Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Indoor and Outdoor Rover with Simultaneous Localization And Mapping (SLAM)

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Alex Albala Diaz

DIRECTOR: Juan Lopez Rubio

DATE: December 1, 2011

Título: Rover con Simultaneous Localization And Mapping (SLAM) para interiores y exteriores

Autor: Álex Albalá Díaz

Director: Juan López Rubio

Fecha: 1 de diciembre de 2011

Resumen

En este documento se describe el diseño, desarrollo e implementación de un sistema SLAM (*Simultaneous Localization And Mapping*) compuesto por dos subsistemas: robot autónomo de bajo coste y una estación de tierra donde se visualizan datos como telemetría e información del robot. El objetivo de un algoritmo SLAM es poder dejar el robot en un entorno desconocido y que él mismo identifique el entorno mediante sensores y extraiga un mapa aproximado. Para ello es muy importante que el propio robot sea capaz de localizarse automáticamente para poder ubicar bien en el espacio todos los datos de los sensores. Así pues el objetivo principal de este proyecto es desarrollar un sistema SLAM de bajo coste. En este trabajo se usará hardware para prototipado rápido para la prueba de concepto y así en un futuro poder usarlo en docencia.

En el documento se comparan diferentes algoritmos de SLAM y se escoge el más adecuado para este sistema. El algoritmo escogido se estudia a fondo y se implementa en el sistema. También se estudia uno de los principales errores comunes a todos los robots terrestres en la localización: los errores de odometría. Se estudian estos errores y las correcciones necesarias.

Más adelante se estudian los componentes hardware necesarios para la construcción del robot autónomo. Se analizan todas las piezas individualmente y se explica qué función realizará cada una. Seguidamente también se explica el diseño del software (tanto en el robot como en la estación de tierra) así como su implementación y sus funcionalidades. El software se separa en pequeñas piezas para que sea más modular y en el documento se explican cada una de esas piezas y sus funcionalidades.

Por último se muestran los resultados obtenidos después del desarrollo del sistema. Se ha diseñado una serie de test y se ha analizado el resultado de cada uno.

Title: Indoor and Outdoor Rover with Simultaneous Localization And Mapping (SLAM)

Author: Alex Albala Diaz

Director: Juan Lopez Rubio

Date: December 1, 2011

Overview

This work describes the design, development and implementation of a SLAM (Simultaneous Localization And Mapping) consists of two subsystems: low-cost autonomous robot and a ground station where telemetry data and information of the robot are displayed. The goal of a SLAM algorithm is to leave the robot in an unknown environment identify the environment through sensors and extract a map. The self-location of the robot is important in order to locate properly in the space all the sensor data. So the main objective of this project is to develop a low cost SLAM. In this work it will be used hardware for fast prototyping for the proof of concept and so can use it in future teaching.

It compares different SLAM algorithms and choosing the most suitable for this system. The chosen algorithm is studied in depth and implemented in the system. Also it is studied one of the common errors in all the terrestrial robots localization: the odometry errors. It is studied these errors and the needed corrections.

Then it is studied the hardware components for the construction of an autonomous robot. All parts are analysed individually and it is explained what task realizes each element. Then also is explained the design of the software (both the robot and the ground station) as well as its implementation and functionalities. The software is separated into small pieces to make it more modular and this document explains each of these parts and their functions.

Finally, it is shown the results obtained after the development of the system. It has designed a series of tests and analysed the results of each one.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

The main objective of this project is to develop the hardware and the software part of a system able to do SLAM tasks in in-door environments as well as out-door environments. The system contains a rover station and a ground station. This objective has to be accomplished by means of a scalable, autonomous and low-cost system.

Simultaneous Localization And Mapping (SLAM) techniques are in charge of knowing the position of a rover placed in an unknown environment. The objective is reached by means of mathematical models, operations and environment sensing using visual sensors like cameras or other ones which provides less information like ultrasonic sensors or infrared sensors which can detect a obstacle at a certain distance from them.

Localization in exterior environments is mostly done by means of a GPS system. In this project the localization must be done in interior environments, so, the GPS is not enough due to its poor signal power in this kind of environments. The localization has to be done using only sensors which have no dependence with other external systems or devices. Some SLAM techniques use a kind of infrared “beacon” in order to establish a reference to the rover. It is an approach like a very simple GPS system, one beacon (like the GPS satellites) and receivers (like GPS navigation receivers). In this project will be studied the way to avoid these kind of elements. The rover developed in this project will try to self-locate without external devices.

Rovers which implement SLAM algorithms are high-technology projects with a lot of features and powerful components. All of that is necessary because all the mathematical models and calculations of the SLAM techniques require high performance components. The sensors information is the most important part of these algorithms. These sensors are in charge of gather all the environment information in order to be able to represent a map of the environment surrounding the rover. Since this information is the base for drawing the map, the common sensors usually used are LASER sensors which have a high detection range, a high resolution and a fast sampling frequency.

Since the goal of this project is to develop a low cost SLAM rover, it is not possible to implement the rover with the hardware that fits and fulfills all these specifications. It is necessary to find low cost hardware able to do the SLAM tasks not with a perfect performance and resolution, but with features enough to achieve the project goals.

The use of low-cost hardware results in a reduction of the system performance. In this case always exists a trade-off between the cost and the final performance. The first problem is the computational complexity. The use of a central microcontroller board with a low computational power limits the implementation of the SLAM algorithm. It will be necessary the implementation of a tiny algorithm for develop SLAM tasks. Also the resolution of the sensors is not so good, it will be necessary to modify the algorithm in order to fit with all the other hardware parts. LASER sensors provides a lot of points with one unique scan, but infrared or ultrasonic sensors (non-expensive sensors) only provides one point for scan. It will be necessary place different sensors and combine the information between them in order to build a multiple point scan.

The communication between the two stations must be done with a wireless protocol. This project will study the best technology to accomplish the bandwidth and the coverage needs (outdoor and indoor communications).

Another problem that will be studied in this project is the movement errors in a 4 wheeled rover. The information about the position is one of the most important parameters for SLAM algorithms. There are different techniques to compute the actual position of the rover, one of them is to use IMU (Inertial Measurement Unit) elements which provide information about the inertial forces is suffering the rover. With different mathematical operations it can be computed the actual position of the rover, but there are two main problems: the high computational cost of this operations, and the high error propagation of the IMU. Also IMU elements use to be expensive and need other sensor inputs in order to correct the errors. Other way to compute the position of the rover is to implement odometry equations from wheel encoders. The encoders allow knowing the angular position of the wheel. The advantage of this system is the low cost of it. The main problem is that the resolution of this elements usually is not the best one and also the errors are accumulative (but with less magnitude than the IMU). In this project will be studied how to reduce the errors and how to unify all the sensor values in order to minimize all the errors.

Finally, this project is a first version of a SLAM system divided in two segments (rover and ground station). Another objective of this work is the possibility of continue this work with different teaching lines.

Summarizing all this points, the main goals of this project are:

- Develop a first version of a SLAM system composed of a ground station and a rover
- Autonomous rover with low power consumption
- Multiple environments such indoor and outdoor
- Wireless communications with good coverage in indoor and outdoor environments
- Low-cost platform

The rest of the document is organized as follows:

- Chapter 2: A first study of other systems and SLAM algorithms.
- Chapter 3: Odometry equations and correction
- Chapter 4: The hardware part of the system
- Chapter 5: The software part of the system
- Chapter 6: The evaluation of the work
- Chapter 7: Conclusions and future work

CHAPTER 2. PREVIOUS WORK

2.1. Introduction

Simultaneous localization and mapping (SLAM) has been an active research area in robotics for the last five years. The primary focus has been to model an indoor environment for Unmanned Ground Vehicles (UGV). In this chapter will be studied different hardware and software references in order to acquire a technical background before starting the development of the work.

First will be studied the robotic vacuum cleaners as a reference of the navigation algorithms and low-cost hardware for implement movement guidelines. Next, the most typical SLAM algorithm as well as the particular algorithm selected.

2.2. Vacuum cleaners

There are several vacuum cleaners in the market. These vacuum systems can be used as a reference to this work because they use low cost hardware for implementing auto-navigation guidelines inside a room. The systems which implement these robot cleaners can be studied and adapted to the rover. Now will be studied one of the most important robot vacuum system, which is the Roomba.

2.2.1. Hardware

The Roomba system is shown in the figure 2.1 and the visible elements are:

- Plastic bumpers which activate mechanical sensors (object sensors)
- Infrared wall sensor which allows following a wall like a reference
- Infrared receiver for localization goals
- Infrared cliff sensors for avoid cliffs in the terrain

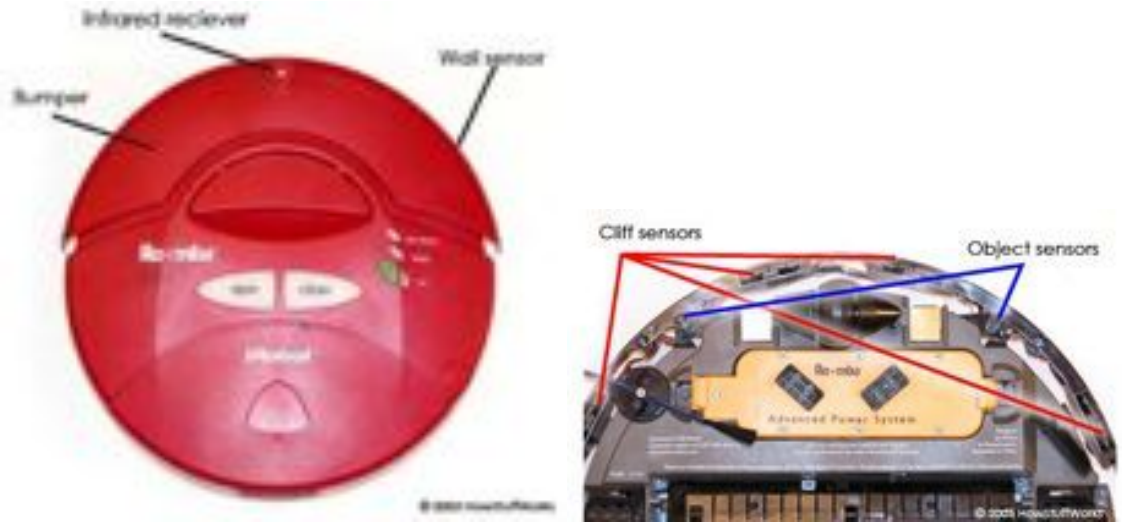


Figure 2.1: Roomba sensors

When Roomba knocks into something, object sensors are activated by the bumpers, and it can correct its position to avoid this object. Also Roomba is able to avoid cliffs by means of the cliff sensors which are sending constantly an infrared signal to the floor, if the signal is not bouncing back it means that there is a cliff, so, Roomba can go back and head into another way. Roomba is equipped with an infrared receiver

which allows the user place virtual walls, or allows Roomba going to the charger when the battery is empty. Finally, the wall sensor is used to follow very close walls or objects without touching them in order to cover all the floor surface.

This sensing method can be used in this project in order to fulfill the needs of the mapping part of the rover. All the sensing and navigation can be done with infrared sensors, LASER sensors are more accurate and faster than infrared ones, but are too expensive for a low-cost system such as the implemented in this project.

2.2.2. Software

Roomba uses iRobot's AWARE^{TM} Robotic Intelligence System to make many decisions for itself. There is no much information about this software, but according to iRobot specifications, Roomba can take about 67 decisions per second which is about 1 decision every 15 milliseconds.

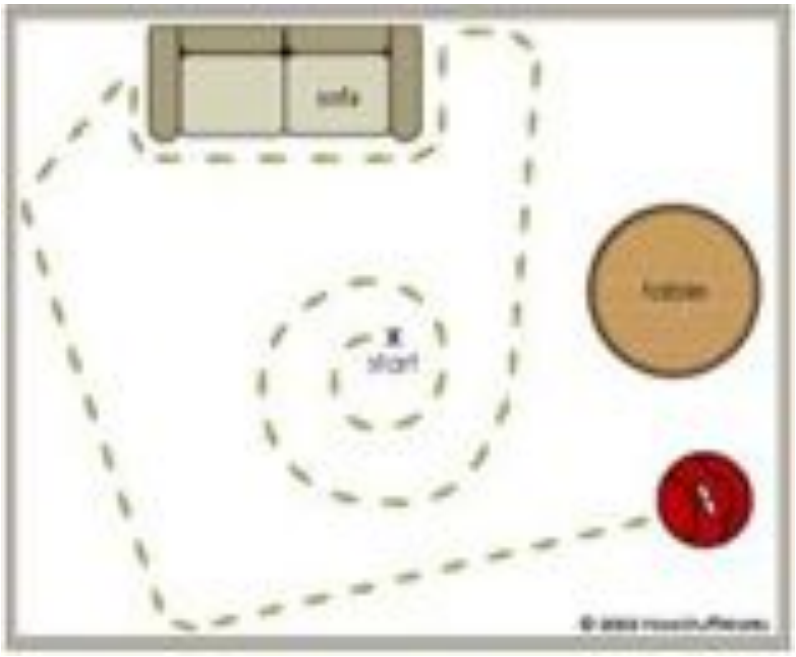


Figure 2.2: Roomba navigation

In the figure 2.2 is shown a normal Roomba path in a square perimeter room with two obstacles. Roomba starts cleaning in an outward-moving spiral and then heads for the perimeter of the room. Once it hits an obstacle, it believes it has reached the perimeter of the room. It then cleans along the "perimeter" until it hits another obstacle, at which point it cleans around it, finds a clear path and proceeds to traverse the room between objects like walls and furniture until the allotted cleaning time is up. The idea is that if it cleans for a certain amount of time, it will cover the whole floor.

The main difference between these kind of systems and our project is the self-localization part. The vacuum cleaners only need to know if there is an obstacle or not, they only implement navigation algorithms. The SLAM rover must have knowledge of its self-position apart from all the obstacles position in the room. Roomba uses infrared beacon to go back to the station to recharge the battery or empty the dust compartment, so, it is not necessary for them to implement localization algorithms, only with navigation guidelines it can do its tasks.

In our project, as it is necessary to locate the vehicle in the environment it will be to added some hardware to get the current position or at least the distance travelled to keep tracking the position. In a first study a motor with encoders seems to be the best option from a low-cost rover point of view, but is necessary to take into account the uncertainty of this element.

2.3. SLAM

The system implemented has to be able to locate itself and detect the main features of the environment. The rover task can be divided in two subtasks: self-localization and mapping. For the self-localization tasks it is necessary to apply different calculations for allow the rover to know its position based on the sensors. The mapping part gathers the information of the different sensors to compute a map of the environment. The mapping algorithm is useless without the localization part so there are different techniques to combine both parts. The resulting algorithm is called a SLAM (Simultaneous Localization And Mapping) algorithm.

SLAM is a current problem in robotics. SLAM tries to solve the existing problems of placing a rover in an unknown position and environment. The autonomous system has to be able to do a map of the environment using mathematical and physical models, different sensors (or different rovers) and actuators.

The main difficulties of SLAM are:

- Sensors error: The accuracy of a sensor is not perfect, and low-cost ones is even worse. This problem can cause detection errors and even position errors which are more poisoning because they are accumulative errors.
- Unexpected elements: More complexity in the environment means a more complex design of the software, even sometimes is not possible to detect some kind of figures and geometries due to the electronics limitations (resolution, systematic errors...)
- Mathematical modeling: It is necessary to represent all the physical and mechanical environment in different mathematical models (one model for the rover odometry, other model for the terrain, other model for the sensors etc.). The needed mathematical toolbox is complex.
- Computational complexity: All the elements enumered above result in a complex software system with several mathematical operations and conversions which have to be deployed in an embedded system with low computational power and memory.

SLAM techniques are mostly used with high resolution sensors like LASER ones which can scan a large range of the environment (i.e. 240^{o} and 5.4 meters of range) with a high resolution in few milliseconds. Since this project consists on a low-cost rover, the sensor will not be a LASER at all. The sensors will be used will not have so good resolution, range, delay and even accuracy, these characteristics are a big constraint for developing a good SLAM algorithm.

There are different SLAM algorithms in the investigation field. The major part of this algorithms are based in probabilistic models which contain information about how much probable is the position of the rover based on the actual calculated position and the sensor data. Also these algorithms usually are based in complex filters like Kalman filter which allows analyze input data over the time (with errors) to compute values which are closer to the reality. Some of the algorithms are FastSLAM and FastSLAM2.0 (see [1, 2]) as an efficient and robust solution of SLAM algorithms and its improved version, L-SLAM (see [3]) which is a low dimension FastSLAM algorithm, CV-SLAM (see [4]) which is an algorithm with a very high accuracy with a single camera input, Cuik-SLAM (see [5]) which is a SLAM algorithm based on kinematics instead of probabilistic models, CoreSLAM (also called tinySLAM) (see [6]) which is a tiny SLAM algorithm implemented in about 200 lines of C code. In this work will be implemented the CoreSLAM algorithm due to its tiny size and its simplicity comparing with the other algorithms.

2.3.1. CoreSLAM

Core SLAM is a SLAM algorithm implemented in less than 200 lines of code. This algorithm were designed to deal with small environments instead of long corridors (like most of the SLAM algorithms). The advantage of this feature is that can be applied the loop closing correction for reposition the rover in the space in order to suppress all the different systematic and non-systematic errors. The Core SLAM is designed as one way to combine LASER data filtered with localization information for mapping purposes.

This algorithm is divided in two main blocks: the map update function and the close loop function

2.3.1.1. Map Update

The map is built by using grey-level scale for determine the obstacles in the environment. The gray-level scale represents the likelihood of an object. White color means no object, black color means there is an

object with a 100% of probability. The map update function is based in attracting holes. For each obstacle detected, the algorithm draws a function with a hole whose lower point is at the position of the obstacle (see figure 2.3).

