED is over a black surface, no light is reflected to the IR receiver, and its output is high. Red LEDs located on the top of the board are wired to the output of each IR receiver, and the anode of each LED is connected to power. When the IR LED is over a white surface, the low signal completes the LED circuit and turns the LED on. Conversely, when the IR LED is over a black surface, the LED receives dual high signals and the LED is off. This allows for easy visual feedback of the Infrared Line Follower's output states. An onboard potentiometer also allows for the easy adjustment of the infrared frequency between 38 and 43 kHz. This allows the sensor to detect different coloured lines, and also allows for the easy adjustment of the sensor to different lighting conditions or mounting positions. [2]

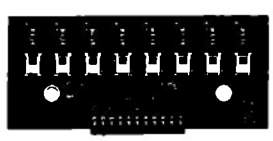


Figure 4.7: Infrared Line Follower

4.8 Robot Design and Implementation

4.8.1 Leader Robot

The leader robot needed to be equipped with the Laser Range Finder to measure the distance between itself and an obstacle in the environment.

However, if the LRF kept just one position the information that could be retrieved from the surroundings would be minimum. That is why it was mounted over a standard servo motor so it could cover the environment in an interval $[0-\pi]$. Now it is capable of acquiring more valuable information.

Another sensor with which it was equipped is the PMB-648 GPS. As it is working in the RMC mode, only the TTL serial output, power and ground wires shall be connected. The GPS is located at the top of the Transport Platform so it has a clear "view" to the satellites to assure liable information. As this robot leads the other ones, it needs far much more "force" to move. Usually, Boe-Bots are equipped with continuous rotation servos with a torque of 2.5 kg-cm. With this torque it could not move freely, that is why those servos were replaced with high torque servo motors that have a 13 kg-cm torque. Finally, as the Thesis' main goal is to transport an object from one place to another, a mechanical structure was added at its back so it could carry part of the platform. The complete implementation is shown in Figure 4.8.

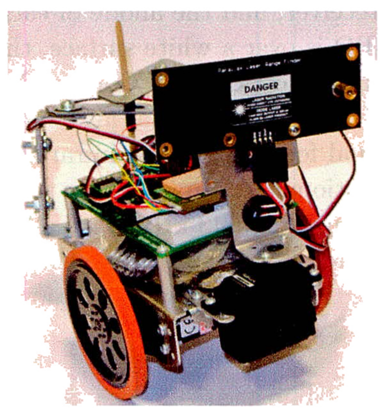


Figure 4.8: Leader Robot

Electronic Schematic

The Leader Robot circuit looks as in Figure 4.9. The Basic STAMP microcontroller is the Robot's core. It is in charge of receiving commands, retrieving data from sensors and performing motion actions. The integrated continuous rotation servo motors the robot has are wired to pins 12 and 13. In the figure they are not represented, nonetheless they are used. The GPS is wired into pin 1, and supplied by 8.4 V. The LRF serial input goes to pin 9 while the output to pin 15, this sensor is quite more complex than any other, that is why it has specific communication ports. The standard servo motor that is in charge of moving the LRF is controlled by an output signal generated in pin 14. The serial communication lines that are used between the Leader and the Transmitter are pins 10 (input) and 11 (output). This wires are not represented in the figure.

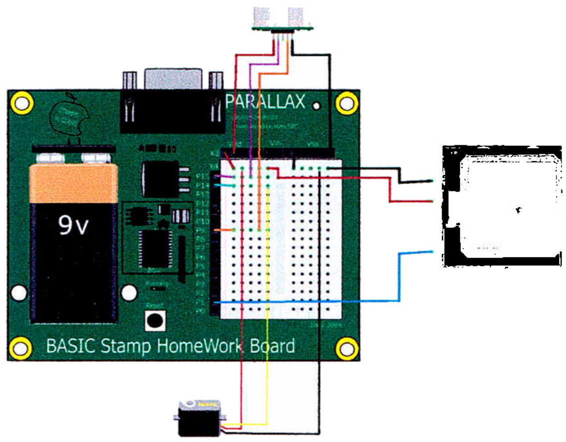


Figure 4.9: Leader Robot Schematic

Mechanic Design

The mechanical structure it has on its back is quite simple, it just provides the robot with a toehold that pulls the Transport platform. It had to be symmetrical so the handled weight could be evenly distributed through both sides of the robot, Another reason was for not affecting any of the motors performance because the desired movement will be affected because of the weight. A wooden stick is used for coupling the structure with the platform.

4.8.2 Transmitter Board

During the initial plans, the leader robot was equipped with all the sensors and communication devices: GPS, LRF, Compass and Bluetooth module, but while implementing it a RAM capacity problem came through: it was not enough. The memory problem will be explained in a deeper way in the next chapter. That is why an extra board was needed to handle the Compass and the Bluetooth module. The hardware split came up with a new need, somehow the leader and the transmitter should communicate. To solve it, serial TX and RX wires were used among them, that way data could flow between them.

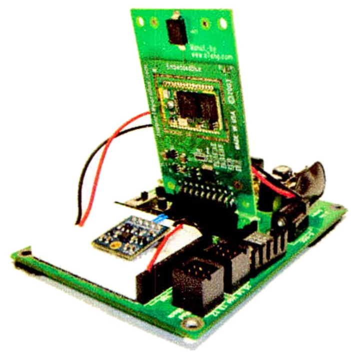


Figure 4.10: Transmitter Board

Electronic Schematic

This schematic is quite simple, there are not much elements in Figure 4.11. The Transmitter collects data from the Compass, sends commands to the Leader, receives information from it to be resent to the Planner. The Compass is an I^2C sensor so it only has four pins: supply, ground, SDA and SCL. SDA stands for Serial Data Line, in it the information flows from the master to the slave and backwards. SCL is a synchronization signal. SDA is connected to pin 9 and SCL to pin 10. The serial communication lines are pin 15 for output and pin 14 for input. The BT module is connected to pin 0 for input and pin 1 for output. Pin 5 is used as an enable pin for starting transmission capabilities to the module.

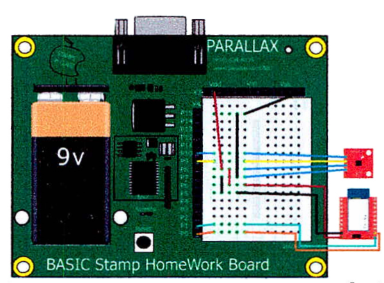
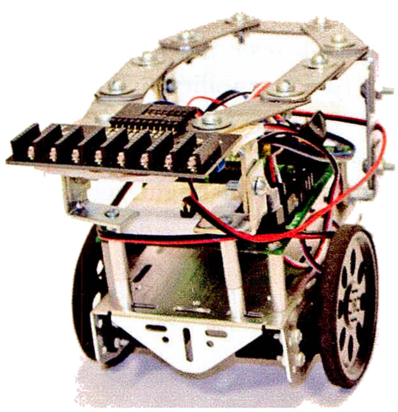


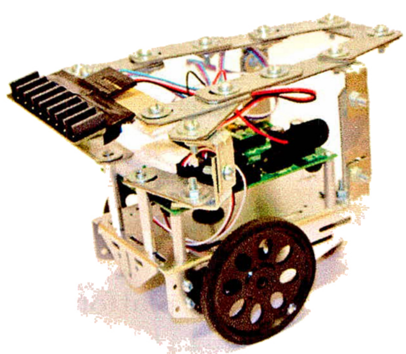
Figure 4.11: Transmitter Board Schematic

4.8.3 Follower Robots

First of all, these robots were equipped with mechanical structures similar to the ones of the leader. But as the followers are the carriers they needed a more robust structure with which the weight will be handled. It goes from the front part, through both sides and it stops at the back. All the structure has the same height and its symmetrical so the object's weight will be evenly distributed among it. The way the followers will pursue the leader is with the Line Followers. Usually they are located at the lower front part, but now they were put at the upper front part. This way the Robot will follow the line's path wherever it goes, in case it does not see any line it will stay still until there is a movement.



(a) Follower 1



(b) Follower 2

Figure 4.12: Follower Robots

Electronic Schematic

This schematic is the most simple of them all. As in the Leader Robot, the Robots' wheels are not included in this schematic but that does not mean they are not wired or used. The Line Follower Sensor has a pin for each of the IR detector. Those outputs are sent to pins 0 to 7, and depending the IR LEDs that are activated the Robot performs a correction action that can be moving frontwards, or adjusting right or left. (Figure 4.13)

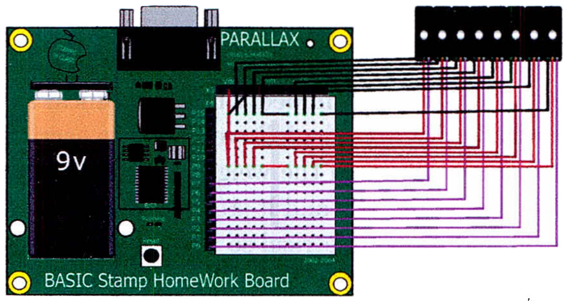


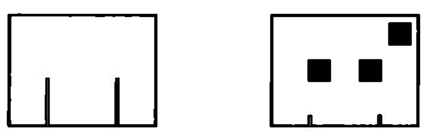
Figure 4.13: Follower Robot Schematic

Mechanic Design

The structure distributes the handled weight all over the Robot's chassis. The whole structure links all the toeholds just for stability purposes. This guarantees that if any joint or screw goes out of place, the robot will still be capable of performing its task. Its main drawback is the sensor location, as it has continuous contact with the platform's surface it might get damaged and malfunction at a certain point.

4.8.4 Transport platform

A platform was needed so the robots could handle the object. At the bottom of it, two lines are placed where the followers are meant to be. They go from the middle to the back of the platform so if they get delayed or one of them moves faster than the other one, they still can follow the leader. At the top of it some Velcro straps were placed where the power circuit, the battery and the Transmitter Board will be located.



(a) Lower view (b) Upper view

Figure 4.14: Transport platform

4.8.5 Multiple Robot System

The system integrates both Follower Robots, the Leader, the Transmitter Board, the Transport Platform and the supply system. The Leader goes at the front of the team, it pulls the Transport Platform. When the platform moves, it guides the way for the followers, which immediately start tracking the leader. The supply system and the Transmitter Board are located at the top, where they are fixed. The Figure 4.15 shows how the Multiple Robot System is mounted during execution time.

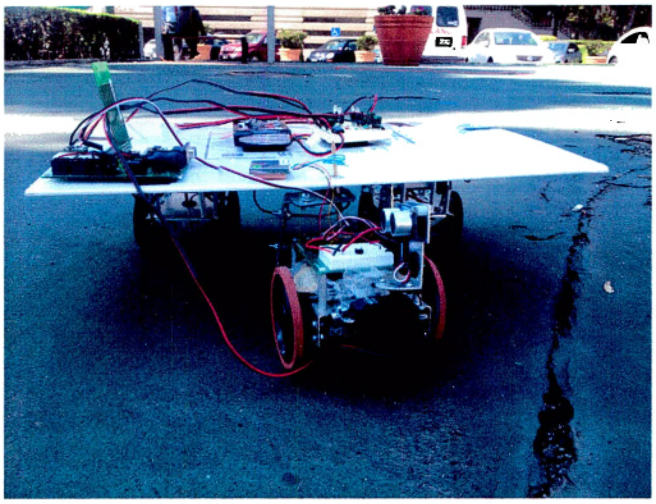


Figure 4.15: Multiple Robot System

As the implemented robot is a prototype it has several drawbacks but lots of virtues. Among the virtues are the usage of low cost robots working together to perform a task, All the employed sensors and actuators are not that expensive and can be found with ease. All the components are user-friendly and they can be easily used, nonetheless it is important to know how they work so all of their capabilities can be exploited. An object (not quite heavy) can be moved between two points, for now just a uniformly distributed weight one. The communications and all of the hardware were carefully selected so they could be robust and quite liable while facing several situations. Durability is a main requirement, robots must be able of working for a long time during different conditions. Almost all devices wear out with usage but the employed ones demonstrated to last long enough while doing lots of tests. There are three main drawbacks, the firs one is the uncertainty that the sensors and actuators have, for many applications high precision data is required, but the employed hardware does not provide it continuously, this is, motion is not always performed as desired, and sensors' measurements may have a large error.

Chapter 5

Software: Design and Implementation

Throughout this chapter the developed software programs will be listed and explained. It will be divided in two main sections: the Robot Programs and the Planner Programs. In both of them it will be explained how data is acquired or sent, communication protocols, movement commands computation and execution, as well as localization, mapping and obstacle avoidance. The computer used for running the Planner Programs is a DELL XPS L502X with an Intel Core I7 @2.20GHz, 8GB RAM and its OS is Windows 7 Home Premium x64. The most important work of this Thesis was implemented here that is why this chapter has a great importance. Actual code is not included here but pseudo-code is. This is for making it comprehensible to almost anyone, as well as if this work is used as a reference it can be easier to understand the implemented algorithms.

The algorithm can be represented as a flowchart. It represents the connections and links that exist between each of the hardware and software elements in the System. All the computing load is performed in the Planner which retrieves information from the Transmitter. It can access all the sensors and elements in the System in a direct or indirect way. The sensors can be accessed by all the elements that have a superior hierarchy, this is, the Planner cannot access directly any element contained in the system, but it can ask for the data. The lower level elements of the System are in charge of providing the Planner all the information it requires. The planner and the Leader robot are indirectly linked, another indirect relationship is the one between the Leader and the Follower Robots.

The diagram in 5.1 shows the overall implementation. The arrows show

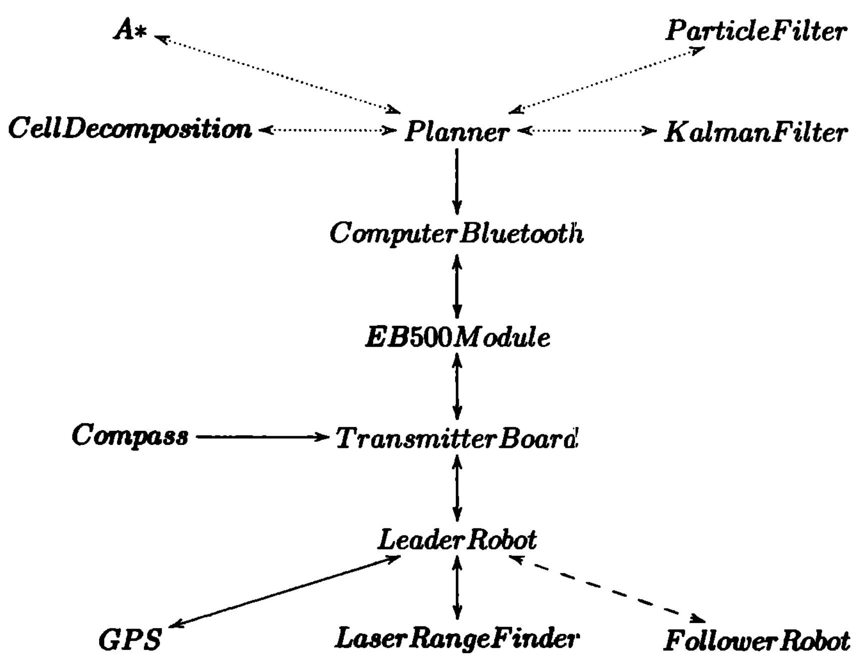


Figure 5.1: Overall Algorithm Block Diagram

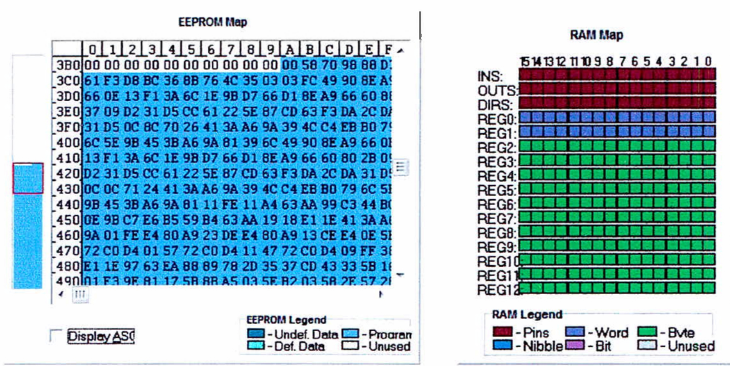
the way the information flows through the various elements of the software implementation. Algorithm interactions are marked with a dotted line, indirect interactions with a discontinuous line and finally direct interactions with a continuous line. Hierarchy is as shown in the Figure. The diagram shows a general interaction diagram, each module will be explained in the following pages.