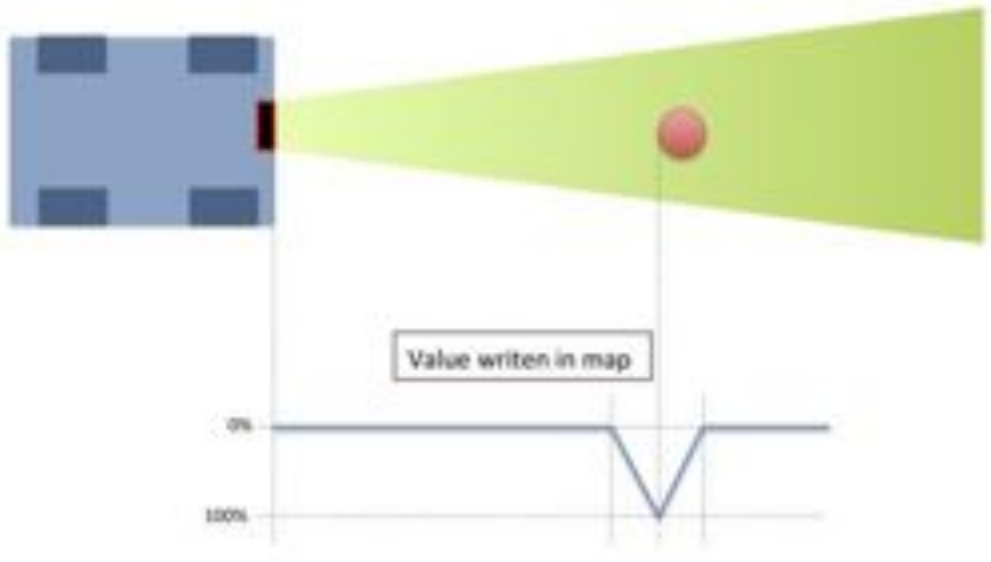


Figure 2.3: Attracting holes

The integration of the values into the map is implemented through an $\alpha\beta$ filter (a low past filter ideal for data smoothing, see the figure 2.4). The output is printed in yellow and the input is printed in blue, the high frequency data of the blue line (sudden changes) is smoothed and results in a line data with lower changes. The result of the rover scans are combined with the map over the time with this filter. If there is an object, all the scans will coincide at the point of impact, thus the values over the time will not change and will pass the filter. If there is an error at the sensor, the following scans will not coincide with this value, so the value over the time will change and this change means a high frequency which will not pass the filter.

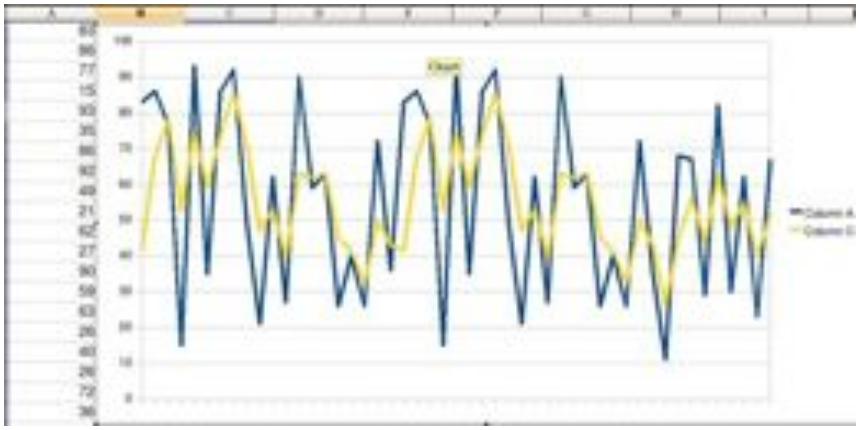


Figure 2.4: $\alpha\beta$ filter example. $\alpha = 0.5$ $\beta = 0.1$. Input in blue, output in yellow

The map update function uses an algorithm to draw the rays of the sensors which are detecting obstacles. These rays (or lines) are computed by means of a Bresenham algorithm (see [7]) in order to draw them in the map with a low computational cost. In the figure 2.5 is shown an example of a Bresenham algorithm result.

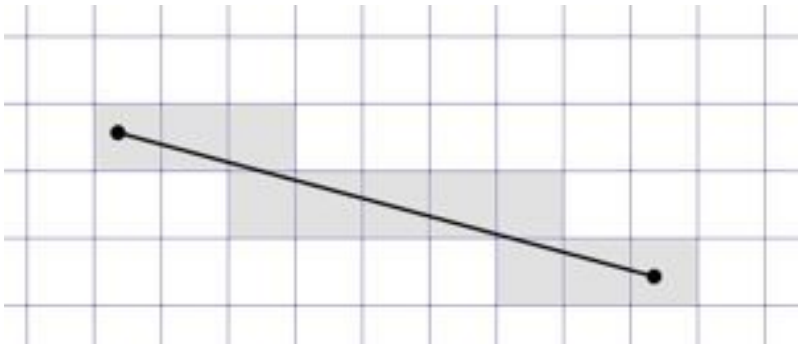


Figure 2.5: Example Bresenham algorithm

The map is a matrix which contains the value of the probability of an object in each cell. There is a trade-off between the resolution of the map (the higher the resolution, the higher the accuracy in the final map) and the memory used. The algorithms explained in this section are applied over the time in each sensor scan. Suppose two consecutive sensor scans like shows the figure 2.6.

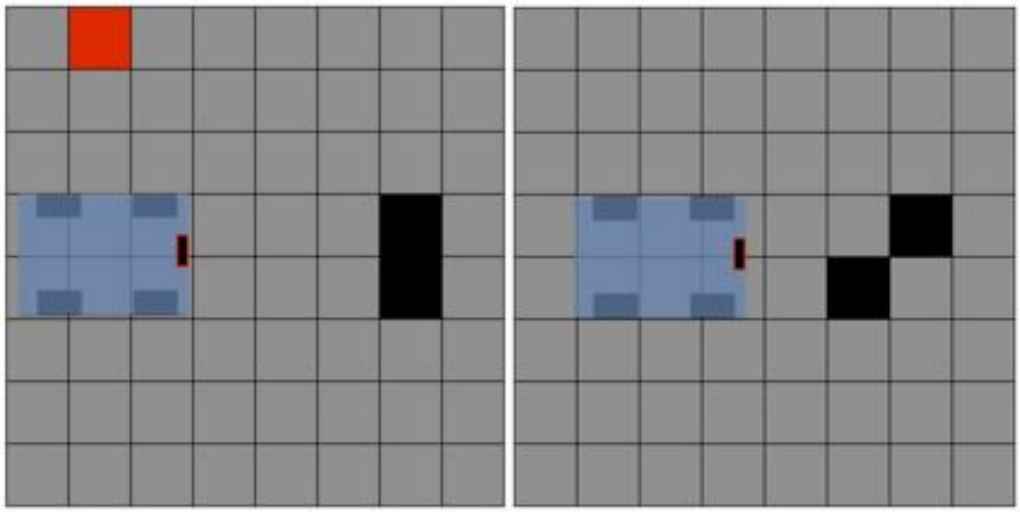


Figure 2.6: Scan 1 (left) and scan 2 (right) matrix

The red pixel in the first scan represents a sensor error, which detects and obstacle when there is not at all. The black pixels in the two scans are the correct detected objects. Notice that in the second scan the rover has travelled some distance and the sensors values can vary their values. For this reason is applied the attracting hole method and the $\alpha\beta$ low pass filter. Applying these algorithms and the Bresenham algorithm the resulting map will be a map where the error pixel does not have so magnitude as the correct values, because the correct ones are present in the two scans. The rays projected by the sensors are printed like 0% of probability until the impact point of the sensor. Notice that before and after the impact point the probability is not 0% due to the attracting hole method. This method results in a cloud of points in which the probability is much higher than the rest of the surrounding points. The higher the number of scans (iterations) the higher the probability difference. Also notice that the error pixel still has a probability but with more scans will decrease until a very low value.

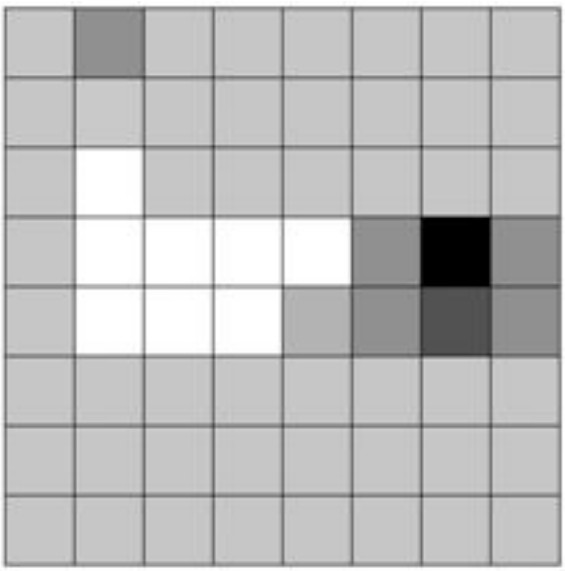


Figure 2.7: Resulting map from two scans

An example of the resulting map combining LASER sensor and odometry information is shown in the figure 2.8.

Figure 2.8: Core SLAM Result

2.3.1.2. Closing the loop

In many cases the SLAM algorithm has a final stage in which the actual calculated position of the rover is corrected for achieve the real position. The self localization of the rover is calculated by means of odometry equations which collects and transforms the encoder data into position information. In CoreSLAM algorithm the loop is closed combining the actual global map with the value of the scanning sensors for determine the correct actual position of the rover.

The calculation consists in a simple sum of all the map values at all the impact points of the scan relative to the position of the rover. The resulting value is called distance and represents the difference between the actual map and the actual value of the sensors scan. The position of the rover is changing randomly to evaluate the probability of each one in order to compare all the surrounding positions to find the best one. Summarizing, it is a method which returns the most probable position for the rover. This method is done comparing several random positions around the actual calculated position following the MonteCarlo algorithm (see [8]) which follows these steps:

1. Calculates the distance value for the actual calculated position
2. Calculates a new position with randomizing the position variables (starting from the actual position of the rover)
3. Calculates the distance of the new position
4. If the new distance is minor than the original distance, this new position is saved as the actual position
5. Repeat from 2 to 4 until a number of iterations is reached (specified by the user). In this work the value used is 10000 iterations.

2.4. Conclusions

One constraint of the CoreSLAM in the project system is the hardware required. The algorithm was designed as one way to combine LASER data filtered with odometry information for mapping purposes. This project is focused on a low-cost rover, so, it is not possible to develop a system based on LASER sensors, therefore the rover will work with infrared sensors. For this reason it will be necessary to modify the algorithm in order to adapts it to the new hardware elements.

The main problem of this algorithm in this work is the uncertainty. This algorithm is not implemented for working with low cost sensors. With the 4 sensors deployed in the rover, the number of points is about 60. The original algorithm is developed with a LASER sensor (Hokuyo URG-04LX) which has a resolution of 1024 angles (360 / 1024 is 0.36º resolution approximately). However the measuring area is 240º from 60 mm to 4095 mm. The impact points are 1024 in 360º, thus in the active area which is 240º the useful impact points are 682 points (much greater than 60). This constraint makes impossible the use of the closing loop feature, because this method needs a lot of impact points for returning a accurate position.

For this reason it is necessary to reduce as much as possible the errors. One element which produces error is the odometry calculations. This calculations must be improved and in this work will be studied how can be corrected the odometry errors.

CHAPTER 3. ODOMETRY

3.1. Introduction

Odometry is defined as the use of data from different sensors to calculate the position (of the rover) over the time. It is important to notice that the value of the position is an estimation of the real one, but due to the error propagation, the more the time the rover is traveling, the higher is the error in the position calculation. Odometry equations can be extracted from different sensors like Inertial Measurement Units, accelerometers, or even motor or wheel encoders.

3.2. Implementation

The sensors used to extract odometry information are incremental encoders. The incremental encoders (see section 4.6.1.) are elements which allows knowing information about the wheels position. It is not possible to know the absolute position of the wheel but know how much the wheel has turned, thus, the distance traveled of this wheel. The encoder output consists on changing levels (ticks) which every output change implies a new position of the wheel. The distance traveled by the rover following a straight line and depending on the encoder ticks is:

$$p[n] = p[n-1] + \triangle P \tag{3.1}$$

$$\triangle P = \frac{wheelPerimeter}{totalTicks} \tag{3.2}$$

where $p[n]$ is the rover 1-dimensional position in the sample n.

For the turning angles there are two ways to make a turn in a 4 wheel rover:

- Braking one side of the rover and moving the other side: the rover will turn over the locked side (see figure 3.1).
- Moving one side in one direction and the other side in the opposite direction: the rover will turn over the center of gravity (see figure 3.2).

Braking one side of the rover has the advantage of non-depending of the center of gravity of the rover, but it has two big disadvantages: the rover needs a lot of space for make a turn and the stopped wheel (front-left in if the turn is to the left) has an unknown contact point of the floor, so in the calculation is added this error.

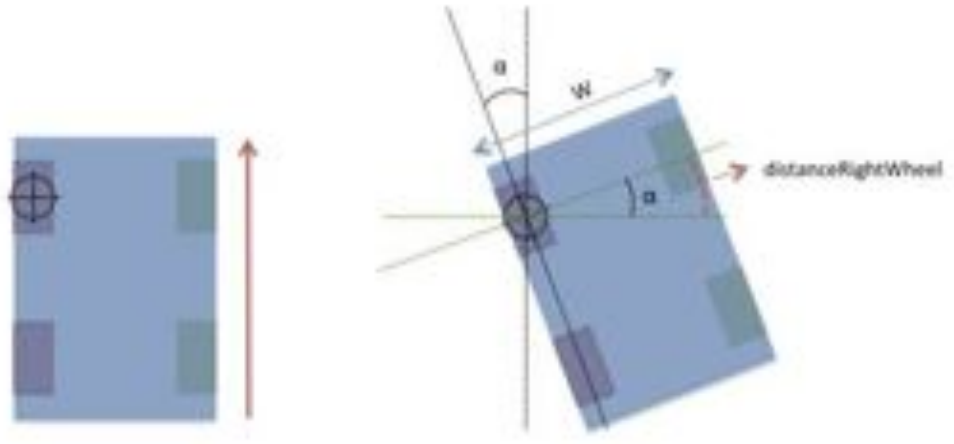


Figure 3.1: Robot turn 1

Since one side is stopped, the drive wheel in the other side will describe an arc with radius the distance between the contact point of the wheels (W) and center in the stopped wheel. The turned angle is calculated using the arc equation:

$$\alpha_{(rad)} = \frac{distanceRightWheel}{W} \tag{3.3}$$

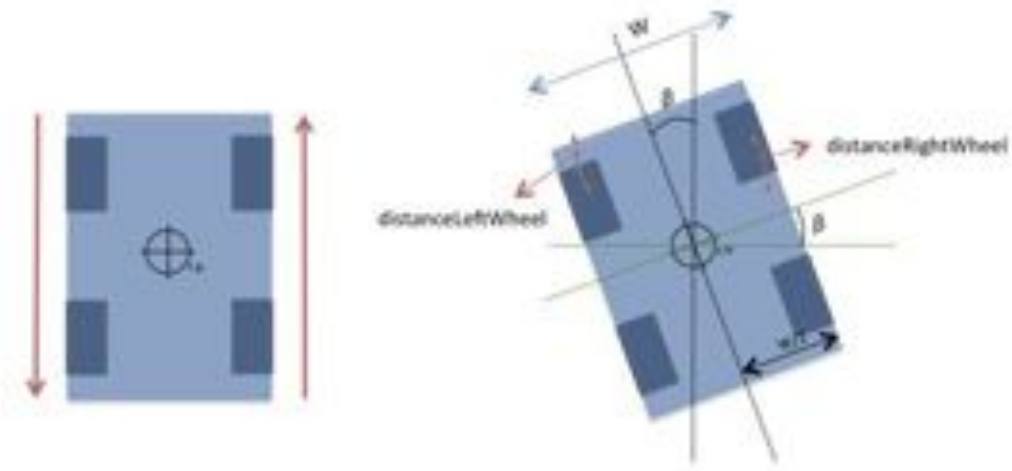


Figure 3.2: Robot turn 2

On the other side, turning the rover over its center of gravity, the wheels (all of them) will describe an arc with radius W/2 (the center of gravity is the geometrical center of the rover) and center in the center of gravity. The turned angle is obtained by:

$$\beta_{(rad)} = \frac{distanceRightWheel}{W/2} = \frac{distanceLeftWheel}{W/2} \tag{3.4}$$

The main disadvantage of this second method is the center of gravity not always will be the geometrical center of the rover, for this reason, since the two methods will introduce errors, it has been applied some mathematical corrections explained in the section 3.3.. The main advantages of the second method is that is faster than the first and the space needed to make a turn is less than the first method.

3.3. Correction

The accumulative errors makes impossible the self-localization of the rover in long-time (even short-time) missions.

The following example will prove the error propagation:

If the rover is programed for run in a square perimeter (four straight sides and 4 turns of 90 degrees) the end position will not be the same as the start position.

These errors in most of the cases are caused by:

- Unequal wheel diameters
- Effective wheel-base (the real distance between the two contact point of the drive wheels)
- Different wheel aligning
- Wheel spinning
- Irregular terrain

These errors can be divided in non-systematic (spinning, irregular terrain) and systematic errors (wheel diameter, wheel-base, wheel aligning). Non-systematic errors are difficult to predict, in fact is a big problem in current rover systems. Systematic errors (see figure 3.3) are specific to the vehicle and do not usually change during the travel or mission, moreover this errors are the main cause of the odometry deviation.

Therefore, the odometry calculations can be improved by means of the reduction of these errors. To reduce the systematic errors it is necessary to measure and correct them in the software level.

For measuring the errors it has been decided to develop the Bidirectional Square Path Experiment: UMBmark defined in [9].

3.4. Bidirectional Square Path Experiment: UMBmark

The experiment consists on running a square path with four turns. The summary of the test is as follows:

1. Measure position (absolute position) of the rover
2. Run the vehicle through a 4x4 m square path in clockwise direction:
   (a) Stop after each 4 m straight leg
   (b) Make a total of four 90º turns
   (c) Avoid slippage running the vehicle with low velocity
3. Measure the final absolute position
4. Compare the starting positions with the final position and take note of the value
5. Repeat from 1 to 4 four more times (a total of five times)
6. Repeat from 1 to 5 in counterclockwise direction

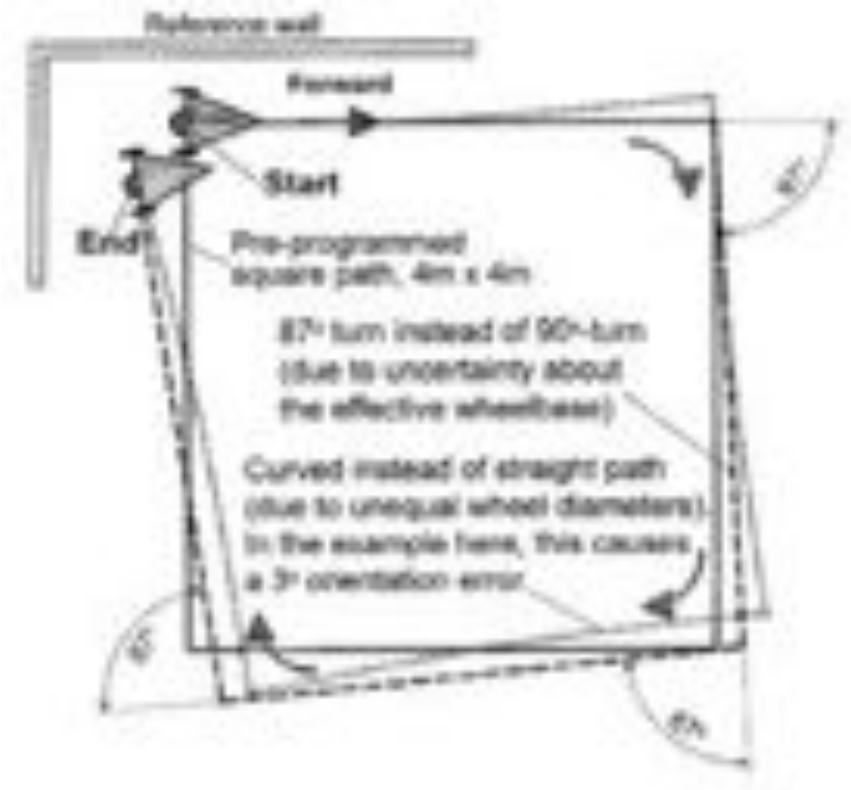


Figure 3.3: Odometry errors 1

The deviation errors of the rover can be divided in two types which both always appears:

- Type A errors: Orientation error that increases or reduces the total amount of rotation during the square path experiment in both CW and CCW direction. See the figure3.4.

- Type B errors: Orientation error that increases or reduces the total amount of rotation during the square path experiment in CW direction but reduces or increases the total amount of rotation in CCW direction. See the figure3.5.

The type A errors are caused by the effective wheel-base problem. The turning ratio of the rover is based on the distance between the drive wheels. This distance is measured taking into account the center point of the wheel width but actually the contact point of the wheels are not this center points. This error causes the rover to turn more (or less) than the required angle.

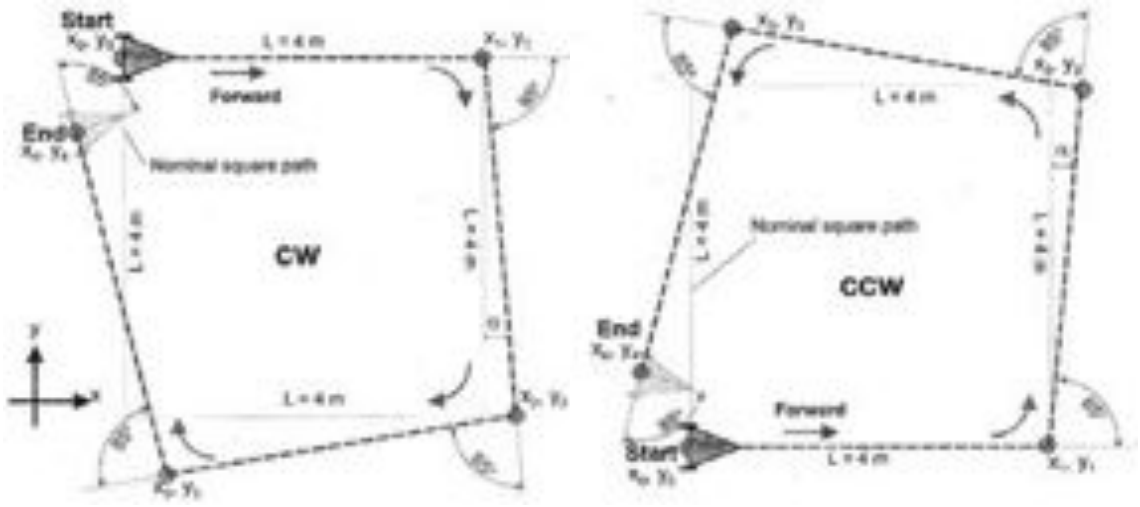


Figure 3.4: Type A errors

The type B errors are caused by the difference in the diameters of the wheels. There is two ways to solve this problem, one way is to make calculations and correct the position of the rover in real time. Other way is to assure that the speed of the wheels is tuned to compensate the diameters error and make the robot go into a perfect straight line.

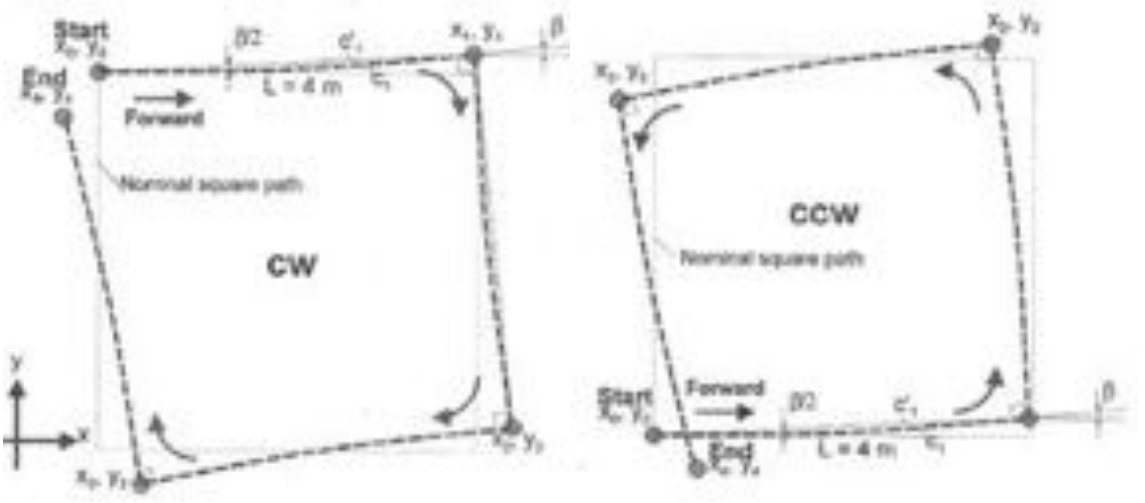


Figure 3.5: Type B errors

Now is described the correction of the type A errors:

In clockwise direction:

$$x_1 = x_0 + L\,; \qquad y_1 = y_0 \tag{3.5}$$

$$x_2 = x_1 + L\sin\alpha \approx L + L\alpha\,; \qquad y_2 = y_1 - L\cos\alpha \approx -L \tag{3.6}$$

$$x_3 = x_2 - L\cos 2\alpha \approx L\alpha\,; \qquad y_3 = y_2 - L\sin 2\alpha \approx -L - 2L\alpha \tag{3.7}$$

$$x_4 = x_3 - L\sin 3\alpha \approx -2L\alpha\,; \qquad y_4 = y_3 + L\cos 3\alpha \approx -2L\alpha \tag{3.8}$$

where α is the amount of erroneous rotation in each nominal 90º turn.
In counterclockwise direction:

$$x_1 = x_0 + L\,; \qquad y_1 = y_0 \tag{3.9}$$

$$x_2 = x_1 + L\sin\alpha \approx L + L\alpha\,; \qquad y_2 = y_1 + L\cos\alpha \approx L \tag{3.10}$$

$$x_3 = x_2 - L\cos 2\alpha \approx L\alpha\,; \qquad y_3 = y_2 + L\sin 2\alpha \approx -L + 2L\alpha \tag{3.11}$$

$$x_4 = x_3 - L\sin 3\alpha \approx -2L\alpha\,; \qquad y_4 = y_3 - L\cos 3\alpha \approx 2L\alpha \tag{3.12}$$

Combining the results of the clockwise and counterclockwise equations:

$$x_{CW} + x_{CCW} = -4L\alpha\,; \qquad y_{CW} - y_{CCW} = -4L\alpha \tag{3.13}$$

so that

$$\alpha = \frac{x_{CW} + x_{CCW}}{-4L}\,; \qquad \alpha = \frac{y_{CW} - y_{CCW}}{-4L} \tag{3.14}$$

Using the well-known odometry equations in [10], the wheelbase b is inversely proportional to the actual amount of rotation:

$$\frac{b_{actual}}{90} = \frac{b_{nominal}}{90 - \alpha} \rightarrow b_{actual} = \frac{90}{90 - \alpha} b_{nominal} \tag{3.15}$$

The Type A errors are corrected applying the factor obtained in the equation 3.15.

3.4.1. Results

The table 3.1 shows the results of the experiment done with the rover developed in this work. The angle showed in the table is not used to make the corrections calculations, because the error of the angle is a direct consequence of the error in X and Y, so it does not provide additional information.

Type	#	X	Y	Angle
CW	1	445	1020	45º
CW	2	515	1300	55º
CW	3	30	1230	60º
CW	4	476	2020	88º
CW	5	654	-310	-5º
CCW	1	-1770	-510	70º
CCW	2	-2830	-615	88º
CCW	3	-1800	-140	80º
CCW	4	-2265	-100	85º
CCW	5	-1510	-240	80º

Table 3.1: UMBMark experiment results

For type B errors, it has been decided to calibrate the speed of the wheels in order to make the rover able to travel in a perfect straight line, so, the correction of type B errors is done by means of an empirical calibration.

For type A errors the correction is done using the equation 3.15. The procedure is as follows:

For the clockwise direction:

$$X_{c.g.CW} = \frac{1}{5}\sum_{i=1}^{5} X_{i.CW} = \frac{1}{5}(445+515+30+476+654) = 424mm \tag{3.16}$$

$$Y_{c.g.CW} = \frac{1}{5}\sum_{i=1}^{5} Y_{i.CW} = \frac{1}{5}(1020+1300+1230+2020-310) = 1052mm \tag{3.17}$$

The same for the counterclockwise direction:

$$X_{c.g.CCW} = \frac{1}{5}\sum_{i=1}^{5} X_{i.CCW} = -2035mm \tag{3.18}$$

$$Y_{c.g.CCW} = \frac{1}{5}\sum_{i=1}^{5} Y_{i.CCW} = -321mm \tag{3.19}$$

Now can be applied equations 3.14 and 3.15 and is obtained a value of $0.17 < \alpha < 0.20\,[rad]$ which is $9.74 < \alpha < 11.46\,[deg]$ (the two values are obtained using CW and CCW experiment). For these alpha values is obtained using 3.15 a correction factor f which is the factor will be applied to the turning angles (a multiplicative factor): $1.12 < f < 1.14$.

CHAPTER 4. HARDWARE IMPLEMENTATION

4.1. Introduction

Before starting with all the hardware parts is necessary to enumerate the main objectives and features the rover must accomplish:

- Movement (software controlled): The rover has to be able to control its own movement.
- Environment sensing: This is the mapping component of the structure. It is in charge of taking measurements of the environment.
- Self-position acknowledgment: This is the localization component. It is in charge of notify the rover position in order to take decisions.
- Wireless: A wireless communication system is needed as well as the design of the protocol in order to interchange all the telemetry information between the two stations.
- Low power consumption: This is one of the main goals of this work. The consumption of the hardware elements have to be lower as possible.

The hardware scheme will be explained in this section is showed in the figure 4.1.

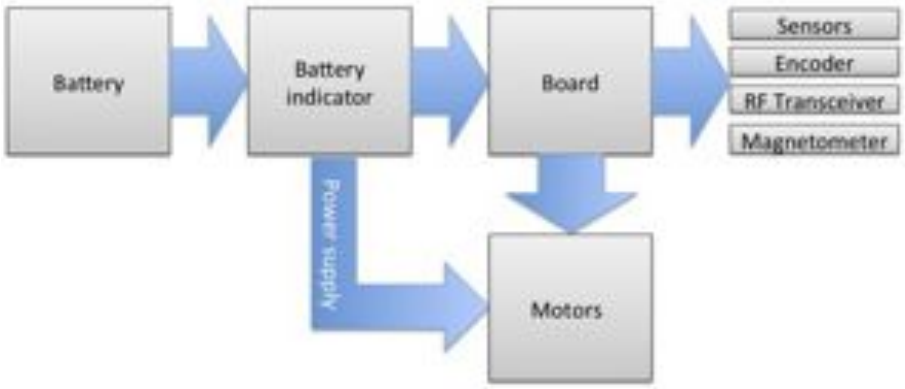


Figure 4.1: Hardware block diagram

The battery and the battery indicator form the power supply block (the battery indicator only notifies to the board the voltage and current levels, the power supply is not affected). The sensors and the transceiver are the payload controlled by the main board or micro controller. The motors are also controlled by the board but have to be supplied by the main power supply of the system.

4.2. Structure

The SLAM rover must be terrestrial, low-cost and with a minimum robustness level. Also it must be taken into account the dimensions for placing all the hardware pieces and the motors must support a wheel encoder kit. The selected kit is DFRobot 4WD Arduino-Compatible Platform w / Encoders from RobotShop.

The kit includes all the necessary elements for the deployment of the rover:

- Body structure
- 4 wheels
- 4 DC Motors
- 1 Encoder Kit (2 encoders)

- 1 Power switch

The specifications of the kit are:

- Wheel diameter: 65mm (Wheel perimeter: 204.203 mm)
- Dimensions
  - Length: 230mm
  - Width: 185mm
  - High: 110mm
- Weight: 614g
- Maximum load: 800g

Figure 4.2: Structure kit

4.3. MicroController

FEZ Panda is a tiny board running Microsoft .NET Micro Framework. Using an Arduino-form-factor (see the figure 4.3), it can be used just about any Arduino shield (see [11]) . In addition, many libraries are already included like FAT file system, networking, threading, USB Client, UART, SPI, I2C, GPIO, PWM, ADC, DAC and many more.

Figure 4.3: FEZ Panda

This board has been selected due to its low cost while offering useful features like multi-threading. For the development of this project will be necessary a powerful board with threading utilities to optimize the performance of the software. Other key feature is the hardware debugging which allows the user to set breakpoints and examine variables. A summary of the key features is:

- 60 IO pins
- 4 TTL UARTs
- User flash 148Kb
- User RAM 64Kb
- 72MHz 32-bit ARM processor
- Multi threading
- Built on Microsoft micro framework 4.1
- Hardware debugging
- High level programming (C#) which is higher than C/C++ so it is useful to teaching or to make fast prototypes.

The extra IO pins (39 IOs comparing with Arduino Deumilanove) have been welded with accessible female header.

4.4. Motors and Driver

The kit includes 4 DC Motors. For controlling the motors it is necessary a DC motor driver. The selected one is the ZX-DCM2 DC Motor driver with 2 channels. The specifications are:

- 2 motor channels with bi-color LED indicator
- L293D H-Bridge driver Maximum load current 600 mA
- Suitable supply voltage +4.8 to 30Vdc. Separate 5V control voltage and Motor supply voltage.
- INEX Standard 3-pin PCB connector.

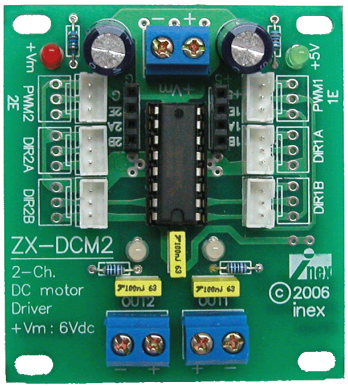


Figure 4.4: ZX-DCM2

The motor driver only has two outputs (then, it can only control 2 different motors), so it will be connected the two motors of the right side in one output and the 2 motors of the other side will be connected in the other output. The DCMotorDriver has 6 inputs (3 inputs for each motor output). These inputs are:

- 1 PWM: Controls the power of the motor (from 0 to 100)

- 2 Digital inputs for control the motors:
  - Forward: True - False
  - Backward: False-True
  - Brake: True-True / False-False

The motor driver will drive current to the motors in order to move them with certain velocity. The figure 4.5 shows the characterization of the motors (the four at the same time) comparing the speed with the current consumption. This characterization has been made with an external 9V power supply unit.