5.1 Robot Programs

All the Robot programs run in a BASIC Stamp 2. It is a handy but limited microcontroller, specially when developing a large and complex implementation. As in any other microcontroller, it has internal auxiliary elements, it is economic, it can handle several peripherals such as sensors and actuators, uses several communication protocols, but most important of all, it can work in an standalone mode, but despite of those advantages, it has a limited program memory and RAM, that is why the code it executes shall be very efficient. As mentioned in the previous chapter, the BS2 has 2 KB EEPROM and 32 bytes RAM, which are quite limited resources for developing a complex program. Knowing that, the BS2 will only be in charge of acquiring data from the sensors and executing movements, no processing is performed in it. Firstly because most of the needed instructions are quite complex for such a microcontroller and valuable information could be lost, lastly because such instructions consume lots of program memory.

5.1.1 Leader Robot

Figure 4.8 shows the actual implementation of the Leader Robot. This Robot is in charge of moving, the GPS and the LRF, but as it has no wireless transmitting capabilities, it must sent the data it has just retrieved to the Transmitter Board. The Leader shows the way to the followers and it plays the Master role, but as it has no decision-making capabilities, it must be a Slave of the Planner. It waits for an indication coming from the Transmitter Board to execute an action. Meanwhile, it waits for an instruction. The general pseudo-code of the Leader is presented in Algorithm 5.



(a) EEPROM Map (b) RAM Map

Figure 5.2: Leader Robot Memory Maps

This code might look quite simple but the amount of memory it uses is significant (Figure 5.2). RAM is used completely by pin definitions, Byte and Word variables used in it. Most of it is consumed while reading the GPS. The EEPROM is occupied in a 53% . No complex functions are used in the code. Mainly just basic instructions like serial transmission/reception, PWM generation and a switch structure were used. Nonetheless a great amount of program memory is occupied. The data transmission between the

Leader and the Transmission is done by the Serial protocol. It is configured to transmit/receive 8 bits, no parity, no end bit and a BaudRate of 9600.

The LRF samples are took by using the laser device as well as a Standard Servo Motor. the $[0 - \pi]$ interval is divided in 44 parts. So when a sample is taken, the servo motor rotates $\frac{\pi}{44}$ radians until the whole environment is covered. Each of them represents ~ 0.07 radians. With that number of samples, a good resolution is reach so if there is a near obstacle it can be perfectly mapped. In case that the Kalman filter is used, the samples are liable for being used in the algorithm. For transmitting the LRF samples, each time one is took, it is sent to the Transmitter Board. This is because it is not affordable to save a set of samples because of memory constraints. Something similar happens when reading the GPS. As seen in (4.1) the output message sent by the device is too long. Some segments must be selected among the whole String. The interest characters are the Latitude, Longitude, and the N/S, E/W indicators. Combined all of them, a set of 18 characters are sent to the transmitter.

The robot uses the non-holonomic model in Figure 3.1. For moving, it just waits the indication. In the case of going front, it moves approximately 55 cm in the same direction it is heading. When going right or left, it executes a $\pm$ 90°turn while moving approximately 60 cm. As it can be expected, movements and sensor readings have a certain degree of uncertainty, the particle filter deals with it.

5.1.2 Transmitter Board

The board can be located in Figure 4.10. The Board mainly plays the role of a semaphore, but it also is in charge off obtaining data from the compass. When started, it waits for a Bluetooth connection coming from the Planner. After it has successfully connected, it waits for an action. The selected action can be: measure the environment, read the GPS or the Compass, or to perform a movement. As it has no direct access to the GPS or the LRF it must request for the data to the Leader. The communication protocol is consistent with the one that was explained just before. The Board and the Planner communicate with each other via Bluetooth, the serial port configuration goes as follows: the BaudRate is of 9600, 8 Data Bits, no parity, and a "CR/LF" terminator. The BS2 has no I^2C communication ports, so they had to be implemented by software using General Purpose I/O Pins. Parallax provides this software implementation.

In this case the RAM was used in an 85% , essentially it was employed in pin definitions, Boolean, Word and Byte variables. The EEPROM was used

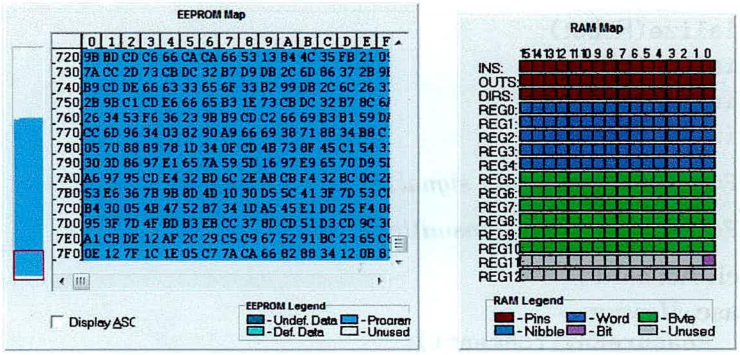
```
Initialize(Robot);
Initialize(LRF);
Initialize(GPS);
Initialize(Serial);
WaitFor(Synchronization signal);
WaitFor(Action from Transmitter Board)
switch Action do
  | case Measure
  |   | MeasureEnvironment();
  |   | WaitFor(Samples);
  |   | Serial.Send(Samples);
  | case Location
  |   | Read(GPS);
  |   | WaitFor(Location);
  |   | Serial.Send(Location);
  | case Front
  |   | Move(Front)
  | case Right
  |   | Turn(Right)
  | case Left
  |   | Turn(Left)
  | otherwise
  |   | Do nothing
  | end
end
```

Algorithm 5: Leader Robot Program

in a 70% mainly because of the software implementation of I^2C. The code is quite similar to the one used in the Leader. The main activity performed by the Board is the continuous data transmission. The most important part of it is the coordination of command/data reception/transmission between the three entities: Leader, Board and Planner.

$$\begin{tikzcd} & TransmitterBoard \arrow[dl, <->] \arrow[dr, <->] & \\ LeaderRobot \arrow[rr, <->, dashed] & & Planner \end{tikzcd}$$

As the diagram shows, there is a direct connection between the Leader



(a) EEPROM Map (b) RAM Map

Figure 5.3: Transmitter Board Memory Maps

and the Board, the Board and the Planner, but an indirect one between the Leader and the Planner. This way the Planner can get data from the Leader's sensors. The I^2C hard-coded interface is necessary because the Compass Module 3-Axis HMC5883L communicates using this protocol. It emulated the behaviour of this type of port by shifting registers and outputting them in a pin, as well as forcing SDA and SCL to a logical '1' and '0' depending on the desired action. I^2C may not be the fastest communication protocol, but it allows to control several devices using only two wires, the Serial Data Line (SDA) and the Serial Clock (SCL). However it has several disadvantages such as time constraints and current consumption that will not be discussed now.

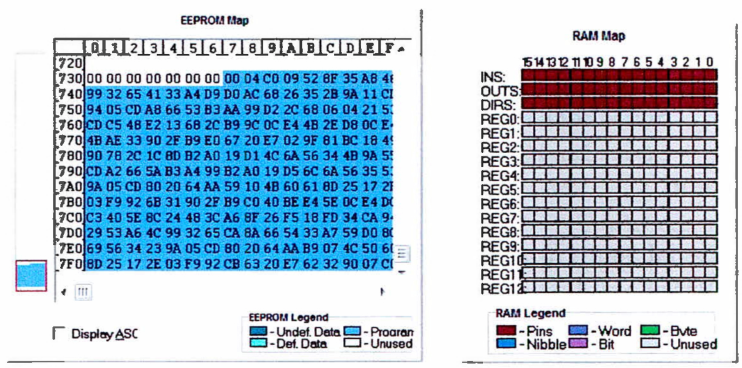
The data retrieved by the Compass comes in the format $X = 100, Y = 200, Z = 300$, so the individual values are sent to the Planner to be processed and the current orientation can be calculated. Originally, this Board was not supposed to be used. The leader had to handle all the sensors and the BT module by itself, but as memory consumption outranged the BS2 capacity, the Board came as a solution. An advantage gained by using it is the distribution of sensor readings and communication management from the software point of view. From the hardware point of view, due to the dimensions of the LRF and the BT module, while scanning the environment, sometimes both devices collided causing the system's malfunction. By being in the Transport platform, the BT module has an free way to the Planner; the compass returns the global orientation of the System, not only the Leader's one. And finally, it made more complex and attractive the problem itself.

```
Initialize(Bluetooth);
Initialize(Compass);
Initialize(Serial);
WaitFor(Connection with Planner);
Serial.Send(Synchronization signal to the Leader)
WaitFor(Action from the Planner)
switch Action do
    case Measure
        Serial.Send(Request to the Leader);
        WaitFor(Acknowledge);
        foreach sample do
            Serial.Receive(Sample from the Leader);
            Serial.Send(Sample to the Planner);
        end
    case Location
        Serial.Send(Request to the Leader);
        WaitFor(Acknowledge);
        Serial.Receive(Location from the Leader);
        Serial.Send(Location to the Planner);
    case Orientation
        Read(Compass);
        Serial.Send(X, Y and Z values to the Planner);
    case Front
        Serial.Send(Command to the Leader);
        WaitFor(Acknowledge);
    case Right
        Serial.Send(Command to the Leader);
        WaitFor(Acknowledge);
    case Left
        Serial.Send(Command to the Leader);
        WaitFor(Acknowledge);
    otherwise
        Do nothing
    end
end
```

Algorithm 6: Transmitter Board Program

5.1.3 Follower Robots

Follower robots are just equipped with the IR Line Follower as seen in Figure 4.12. Their main and only task is to pursue the Transport platform wherever it goes. From the hardware employed for tracking paths, no big algorithm is needed to implement this kind of Slave behaviour. An ordinary Line Follower code can be a good reference of how the Followers work. Their memory (RAM and EEPROM) usage is minimum, that way the will only be focused on always tracking any movement at all.



(a) EEPROM Map (b) RAM Map

Figure 5.4: Follower Robots Memory Maps

The only big modification that had to be made was the sensor's threshold of when to detect that a line is present. This is, with different illumination sources the sensor can behave differently that is why it had to be calibrated so it could work properly in several locations with diverse luminous intensity.

5.2 Planner Programs

The Planner runs in a remote computer. Its task is to compute the employed algorithms: A*, Particle Filter, Kalman Filter, and Cell Decomposition, It requires additional information from the environment, such as the robot's location, orientation and obstacle presence. That information comes from the Leader Robot's and Transmitter Board's sensors. After the algorithm's output is computed, it must translate it to a primitive command for the Robot to execute. All the planner runs in MATLAB 2012a. The communication protocol is Bluetooth with the BaudRate of 9600, 8 Data Bits, no parity, and a "CR/LF" terminator. The system's intelligence is contained here, most of the calculations are done in quite large loops, which are not recommended to implement in an autonomous system with small memory. Some other calculations require a more powerful processor, examples can be an arctangent, float point operations, array manipulation, linear algebra or random number generation.

5.2.1 Data Acquisition

All the algorithms used in this Thesis require environment information. The Kalman filter needs to sample to know if there is any moving object, so it can estimate where it will be at the time $t+1$. A* requires to know where the robot is located and if there are obstacles in $\mathcal{W}$ so they can be evaded. The Particle Filter requires a location and orientation to estimate the real robot's position.

GPS Data

The PMB-648 GPS returns the data it retrieves in a format as shown in Equation 4.1. Not all the information is useful, so only a part of it is retrieved for usage. From the entire message, only 19 characters are received from the Leader. The string looks like: `37232475N121583416W`. The first part contains the latitude information. The first two characters are the hours, the next two the minutes, and the last four the seconds, the last letter describes if the location is at the North or the South. Similarly, the latitude is expressed in the same way, but the hours are contained in three characters and the final letter represents East or West. This format is only useful for navigation and people with an expertise in it can easily interpret them. Usually they are read in the degree representation, such format is used in different types of applications such as Google Maps, Foursquare, and Google Places.

To transform the data from degrees to the decimal representation the following operation must be done:

$$\text{Decimal} = \begin{cases} Hours + \frac{Minutes}{60} + \frac{Seconds}{3600} & \text{If North or East} \\ 1 \times Hours + \frac{Minutes}{60} + \frac{Seconds}{3600} & \text{If South or West} \end{cases} \tag{5.1}$$

With data being transformed to this representation. it can be manipulated as if in a planar (x, y) coordinate system. Now that the concept has been mentioned and taking advantage that the GPS Location theme is being treated. The environment in which the Robots will work is located at

"Explanada CEDETEC". It is usually used for different type of events such as cultural, medical and sports. This place has the great advantage that its form is like two contiguous rectangles. They can be used as a semi-ideal $\mathcal{W}$. Each of them will be used separately, just to keep its *ideal* shape.

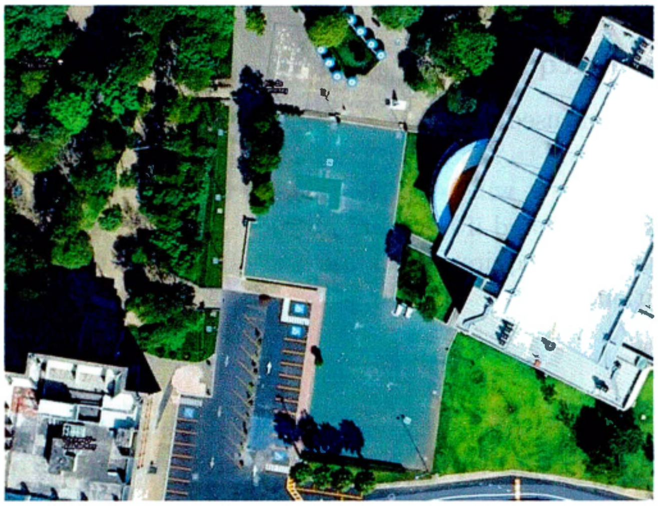


Figure 5.5: Real Workspace

Just for now, another theme will be treated but it has to be mentioned in this section. Landmarks are needed for implementing the Particle Filter. They are used as a reference for the robot to know where it is located. In environments with no regular shape, landmarks are set by the experience of the designer, but when it can be approximated to a regular one the designer must take advantage of it.

Each of the corners of the rectangles will be used as a landmark. The landmarks will be stored in an a 4×2 matrix. The order of the corners used as landmarks is: Upper right, Lower right, Lower left and Upper left.

They will look this way for the upper rectangle:

$$L_1 = \begin{bmatrix} 19.283587 & -99.135622 \\ 19.283265 & -99.135668 \\ 19.283301 & -99.135947 \\ 19.283622 & -99.135788 \end{bmatrix} \tag{5.2}$$

And for the lower one:

$$L_2 = \begin{bmatrix} 19.283201 & -99.135535 \\ 19.282911 & -99.135567 \\ 19.282953 & -99.135838 \\ 19.283276 & -99.135788 \end{bmatrix} \quad (5.3)$$

LRF Data

The LRF returns data in a $[0-99]$cm interval. As mentioned before, it samples the environment each $\frac{\pi}{44}$ radians. If the sampling takes place in an empty $\mathcal{W}$ then all the samples will have a magnitude of 99 in all the angles. If this data is plotted it resembles the form of a semicircle.

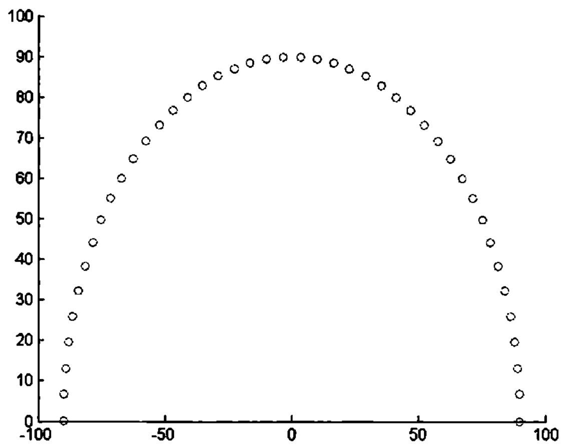


Figure 5.6: Laser Range Finder Data

Whenever an obstacle is located, a different value rather than 99 is returned. That way it can be mapped so A* can use it so it can be avoided. The retrieved data is also employed while using the Kalman Filter, its function will be explained later.

Compass Data

The Compass Module 3-Axis HMC5883L returns X, Y and Z values corresponding to Earth's magnetic fields. These data comes in a three dimensional Cartesian form. Assuming that our planet has a spherical shape, the information can be transformed into a spherical coordinate system. This is, in a Cartesian system a point is specified by its coordinates in each plane while in a spherical system by r(radius), θ(inclination) and φ (Azimuth). They can be calculated by:

$$
\begin{aligned}
r &= \sqrt{x^2 + y^2 + z^2} \\
< \quad \theta &= \arctan\left(\frac{z}{r}\right) \\
\varphi &= \arctan\left(\frac{y}{x}\right)
\end{aligned}
\tag{5.4}
$$

All the information can be handy at certain point, but the most important component of it is the Azimuth. It tell us the inclination it has from the north. After calculating the Robot's current orientation it must be manipulated so it can be used in a more natural way. In the planar coordinate system used as $\mathcal{W}$ the 0^o marks is located exactly at the East, so 90^o had to be summed to the original computed orientation. The obtained angle by itself gives information of where the Robot is facing, but if used with other information it can provide further knowledge of where certain places are located, specifically, the landmarks' location.

That information is used by the Particle Filter. To compute the landmarks' angles, latitude and longitude must be obtained from the GPS. After obtaining it the following operations must be done:

$$
\begin{aligned}
\Delta\text{lat}_i &= L_{i,lat} - \text{lat}_{GPS} \\
\Delta\text{lon}_i &= L_{i,lon} - \text{lon}_{GPS} \\
\theta_i &= \arctan\left(\frac{\Delta\text{lon}_i}{\Delta\text{lat}_i}\right)
\end{aligned}
\tag{5.5}
$$

In the previous lines, i is the i-th landmark used in $\mathcal{W}$. The differences calculated for the latitude and longitude are used for knowing in which quadrant is the landmark located. Lets remember that when using the arctan trigonometrical function the sign of that difference determines the quadrant it will be mapped. Conventional arctan just work in the first and second quadrant, that is why the $\arctan 2$ function was developed, with it, the angle can be spotted correctly in the corresponding quadrant. In the Particle Filter, all the angles are in radians, but for explaining what was done, degrees will be used.

Lets suppose that an angle of 168° was calculated using the Azimuth formula. The GPS returned a latitude of 19.283529 and a longitude of -99.135783. The first row of L_1 will be used:

$$
\begin{aligned}
\Delta \mathrm{lat}_1 &= 19.283587 - 19.283529 = 0.000058 \\
\Delta \mathrm{lon}_1 &= -99.135622 - (-99.135783) = 0.000160 \\
\theta_1 &= \arctan\left(\frac{0.000160}{0.000058}\right) = 70
\end{aligned}
\tag{5.6}
$$

This way the angles for all the landmarks are computed. If the angles are plotted, the can be seen as:

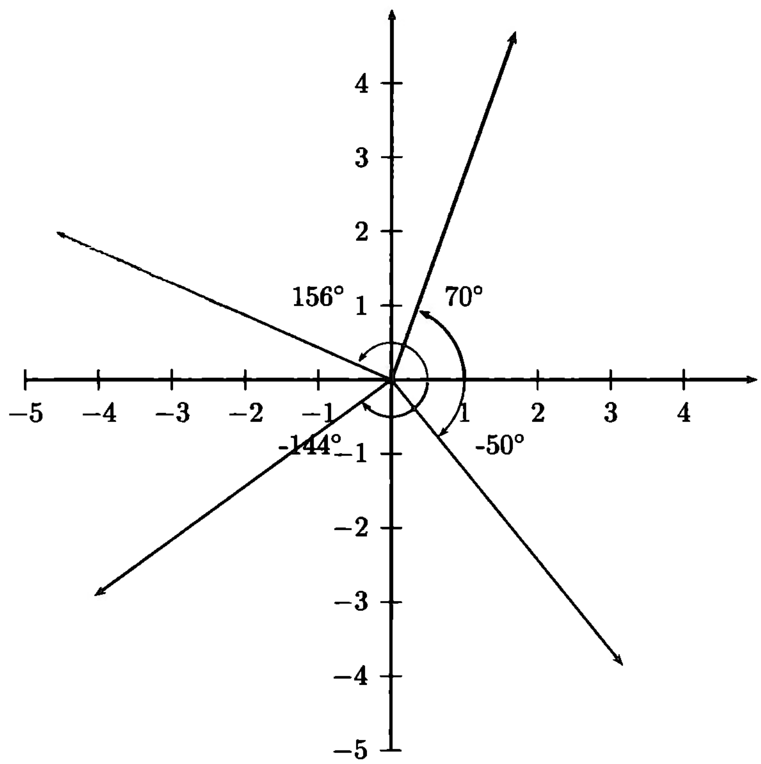


Figure 5.7: Measured angles from the landmarks

The next step is to subtract the original angle, the data is in the [0-360] range, so what the Particle receives looks like:

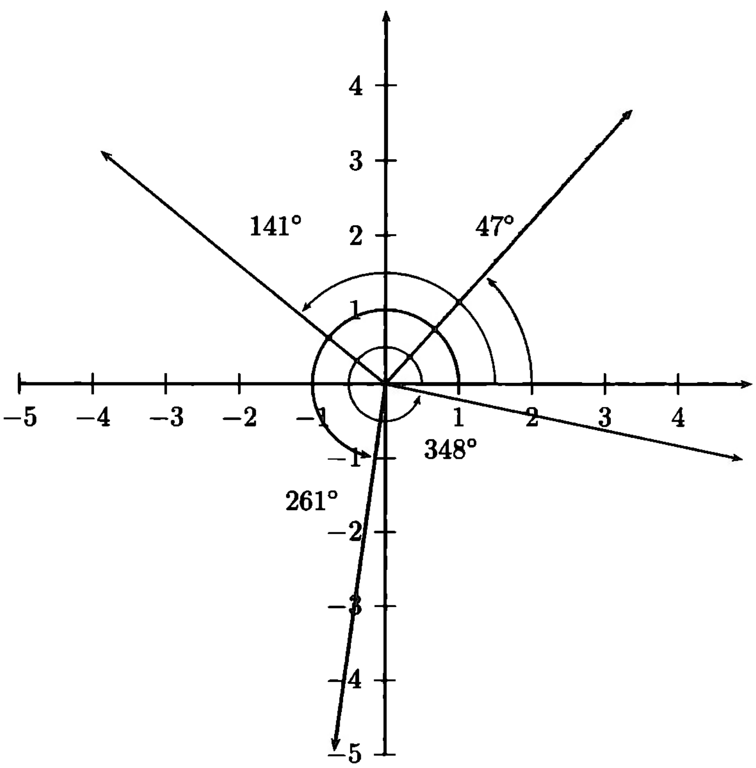


Figure 5.8: Computed Angles

5.2.2 Cell Decomposition implementation

The Approximate Cell Decomposition Algorithm will be used to represent the location shown in 5.5. The algorithm that was explained in Chapter 3 subdivided M-cells until a path was found from $\mathcal{P}_i$ to $\mathcal{P}_f$. In this case, as there is no clue if there are obstacles in $\mathcal{W}$, the cell size must be set to a very small value ϵ, this way it can be semi-warranted to find a path. A relationship must be obtained to have an equivalence between the cell size and a decimal degree, that is why using experience from Mobile Device programming, the GPS decimal degree coordinates will be multiplied by 10^6. So for example from L_1 the first row will go from [19.283587 − 99.135622] to [19283587 − 99135622].

Now, this data is too ambiguous for representing it in a $m \times n$ matrix, specially because in almost all programming languages memory usage is an important criteria for its efficiency. Because of that, an adaptation of the decimal degree representation must be done. First of all, the minimum latitude and longitude are computed from the landmarks L_1 or L_2, depending which $\mathcal{W}$ is currently used. After that, min_{lat} and min_{lon} are subtracted from L_i, so the new landmarks are:

$$L_1 = \begin{bmatrix} 326 & 323 \\ 280 & 1 \\ 1 & 37 \\ 52 & 358 \end{bmatrix} \tag{5.7}$$

The set of values can be seen as the corners of a rectangle so assuming Explanada Cedetec is a perfect rectangle, this is a valid and functional representation of $\mathcal{W}$. Everything outside the rectangle's borders will be considered as an obstacle.

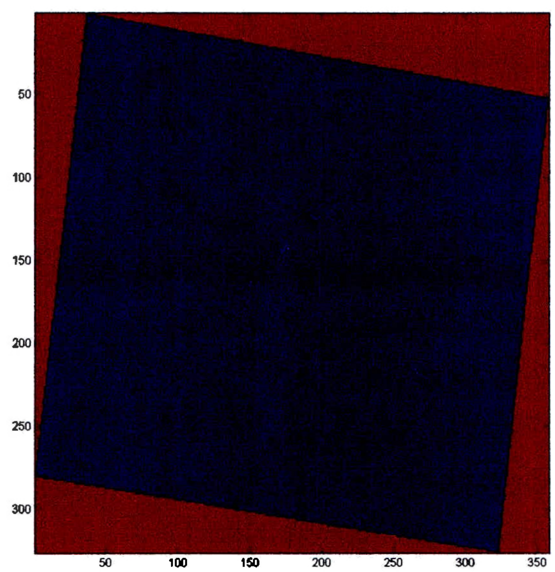


Figure 5.9: Cell Decomposition Implementation

5.2.3 A* implementation

Having explained how $\mathcal{W}$ is represented, A* will now get into action. Let us remember that an Informed Search Algorithm requires knowledge that will aid it to find an optimal solution to certain problem. To find an optimal path we want to get from P_i to P_f in the minimum number of steps possible, so our heuristic shall penalize the number of steps taken. So, it will find a way to reach the goal quicker. It must be said that the robot's environment is defined if Manhattan Distance, and it can only move in a vertical or horizontal way, no direct diagonals are allowed. With that said, the implemented heuristic shall conduce all movements to goal. While the robot is far from the goal it will greedily try to find a way to reach it. A paraboloid was used to generate it. Its mathematical expression is:

$$z = \frac{(x - g_x)^2}{a} + \frac{(y - g_y)^2}{b} \tag{5.8}$$

Where $g_{x,y}$ is the goal's coordinate, a and b are set to 1. This way the center of the paraboloid will be located at the goal, so it will always converge if a path exists, else it will return *failure*. In a grid of 50×50, with the goal located at $(25, 25)$ the paraboloid will look as follows:

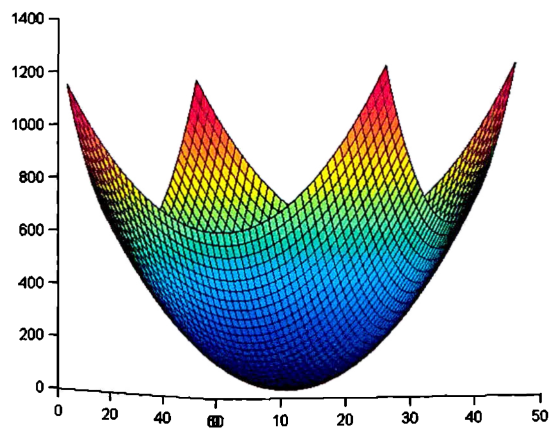


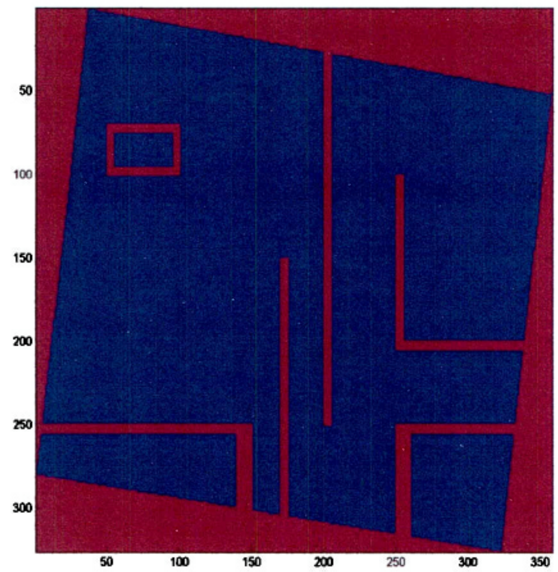
Figure 5.10: Employed Heuristic

Now, as mentioned in the previous section, $\mathcal{W}$ is defined in a $m \times n$ cell matrix, $\mathcal{W}_{free}$ has a value of '0', in the other case if an obstacle is located at that point $\mathcal{OW}$ contains a '1'. In an hypothetical environment like the one showed in Figure 5.11 (a) the blue cells are empty, the other ones are obstacles. A* will only look for an optimal path through $\mathcal{W}$. After it has performed the search, it will return primitive instructions such as the following:

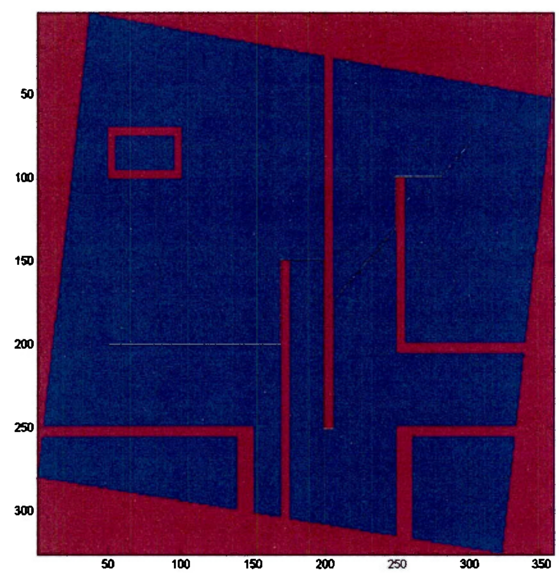
$$\rightarrow\rightarrow\rightarrow\rightarrow\uparrow\rightarrow\uparrow\rightarrow\uparrow\leftarrow\leftarrow\leftarrow\ \star \tag{5.9}$$

Each arrow describes the direction of where there robot shall move. As it can be inferred, the actions are up ($\uparrow$), down ($\downarrow$), right ($\rightarrow$) and left ($\leftarrow$).

When goal is reached, a $\star$ is returned.



(a) $\mathcal{W}$ with obstacles



(b) Computed Path

Figure 5.11: A* implementation

A run of the algorithm is shown in Figure 5.11 (b), the paraboloid's center (heuristic) is located at the goal, the computed path is shown as a white line.