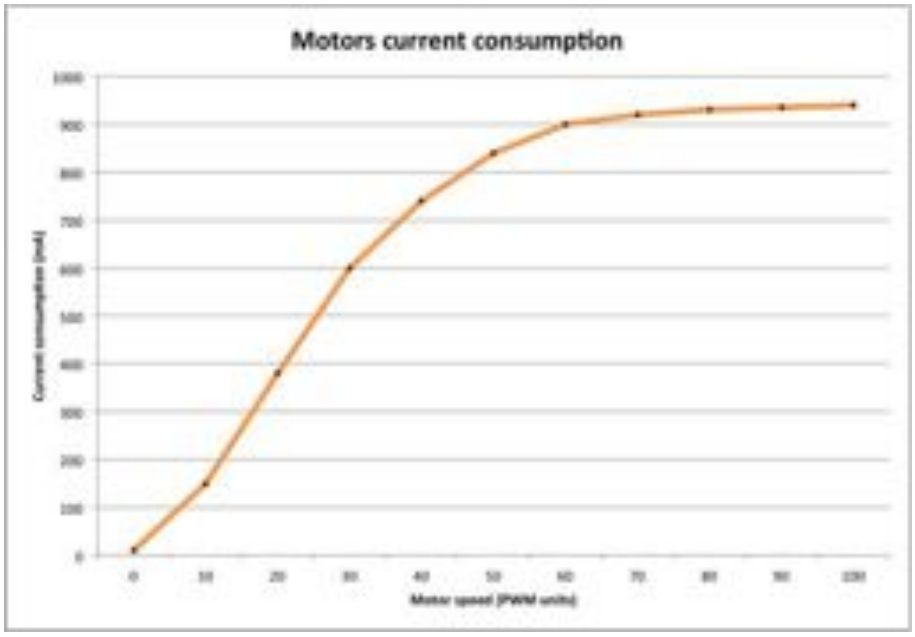


Figure 4.5: Motor characterization curve

4.5. Infrared sensors

Figure 4.6: Infrared SHARP sensor

The mapping part of the Rover needs sensors which collect the environment information and send it to the board. The sensors selected are 4 small, low power and low cost infrared sensors:

- 2xSHARP GP2D120
- 2xSHARP GP2Y0A21

4.5.1. Working principle

The selected sensors are reflective sensors, they are composed by two elements: a LED which emits infrared light, and a photodiode which receives infrared light.

The working principle of the two sensors is the same. The distance is measured by means of the time delay between the LED sends one frame and the photodiode receives this frame. The two sensors have the same time response which is showed in the figure 4.7.

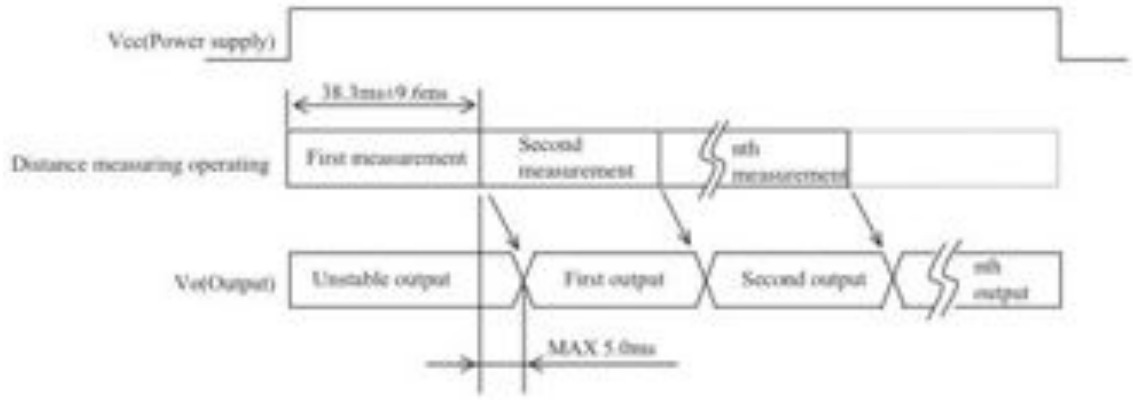


Figure 4.7: Sensor timing

4.5.2. SHARP GP2D120

This sensors has an effective range from 4 to 30 cm and analog voltage output. The figure 4.8 shows the output curve respect to the distance measured. Notice that the range of the sensor is delimited by the linear part of the inverse distance function.

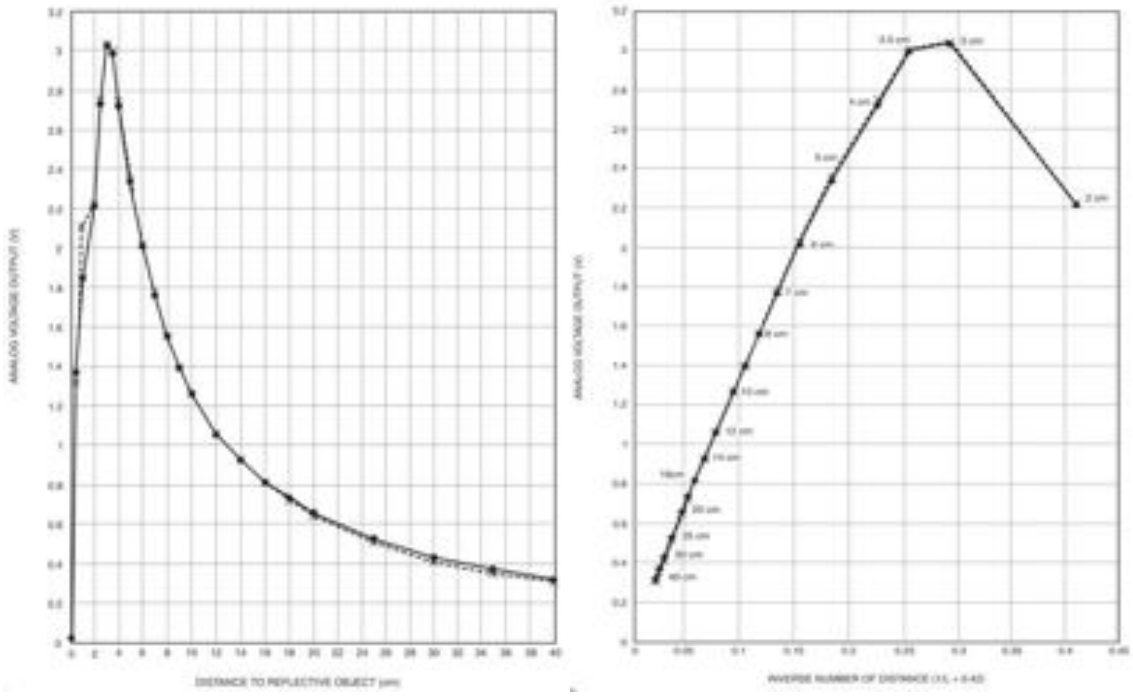


Figure 4.8: GP2D120 response

4.5.3. SHARP GP2Y0A21

This sensors has an effective range from 10 to 80 cm and analog voltage output. The figure 4.9 shows the output curve respect to the distance measured. As same as above, the range of the sensor is delimited by the linear part of the inverse distance function.

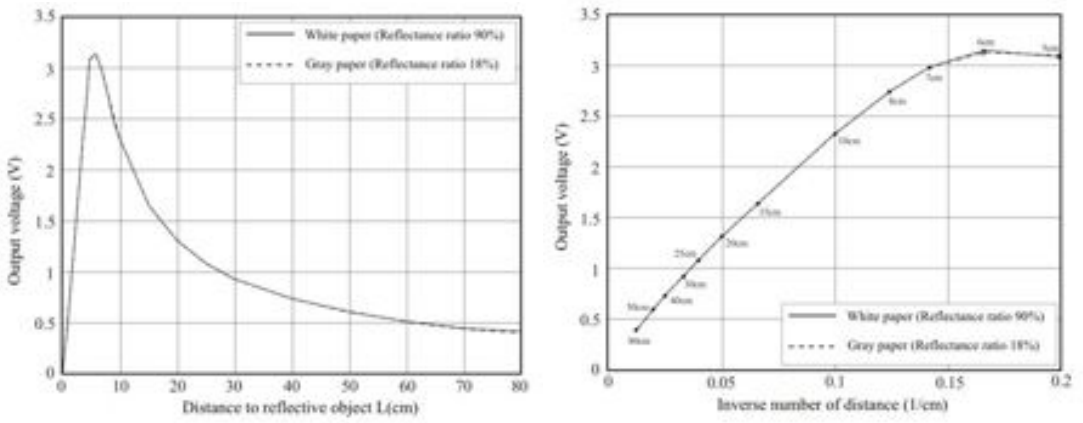


Figure 4.9: GP2Y0A21 response

4.6. Encoders

The encoders is one of the elements in charge of the self localization of the rover. An encoder is a digital sensor placed in the motors which allows knowing the actual position of the motor. There are two types of encoders:

- Incremental position encoders
- Absolute position encoders

The selected kit for developing the Rover includes two incremental position encoders.

4.6.1. Incremental encoders

Consists on three pieces: Photodiode, LED, and a plastic circular piece with transparent and opaque zones.

Figure 4.10: Plastic wheel encoder

The plastic piece is placed between the Photodiode and the LED and spins with the motor, when the light is received by the Photodiode, it means that the plastic piece is placed in such way the light is passing through a hole, on the other hand, if the photodiode is not receiving anything, the plastic piece is placed such a way the light is not passing at all. This results in a rectangular pulse output signal indicating when the light is passing throw the hole or not like shows the figure 4.11.

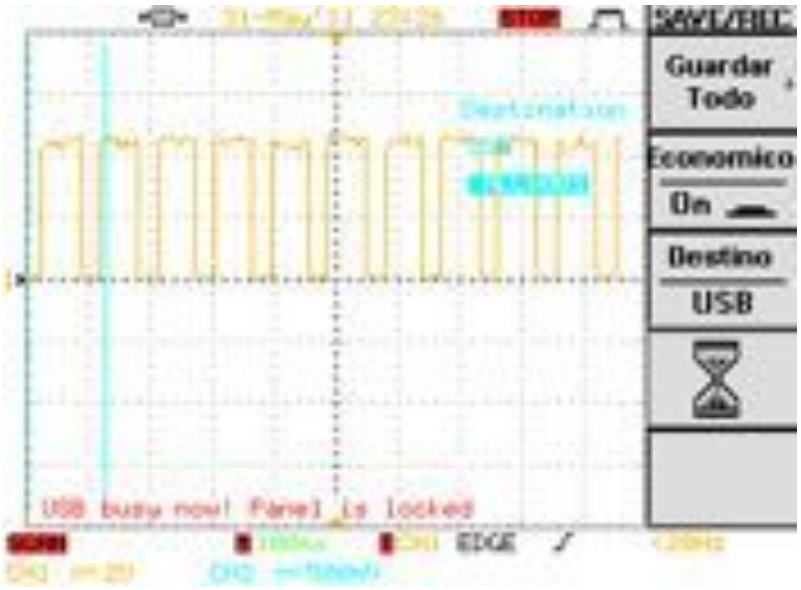


Figure 4.11: Encoder output

Every change in the signal output (from high to low or vice-versa) can be taken into account and it means that the encoder has change the position in one tick. If the encoder has 10 'holes', it means that one entire turn will output a square signal of 10 cycles: 10 transitions from HIGH to LOW and 10 transitions from LOW to HIGH, so a total of 20 transitions. It is important to count the two different transitions because the resolution is improved in a factor of 2. The resulting resolution is as follows:

$$\triangle\alpha = \frac{360º}{TotalTicks} = \frac{360º}{20} = 18º \tag{4.1}$$

Then, this encoder is able to detect minimum changes of 18 degrees of wheel rotation. The resolution in the distance traveled in straight line is:

$$\triangle d = \triangle\alpha_{(rad)} \cdot radius = 18º \frac{2\pi}{360º} 32.5mm \simeq 10.23mm \tag{4.2}$$

4.7. XBee modules

Figure 4.12: MaxStream XBee

XBee modules are hardware components which implement the IEEE 802.15.4 standard (low-rate wireless personal area network: LR-WPAN and is the ZigBee basis). XBee modules work with low supply current (45 mA in transmission and 50 mA in reception) and offers a transmission bandwidth of 250 kbps working in the S band (2.4 GHz).

The range data integrity is 30 m in indoor or urban environment and 100 m in outdoor line-of-sight environment. The output power is 0 dBm and the sensitivity in reception is -92 dBm.

4.8. Battery

Before studying the battery it is necessary to evaluate the overall current consumption of the system in order to choose the battery which fits the needs of the hardware. The measured current of all the elements is shown in the figure 4.13. The missing element is the DC Motors. The speed of the motors following a straight line is 23%, thus, like shows the figure 4.5 where the motors are characterized, the consumption at this speed will be approximately 450 mA.

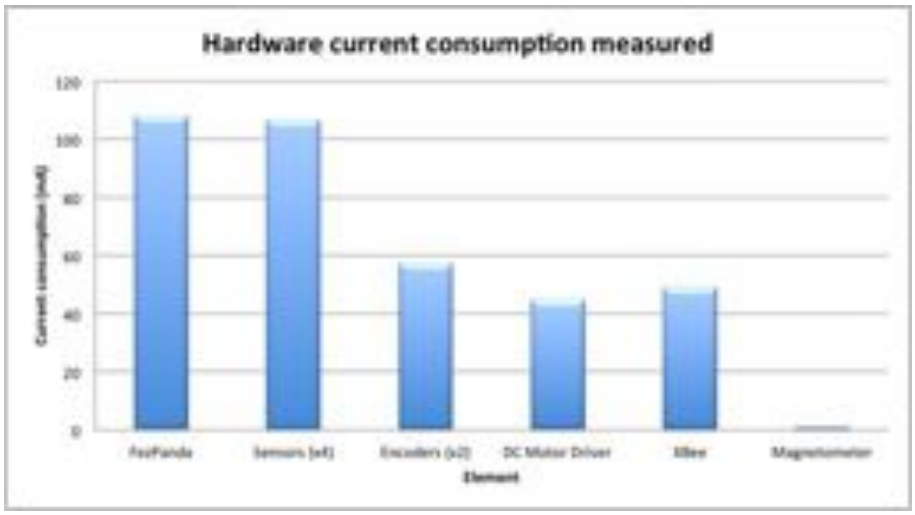


Figure 4.13: Hardware current consumption

The measured current compared with the theoretical current is shown in the figure 4.14.

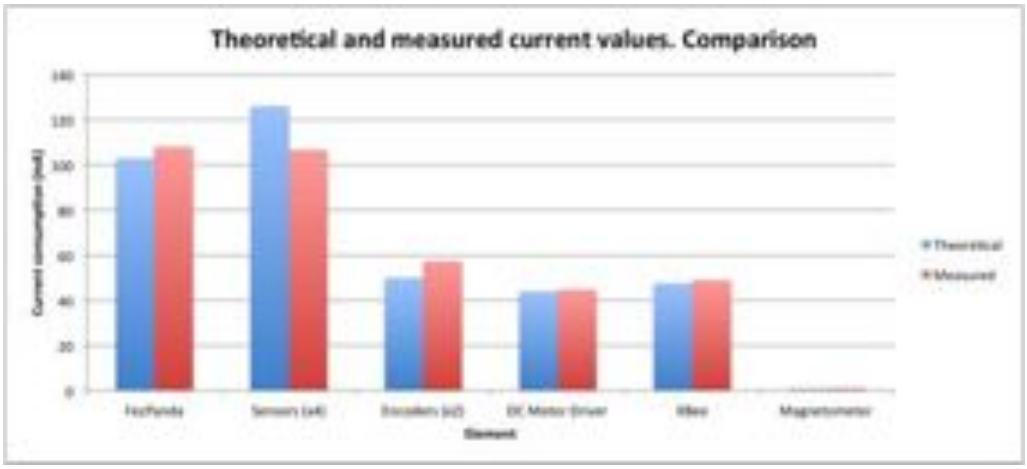


Figure 4.14: Theoretical and measured current consumption comparison

The overall current consumption of the system is around 371.4 mA theoretical, 365.8 mA is the sum of all the elements measured, and 354 mA is the total experimental measured. The variation in these values is due to the tolerance of the hardware, the activity/inactivity current consumption of the board, the changes in temperature and also the resolution of the amperimeter. The total consumption following a straight line will be about 800 mA (taking into account the 450 mA consumption of the DC Motors).

The battery must supply the Panda board and the motor driver. The input voltage of the motor driver is from +4.8Vdc to +30Vdc. The input voltage of the FEZ Panda is from +7Vdc to +12Vdc. The selected battery is a Fullymax with the following features:

- 11.1 V

- 4000 mAh
- 3SP1 (3 cells in series, 1 cell in parallel)
- Lithium-ion polymer battery

The absolute maximum of a Li-Po cell is 4.2V, thus the maximum voltage of the battery is $3 \cdot 4.2V = 12.6V$. The absolute minimum is 3.0V, so the minimum voltage is $3 \cdot 3.0V = 9.0V$. Now can be computed the battery life estimation in two cases: stopped mode (all the elements running but not the DC motors) and moving in a straight line mode.

In a stopped mode the consumption is about 354 mA (taking into account the total experimental measured), so, the battery life will be:

$$t_{hours} = \frac{capacity}{consumption} = \frac{4000mAh}{354} = 11.3h \Rightarrow 11h17m \tag{4.3}$$

Following a straight line the consumption is about 800 mA so the battery life will be:

$$t_{hours} = \frac{capacity}{consumption} = \frac{4000mAh}{800} = 5h \tag{4.4}$$

After the battery it has been placed a battery indicator in order to extract information about the supply current and voltage to the circuit. The battery indicator used has two analog outputs. One output yields a voltage from 0V to 3.3V depending on the input voltage (from 0V to 50V). The other output yields a voltage also from 0V to 3.3V depending on the input current (from 0A to 90A). So, the output of the sensor is: 63.69mV / Volt for the battery voltage and 36.60mV / Amp for the current consumption.

4.9. Installation

In the figure 4.15 is described the hardware system installation. Notice the DC Motor driver, the DC Motors and the encoders are below the “second floor” of the platform. The board FEZ Panda, the magnetometer, the battery, the battery indicator and the XBee module are placed in the top of the platform. Finally there are four sensors:

- Two short range sensors (GP2D120) placed in the left part of the rover, in the top of the platform and parallel to the structure limit. This sensors will be used to detect some kind of wall and use it for a movement reference
- Two large range sensors (GP2Y0A21). One is placed in front of the rover in order to sample all the front part of the system. The other is placed in the right side of the rover with a rotation angle of 45 degrees. This sensor combined with the front sensor will cover space enough to consider the front area analyzed.