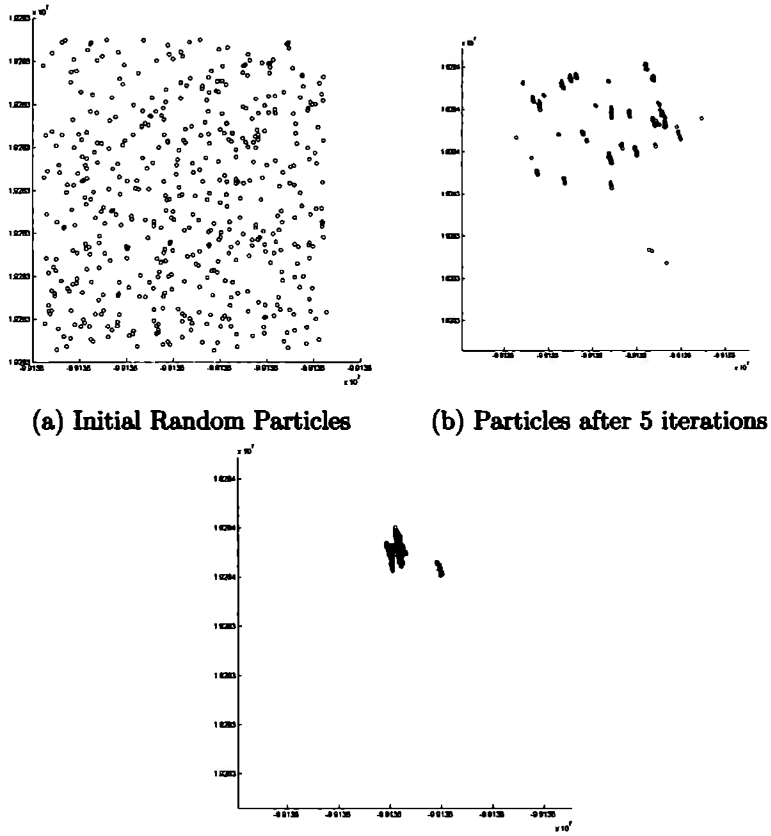
5.2.4 Particle Filter implementation

The Particle Filter shows the general scheme described in Algorithm 4 but it suffers some little modifications so it can be coupled to the current problem. First of all some noise constants must be defined: bearing, steering and distance. Robot motion usually is not as exact as one could wish, that is why these parameters shall be set according to observed uncertainty. PF deals with Gaussian noise in sensor readings and motion, so the noise constants can be seen as σ^2 for a reading μ. Next, the number of particles shall be specified. This number shall be proportional to the difference of the maximum and minimum data in which the algorithm will run. High precision is reached by using a large number of particles, but the disadvantage is the time consumption. For our case $N = 500$ particles worked nice enough. A random initial population of N particles is thrown to the environment. They have a random location and orientation:

$$p_i = \begin{bmatrix} \text{random}\,(\max(L_{j,lat}) - \min(L_{j,lat})) \\ \text{random}\,(\max(L_{j,lon}) - \min(L_{j,lon})) \\ \text{random}\,(2\pi)) \end{bmatrix}^T \tag{5.10}$$

The big next step is to perform a movement according to the robot's motion. Each of the particles moves and rotates from their own location. At this point, distance and steering noise are critical values because they define how much does each particle really moved. Then, sensor data is acquired from the GPS and the Compass. Data is returned in the format that has been explained previously. The measurement error stage takes place. Here, the error is calculating by obtaining the difference between the physical measurement and all the particles computed measurement. Finally a resampling takes place, only particles with a very small error survive, the ones that have a big error are relocated near the real robot location following a Gaussian distribution.

A brief example will be given. Using Explanada Cedetec's upper rectangle as $\mathcal{W}$, the robot will locate itself. It will perform no movements at all, neither planar nor rotational. The robot's real location is [19.283529, −99.135783] with an orientation of 168.19°. The previously explained steps are taken. After 5 iterations the particle population has dramatically reduced into a radius surrounding the real location: [19.283533 – 99.135780] with an orientation of 167.17°. As five more iterations are done, the Gaussian estimate of the robot's location is narrowly closed to the real one, as an estimate of [19.283527 – 99.135791] and 167.47° is computed.



(a) Initial Random Particles (b) Particles after 5 iterations

(c) Particles after 10 iterations

Figure 5.12: Particle Filter implementation

5.2.5 Kalman Filter implementation

Kalman Filter is tightly related to LRF samples. At first, the environment is completely sampled. If there is an object detected, there will be 5 more environment samples taken. The centroid of the detected object will be computed so just one x and y value per sample is used. After that, the data is set into the KF to esteem where will the moving object will be located at

$t + 1$. The Employed matrices look as follows:

$$\mu_{t-1} = \begin{bmatrix} -4.6236 \\ 13.7416 \\ 0 \\ 0 \end{bmatrix} \tag{5.11}$$

$$\Sigma_{t-1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 1000 \end{bmatrix} \tag{5.12}$$

$$u_t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.13}$$

$$A_t = \begin{bmatrix} 1 & 0 & 2.7 & 0 \\ 0 & 1 & 0 & 2.7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.14}$$

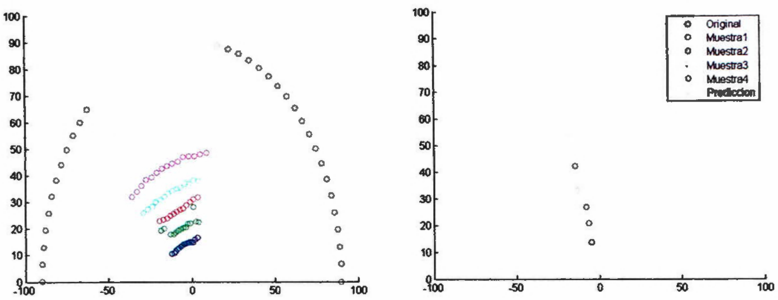
$$R_t = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{5.15}$$

$$H_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.16}$$

For example, the Robot will track a moving object that is moving away from it. It takes an initial sample that will be used as μ. The samples z_t are taken each 2.7 seconds, which is the time the LRF scans completely the environment. After sampling 5 times the environment, the result is really impressive, it computes the estimated output with great precision, such as Figure 5.13 (b).

5.3 Global Algorithm

The whole algorithm implementation required a coupling strategy so all of them could employ the same (or very similar) data. Through the previous chapter all algorithms outputs were shown individually, now the final results are presented. The final implementation will be explained in detail.



(a) LRF's Environment Samples (b) Kalman Filter Estimate

Figure 5.13: Kalman Filter implementation

The program starts by asking which $\mathcal{W}$ of Explanada Cedetec will be used (Figure 5.5), if the upper or the lower one, each one of them has different landmarks coordinates. Whichever was chosen, it will be conditioned to be used by all the algorithms, so it is multiplied by 10^6 and rounded to the nearest integer. With those digits the resolution is quite good enough. The step is to calculate the minimum and maximum values of the latitude and longitude that define each $\mathcal{W}$.

The next big step is to define all of the constants that will be used, these are:

1. Particle Filter - Particle number definition (500), the robot's length (1), bearing noise (1), steering noise (0.1) and distance noise (1). The measurements is initialized to an empty array, the initial motion is set to stay still.

2. Kalman Filter - Sampling time (2.7 seconds), μ_{t-1}, Σ_{t-1}, u_t, z_t, A_t, C_t and Q_t. Also the angles in which the LRF will point are set.

3. Cell Decomposition - Set the corners to a parametrized $\mathcal{W}$ representation, this is, it will not have the conventional GPS decimal degree representation, but a smaller number not bigger than 350. Everything outside $\mathcal{W}$ will be set as an obstacle. (Figure 5.9)

4. A* Search - The goal location is defined if and only if it is a valid one inside $\mathcal{W}$.

5. Serial Port - 9600 Baudrate, 8 Data bits, no parity, CR/LF terminator and a time out of 1.

The initial set of particles is thrown an plotted (Figure 5.12 (a)). A while loop starts just at this point, it will only stop if the robot is located at the goal. Particles move according to their location and orientation. After it, the GPS and Compass are read for obtaining the errors between the expected and the actual measurement. Particles with small error will survive, the rest of them will be re-sampled until they are located near the actual location Gaussian.

Now the Kalman Filter is used to map any obstacle present in the surroundings of the robot, in case it is a mobile object, its trajectory is estimated. The previously computed position is used as the initial location of the robot to start the A* search for an optimal path. After it, the movement is performed and the while loop repeats until the robot completes the task. Algorithm 7 shows the pseudo code of what has just been described.

```
input: Goal
Initialize(Constants);
Initialize(Bluetooth);
Create(Environment);
Create(Particles);
repeat
    ParticleFilter.Move(Particles);
    Read(GPS);
    Read(Compass);
    ParticleFilter.Compute(Angles);
    ParticleFilter.Compute(Error);
    ParticleFilter.Resample(Particles);
    ParticleFilter.Get(Location)
    if Location == Goal then
        return Success
    end
    KalmanFilter.Sample(Environment);
    if Moving Object then
        KalmanFilter.Track();
    else
        KalmanFilter.Map();
    end
    A*.Compute(Path);
    if Path exits then
        Move(Robot);
    else
        return Failure
    end
until Success or Failure ;
```

Algorithm 7: Final Program Implementation

Chapter 6

Results

In this chapter results will be presented throughout the most representative cases. All the problems that were faced are also mentioned here. Lots of images will be used so the understanding process can be done easily. Due to workspace availability, all the tests were done at Explanada Cedetec secondary area, but they could have been easily done in another location.

6.1 Implementation Runs

Four cases will be presented in this section. Each of them shows the algorithms functionality in various situations. They were carefully chosen so they could represent all the cases that the robot can face. It must be said that the robot executes the Particle Filter first of all so it can locate itself at $\mathcal{W}$ before any motion action is performed, this way collisions are avoided as well as creating more uncertainty. Once the robot's location is identified a Path is computed using A* and it is translated to a primitive action so the robot can perform it.

6.1.1 Robot's P_i is P_f

The first situation is one of the most simple of them all. The robot is already located at the goal, so when the robot initializes and determines its own location it will know that it is already located at the final point so it will return *Success* immediately. In Figure 6.1a it can be seen the initial robot location in a Google Maps View. The initial particle guess shows that all were thrown at random locations, Figure 6.1b. After 10 iterations (Figure 6.1c) there is no good guess of where the robot is located so the Particle

Filter must continue working to improve it. Figure 6.1d shows similar results as in 10 iterations, but there is a significant particle cluster formed. Ten iterations later (Figure 6.1e) a good approximate is computed, but if there is a considerable amount of noise while reading sensors this believe may have several peaks. Forty iterations returns a fairly good approximate that shows the robot's current location.

After the robot finishes the self-localization stage, it checks out if it is located at the goal. In this particular case that condition is true, so immediately it returns *Success*.

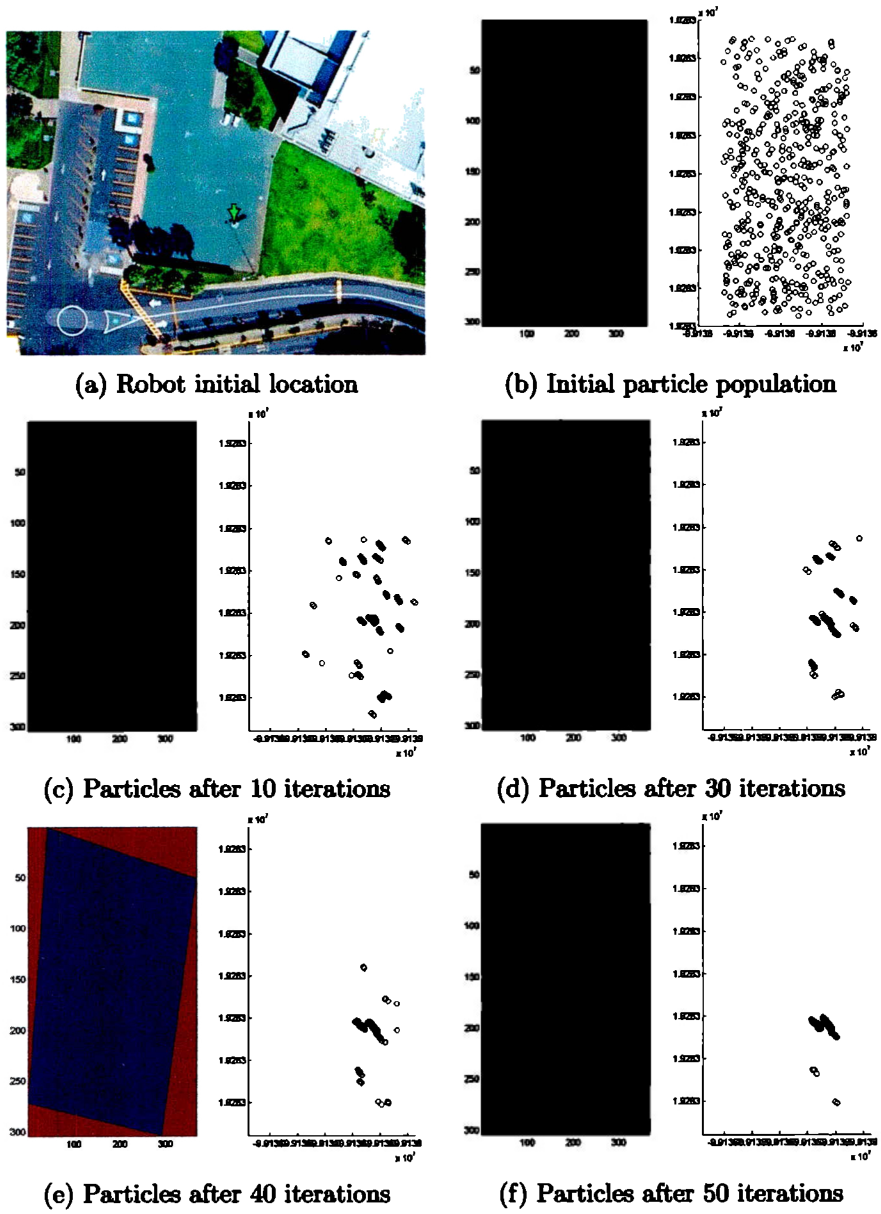
6.1.2 Robot's P_i is outside of the $\mathcal{W}$

This case is a particularity of the previous one. The Particle Filter finds the robot location given the sensor's data. In Figure 6.2a there are two pointing arrows. The leftmost one shows the robot's location, the one at the bottom right is the goal location. As the particles converge to the robot location, it can be seen that from Figure 6.2c to Figure 6.2f the particle's centroid is located outside the $\mathcal{W}$. So when the localization stage finishes, the A* search returns *Failure*, and immediately the program stops.

6.1.3 Path planning

Now, the previous examples have shown how the self-location stage works. The particle filter executes with a movement vector of $\begin{bmatrix}0 & 0\end{bmatrix}$. The first element represents the travelled distance, the second one the rotation angle. In other words, it stays still. Now that that the robot location has been found no more images from this stage will be shown except for the initial belief and the located robot in the 40th iteration. The next step consists in finding a path from the current location to the goal state. Each time a movement is performed all the particles move in the same direction as the robot does. The A* path is also refreshed so it fits the actual location and orientation. As movements go by, the path gets reduced and the particles approach the goal state.