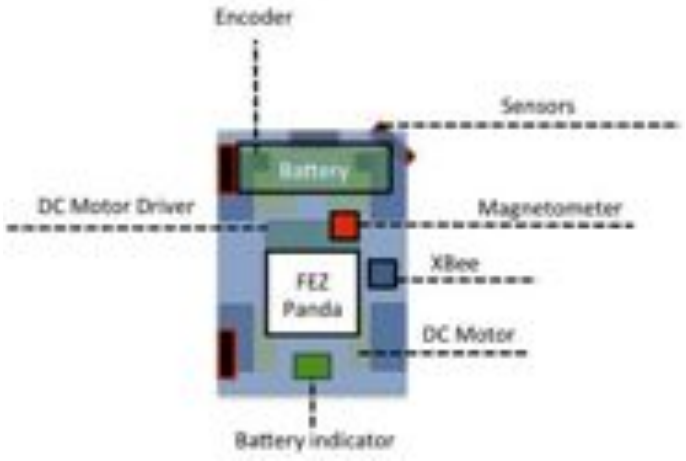


Figure 4.15: Installation scheme

4.10. General electronic scheme

The figure 4.16 shows the overall hardware connections of all the system.

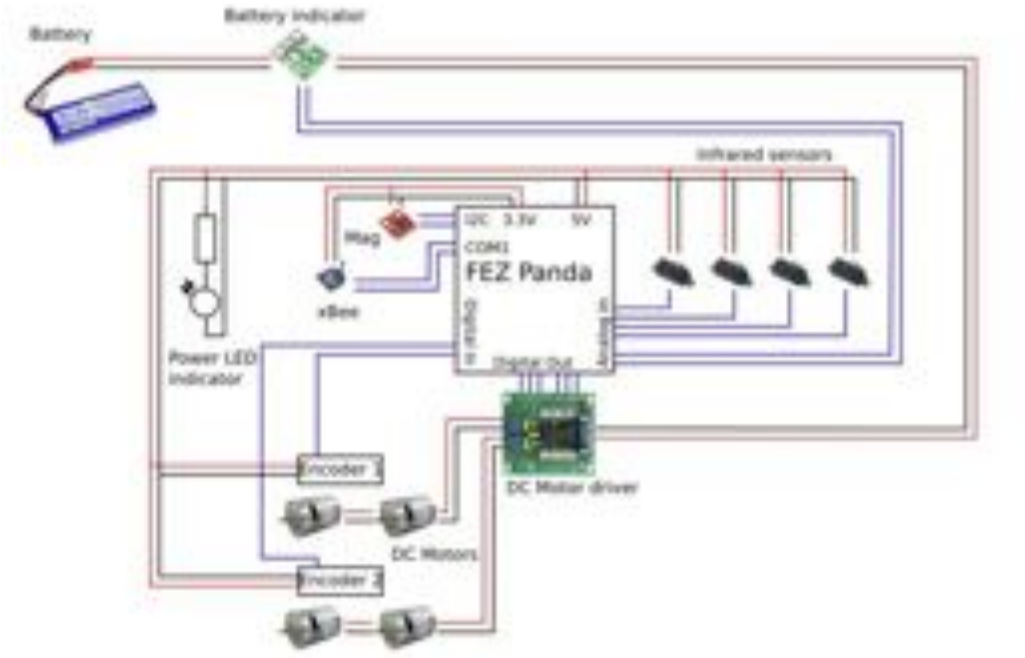


Figure 4.16: General scheme. Red and black = supply voltage; blue = control

CHAPTER 5. SOFTWARE IMPLEMENTATION

5.1. Introduction

The solution will be implemented in C# language with the framework .NET Micro Framework version 4.1. Micro Framework offers .NET support for resource limited devices. It is an object oriented language designed for create complex applications in a modular way. In order to take benefit of the Micro Framework scalability the code will be designed in several simple blocks interacting with each other if necessary.

5.2. Rover software

The software developed in the rover can be divided in three main groups: Core, Drivers and Communications. The figure 5.1 shows the distribution of these parts. The core is composed by the managers (which are in charge of manage and control de drivers) and an element called Engine (which contains all the logical part of the movement, access to the radio link and manage all the information provided by the managers).

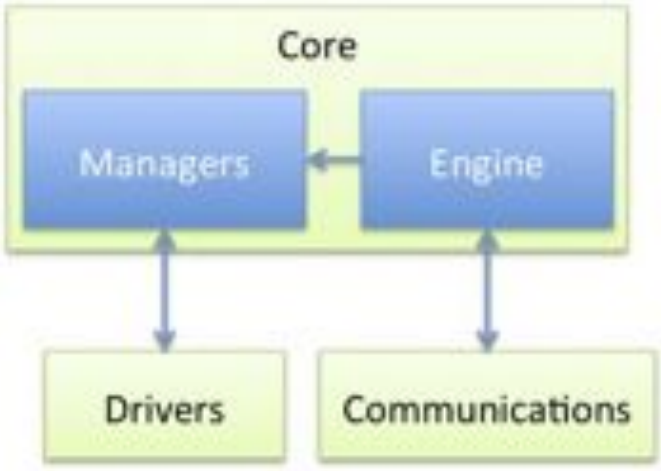


Figure 5.1: Rover architecture

5.2.1. Drivers

The drivers are in charge of provide to the core part an interface to interact with the sensors. The drivers must provide an abstraction layer to the upper layers of the software. With this modular implementation is obtained a reliable and scalable software. For this robot the needed drivers are:

- Infrared sensors driver
- Encoders driver
- Motors driver
- Magnetometer driver
- Battery indicator driver

5.2.1.1. Infrared sensors driver

The infrared sensor driver is called DistanceDetector. The driver is in charge of translate the output voltage of the analog sensor into a distance in centimeters. This conversion is achieved with the implementation of the linear part of the inverse curves of the infrared sensors (see figures 4.8 and 4.9). The figure 5.2 shows the

class diagram of the driver, the public method GetDistance_cm() returns the value of the distance detected in centimeters.

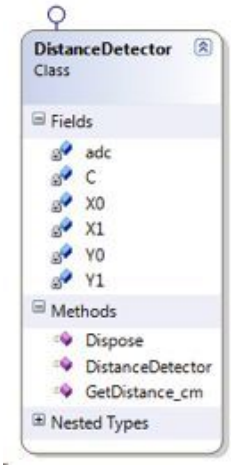


Figure 5.2: Distance detector driver

5.2.1.2. Encoders driver

The encoder driver is in charge of counting the encoder ticks and computing the distance traveled by the wheel associated to the encoder. The driver is composed by a thread which is executing a loop (Run method) which is counting the ticks with a pooling technique. This driver can be improved using interruptions but the FEZ Panda interruption pins when the motor is moved starts to detect a lot of interruptions even when the oscilloscope is showing a normal square wave. This is a hardware error and it is reported to GHI. The figure 5.3 shows the class diagram of the driver. The property distance_mm contains the distance traveled by the wheel. The driver provides the resetDistance() method to reset this property when needed.

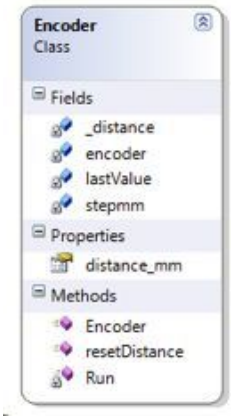


Figure 5.3: Encoder driver

5.2.1.3. Motors driver

The motor driver task is to communicate with the DC Motor Driver in order to control the motors. The figure 5.4 shows the class diagram of this driver. The provided methods are Move() and MoveRamp(). The method Move() moves the motors with a specified velocity. The method MoveRamp() moves gradually the motors starting with the initial speed until the specified speed is achieved. The slope of the ramp also can be specified.

Figure 5.4: Motor driver

5.2.1.4. Battery indicator driver

The battery indicator driver is called VoltageCurrentSensor. This driver is in charge of translate the analog outputs of the sensor into voltage and current values. The conversion is done by means of the linear transformations provided by the data-sheet. The figure 5.5 shows the class diagram of the driver. The provided methods are: getActualCurrent(), getActualVoltage(), which provides instant information of the actual values, getBatteryCapacity() and getEstimation() which provides the actual capacity of the battery and the estimated time left taking into account the capacity and the current consumption.

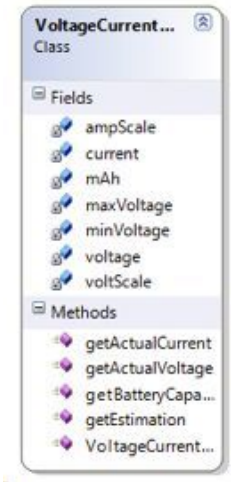


Figure 5.5: Battery indicator driver

5.2.2. Core

The core module contains all the logical part of the software. It is divided in 4 parts:

- Navigation manager
- Battery manager
- Contingency
- Engine

5.2.2.1. Battery manager

Battery manager is in charge of provide for the core part public methods to interact with the battery driver. It is composed by a VoltageCurrentSensor class (vcs) and a boolean property called lowBattery that indicates if the battery capacity is lower than 5%. The figure 5.6 shows the class diagram of the BatteryManager.

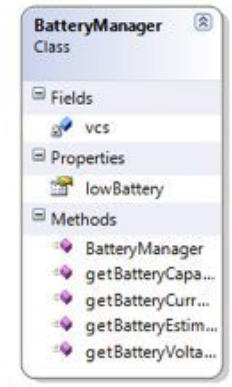


Figure 5.6: Battery manager class diagram

5.2.2.2. Navigation manager

The navigation manager is in charge of all the movements of the rover. The navigation control contains two encoder drivers, one Motor driver, one magnetometer driver and one Contingency element. This core element provides different features:

- Manual movement (from the ground station, specifying normal speed and turning speed)
- Normal movement (not from the ground station, specifying normal and turning speed, or not)
- Movement combining sensors (move forward until object, turn right until stand parallel to an object or wall ...)
- Position management combining the two encoders values
- Magnetic heading and relative magnetic heading feature
- Contingency control

The figure 5.7 shows the class diagram of the module.

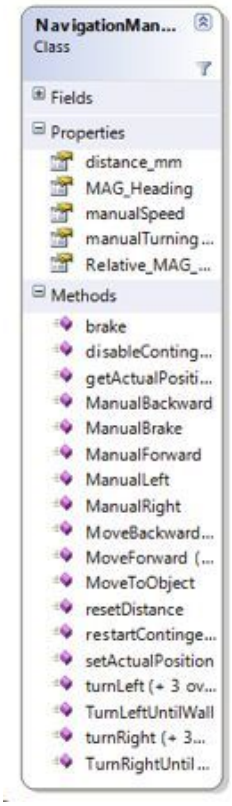


Figure 5.7: Navigation manager class diagram

5.2.2.3. Contingency

The contingency module is in charge of assure the correct movement of the rover. This module is composed by two different threads: one is testing if the theoretical movement of the rover equals to the real movement (i.e. if the rover is moving forward, the encoders should increment its distance). The other thread is in charge of correct the rover if the encoders are not consequently with the theoretical movement. When the first thread detects some contingency, a flag is activated and the second thread corrects the movement of the rover applying the opposite movement the rover was doing just before the detection. In order to avoid a blocking in the movement of the rover, when is detected several contingencies in a short time is applied a forward movement with maximum velocity and if the contingencies still continue is applied movements forward, right and left at maximum velocity. The figure 5.8 shows the class diagram of the Contingency module.

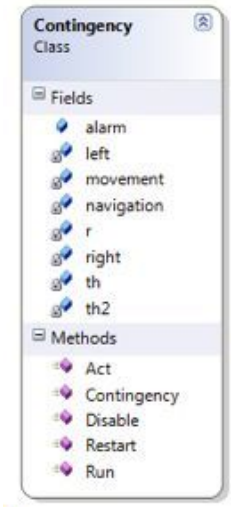


Figure 5.8: Contingency class diagram

5.2.2.4. Engine

The engine module is the higher level one. It controls all the core parts and manages the communication with the ground station. Is implemented with a singleton model. The features of the Engine part are:

- Manages all the timers
  - Transmitting position
  - Transmitting sensor values
  - Transmitting battery levels
- Allows manual movement
- Provides three level log: INFO, DEBUG and ERROR
- Contains the sampling algorithm
  - Movement guidelines
  - Sensing timing

The figure 5.9 shows the class diagram of the Engine.

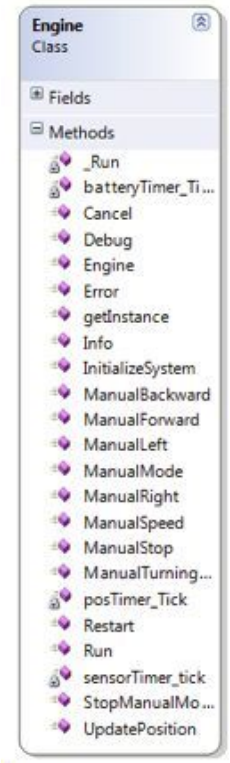


Figure 5.9: Engine class diagram

The Engine class manages all the managers and implements all the logical part. The relationship between all this classes is shown in the figure 5.10.

Figure 5.10: Core scheme

5.2.3. Movement guidelines

The movement guidelines of the rover are implemented in the Engine class. There are three modes of movement:

- Manual movement: controlled from the ground station
- Searching for a wall: is the starting mode. In this first version the rover is supposed to be placed in an empty room. The first movement is to search the limit of the room in order to follow this limit to describe the perimeter. Once the rover has detected a wall, turns right itself until is placed parallel to this wall (in this turn the sensors have to detect when the rover is parallel, when the two wall sensors detect the same distance it means that the rover is in parallel). At this moment the rover starts with the third mode:
- Follow wall: the rover maintain a certain distance to the wall and tries to follow the wall. The rover also is sensing if there is an object in front of it in order to detect when another wall is near. When another wall is detected the rover turns right another time and so on.

The figure 5.11 shows the flow of the thread in charge of the movement of the rover. Notice that when the rover is following the wall it tends to be in parallel of the wall and also maintains a minimum distance to the wall in order to avoid a collision.

Figure 5.11: Movement guidelines flowchart

5.2.4. Communications

The communications part is divided in two layers: coder layer and transport layer.

The coder layer is in charge of serialize the data and implementing a proprietary protocol for the internal communications of the rover with the ground station. In the section 5.3.4. is explained the protocol.

The transport layer implements the serial communication from the board to the transceiver device. This transport is called “SerialTransport” and it has been extracted from MAREA middleware (see [13]) .

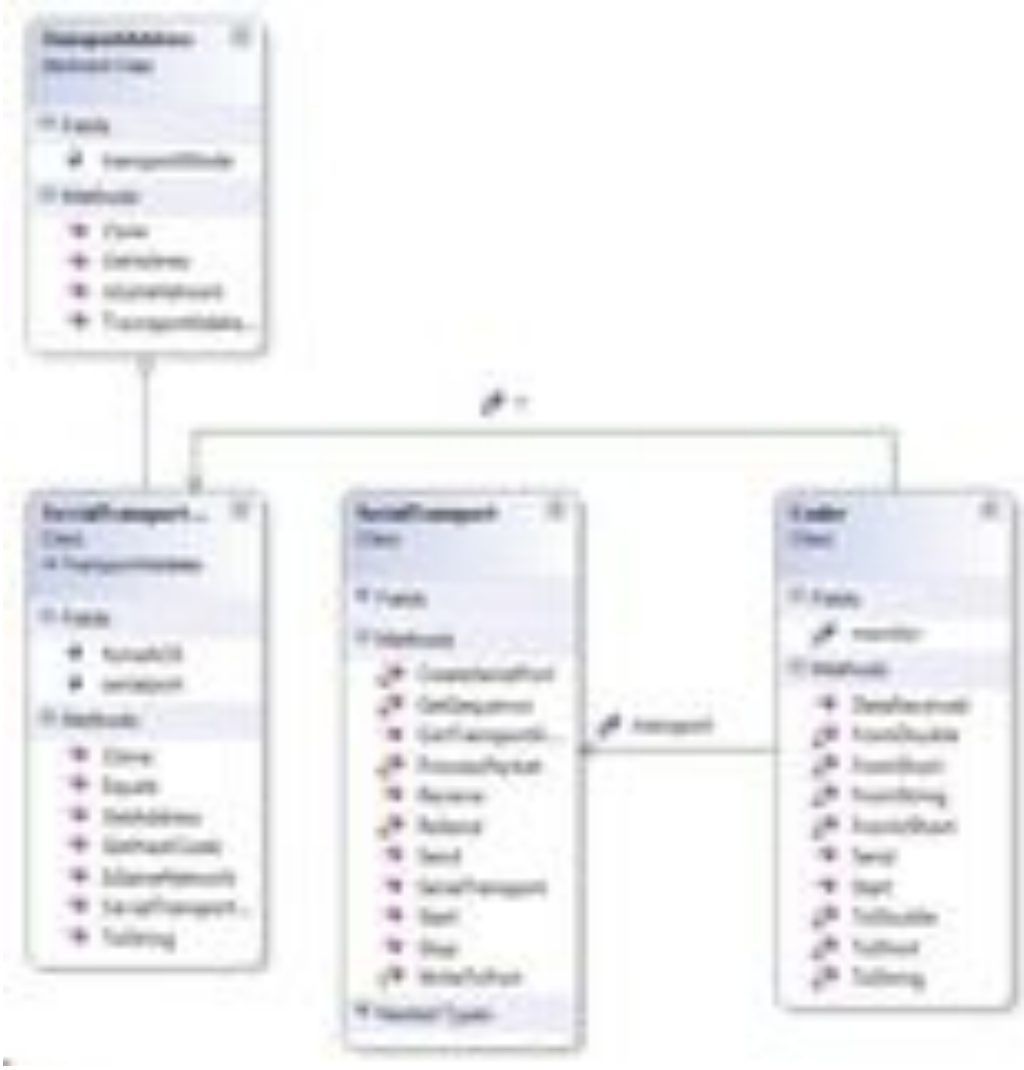


Figure 5.12: Communications scheme

5.2.5. Configuration

There are two static classes in the project which allows the user configure the parameters of the system. One file configures all the global variables and the other file configures the port map (where are all the inputs/outputs connected in the board) of all the hardware elements.