The next three figures show some snapshots from an algorithm run. They illustrate how the robot moves. The left column shows the robot orientation at a Google Maps location while in the right one the estimated location the Particle Filter returns and the computed path that A* returns. The initial location (Figure 6.3a and ??) shows where the robot is at a time $t = 0$. As the robot performs some movements, it can be seen that it does not perform as exactly as planned due to several causes such as loose earth and



(a) Robot initial location

(b) Initial particle population

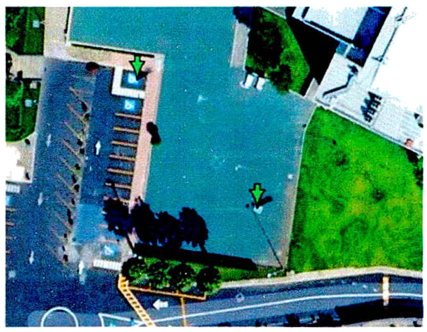
(c) Particles after 10 iterations

(d) Particles after 30 iterations

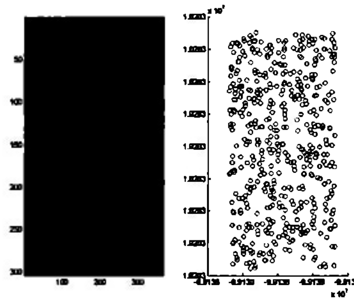
(e) Particles after 40 iterations

(f) Particles after 50 iterations

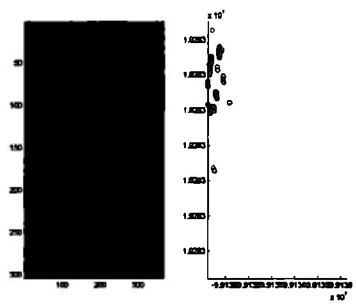
Figure 6.1: Static Robot Example



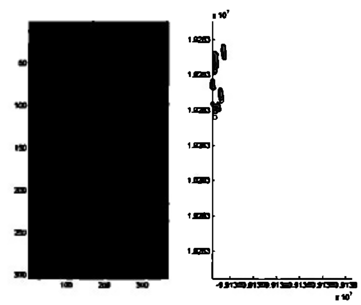
(a) Robot initial location



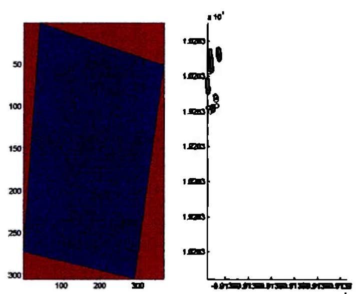
(b) Initial particle population



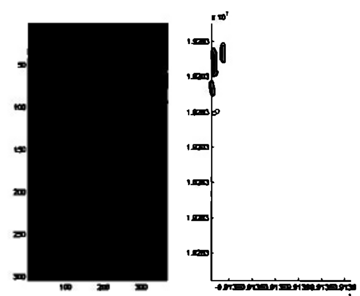
(c) Particles after 10 iterations



(d) Particles after 30 iterations



(e) Particles after 40 iterations



(f) Particles after 50 iterations

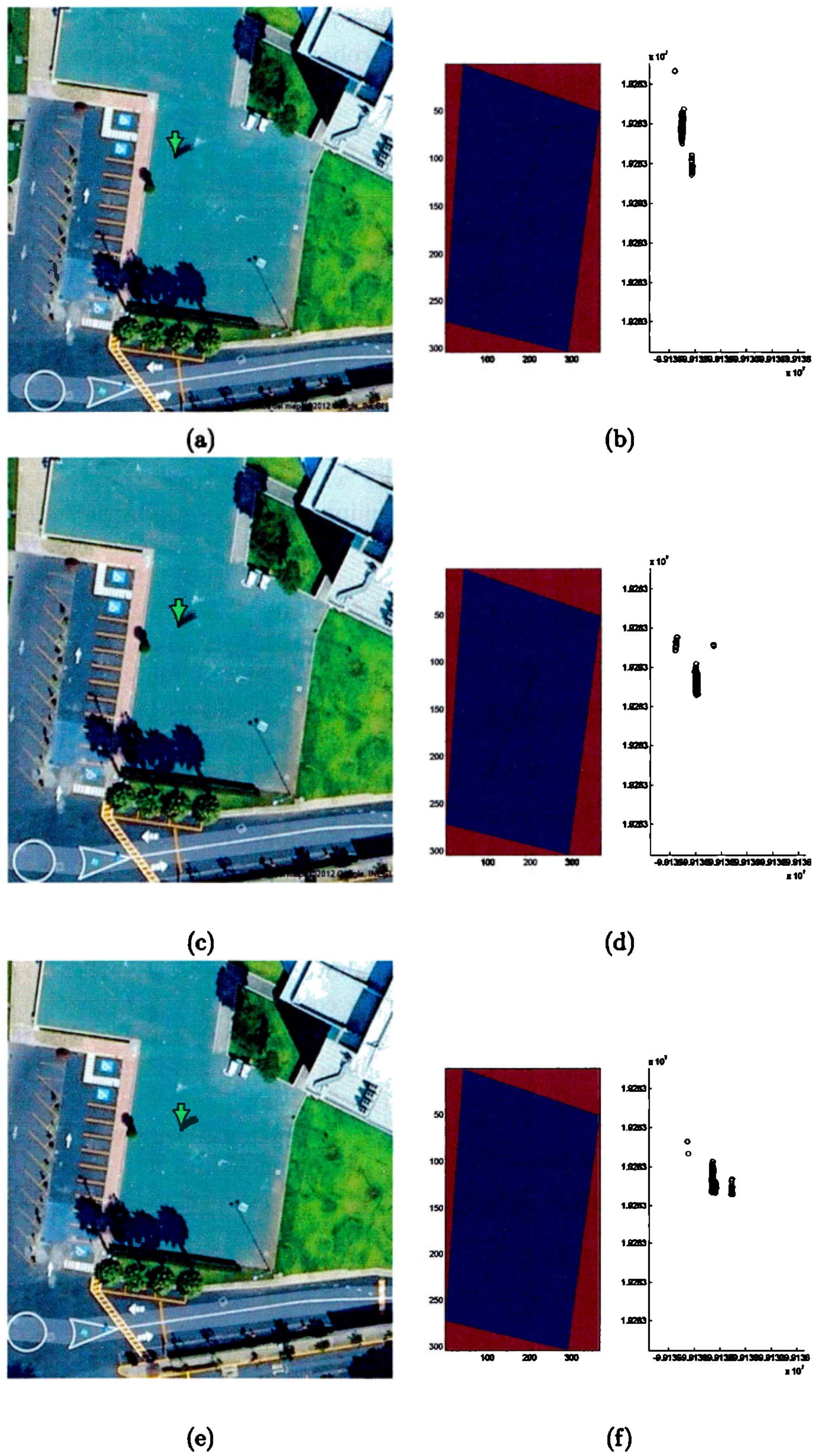
Figure 6.2: Out of bounds Robot Example

irregular ground. So each time the robot executes a movement the path is recalculated dynamically. When the robot reaches the goal location it returns success and the run is aborted.

6.2 Faced Problems

Throughout the development of the final implementation lots of problems, hardware and software, emerged. Some of them will be listed now. Hardware design was the first obstacle that was faced, some modifications had to be done to the Boe-Bot Robots so each one of them could perform its role in a better way. Several designs were done trying to give enough capabilities to the Follower Robots. Initially their task was to push the object in the direction the Leader indicated, but as there could be some movements that over exploited a Follower, that idea was replaced with the final one. As Figure 4.12 shows, the robots were equipped with a certain 'exoskeleton' that distributes the carried weight all over the structure. It suffered several modifications due to the resources that were available at the time. At the beginning. four metal tubes held a small platform. That design was thrown away because of the huge overall weight of the Robots. The structure design came from considering how could the weight be distributed evenly into the robot's chassis while having enough surface contact with the Transport Platform. Another problem with this robots was to figure out how will they follow the Leader. What people usually do is to "feel" where the Leader is going, but as it is quite too mainstream something different could be a big breakthrough. While watching how light-follower car-like robots worked, the idea came: there could be a form in which the robot can keep following a signal. In this case, following a light was not a good choice because in different environments, the light-source intensity is variable and it could not always work. So, if a platform was used above the robots, it shall help and indicate them where it is moving. The low cost solution was to place a line follower sensor over the Followers and a line under the Platform. It results easier to calibrate a line than a light follower. As under the Platform a shadow is projected, the illumination changes would not affect that much the Follower. That is why the Follower Robots are always waiting for a Leader's movement, wherever it goes they will follow him.

Sensor data gave lots of trouble. Each of them had its own degree of uncertainty. For example, the employed Compass is not as precise as one could wish. As it measures magnetic field's strength, any electronic device or electric object interfered with the sensor's readings. Most of the times it

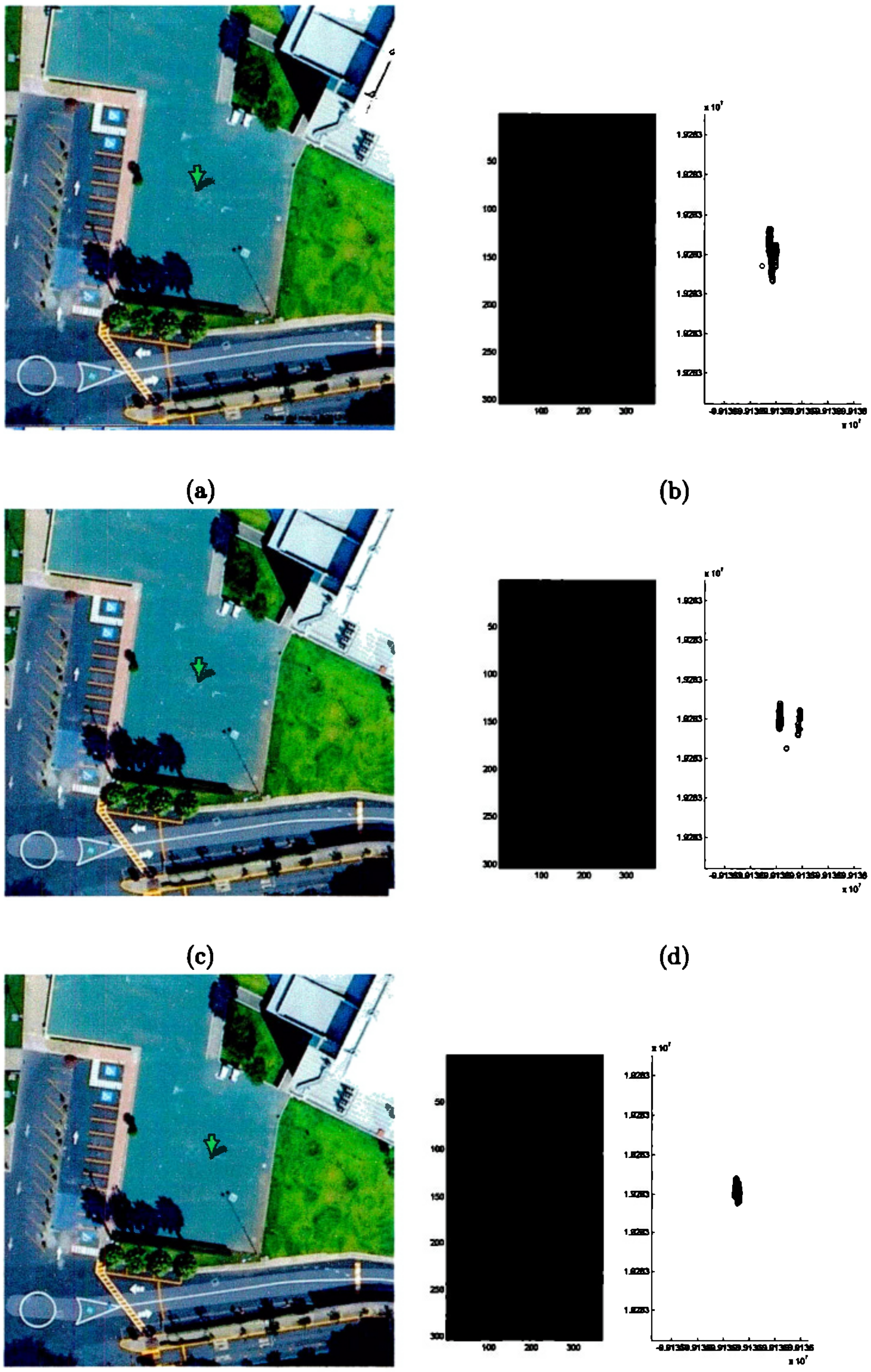


(a) (b)

(c) (d)

(e) (f)

Figure 6.3: Path Planning Example part 1

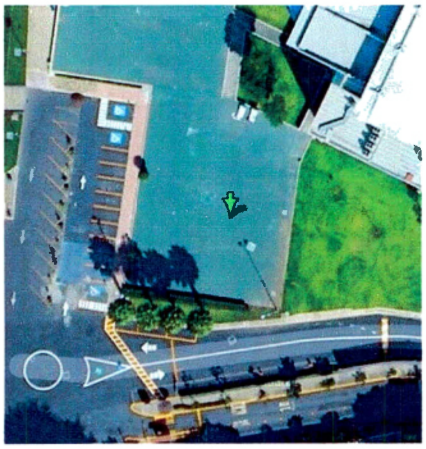


(a) (b)

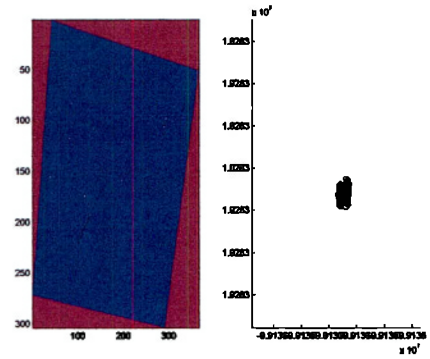
(c) (d)

(e) (f)

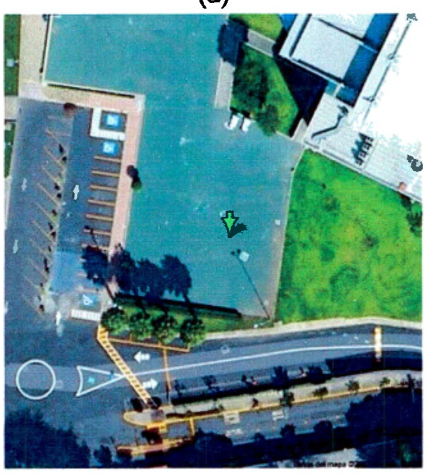
Figure 6.4: Path Planning Example part 2



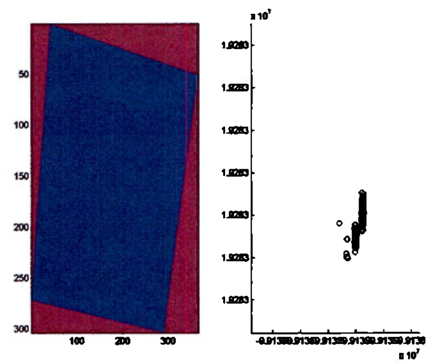
(a)



(b)



(c)



(d)

Figure 6.5: Path Planning Example part 3

gave a very large mistake from the current orientation to the measured one. Fortunately, the particle filter leads with this mistake, so it did not affect a lot the System's performance. The range sensor gave almost no error at all, so it represented no big deal. In the other hand, the GPS gave lots of trouble. First of all because it depends on how good its connection is to any of the satellites, During rainy or clouded days its measurements were not liable at all. It returned an error from 2 to 10 meters, which was not desirable. Another problem with it was that as it was embedded in the Leader Robot, during certain movements the high-torque servo motors demanded quite too much current from the power source which provoked that the GPS loosed connection with the satellites, increasing the measured error. The solution to this problem was to implement a local power source for the GPS so it could have a constant current feed. With this modification, the readings it returned were quite accurate.

The Leader had some common problems with the Followers. At first, it was going to pull the object. After the Slave Robots were given an Exoskeleton, it needed one too. It only was equipped with a small contact surface what enable it to carry part of the platform's weight while leading the way. During the first tests the Leader had many problems to pull the platform, the Servo Motors it had did not give it the necessary torque. They were changed for some more powerful motors, but now another problem emerged. As all the weight was distributed in the robot's back, the wheels could not create enough friction to start the movement. A counterweight was placed at the robot's front part. Now there were no motion problems.

Communication was critical among the Planner, the Board and the Leader. The Leader and the Board use the Serial protocol. Parallax shows off with its hardware because every I/O pin is capable of generating a PWM signal and communicating via Serial. In its datasheet it specifies that any pin can be used as a serial input and output port. In practice, it does not work as the manufacturer assumes. So separate input/output pins were used for communication. The BT communication between the Board and the Planner presented almost no problems. Except for the Time out MATLAB requires for waiting data.

Algorithm coupling was not that complex as it sounds. Almost all of them are very flexible and can manipulate data with almost no problem. The real challenge was to parametrize data. This is, the Particle Filter had no problem while using Decimal Degree coordinates, but for A*'s $\mathcal{W}$ the landmark's location was too broad, so they had to be set from 1 to a maximum of 400.

Chapter 7

Future Work and Conclusions

The obtained results were quite satisfactory, all the proposed objectives were accomplished. An online centralized low-cost non-holonomic object carrier Multiple Robot System was developed. All the algorithms employed worked as expected, plus they were all integrated and modified so they could coexist and work together. This project can grow as much as it can be wanted. The robots can be changed for more complex ones just like a Pioneer Robot. These type of robots are capable of moving any distance that it is specified, as well as rotating in any direction. Obviously their motion actions can be represented as Gaussian, not always they will perform exactly what they were commanded but they are much more liable. They have really big motors capable of handling large weights. And most important of all, they can be structurally modified with certain ease. The only disadvantage they can face is the presence of a computer that manipulates them, but due to wireless communications as XBee that is a minor problem.

Another improvement that the System can have is the employment of better sensors. LRF is not that liable for measuring large distances. Obstacle detection is critical for robot navigation so a faster and more precise sensor is required. Ultrasonic sensors are a good choice but they are not punctual, so noise can interfere into measurements, laser looks like a great choice but its limitations are the sampling time. The employed compass is quite accurate but it sometimes malfunctions because of the presence of some magnetic field, there shall be no problem is a shield is built for protecting it. A better and more accurate GPS can be used. The current one has not a good functioning while being indoors, so if an external antenna is

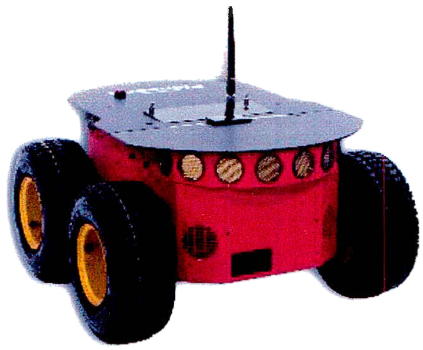


Figure 7.1: Pioneer Robot

connected that problem can be considered as solved.

An extension to the problem that was solved here can be the implementation of a Multiple Robot-Team System where several teams work together to deal with objects in the environment. With this extension a decentralized schema can be used. Each team will know the global goal and will plan its actions individually. Each team can have N robots depending of what it is going to transport, thus, some team members can be equipped with a robot-arm so the object is mounted and dismounted from the platform. An option for the robot's processor can be the employment of a mobile device such as a tablet. Currently, these devices are capable of doing lots of calculations at the time, have a nice user interface, Bluetooth, can access a remote server via Wi-Fi , and most important of all, they got the minimum required sensors used for navigation: GPS, magnetic field sensor and altimeter. Extended Reality as well as human interaction are the most attractive capabilities it has.

While talking about software improvements, the path can be computed while using Dynamic programming, this way a faster online algorithm can be used, and there will be less impact of motion uncertainty. The only disadvantage is that all actions are computed at the time so there is a large computing-load. The localization and mapping done here by the Particle Filter, can be replaced with a modified-SLAM algorithm, that is quite more efficient than SLAM and it maps all the environment. Reducing computing load is important if a decentralized architecture is used.

The job done to develop this project was demanding but at the same time satisfying. Hardware and Software resources were exploited to almost its maximum capabilities. The selected workspace helped a lot due to its

Figure 7.2: Samsung Galaxy SIII

rectangular shape and because it was in an open space so GPS could have a direct access to the satellites. It also aided that there are no big magnetic sources near there to interfere with the Compass readings. There were also lots of learnings. The author has a Biomedical Engineer background so much of the themes that where developed in this Thesis resulted new and abstract, so there was the need to check lots of bibliography to get up to date. Robotics is a field that is not the future, but the present of human race. Much of the risky activities that were made by humans are now done by robots. They are also capable of exploring certain areas that have been affected by a natural disaster or automating processes. What will come tomorrow may be autonomous robots just as the ones that only exist in films, but who knows, maybe one day The Jetsons world will be real.

Appendix A

Basic STAMP Programs

A.1 Boe-Bot Leader Source Code

```
' {$STAMP BS2}
' {$PBASIC 2.5}

LrfServo      PIN   14      ' Servo que mueve el Ping)))
LRF_TX        PIN   9       ' Serial output to LRF (connects to SIN)
LRF_RX        PIN   15      ' Serial input from LRF (connects to SOUT)

minimo        CON   280     ' Ancho de pulso para 0°
maximo        CON   1140    ' Ancho de pulso para 180°
salto         CON   20      ' Ancho de pulso para 0°

accion        VAR   Byte
i             VAR   Byte
iteraciones   VAR   Byte
LatGrad       VAR   Byte (2)
LatMin        VAR   Byte (2)
LatMinD       VAR   Byte (4)
NS            VAR   Byte
LonGrad       VAR   Byte (3)
LonMin        VAR   Byte (2)
LonMinD       VAR   Byte (4)
EW            VAR   Byte
pulso         VAR   Word
range         VAR   Word

Inicializacion:
  DEBUG "Iniciando al robot líder...", CR
  GOSUB Init_lrf
  DEBUG "Moviendo servo del LRF a posición inicial", CR
  GOSUB MotorPingInicial
```

```
  DEBUG "Esperando señal de sincronización...", CR
  SERIN 11, 84, [WAIT("start")]
  PAUSE 10
  SEROUT 10, 84, ["ack"]
  DEBUG "Señal recibida!!", CR, CR

Poll_accion:
  DEBUG CR, "Esperando accion", CR
  SERIN 11, 84, [accion]
  PAUSE 100

  IF (accion = "m") THEN
    DEBUG "Medir ambiente", CR
    SEROUT 10, 84, ["ack"]
    GOSUB Mide_ambiente
  ELSEIF (accion = "g") THEN
    DEBUG "Leer GPS", CR
    SEROUT 10, 84, ["ack"]
    GOSUB lee_GPS
  ELSEIF (accion = "f") THEN
    DEBUG "Mueve al frente", CR
    SEROUT 10, 84, ["ack"]
    GOSUB Frente
  ELSEIF (accion = "i") THEN
    DEBUG "Gira izquierda", CR
    SEROUT 10, 84, ["ack"]
    GOSUB Izquierda
  ELSEIF (accion = "d") THEN
    DEBUG "Gira derecha", CR
    SEROUT 10, 84, ["ack"]
    GOSUB Derecha
  ENDIF
  GOTO Poll_accion

Frente:
  FOR i = 0 TO 150
   PULSOUT 13, 620                                  'izquierdo quieto 720
   PULSOUT 12, 860                                  'derecho   quieto 740
    PAUSE 20
  NEXT
  PAUSE 500
  RETURN

Izquierda:
  FOR i = 0 TO 145
      PULSOUT 12, 960
      PULSOUT 13, 660
      PAUSE 20
  NEXT
```

```
  PAUSE 500
  RETURN

Derecha:
  FOR i = 0 TO 105
    PULSOUT 12, 810
    PULSOUT 13, 610
    PAUSE 20
  NEXT
  PAUSE 500
  RETURN

Mide_ambiente:
  GOSUB MotorPingInicial
  pulso = minimo
  FOR i = 1 TO 44
    SEROUT LRF_TX, 396, ["R"]
    SERIN  LRF_RX, 396, 3000, No_Response, [WAIT("D = "), DEC4 range]
    range = range MAX 990 MIN 160
    SEROUT 10, 84, [DEC range/10]
    PULSOUT LrfServo, pulso
    pulso = pulso + salto
    PAUSE 20
  NEXT
  RETURN

Init_lrf:
  DEBUG "Iniciando LRF... "
  PAUSE 500
  SEROUT LRF_TX, 396, ["U"]
  SERIN LRF_RX, 396, [WAIT(":")]
  DEBUG "Listo!", CR
  RETURN

No_Response:
  PAUSE 1000
  SEROUT LRF_TX, 396, ["U"]
  SERIN LRF_RX, 396, [WAIT(":")]
  RETURN

MotorPingInicial:
  FOR i = 1 TO 50
    PULSOUT LrfServo, minimo
    PAUSE 20
  NEXT
  RETURN

lee_GPS:
  SERIN 1,188,[WAIT("RMC,"),SKIP 9,STR LatGrad\2, STR LatMin\2, SKIP 1,
```

```
    STR LatMinD\4, SKIP 1, STR NS\1, SKIP 1, STR LonGrad\3, STR LonMin\2,
    SKIP 1, STR LonMinD\4, SKIP 1, STR EW\1]

  SEROUT 10, 84, [STR LatGrad\2]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR LatMin\2]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR LatMinD\4]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR NS\1]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR LonGrad\3]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR LonMin\2]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR LonMinD\4]
  SERIN 11, 84, [WAIT("ack")]
  SEROUT 10, 84, [STR EW\1]
  SERIN 11, 84, [WAIT("ack")]

  RETURN
```

A.2 Transmitter Board Source Code

```
' {$STAMP BS2}
' {$PBASIC 2.5}

SDA             PIN     9           ' SDA of compass to pin P0
SCL             PIN     10           ' SCL of compass to pin P1

WRITE_Data      CON     $3C          ' Requests Write operation
READ_Data       CON     $3D          ' Requests Read operation
MODE            CON     $02          ' Mode setting register
X_MSB           CON     $03          ' X MSB data output register

I2C_LSB         VAR     Bit
accion          VAR     Byte
i               VAR     Byte
iteraciones     VAR     Byte
range           VAR     Byte
opcion          VAR     Byte
I2C_DATA        VAR     Byte
I2C_REG         VAR     Byte
I2C_VAL         VAR     Byte
datosGPS        VAR     Byte (4)
X               VAR     Word
Y               VAR     Word
Z               VAR     Word
```

```
rawl            VAR       Word
rawh            VAR       Word

Inicializacion:
  PAUSE 1000
  SEROUT 1, 84,["con 88:53:2e:70:15:88", CR]
  SERIN 0, 84,[WAIT("ACK",CR)]

  WaitForConnection:
    IF IN5 = 0 THEN
      DEBUG HOME, "Iniciando BT...", CR,"esperando conexion...",CR
      GOTO WaitForConnection
    ENDIF
  DEBUG "Se conecto!!", CR, CR

  DEBUG "Iniciando HMC5883L... "
  I2C_REG = MODE
  I2C_VAL = $0
  GOSUB I2C_Write_Reg
  DEBUG "Listo!", CR

  PAUSE 1000
  DEBUG "Sincronizando micros...", CR
  SEROUT 15, 84, ["start"]
  SERIN 14, 84, [WAIT("ack")]
  DEBUG "Micros sincronizados!!!", CR, CR
  PAUSE 100

Poll_accion:

  DO
    DEBUG "Esperando accion", CR
    SERIN 0, 84, [accion]

    IF (accion = "M") THEN
      DEBUG "Medir ambiente...", CR
      SEROUT 15, 84, ["m"]
      SERIN 14, 84, [WAIT("ack")]
      GOSUB Recupera_lrf
      SEROUT 1, 84, [DEC 255]
    ELSEIF (accion = "G") THEN
      DEBUG "Leer GPS...", CR
      SEROUT 15, 84, ["g"]
      SERIN 14, 84, [WAIT("ack")]
      GOSUB Recupera_gps
      SEROUT 1, 84, [DEC 255]
    ELSEIF (accion = "C") THEN
      DEBUG "Leer HMC5883L...", CR
      GOSUB GetRawReading
```

```
      SEROUT 1, 84, ["x", SHEX x]
      SEROUT 1, 84, ["y", SHEX y]
      SEROUT 1, 84, ["z", SHEX z]
      SEROUT 1, 84, ["v", DEC 255,CR]
    ELSEIF (accion = "F") THEN
      DEBUG "Mover al frente", CR
      SEROUT 15, 84, ["f"]
      SERIN 14, 84, [WAIT("ack")]
    ELSEIF (accion = "I") THEN
      DEBUG "Gira izquierda", CR
      SEROUT 15, 84, ["i"]
      SERIN 14, 84, [WAIT("ack")]
    ELSEIF (accion = "D") THEN
      DEBUG "Gira derecha", CR
      SEROUT 15, 84, ["d"]
      SERIN 14, 84, [WAIT("ack")]
    ENDIF
  LOOP

Recupera_lrf:
  FOR i = 1 TO 44
    SERIN 14, 84,[STR range\2]
    SEROUT 1, 84, [STR range\2, CR]
  NEXT
  RETURN

Recupera_gps:

  ' LATITUD GRADOS
  SERIN 14, 84, [STR datosGPS\2]
  SEROUT 1, 84, [STR datosGPS\2, CR]
  SEROUT 15, 84, ["ack"]
  ' LATITUD MINUTOS
  SERIN 14, 84, [STR datosGPS\2]
  SEROUT 1, 84, [STR datosGPS\2,CR]
  SEROUT 15, 84, ["ack"]
  ' LATITUD DECIMAS DE MINUTO
  SERIN 14, 84, [STR datosGPS\4]
  SEROUT 1, 84, [STR datosGPS\4,CR]
  SEROUT 15, 84, ["ack"]
  ' NORTE/SUR
  SERIN 14, 84, [STR datosGPS\1]
  SEROUT 1, 84, [DEC datosGPS,CR]
  SEROUT 15, 84, ["ack"]
  ' LONGITUD GRADOS
  SERIN 14, 84, [STR datosGPS\3]
  SEROUT 1, 84, [STR datosGPS\3,CR]
  SEROUT 15, 84, ["ack"]
  ' LONGITUD MINUTOS
```

```
  SERIN 14, 84, [STR datosGPS\2]
  SEROUT 1, 84, [STR datosGPS\2,CR]
  SEROUT 15, 84, ["ack"]
  ' LONGITUD DECIMAS DE MINUTO
  SERIN 14, 84, [STR datosGPS\4]
  SEROUT 1, 84, [STR datosGPS\4,CR]
  SEROUT 15, 84, ["ack"]
  ' ESTE/OESTE
  SERIN 14, 84, [STR datosGPS\1]
  SEROUT 1, 84, [DEC datosGPS,CR]
  SEROUT 15, 84, ["ack"]
  RETURN

GetRawReading:
  PAUSE 400                          ' Wait for new data
  ' Send request to X MSB register
  GOSUB I2C_Start
  I2C_DATA = WRITE_Data
  GOSUB I2C_Write
  I2C_DATA = X_MSB
  GOSUB I2C_Write
  GOSUB I2C_Stop

  'Get data from register (6 bytes total, 2 bytes per axis)
  GOSUB I2C_Start
  I2C_DATA = READ_Data
  GOSUB I2C_Write

  ' Get X
  GOSUB I2C_Read
  rawH = I2C_Data
  GOSUB I2C_ACK
  GOSUB I2C_Read
  rawL = I2C_Data
  GOSUB I2C_ACK
  X = (rawH << 8) | rawL

  ' Get Z
  GOSUB I2C_Read
  rawH = I2C_Data
  GOSUB I2C_ACK
  GOSUB I2C_Read
  rawL = I2C_Data
  GOSUB I2C_ACK
  Z = (rawH << 8) | rawL

  ' Get Y
  GOSUB I2C_Read
  rawH = I2C_Data
```

```
  GOSUB I2C_ACK
  GOSUB I2C_Read
  rawL = I2C_Data
  GOSUB I2C_NACK
  Y = (rawH << 8) | rawL

  GOSUB I2C_Stop
  RETURN


'---------I2C functions------------
' Set I2C_REG & I2C_VAL before calling this
I2C_Write_Reg:
  GOSUB I2C_Start
  I2C_DATA = WRITE_DATA
  GOSUB I2C_Write
  I2C_DATA = I2C_REG
  GOSUB I2C_Write
  I2C_DATA = I2C_VAL
  GOSUB I2C_Write
  GOSUB I2C_Stop
  RETURN

' Set I2C_REG before calling this, I2C_DATA will have result
I2C_Read_Reg:
  GOSUB I2C_Start
  I2C_DATA = WRITE_DATA
  GOSUB I2C_Write
  I2C_DATA = I2C_REG
  GOSUB I2C_Write
  GOSUB I2C_Stop
  GOSUB I2C_Start
  I2C_DATA = READ_DATA
  GOSUB I2C_Write
  GOSUB I2C_Read
  GOSUB I2C_NACK
  GOSUB I2C_Stop
  RETURN

I2C_Start:
  LOW SDA
  LOW SCL
  RETURN

I2C_Stop:
  LOW   SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```

I2C_ACK:
  LOW   SDA
  INPUT SCL
  LOW   SCL
  INPUT SDA
  RETURN

I2C_NACK:
  INPUT SDA
  INPUT SCL
  LOW   SCL
  RETURN

I2C_Read:
  SHIFTIN SDA, SCL, MSBPRE, [I2C_DATA]
  RETURN

I2C_Write:
  I2C_LSB = I2C_DATA.BIT0
  I2C_DATA = I2C_DATA / 2
  SHIFTOUT SDA, SCL, MSBFIRST, [I2C_DATA\7]
  IF I2C_LSB THEN INPUT SDA ELSE LOW SDA
  INPUT SCL
  LOW SCL
  INPUT SDA
  INPUT SCL
  LOW SCL
  RETURN
```