The configurable parameters are:

- Physical measures
  - wheel perimeter
  - height, width and length
  - distance between wheels (and correction factor extracted from UMBmark test)
  - Number of ticks the encoders have
- Normal speed and turning speed (these values are used if the user does not specifies a speed value. In manual mode is mandatory to specify a speed value)
- Distance to consider an object detected

- Minimum distance to follow the wall
- Hysteresis to avoid noise in the movement (some movement decisions are taken depending on some threshold value in the infrared sensors, so with the hysteresis is reduced this noise)
- Timing:
  - Sensor transmission period
  - Position transmission period
  - Contingency update period
  - Battery status transmission period

5.2.6. General scheme

The figure 5.13 shows the general layer scheme of the rover software. The three main parts are Navigation manager, Sensor manager and Battery manager. The contingency module also is important and it is implemented in the navigation manager because the contingency part only refers to movements of the rover. All the drivers are placed in the different managers depending on the sensors and its functionalities. Finally the communications are done through the coder and transmitted to the xBee serial port using the transport.

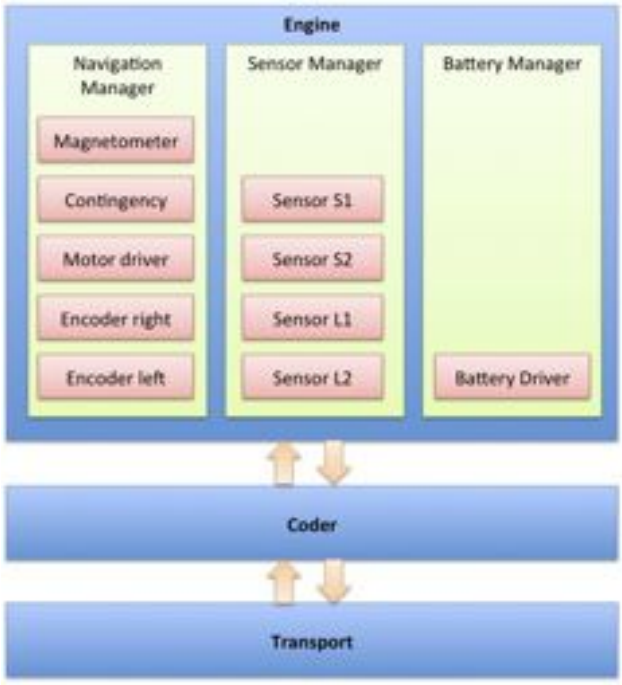
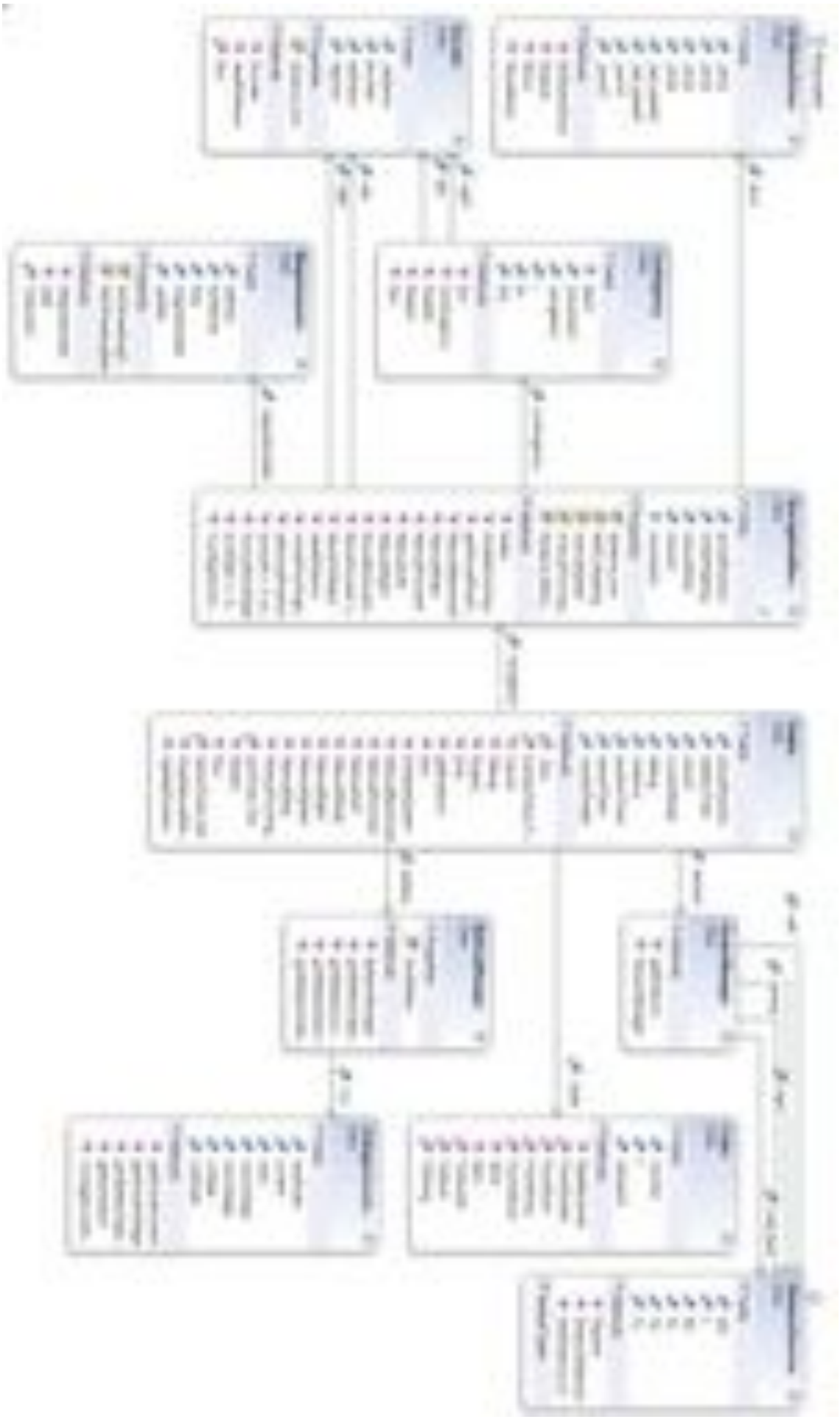


Figure 5.13: General software scheme

All the software components relationship is described in the figure 5.14.



The general flow of the software is shown in figure 5.15. It can be seen the initializing part, the movement guidelines module and the three threads (implemented with timers) which are in charge of sending telemetry to the ground station. The battery thread also takes benefit of the Battery Manager capabilities and if the battery charge is low, it can stop all the working threads and processes in order to avoid battery damage.

Figure 5.15: General flowchart

5.2.7. Algorithm distribution

As explained above, FEZ Panda has 148KB of user flash and 62KB of user RAM. The SLAM algorithm implemented not needs so much power calculation, but a lot of memory RAM. When the algorithm is starting, the map is created (which is an array of numbers), and the scan process also starts (different arrays of numbers and decimal numbers). All this objects can not be allocated in the user free RAM memory available. There are two options: reduce considerably the resolution and the size of the map, or distribute the software between the ground station and the rover station. The first option was studied but the reduction must be so large that the resulting map is almost unintelligible due to its low resolution. The second option is the deployed one in order to take benefit of the SLAM algorithm and draw maps with better resolution. The disadvantage of this scheme is the more use of the xBee bandwidth, but in this project is not an issue.

5.3. Ground station software

The software developed in the ground station also is implemented in C# language. The ground station is a visual application and the WPF technology will be used. WPF (Windows Presentation Foundation) is a technology for rendering user interfaces in Windows applications (see [14, 15]). WPF uses DirectX and employs XAML (eXtensible Application Markup Language) documents, a derivative of XML, in which all the

control and user interface objects are defined and linked. DirectX interacts directly with the graphic controller instead of the operative system (like Windows Forms do). For this reason the performance is increased and the higher the version of DirectX the lower the interaction with the operative system and the higher the interaction with the graphical controller.

For the developing of the user GUI it has been used the software Expression Blend 4 (see [16]), which provides an user interface to edit all the component properties and exports it to a XAML document.

5.3.1. AvalonDock

AvalonDock (see [17]) is a WPF controls library which can be used to create a docking layout system like that is present in VisualStudio. It supports fly-out panes, floating windows, multiple docking manager in same window, styles and themes and it can host WinForms controls.

Figure 5.16: AvalonDock library demo

The structure is defined in different levels:

- DockingManager: The root control
  - ResizingPanel: Control which defines the area in which can be placed different windows (resizeables and floating)
    - DocumentPane: The main pane of the application.
      - DocumentContent: The content of the main pane. Here is defined all the user controls.
    - DockablePane: The same as DocumentPane but is not the main pane
      - DockableContent: The content of the DockablePane

5.3.2. Ground station layout

The ground station is divided in several parts: Map, Manual control, telemetry, SLAM information, Battery status, Log and Sensors map. The 5.17 shows the ground station layout.

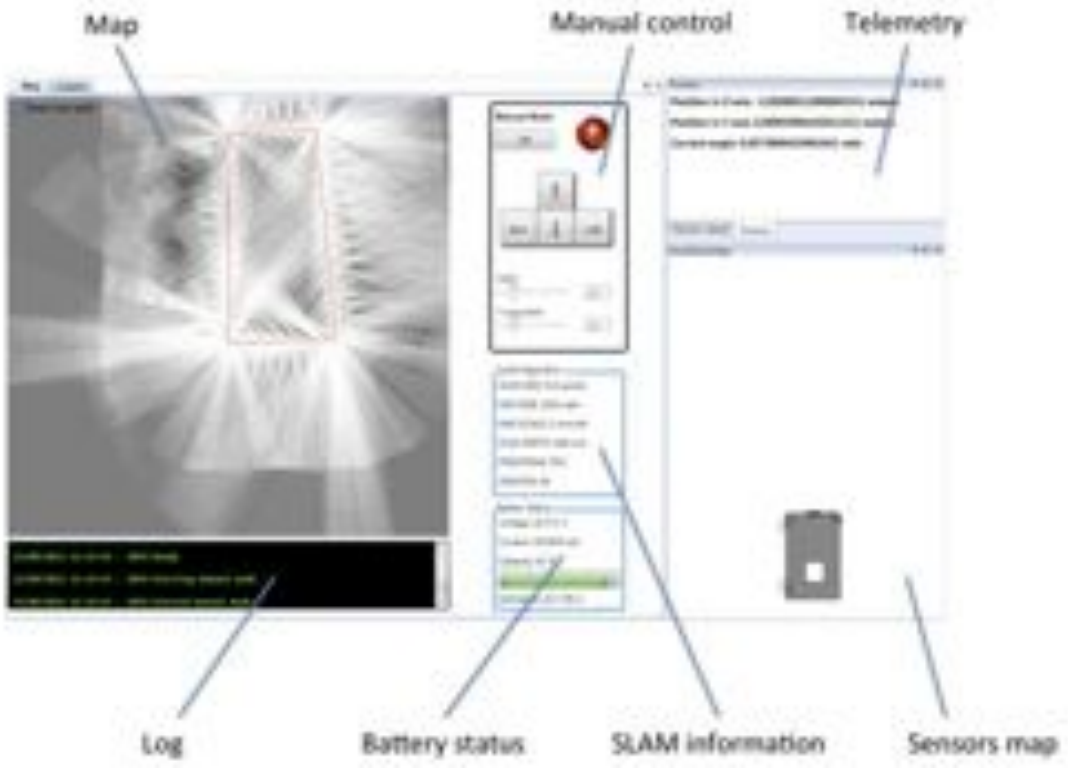


Figure 5.17: Ground station

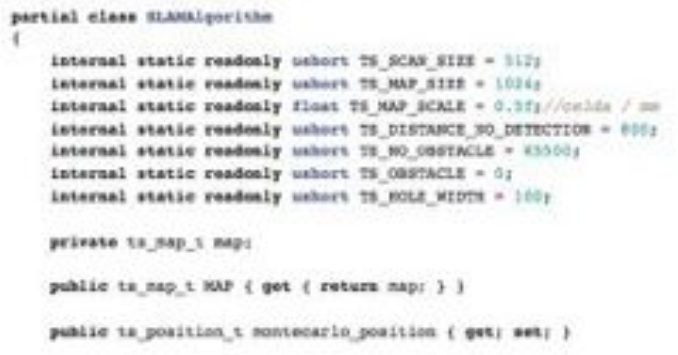
The Map control is the DocumentPane of the application. It contains two DocumentContent: Map and Graphs. The map content contains the resulting map of the SLAM algorithm. The byte array out of the algorithm is computed as an image and is showed in an Image control. Also the rover path is drawn over this image. The map content also contains the log control which shows all the log messages from the rover station. Since this content is the main one also is placed the manual control, the battery status and the SLAM algorithm information.

The graphs content contains graphical information about the position (x, y and angle) in order to improve debugging tasks.

In this layout also is defined other ResizingPane with two DockablePane. One DockablePane contains two DockableContent: Sensor values and Position. These two controls are only in charge of show the information (numerical information) of the sensors and the position respectively. The other DockablePane contains only one DockableContent which is the sensor map of the rover. In this sensor map is drawn the rover and the sensor distances detected.

5.3.3. CoreSLAM algorithm

As explained in section 5.2.7. the SLAM algorithm is implemented in the ground station due to the FEZ Panda limitations. The algorithm is composed by configuration variables like shows the figure 5.18.



```
partial class SLAMAlgorithm
{
    internal static readonly ushort TS_SCAN_SIZE = 512;
    internal static readonly ushort TS_MAP_SIZE = 1024;
    internal static readonly float TS_MAP_SCALE = 0.5f;//celda / mm
    internal static readonly ushort TS_DISTANCE_NO_DETECTION = 800;
    internal static readonly ushort TS_NO_OBSTACLE = 65500;
    internal static readonly ushort TS_OBSTACLE = 0;
    internal static readonly ushort TS_HOLE_WIDTH = 100;

    private ts_map_t map;

    public ts_map_t MAP { get { return map; } }

    public ts_position_t montecarlo_position { get; set; }
```

Figure 5.18: Configuration variables

- The TS_OBSTACLE and the TS_NO_OBSTACLE are the numbers representing the probability values of 100% obstacle and 0% obstacle (respectively).
- TS_SCAN_SIZE is the maximum number of points can be stored in a scan.
- TS_MAP_SIZE is the length (in pixels) of the side of the square form map. The number of total pixels is $TS\_MAP\_SIZE^2$.
- TS_MAP_SCALE is the resolution of the map, so, the number of cells (or pixels) which represents 1 mm of the reality.
- TS_HOLE_WIDTH is the size of the attracting hole.
- TS_DISTANCE_NO_DETECTION is the maximum distance the sensor can be detected, so in this work is 80 cm (800 mm).

Also notice the variable of type ts_map_t which is the map. This type is composed by the matrix of size $TS\_MAP\_SIZE^2$. The figure 5.19 shows this matrix. It is implemented with a 1-Dimensional array and it will contain numbers (between TS_OBSTACLE and TS_NO_OBSTACLE) representing the probability of the existence of an object.

```
public class ts_map_t
{
    public ushort[] map;

    public ts_map_t()
    {
        map = new ushort[SLAMAlgorithm.TS_MAP_SIZE * SLAMAlgorithm.TS_MAP_SIZE];
    }
}
```

Figure 5.19: CoreSLAM map

The map is updated every time is generated a scan. The scan consists in three arrays with the values of the position and value of the impact points of the sensor. The figure 5.20 shows the three arrays, also notice that in the startup of the scan, all the values are initialized with a 50% of probability of an object: $\frac{TS\_OBSTACLE+TS\_NO\_OBSTACLE}{2}$. It is the same procedure done in the map (which is also initialized with this value).

Figure 5.20: CoreSLAM scan

Also there are four important methods which are:

- ts_map_laser_ray: It is the method which compute the “ray” from the position of the sensor to the impact point. It is called for every impact point of a scan
- ts_map_update: It is the method in charge of combine a new scan with the actual map of the algorithm. In this method is applied the $\alpha\beta$ filter and the Bresneham algorithm explained in 2.3.1.1..
- ts_monte_carlo search: It is the method which implements the MonteCarlo algorithm explained in 2.3.1.2..
- ts_distance_scan_to_map: It is the method used by the MonteCarlo algorithm to evaluate how much one scan is similar to the actual map.

5.3.4. Communications

5.3.4.1. Introduction

The ground station is in charge of printing all the environment data and all the rover telemetry. All the mathematical calculations are done in MaCRo segment, so, a communications protocol has to be defined and implemented. It will be implemented a serial communication protocol solution using the XBee modules described in the section 4.7..

5.3.4.2. Protocol

The protocol is described in the table 5.1.