A.3 Boe-Bot Follower Source Code

```
' {$STAMP BS2}
' {$PBASIC 2.5}

DO
  SELECT INL                                   ' Select line follower states
    CASE %00011000, %00001100, %00110000
      DEBUG HOME, "ADELANTE        "
      PULSOUT 13, 1000
      PULSOUT 12, 500
    CASE %00000011, %00000111, %00000001, %00000110, %00111000
      DEBUG HOME, "AJUSTA IZQUIERDA "
      PULSOUT 12, 450
'      PULSOUT 12, 750
    CASE %11000000, %11100000, %10000000, %01100000, %00011100
      DEBUG HOME, "AJUSTA DERECHA   "
'      PULSOUT 13, 750
```

```
      PULSOUT 13, 1050
  ENDSELECT
LOOP
```

Appendix B

MATLAB Programs

B.1 Main Source Code

```
close all; clear all; clc
format longEng
%% SELECCIÓN DEL ESPACIO DE TRABAJO
espacio = 0;

while 1
    clc
    disp('Selecciona un espacio de trabajo:');
    disp('1 - Rectángulo superior');
    disp('2 - Rectángulo inferior');
    disp('3 - Dentro CEDETEC');
    disp('4 - Canchas Prepa')
    espacio = input('Tu opción es: ');
    if espacio == 1 || espacio == 2 || espacio == 3 || espacio == 4
        break
    end
end
disp(strcat('Se cargará el espacio de trabajo: ', num2str(espacio)));

switch espacio
    case 1
        % CEDETEC 1
        landmarksGPS = [19.283587845330565 -99.13562268018722;   % Arriba Derecha
                19.283265050042573 -99.13566827774048;      % Abajo Derecha
                19.283301760127006 -99.13594722747803;      % Abajo Izquierda
                19.283622023617926 -99.13589626550674];     % Arriba Izquierda
        goal = [19.283537, -99.135785];
    case 2
        % CEDETEC 2
```

```
        landmarksGPS = [19.283201756774254   -99.13553550839424;
% Arriba Derecha
            19.282911873292704 -99.13556769490242;    % Abajo Derecha
            19.282953646926493 -99.13583859801292;    % Abajo Izquierda
            19.28327644282828  -99.13578897714615];   % Arriba Izquierda
        goal = [19.283011876822485, -99.1356173157692];
    case 3
        % DENTRO CEDETEC
        landmarksGPS = [19.283651138449677, -99.1348448395729;
            19.283035928295213, -99.13514792919159;
            19.283208086102178, -99.13551270961761;
            19.28382076388827, -99.13519889116287];
        goal = [19.28329669666758, -99.13522034883499];
    case 4
        % CANCHA PREPA
        landmarksGPS = [19.285181558622586, -99.13645818829536;
            19.285029656515284, -99.13656547665596;
            19.285163836717327, -99.13684576749801;
            19.285320802097164, -99.13675993680954];
        goal = [19.2852195341274, -99.13674920797348];
end

disp('Cargando constantes de W... ')
landmarks = fliplr(floor(10^6 * landmarksGPS));
goal = fliplr(floor(10^6 * goal));
goal2 = goal;

minLat = min(landmarks(:,2));
maxLat = max(landmarks(:,2));
minLon = min(landmarks(:,1));
maxLon = max(landmarks(:,1));


%% CTES FILTRO PARTÍCULAS
disp('Cargando constantes del Filtro de Partículas... ')

N = 500;
length = 1;
motions = [0 0];
measurements = [];
bearing_noise = 1;
steering_noise = 0.1;
distance_noise = 1;

%% A*

disp('Creando mundo para usar en A* ... ')
mundo_land(:,1) = 1 + landmarks(:,1) - minLon;
mundo_land(:,2) = 1 + landmarks(:,2) - minLat;
```

```
goal(1) = goal(1) - minLon;
goal(2) = goal(2) - minLat;

mundo = zeros(max(mundo_land(:,1)), max(mundo_land(:,2)));

for i = 1:size(mundo,1)
    for j = 1:size(mundo,2)
        if ~inpolygon(i, j, mundo_land(:,1), mundo_land(:,2))
            mundo(i,j) = 1;
        end
    end
end

subplot(1,2,1)
imagesc(mundo)

%% KALMAN FILTER
disp('Matrices␣para␣Filtro␣de␣Kalman')

dt = 2.7;
P = [0 0 0 0; 0 0 0 0; 0 0 1000 0; 0 0 0 1000];
u = [0; 0; 0; 0];
F = [1 0 dt 0; 0 1 0 dt; 0 0 1 0; 0 0 0 1];
H = [1 0 0 0; 0 1 0 0];
R = [0.1 0; 0 0.1];

ancho = 280:20:1140;
angulo = 0:180/(size(ancho,2)-1):180;
angulo = deg2rad(angulo');

%% SERIAL

disp('Iniciando␣el␣puerto␣serial...')
s1 = serial('COM4');
set(s1, 'Baudrate', 9600);
set(s1, 'DataBits', 8);
set(s1, 'Parity', 'none');
set(s1, 'Terminator', 'CR/LF');
set(s1, 'OutputBufferSize', 2);
set(s1, 'InputBufferSize', 1000);
set(s1, 'Timeout', 1);
fopen(s1)
disp('Conexión␣exitosa␣a␣eb500!!')

%% CREACIÓN DE PARTÍCULAS
disp('Creando␣Partículas...')

for i = 1:N
    p(i,:) = robot(minLat + (maxLat-minLat)*rand(), minLon + (maxLon-minLon)*rand(), r
```

```
end

% VEO PARTICULAS ALEATORIAS
subplot(1,2,2)
scatter(p(:,2), p(:,1))

saveas(gcf,strcat('C:\Users\Rolix\Dropbox\Tesis\Programas finales\MATLAB\prueb
        num2str(1),'.jpg'))

disp('Pausa.... Presiona "ENTER" para continuar')
% pause

%% ALGORITMO PRINCIPAL

disp('Ubicando robot')
for iteracion = 1:100
    disp(strcat('Iteración.... ',num2str(iteracion)))
    disp('Moviendo partículas...')
    for i = 1:N
        p2(i,:) = move(p(i,:), motions(end,:), distance_noise, steering_noise,
    end
    p = p2;

    % LECTURA DE GPS
    disp('Leyendo GPS')
    gps = leeGPS(s1);
    [lat lon] = convierteGPS(gps);

    lat = floor(lat * 10^6);
    lon = floor(lon * 10^6);
    gpsDatos{iteracion} = [lat lon];

    disp('Leyendo Brújula')
    % LECTURA DE BRÚJULA
    [x y z] = leeCompass(s1);
    [azimuth elev r] = cart2sph(x, y, z);
    azimuth = mod(azimuth + pi/2, 2*pi);
    compassDatos{iteracion} = azimuth;

    disp('Obteniendo ángulos...')
    % OBTENCIÓN DE ÁNGULOS
    Z = [];
    for i = 1:size(landmarks,1)
        deltax = landmarks(i,2) - lat;
        deltay = landmarks(i,1) - lon;
        temp = atan2(deltay, deltax) - azimuth;
        temp = mod(temp, (2*pi));
        Z = [Z temp];
```

```
    end

    % CALCULO ERROR
    measurements(end + 1, :) = Z;

    disp('Encontrando_error...')
    for i = 1:N
        w(i,:) = measurement_prob(p(i,:), landmarks, measurements(end,:), bearing_nois
    end

    disp('Re-sampling...')
    index = floor(rand() * N) + 1;
    beta = 0;
    mw = max(w);
    for i = 1:N
        beta = beta + rand() * 2 * mw;
        while beta > w(index)
            beta = beta - w(index);
            index = mod(index , N) + 1;
        end
        p3(i,:) = p(index,:);
    end
    p = p3;

    [lat-minLat lon-minLon]

    subplot(1,2,2)
    scatter(p(:,2), p(:,1))
    axis([minLon maxLon minLat maxLat])

    saveas(gcf,strcat('C:\Users\Rolix\Dropbox\Tesis\Programas_finales\MATLAB\prueba4\c
        num2str(iteracion+1),'.jpg'))

    disp('Pausa..._Presiona_"ENTER"_para_continuar')

    particulas{iteracion} = p;

end

disp('Iniciando_algoritmo_principal...')
iteracion = iteracion + 1;
while 1
    disp(strcat('Iteración..._',num2str(iteracion)))

    for k = 1:2
        disp('Moviendo_partículas...')
        for i = 1:N
            p2(i,:) = move(p(i,:), motions(end,:), distance_noise, steering_noise, len
        end
```

```
    p = p2;

    % LECTURA DE GPS
    disp('Leyendo␣GPS')
    gps = leeGPS(s1);
    [lat lon] = convierteGPS(gps);

    lat = floor(lat * 10^6);
    lon = floor(lon * 10^6);

    disp('Leyendo␣Brújula')
    % LECTURA DE BRÚJULA
    [x y z] = leeCompass(s1);
    [azimuth elev r] = cart2sph(x, y, z);
    azimuth = mod(azimuth + pi/2, 2*pi);

    disp('Obteniendo␣ángulos...')
    % OBTENCIÓN DE ÁNGULOS
    Z = [];
    for i = 1:size(landmarks,1)
        deltax = landmarks(i,2) - lat;
        deltay = landmarks(i,1) - lon;
        temp = atan2(deltay, deltax) - azimuth;
        temp = mod(temp, (2*pi));
        Z = [Z temp];
    end

    % CALCULO ERROR
    measurements(end + 1, :) = Z;

    disp('Encontrando␣error...')
    for i = 1:N
        w(i,:) = measurement_prob(p(i,:), landmarks, measurements(end,:), 
    end

    disp('Re-sampling...')
    index = floor(rand() * N) + 1;
    beta = 0;
    mw = max(w);
    for i = 1:N
        beta = beta + rand() * 2 * mw;
        while beta > w(index)
            beta = beta - w(index);
            index = mod(index , N) + 1;
        end
        p3(i,:) = p(index,:);
    end
    p = p3;
```

```

        subplot(1,2,2)
        scatter(p(:,2), p(:,1))
        axis([minLon maxLon minLat maxLat])
    end

    [x y theta] = get_position(p);
    init = floor([y x]);

    init(1) = abs(init(1) - minLon);
    init(2) = abs(init(2) - minLat);

    disp('Verificando se se llegó a la meta...')
    if ((init(2) < (goal(2) * 1.2)) && (init(2) > (goal(2) * .8))) ...
            && ((init(1) < (goal(1) * 1.2)) && (init(1) > (goal(1) * .8)))
        disp('¡Meta alcanzada!')
        break
    end

    disp('¡Meta aún no alcanzada!')
    disp('Continúa algoritmo...')

    disp('Detectando obstáculos...')

    [mundo mov] = mappea(s1, mundo, round(x - minLat), round(y  - minLon), theta);

    disp('Calculando trayectoria...')
    [path path_inst linea] = astar2(mundo, init, goal);

    accion = movBoeBot(mov, path_inst(1));

    subplot(1,2,1)
    imagesc(mundo)
    line(linea(:,2),linea(:,1),'Color','white')

    % mayor a 315° o menor a 45°
    if theta <= deg2rad(45) || theta > deg2rad(315)
        temp = 'v';
        % mayor a 45° o menor a 135°
    elseif theta <= deg2rad(135) && theta > deg2rad(45)
        temp = '<';
        % mayor a 135° o menor a 225°
    elseif theta <= deg2rad(225) && theta > deg2rad(135)
        temp = '^';
        % mayor a 255° o menor a 285°
    elseif theta <= deg2rad(315) && theta > deg2rad(225)
        temp = '>';
    end

```

```
    disp('Moviendo robot')
    policy_actions(1:2) = [temp path_inst(1)];
    temp = policy_actions(2);
    mov = trad_mov(policy_actions(1:2));

    if strcmp(mov(1),'F')
        motions(end+1,:) = [12 0];
    elseif strcmp(mov(1),'D')
        motions(end+1,:) = [12 3*pi/4];
    else
        motions(end+1,:) = [12 pi/4];
    end

    for gg=1:size(mov,2)
        fwrite(s1,mov(gg));
        pause(5)
    end

    iteracion = iteracion + 1;
    saveas(gcf,strcat('C:\Users\Rolix\Dropbox\Tesis\Programas finales\MATLAB\p
        num2str(iteracion),'.jpg'))

    disp('Pausa... Presiona "ENTER" para continuar')
    %     pause
end

%% FIN
disp('Terminando conexión...')
fclose(s1),delete(s1),clear s1;
disp('Fin de conexión')
```

B.2 Robot Creation Function

```
function r = robot(x, y, z)

r = [x y z];

end
```

B.3 Particle Filter Movement Function

```
function result = move(r, motion, distance_noise, steering_noise, length)

alfa = motion(1) + randn() * steering_noise;
d = motion(2) + randn() * distance_noise;

x = r(1);
y = r(2);
```

```
theta = r(3);

beta = (d / length) * tan(alfa);
if beta == 0
    beta = 0.001;
end

R = d / beta;
cx = x - (R * sin(theta));
cy = y + (R * cos(theta));

x = cx + (R * sin(theta + beta));
y = cy - (R * cos(theta + beta));
theta = mod(theta + beta, 2*pi);

result = robot(x, y, theta);

end
```

B.4 Particle Filter Measurement Error Function

```
function error = measurement_prob(r, landmarks, measurements, bearing_noise)

predicted_measurements = sense(r, landmarks);

error = 1;
for i = 1:size(measurements,2)
    error_bearing = abs(measurements(i) - predicted_measurements(i));
    error_bearing = mod((error_bearing + pi), 2*pi) - pi;

    error = error * (exp(-(error_bearing^2) / (bearing_noise^2) / 2) / ...
        sqrt(2 * pi * (bearing_noise^2)));
end

end
```

B.5 Particle Filter Get Position Function

```
function [x y orientation] = get_position(p)

x = 0;
y = 0;
orientation = 0;

for i = 1:size(p,1)
    x = x + p(i,1);
    y = y + p(i,2);
    orientation = orientation + mod((p(i,3) - p(1,3) + pi), (2*pi)) + p(1,3) - pi;
```

```
end

x = x / size(p,1);
y = y / size(p,1);
orientation = orientation / size(p,1);

end
```

B.6 GPS Data Acquisition Function

```
function gps2 = leeGPS(s1)

pause(2)
gps = [];
fwrite(s1, 'G');

while (true)
    if s1.BytesAvailable ~= 0
        a = fscanf(s1, '%3u');
        gps = [gps,a];
        a = [];
        if size(find(gps == 255),1)==1
            break
        end
    end
end

gps';

gps2 = '';

for i = 1:size(gps,1)-1
    if i == 2 || i == 7
        if size(num2str(gps(i)),2) == 1
            gps2 = strcat(gps2, '0',num2str(gps(i)));
        else
            gps2 = strcat(gps2, num2str(gps(i)));
        end
    elseif i == 3 || i == 8
        if size(num2str(gps(i)),2) == 1
            gps2 = strcat(gps2, '00',num2str(gps(i)));
        elseif size(num2str(gps(i)),2) == 2
            gps2 = strcat(gps2, '0',num2str(gps(i)));
        else
            gps2 = strcat(gps2, num2str(gps(i)));
        end
    elseif i == 5
        if gps(i) == 78
```

```
            gps2 = strcat(gps2, 'N');
        elseif gps(i) == 83
            gps2 = strcat(gps2, 'S');
        end
    elseif i == 6
        if size(num2str(gps(i)),2) == 2
            gps2 = strcat(gps2, '0',num2str(gps(i)));
        else
            gps2 = strcat(gps2, num2str(gps(i)));
        end
    elseif i == 10
        if gps(i) == 87
            gps2 = strcat(gps2, 'W');
        elseif gps(i) == 69
            gps2 = strcat(gps2, 'E');
        end
    else
        gps2 = strcat(gps2, num2str(gps(i)));
    end

end

end
```

B.7 GPS Data Conversion Function

```
function [latitud longitud] = convierteGPS(cadena)

latDeg = cadena(1:2);
latMin = str2double(cadena(3:4));
latMinD = str2double(cadena(5:8));
NS = cadena(9);

lonDeg = cadena(10:12);
lonMin = str2double(cadena(13:14));
lonMinD = str2double(cadena(15:18));
EW = cadena(19);

w1 = (latMin*1000/6) + latMinD/60;
w1s = num2str(w1);

w2 = (lonMin*1000/6) + lonMinD/60;
w2s = num2str(w2);

for i = 1:size(w1s,2)
    if strcmp(w1s(i),'.')
        w1s(i) = [];
        w1s = strcat('.',w1s);
```

```
        break;
    end
end
w1s = strcat(latDeg,w1s);

latitud = str2double(w1s);
if strcmp(NS, 'S')
    latitud = -1 * latitud;
end

for i = 1:size(w2s,2)
    if strcmp(w2s(i),'.')
        w2s(i) = [];
        w2s = strcat('.',w2s);
        break;
    end
end
w2s = strcat(lonDeg,w2s);

longitud = str2double(w2s);
if strcmp(EW, 'W')
    longitud = -1 * longitud;
end

end
```

B.8 Compass Data Acquisition Function

```
function [x y z] = leeCompass(s1)

pause(2)
valores = [];
fwrite(s1,'C');

while (true)
    if s1.BytesAvailable ~= 0
        a = fscanf(s1, '%3u');
        valores = [valores; a];
        a = [];
        if size(find(valores == 255),1)==1
            break
        end
    end
end

cy = find(valores == 'y');
cz = find(valores == 'z');
cw = find(valores == 'w');
```

```

xh = valores(2:cy-1);
yh = valores(cy+1:cz-1);
zh = valores(cz+1:cw-1);

if strcmp(xh(1), '-')
    x = -1 * hex2dec(xh(2:end));
else
    x = hex2dec(xh);
end

if strcmp(yh(1), '-')
    y = -1 * hex2dec(yh(2:end));
else
    y = hex2dec(yh);
end

if strcmp(zh(1), '-')
    z = -1 * hex2dec(zh(2:end));
else
    z = hex2dec(zh);
end

end
```

B.9 LRF Data Acquisition Function

```
function valores = lrf(s1)

pause(2)
valores = [];
fwrite(s1, 'M');

while (true)
    if s1.BytesAvailable ~= 0
        a = fscanf(s1, '%3u');
        valores = [valores; a];
        a = [];
        size(valores)
        if size(find(valores == 255),1) == 1
            break
        end
    end
end

valores = valores(1:end-1)

end
```

B.10 A* Search Function

```
function [policy policy_actions puntos] = astar2(grid, init, goal)

puntos = [];

cost = 1;
delta = [-1 0; 0 -1; 1 0; 0 1];
delta_name = ['^'; '<'; 'v'; '>'];

closed = zeros(size(grid));
closed(init(1),init(2)) = 1;
action = -1 * ones(size(grid));
expand = action;

heuristica = gridFire(size(grid,1), size(grid,2), goal);

x = init(1);
y = init(2);
h = heuristica(x,y);
g = 0;
f = g+h;
%open =[g x y];
open =[f g h x y];

found = 0;
resign = 0;
count=0;

while found == 0 && resign == 0
    if size(open, 1) == 0
        resign = 1;
        disp('fail')
    else
        open = flipud(sortrows(open));
        next = open(end,:);
        open(end,:) = [];

        x = next(4);
        y = next(5);
        g = next(2);

        expand(x,y)=count;
        count=count+1;

        if x == goal(1) && y == goal(2)
            found = 1;
        else
```

```
                for i = 1:size(delta,1)
                    x2 = x + delta(i,1);
                    y2 = y + delta(i,2);
                    if (x2>=1) && (x2<=size(grid,1)) && (y2 >=1) && (y2<=size(grid,2))
                        if closed(x2,y2) == 0 && grid(x2,y2) == 0
                            g2 = g + cost;
                            h2=heuristica(x2,y2);
                            f2=g2+h2;

                            open = [open; [f2 g2 h2 x2 y2]];
                            closed(x2,y2) = 1;
                            action(x2,y2) = i;
                        end
                    end
                end
            end
        end
    end
policy(size(grid))='␣';
x=goal(1);
y=goal(2);

policy(x,y)='*';
policy_actions = '*';

grid2 = zeros(size(grid));
while x ~=init(1) || y~=init(2)
    x2 = x - delta(action(x,y),1);
    y2 = y - delta(action(x,y),2);
    puntos = [puntos; x2 y2];
    policy(x2,y2) = delta_name(action(x,y));
    policy_actions = [policy_actions, delta_name(action(x,y))];
    x=x2;
    y=y2;
end

policy_actions = fliplr(policy_actions);

end
```

B.11 A* Search Heuristic Function

```
function [heuristica]=gridFire(m,n,centro)
%genera una matriz con valores crecientes alrededor del punto centro
    heuristica = zeros(m,n);
    fin = centro;

    for i = 1:size(heuristica,1)
```

```
        for j = 1:size(heuristica ,2)
            heuristica(i,j) = ((i-fin(1)))^2 + ((j-fin(2)))^2;
        end
    end

end
```

B.12 Kalman Filter Measurement Function

```
function [x P] = KalmanFilterMeasurement(x, P, Z, H, R)

    y = Z' - H*x;
    S = H*P*H' + R;
    K = P*H'*pinv(S);
    x = x + K*y;
    P = (eye(size(P)) - (K*H))*P;

end
```

B.13 Kalman Filter Prediction Function

```
function [x P] = KalmanFilterPrediction(x, P, F, u)
    x = F*x + u;
    P = F*P*F';

end
```

B.14 Boe-Bot Movement Function

```
function [mov]=trad_mov(policy)
%Esta funcion recibe una policy de la forma string lineal
%
%       policy='>v>>>>vvv* ';
%
%y regresa los moviemientos para el robot
%
%        s - seguir derecho
%        l - girar 90° izquierda
%        r - girar 90° derecha
%        f - dar vueltas de Felicidad, porque llegaste al final
%
% el resultado lo regresa en mov como string
%
%        [mov]=trad_mov(policy)


%estado=0;
```

```
%mov='s ';
switch policy(1)
    case '>'
        estado=0;
        mov='F';
    case '<'
        estado=1;
        mov='F';
    case 'v'
        estado=2;
        mov='F';
    case '^'
        estado=3;
        mov='F';
    case '*'
        estado=4;
        mov='F';
end

for i=2:size(policy,2)
    switch policy(i)
        case '>'
            estadon=0;
        case '<'
            estadon=1;
        case 'v'
            estadon=2;
        case '^'
            estadon=3;
        case '*'
            estadon=4;
    end
    if estado~=estadon
        %Si estaba viendo a la derecha
        if estado==0 && estadon==2
            %                mov=strcat(mov, 'rs ');
            mov='DD';
        elseif estado==0 && estadon==3
            %                mov=strcat(mov, 'ls ');
            mov='II';
        elseif estado==0 && estadon==1
            %                mov=strcat(mov, 'lls ');
            mov='III';
            %Si estaba viendo a la izquierda
        elseif estado==1 && estadon==2
            %                mov=strcat(mov, 'ls ');
            mov='II';
        elseif estado==1 && estadon==3
            %                mov=strcat(mov, 'rs ');
```

```
            mov='DD';
        elseif estado==1 && estadon==0
            %               mov=strcat(mov,'rrs');
            mov='DDD';
            %Si estaba viendo hacia arriba
        elseif estado==3 && estadon==2
            %               mov=strcat(mov,'rrs');
            mov='DDD';
        elseif estado==3 && estadon==1
            %               mov=strcat(mov,'ls');
            mov='II';
        elseif estado==3 && estadon==0
            %               mov=strcat(mov,'rs');
            mov='DD';
            %Si estaba viendo hacia abajo
        elseif estado==2 && estadon==3
            %               mov=strcat(mov,'rrs');
            mov='DDD';
        elseif estado==2 && estadon==1
            %               mov=strcat(mov,'rs');
            mov='DD';
        elseif estado==2 && estadon==0
            %               mov=strcat(mov,'ls');
            mov='II';
        elseif estadon==4
            %               mov=strcat(mov,'f');
            mov='F';
        end
    else
        %           mov=strcat(mov,'s');
        mov='F';
    end
    estado=estadon;
end
end
```

Bibliography

[1] *Compass Module 3-Axis HMC5883L*. Parallax Inc., 1.0 edition, April 2011.

[2] *Infrared Line Follower Kit*. Parallax Inc., 1.0 edition, January 2011.

[3] Boe-bot robot kit. `http://www.parallax.com/Store/Robots/AllRobots/tabid/128/CategoryID/3/List/0/SortField/0/Level/a/ProductID/296/Default.aspx`, October 2012.

[4] Crawler kit for the boe-bot robot. `http://www.parallax.com/Store/Robots/AllRobots/tabid/755/CategoryID/3/List/0/SortField/0/Level/a/ProductID/314/Default.aspx`, October 2012.

[5] Gripper kit of the boe-bot robot. `http://www.parallax.com/Store/Robots/AllRobots/tabid/755/CategoryID/3/List/0/SortField/0/Level/a/ProductID/311/Default.aspx`, October 2012.

[6] Isaac Asimov. *The Foundation Novels*. Bantam Dell, 2001.

[7] Karel Capek. *R.U.R. (Rossum's Universal Robots)*. Dover Thrift Editions, 2001.

[8] Khac Duc Do Khiang-Wee Lim Cheng-Heng Fua, Shuzhi Sam Ge. Multirobot formations based on the queue-formation scheme with limited communication. *IEEE Transactions On Robotics*, 23(6):1160–1169, 2007.

[9] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principle of Robot Motion: Theory, Algorithms, and Implementation*. The MIT Press, 2005.

[10] Víctor de La Cueva Hernández. *Planificación de trayectorias*. Instituto Tecnológico y de Estudios Superiores de Monterrey, 2011.

[11] A7 Engineering. *EmbeddedBlue 500 User Manual.* Parallax Inc., 12860 C Danielson Court, revision e edition, April 2005.

[12] Domenico Prattichizzo Gian Luca Mariottini, Giuseppe Oriolo. Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. *IEEE Transactions On Robotics*, 23(1):87–100, February 2007.

[13] Domenico Prattichizzo Nicholas Prattichizzo Nicholas Vander Valk Nathan Michael George Pappas Gian Luca Mariottini, Fabio Morbidi and Kostas Daniilidis. Vision-based localization for leader-follower formation control. *IEEE Transactions On Robotics*, 25(6):1431–1448, 2009.

[14] Joh Wen He Bai. Cooperative load transport: A formation-control perspective. *IEEE Transactions on Robotics*, 26(4):742–750, August 2010.

[15] Manabu Sato Kazuhiro Kosuge. Transportation of a single object by multiple decentralized-controlled nonholonomic mobile robot. In *Proceedings of the 1999 IEEE/RSJ*, pages 1681–1686.

[16] Shoichi Maeyama Shin'ichi Yuta Kazunori Ohno, Takashi Tsubouchi. A mobile robot campus walkway following with daylight-change-proof walkway color image segmentation. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 77–83, Mani, Hawaii, USA, Oct. 29 - Nov. 03 2001. IEEE.

[17] Jean-Claude Latombe. *Robot Motion Planning.* The Kluwer International Series In Engineering And Computer Science. Kluwer Academic Publishers, 1991.

[18] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[19] Andy Lindsay. *Robotics with the BOe-Bot.* Parallax Inc., version 3.0 edition, 2012.

[20] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions On Computers*, 32(2):108–120, February 1983.

[21] George Lucas. *Star Wars: A New Hope.* Ballatine Books, 1976.

[22] ZhiDong Wang Eiji Nakano Majid Nili Ahmadabadi, Saahin Mehdinezhad. A constrain-move based distributed cooperation strategy for four object lifting robots. In *Proceedings of the 2000 IEEE/RSJ*, International Conference on Intelligent Robots and Systems, pages 2030–2035. IEEE, 2000.

[23] John McPhee Mike Peasgood, Christopher Michael Clark. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, (2):283–292, April 2008.

[24] J. Saito S. Yamada. Action selection without explicit communication for multi-robot box-pushing. In *Proceedings of the 1999 IEEE/RSJ*, International Conference on Intelligent Robot and Systems, pages 1444–1449. IEEE, 1999.

[25] Dieter Fox Sebastian Thrun, Wolfram Burgard. *Probabilistic Robotics*. The MIT Press, 2006.

[26] Inc SiRF Technology. *NMEA Reference Manual*. Number 1050-0042. SRIF, 217 Devcon Drive, revision 2.2 edition, November 2008.

[27] Peter Norvig Stuart Russel. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., 2010.

[28] Prasertsak Detudom Thavida Maneewarn. Mechanics of cooperative nonprehensile pulling by multiple robots. In *Proceedings of the 2005 IEEE/RSJ*, International Conference on Intelligent Robot and Systems, pages 1319–1324. IEEE, 2005.

[29] H. Asama H. Kaetsu I. Endo Y. Arai, T. Fujii. Realization of autonomous navigation in mulirobot environment. In *Proceedings of the 1998 IEEE/RSJ*, International Conference on Intelligent Robots and Systems, pages 1999–2004, Victoria B.C., Canada, October 1998. IEEE.

[30] Clarance W. de Silva Ying Wang. Multi-robot box-pushing: Single-agent-q-learning vs. team q-learning. In *Proceedings of the 2006 IEEE/RSJ*.

[31] Y. Charlie Hu C. S. George Lee Yongguo Mei, Yung-Hsiang Lu. Deployment of mobile robots with energy and timing contraints. *IEEE Transactions on Robotics*, 22(3):507–522, June 2006.

[32] Kazuhiro Kosuge ZhiDong Wang, Yasuhisa Hirata. Control multiple mobile robots for object caging and manipulation. In *Proceedings of the 2003 IEEE/RSJ*.

[33] Lei Li Nong Gu Shuo Wang Zhiqiang Cao, Min Tan. Cooperative hunting by distributed mobile robots based on local interaction. *IEEE Transactions on Robotics*, (2):403–407, April 2006.