Command	Direction	Type	Description
'tS'	Rover to Ground	short	distance detected by sensor GP2D120 #1 (mm)
'ts'	Rover to Ground	short	distance detected by sensor GP2D120 #2 (mm)
'tL'	Rover to Ground	short	distance detected by sensor GP2Y0A21 #1 (mm)
'tl'	Rover to Ground	short	distance detected by sensor GP2Y0A21 #2 (mm)
'px'	Rover to Ground	double	position of the rover in X axis (meters)
'py'	Rover to Ground	double	position of the rover in Y axis (meters)
'pa'	Rover to Ground	double	angle of the rover (rads)
'bv'	Rover to Ground	double	current voltage of the battery
'bi'	Rover to Ground	double	current consumption of the battery
'bp'	Rover to Ground	ushort	actual capacity of the battery (%)
'be'	Rover to Ground	ushort	time estimation of the battery (minutes)
'LI'	Rover to Ground	string	Log message with INFO level
'LD'	Rover to Ground	string	Log message with DEBUG level
'LE'	Rover to Ground	string	Log message with ERROR level
'f'	Ground to Rover	N/A	Manual movement. Forward
'b'	Ground to Rover	N/A	Manual movement. Backward
'l'	Ground to Rover	N/A	Manual movement. Turn left
'r'	Ground to Rover	N/A	Manual movement. Turn right
's'	Ground to Rover	N/A	Manual movement. Stop (brake)
'm'	Ground to Rover	N/A	Start manual mode
'M'	Ground to Rover	N/A	Stop manual mode
'v'	Ground to Rover	short	Configure manual turning speed
'V'	Ground to Rover	short	Configure manual speed

Table 5.1: Serial protocol

5.3.4.3. Study of bandwidth

The information sent to ground station is:

- Sensor information: There are 4 sensors. The bytes sent are for each sensor 2 bytes for the command and the size of the short which is other 2 bytes.
- Position information: The position information consists on three doubles (x,y and theta). The doubles are coded as a string because the Micro Framework 4.1 does not implement efficient conversion methods from double to byte. The worst case is a number with 18 figures. The bytes sent are for each variable(x,y and theta), 18 bytes and 2 bytes for the command.
- Battery information: The battery information are the current, the voltage, the capacity and the estimation. The current and the voltage are doubles and the capacity and estimation are ushorts. The bytes sent are: for the current and the voltage 2 bytes of command and 18 bytes of double size for each one, and for the capacity and estimation 2 bytes of command and 2 bytes of the ushort size for each one.

The table 5.2 summarizes the bandwidth estimation.

Name	Total bytes	Periodicity	Bandwith
Sensors values	$b = 4\,[2+2] = 16$	150 ms	$BW = \frac{16bytes}{150e-3} \simeq 106.67bps$
Position values	$b = 3\,[2+18] = 60$	100 ms	$BW = \frac{60bytes}{100e-3} = 600bps$
Battery information	$b = 2[2+2]+2[2+18] = 48$	5000 ms	$BW = \frac{48bytes}{5} = 9.6bps$
			$TotalBW \simeq 716.27bps$

Table 5.2: Bandwidth

5.4. Conclusions

The software deployed in the rover and in the ground station is simple and very modular. Every element can be reused in other rovers and it is easy to introduce new elements in the system. As seen before the CoreSLAM algorithm has moved to the ground station to take benefit of the computer power. The figure 5.21 shows the final architecture of the system.

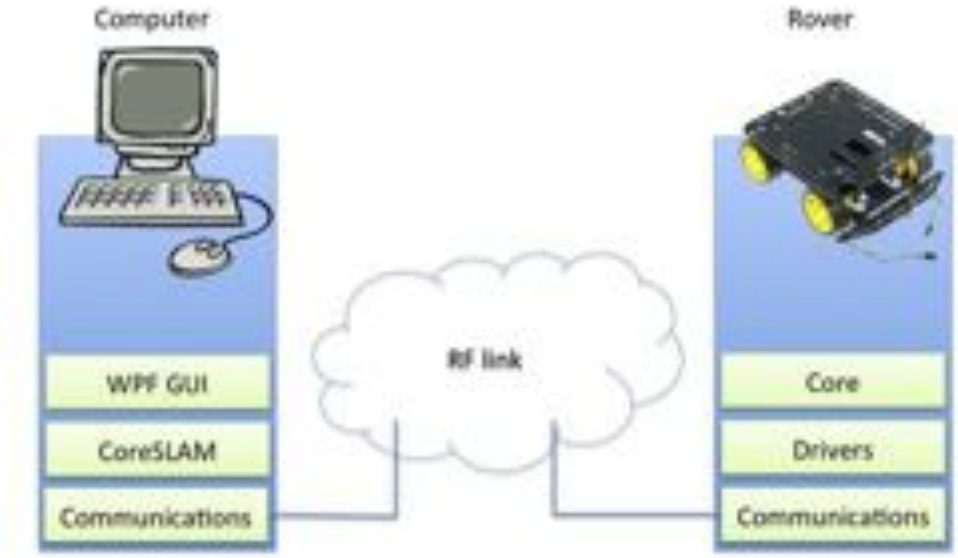


Figure 5.21: System architecture

CHAPTER 6. EVALUATION

6.1. Introduction

In this chapter will be evaluated all the system deployed. First of all will be analyzed the final rover platform implemented and then the overall system including the ground station with the SLAM algorithm.

6.2. Platform

Considering all the hardware explained in chapter 4 the final platform is shown in the figure 6.1.

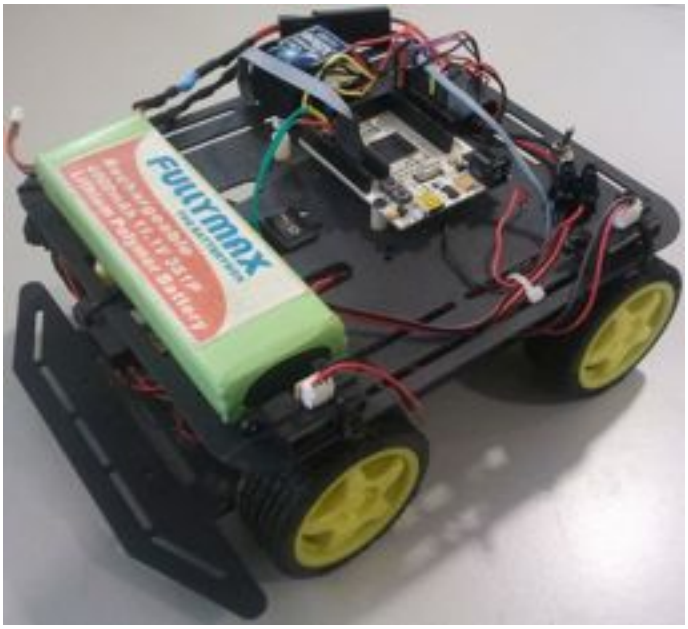


Figure 6.1: Platform

In the Annex IV is shown the assembly of the physical prototype. Also is shown several images of the final platform after the implementation of the design showed in the section 4.9..

6.3. System test

6.3.1. Odometry correction: UMBMark experiment

The first test for the odometry information is to draw a square path in manual mode. The resulting drawn path should be a form similar to a square path. Since the odometry information has systematic errors, the turning angles are not perfect (i.e. the rover has turned 90º but the navigation manager says 80º or less). The resulting form is the figure 6.2. The turning angles are not calculated correctly and the square path is not closed.

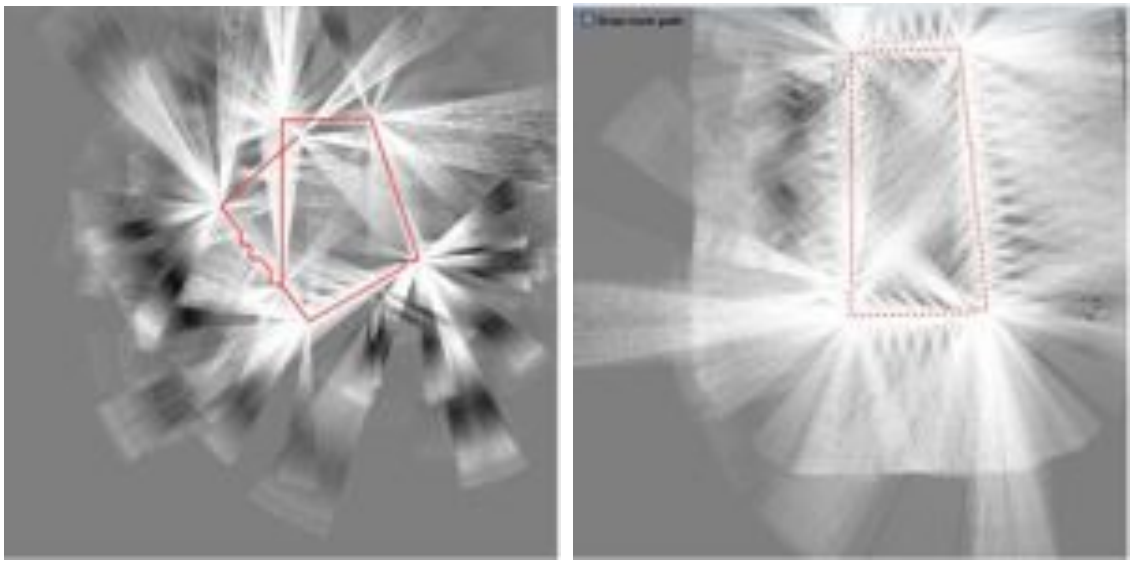


Figure 6.2: Square test before (left) and after (right) correction

After doing the UMBmark test explained in the section 3.4. the deviation angles are reduced. However, since this errors are accumulative, it is very difficult to obtain a zero-error angle, so some little errors still remain in the square path of the rover. The longer the path, the higher the error. The figure 6.2 shows the result mentioned.

6.3.2. SLAM testing

The SLAM algorithm has been testing placing the rover in two different circuits. One simple rectangular circuit, and other more complex with an "L" form. It has been tested in automatic mode and in manual mode. The two mentioned circuits are shown in figure 6.3 and figure 6.4.

Figure 6.3: Rectangular circuit

Figure 6.4: L form circuit

The figure 6.5 shows the first test. In the left appears the resulting map of the rover in automatic mode. In the left appears the same map but made in manual mode.

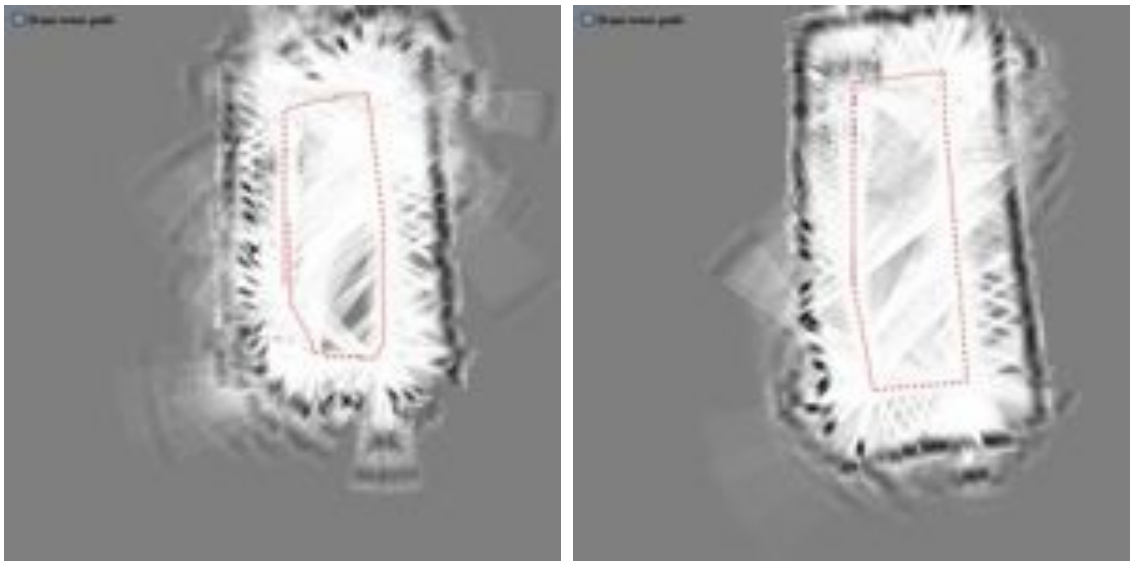


Figure 6.5: Rectangular circuit, automatic (left) and manual (right) mode

In automatic mode the rover makes several corrections in order to follow correctly the wall, maintain certain distance and in general avoid situations where the sensors do not work properly (too close to the wall and so on). Every correction is taken into account by the encoders, but since the resolution of these elements is not good, in every correction there is a little error which is added over the time. In the manual mode this corrections are not done because the user is viewing the rover and the distances to the walls, the driving is more smoothed, so the amount of corrections is zero or almost 0. For this reason the errors produced are less than the automatic mode and the resulting map is closer to the reality.

The figure 6.6 shows the same results above but in the “L” form circuit. Another time the manual mode map is more accurate than the automatic mode map. Notice that in this time the two maps are not as well defined as in the rectangular circuit, because the “L” circuit is a little more complex due to the two different corners in both directions (right and left)

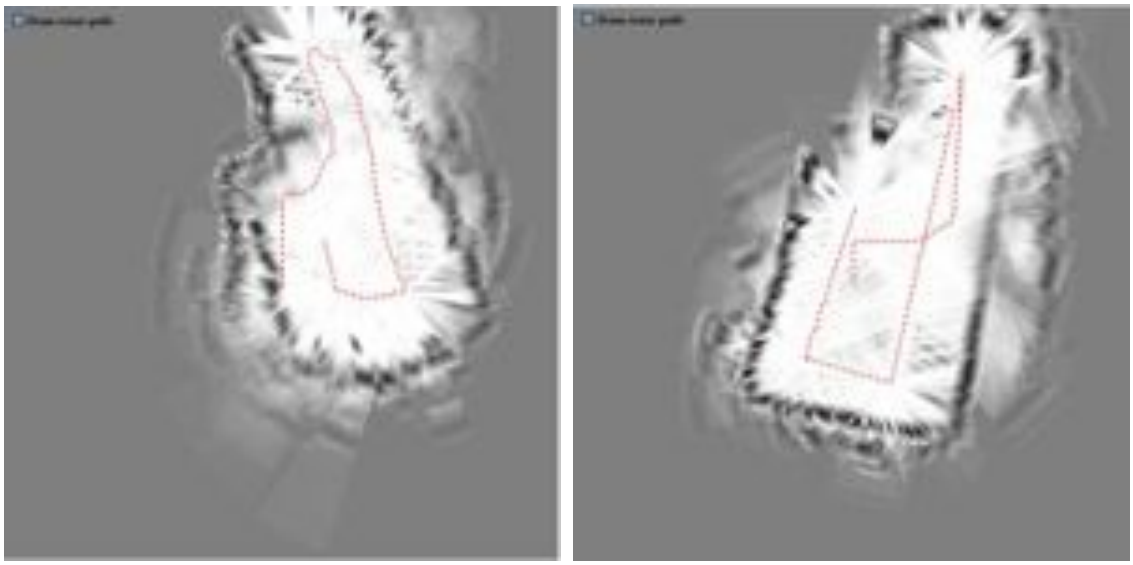


Figure 6.6: “L” form circuit, automatic (left) and manual (right) mode

The final test is about the error propagation. Since the error grows over the time because are accumulative errors, the larger the distance, the higher the error, thus if the rover makes several laps in the circuit it will

become a “cloud” of points which are not correct at all. The figure 6.7 shows this effect. Notice that there are several rectangles but in many different orientations (actually is always the same rectangle).

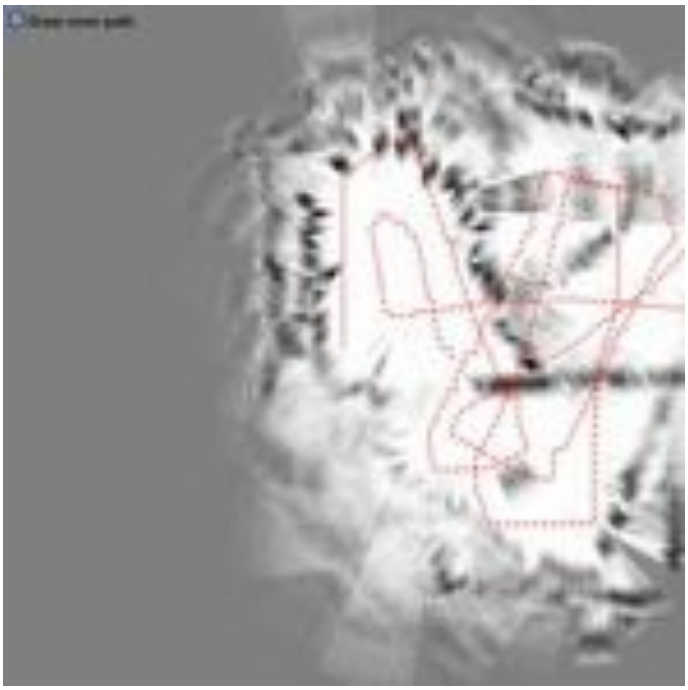


Figure 6.7: Rectangular circuit and several laps

Summarizing all the results obtained in this chapter, the accumulative errors of the odometry (mostly in the robot turns) provokes a map which is not correct. On the other hand, if the distance travelled is not so high, the errors are not high enough and the map generated in both automatic and manual modes represents (with a certain accuracy) the correct circuit.

CHAPTER 7. CONCLUSIONS & FUTURE WORK

7.1. Conclusions

SLAM algorithms is a current subject in robotics which is in continuous investigation. Usually these system are implemented in order to map long corridors with a high accuracy and precision. The important goal of this work has been to develop a first version of a SLAM system composed by a ground station and a rover, and the main difference between that objective and the current SLAM systems is that in this work has not been used high accuracy technology, because other goal of the work was that the platform must be a low-cost one. The objective has been reached by means of different low-cost hardware elements (usually used for learning applications and prototyping) in order to build the hardware platform as simple as possible. It has been selected all the elements comparing the trade-off of price and accuracy, and also minimizing the current consumption and the size as possible.

Nowadays there are different SLAM algorithms with different hardware elements, different probabilistic methods and different mathematical models. In this work the most appropriate algorithm is a tiny algorithm with low computational complexity and it is not necessary to be so accurate because the hardware elements used are not accurate at all, so in the tradeoff accuracy versus speed of the algorithm, in this work has been prioritized the speed. The algorithm used is CoreSLAM.

The logical part of the rover (which is in charge of the movement autonomy of itself) has been developed in a .NET Micro Framework environment in a FEZ Panda board (based on Arduino). .NET Micro Framework allows a fast prototyping and programming ideal also for learning purposes due to its high-level programming (higher than C/C++). In the ground station has been used .NET Framework which offers the same of .NET Micro Framework but with more capabilities for more powerful equipments.

The sensing method used has been selected in such a way that it can be used in indoor and outdoor environments. The rover does not need external elements to self-locate, but infrared sensors and encoders has been used to locate and map the environment. The encoders are in charge of the localization part (odometry) and the most important part of these pieces is the error reduction needed because they introduce accumulative errors which can be very significant over the time. With the UMBMark experiment which uses mathematical equations to reduce the error it has been reduced significantly the systematic errors of the encoders, but some non-systematic errors like slippage, irregular terrain and so on still is a big constraint to the self-location part. For this reason it has been placed a magnetometer which allows knowing an absolute value of the rover orientation in order to correct the errors described before.

Between the two parts must be a communication layer which allows exchanging information for the correct working of the system. This layer has been implemented with a serial protocol over the xBee modules in the S band with a maximum throughput of 256 kbps. The information traveling in the communication layer is telemetry (position, sensor values, battery levels, orientation) and also commands for the manual movement control of the rover. One important information which is sent by the rover is its position and its sensor values, with these information, the ground station is able to apply the SLAM algorithm which is not implemented in the rover station due to its computational complexity and memory requirements.

The graphical part of the ground station software has been implemented with Windows Presentation Foundation (WPF) which is a technology implemented over DirectX and allows the user designing graphical user interfaces. Employs a XAML (eXtensible Application Markup Language) document which contains the description of all the visual elements. The DirectX library interacts more directly with the graphical controller and for this reason has a better performance than Windows Forms, which interacts with the operative system. The graphical part has been designed with Microsoft Expression Blend 4 which is an application that allows designing graphically the interface and exports that design into an editable XAML file.

About the low-cost objective, the table 7.1 shows a simple budget with the cost of the rover platform.

Element	Units	Price (€)/Unit	Price (€)
Structure kit	1	71.43	71.43
FEZ Panda	1	34.95	34.95
DC Motor Driver	1	16.95	16.95
Distance detector	4	12.95	51.80
xBee RF module	2	16.99	33.98
Battery indicator	1	14.75	14.75
Battery	1	68.95	68.95
Total			**292.81**

Table 7.1: Budget

The total price does not exceed the price of 300€ which is extremely lower than the current SLAM systems (which uses high cost LASERs about 6000€ or high resolution cameras).

7.2. Future work

About the future work-lines the elements can be improved are the sensors. Their very low-cost implies a not so good resolution and that introduces error into the system. In the work it has been proved that the limitation is the turns. The encoders do not have resolution enough to reduce the errors. The improvement points are:

- Encoders resolution. The actual low resolution of the encoders introduces slight errors (mostly in turns) but since they are accumulative they become a large errors over the time
- Use of magnetometers or IMUs. In the Annex I is shown the study and the drivers developed for a magnetometer. It has been implemented all the necessary software elements but it is not installed yet because it does not have time enough.
- More sensing points. The infrared sensors installed in the system have a low number of detection points. With a larger number the SLAM algorithm can be improved and even can be correct the rover position with closing the loop of the system.

Another improvement is the installation of a more powerful board in order to implement the SLAM algorithm in the rover station instead of the ground station. This will turn the ground station more independent from the rover station, and consequently a more modular system.

Other future work-line is to implement a visual camera in the system (or maybe more than one). The goal of this is to combine the tinySLAM with other SLAM algorithms based on visual cameras. Also can be installed stereoscopic systems (such Microsoft Kinect) for 3D SLAM.

Finally the system can be integrated into a middleware able to distributing all the software in small pieces in different machines. MAREA is a middleware based on services which follows the publish/subscribe policy in which all the services can be producers or consumers of different kind of data. The ground station and the rover software have been implemented in a modular way, thus the integration from this work to the middleware can be done converting the current described pieces into services and managing the exchange of data of each service with each other. Right now it has being used its transport component for sending the messages between the rover and the ground station. This improvement allows the user configure which elements will be present in the mission and which not, and the most important part is that is a system 100% modular, so, one service can be moved to the ground station and the resultant system is exactly the same than the actual one. Other important feature is that can be added other rovers to do collaborative tasks or just improve the SLAM algorithm.

7.3. Environmental impact

At last but not least it is necessary to talk about the environmental impact of the work described in this document. First, all the hardware pieces installed in the rover are tagged as RoHS (Restriction of Hazardous Substances) hardware. This tag means that the elements are not containing excessively the following hazardous materials:

- Lead (Pb)
- Mercury (Hg)
- Cadmium (Cd)
- Hexavalent Cromium (CrVI)
- Polybrominated diphenyl ether (PBDE)
- Polybrominated biphenyls (PBB)

The limits of these materials are: 1000 ppm (cadmium 100 ppm) by weight of homogeneous material (the limits are not applied to the whole element but to any single substance that could be separated mechanically).

Also this rover can take care of the environment doing “green” tasks. Nowadays it has been investigated different robots and algorithms to work in nuclear plants for instance like Fukushima one. These robots are in charge of measure the radioactivity level without compromising the human health.

BIBLIOGRAPHY

[1] Michael Montemerlo - Robotics Institute, Carnegie Mellon University, July, 2003. ”FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association”

[2] Michael Montemerlo - School of Computer Science Carnegie Mellon University Pittsburgh, Daphne Koller and Ben Wegbreit - Computer Science Department Stanford University Stanford. ”FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges”.

[3] V. Petridis, N. Zikos, IEEE members. ”L-SLAM: reduced dimensionality FastSLAM algorithms

[4] WooYeong Jeong - School of Radio Science and Communications Hongik University, Kyoung Mu Lee - School of Electrical Engineering and Computer Science Seoul National University. ”CV-SLAM: A new Ceiling Vision-based SLAM technique”

[5] Josep M. Porta - Robotic institute and informatics (UPC-CSIC). ”CuikSLAM: A Kinematics-based Approach to SLAM”

[6] Bruno Steux, Oussama El Hamzaoui Mines ParisTech - Center of Robotics, Paris, FRANCE. “CoreSLAM : a SLAM Algorithm in less than 200 lines of C code”

[7] J.E.Bresenham - ”Algorithm for computer control of a digital plotter”

[8] Nicholas Metropolis, S. Ulam - ”The Monte Carlo Method”. ournal of the American Statistical Association, Vol. 44, No. 247. (Sep., 1949), pp. 335-341.

[9] Johan Borenstein, Liqiang Feng - “Measurement and Correction of Systematic Odometry Errors in Mobile Robots”

[10] J. Borenstein, B. Everett, L. Feng - “Navigating Mobile Robots: Systems and Techniques”. Wellesley, MA: A.K. Peters, 1996

[11] Arduino Home Page: http://www.arduino.cc/

[12] Freescale Semiconductor. “3-Axis, Digital Magnetometer”, Document number MAG3110, Revision 2.0, February 2011

[13] Juan López Rubio. "Service Oriented Architecture for Embedded (Avionics) Applications". Phd Thesis. Technical University of Catalonia (UPC), 18th March 2011.

[14] WPF Home page: http://windowsclient.net/wpf/

[15] WPF Tutorial: http://www.wpftutorial.net/

[16] Expression Blend overview: http://www.microsoft.com/expression/products/blend_overview.aspx

[17] AvalonDock library: http://avalondock.codeplex.com/

[18] Nguyen Ho Quoc Phuong, Hee-Jun Kang, Young-Soo Suh, Young-Sik Ro - “A DCM Based Orientation Estimation Algorithm with an Inertial Measurement Unit and a Magnetic Compass”. Journal of Universal Computer Science, vol. 15, no. 4, 2009

[19] William Premerlani, Paul Bizard - ”Direction Cosine Matrix IMU: Theory”

[20] ArduPilot main page: http://diydrones.com/profiles/blogs/ardupilot-main-page

ANNEX I: MAGNETOMETER

Hardware

The encoders have a limited resolution, and always will introduce different errors depending on the resolution. The main problem is that this errors are accumulative and always one error is added to each other an so on. When the rover has made several turns, the final position has an error in the angle variable. For this reason it has been installed a magnetometer which allows knowing an absolute angle heading to correct these errors.

The magnetometer used is the MAG3110FCR1. This magnetometer is a small, low-power, digital 3-axis magnetometer. It features a standard I2C serial interface output and smart embedded functions. It is placed in a board which breaks out all of the pins for the MAG3110FCR1 to a standard 0.1" header and also supplies the necessary filtering capacitors. The main features are:

- 1.95V to 3.6V Supply Voltage
- 7-bit I2C address = 0x0E
- Full Scale Range ±1000 μT
- Sensitivity of 0.10 μT

The figure 7.1 illustrates the installed board.

Figure 7.1: Magnetometer

The communications are done by means of a standard I2C interface. MAG3110FCR1 provides different registers (R or R/W registers) for extract the magnetic information and for configuring different parameters in order to fulfill specific needs. The used registers are specified in the table7.2.

Name	Type	Register Address	Default Value	Description
OUT_X_MSB	R	0x01	data	Bits [15:8] of X measurement
OUT_X_LSB	R	0x02	data	Bits [7:0] of X measurement
OUT_Y_MSB	R	0x03	data	Bits [15:8] of Y measurement
OUT_Y_LSB	R	0x04	data	Bits [7:0] of Y measurement
WHO_AM_I	R	0x07	0xC4	Device ID number
CTRL_REG1	R/W	0x11	0000 0000	Operation modes
CTRL_REG2	R/W	0x12	0000 0000	Operation modes

Table 7.2: Used MAG3110FCR1 registers from

The MAG3110FCR1 has an I2C interface of 3.3V levels. The FEZ Panda works with 5V levels. Since the I2C bus is a bidirectional interface, it is necessary to install a logic voltage converter. It can be done with a

MOSFET transistor (which are bidirectional) and two pull-up resistors. The figure 7.2 illustrates the scheme used. With this circuit the input 3.3V levels (from left to right) will output in 5V levels (right side), and the 5V levels input (from right to left) will output in 3.3V levels (left side).

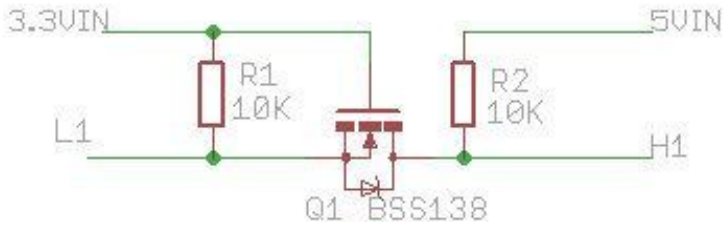


Figure 7.2: Logic level conversion

The circuit must be deployed in the two I2C lines (SDA and SCL).

Software

The magnetometer driver is in charge of configure and extract information from the MAG3110FCR1. The driver establishes the I2C communication with the device and provides two propierties (MAGHeadingDeg and MAGHeadingRad) which are updated periodically with the magnetic heading information. The figure 7.3 shows the class diagram of the driver. The magnetometer has several registers for configuration, and for the data information. The registers are specified in the data-sheet (see [12], page 10), the used in this driver are: from 0x01 to 0x07 for the data values, the 0x07 register for communication testing, and the 0x11 and 0x12 for the magnetometer configuration (start up magnetometer, set working mode and so on).

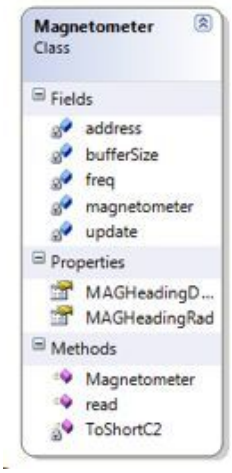


Figure 7.3: Magnetometer driver

ANNEX II: IMU

Inertial Measurement Unit (IMU) is an electronic element which provide information about inertial forces: acceleration, angular velocity and magnetic field. A 3-axis IMU has been tested in this project to be a sensor for auto-locate the rover. The auto-location done with IMUs has a big constraint that is the accumulative errors. The data of the accelerometer and the gyroscopes have to be integrated in order to obtain position units, and the accelerometer errors (even if are very low errors) over the time due to the integration provokes a big error. The figure 7.4 shows the IMU used: MEMSENSE NanoIMU. Is an I2C IMU with 3-axis accelerometer, 3-axis gyroscopes and 3-axis magnetometer. Also provides temperature information of the sensors and also a timer reference.

Figure 7.4: IMU

Moreover other IMU useful information is the Euler angles which are the Roll, the Pitch and the Yaw angles. One method to extract these angles are the Direction Cosine Matrix (DCM) which combines all the IMU data to extract the angles (see [18, 19]).

The DCM implemented in this project has been extracted from the ArduPilot (see [20]). This DCM extracts correctly the angles but in our project the computational cost (the I2C bus, the data extraction and the DCM calculation) is too high and the information of the position is not accurate enough, due to the difficulty to correct the accumulative errors. for these reasons the IMU has been uninstalled.

ANNEX III: GROUND STATION DESIGN WITH EXPRESSION BLEND® 4

The figure 7.5 shows the Expression Blend application. The interface allows designing the GUI from a graphical point of view.

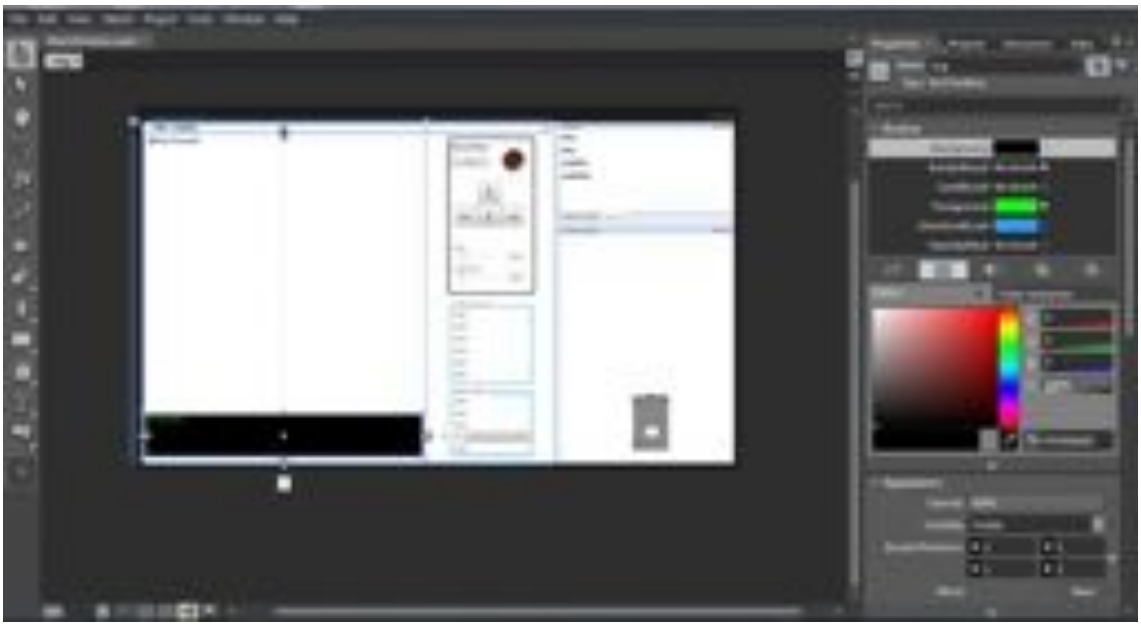


Figure 7.5: Ground station design: visual

Also can be designed from the XAML document like shows the figure 7.6.

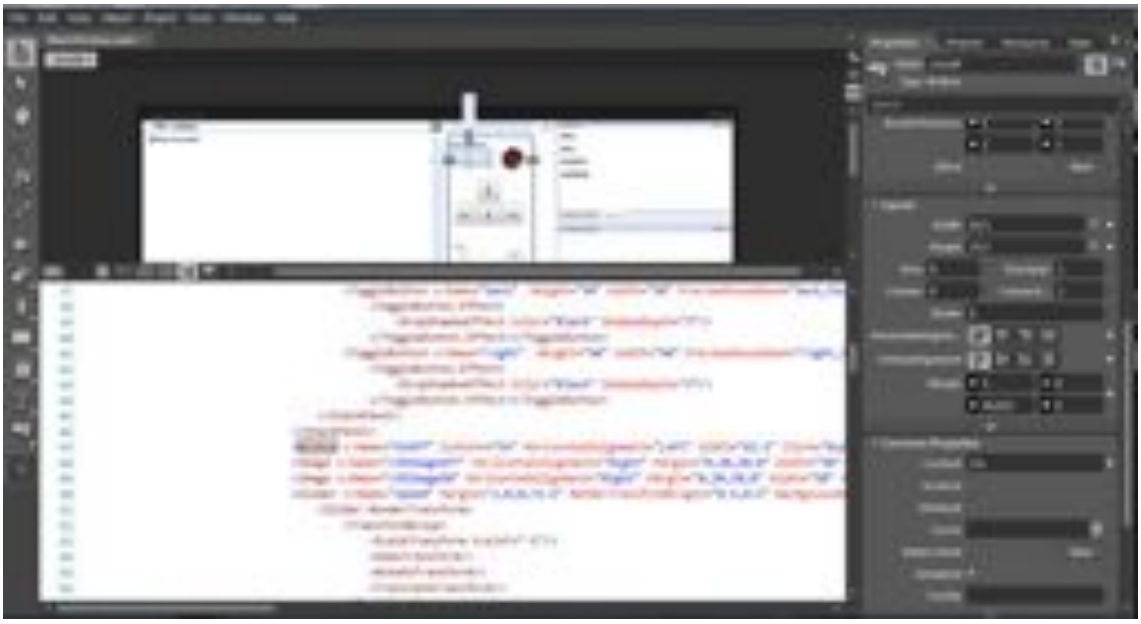


Figure 7.6: Ground station design: visual and XAML

Expression Blend offers all the available controls in a toolbox. The figure 7.7 shows this.

Figure 7.7: Expression Blend controls

Expression Blend works over the same solution as the Visual Studio, in fact are exactly the same files and can be edited from the two applications. The figure 7.8 shows a comparison between the Expression Blend solution and the Visual Studio solution.

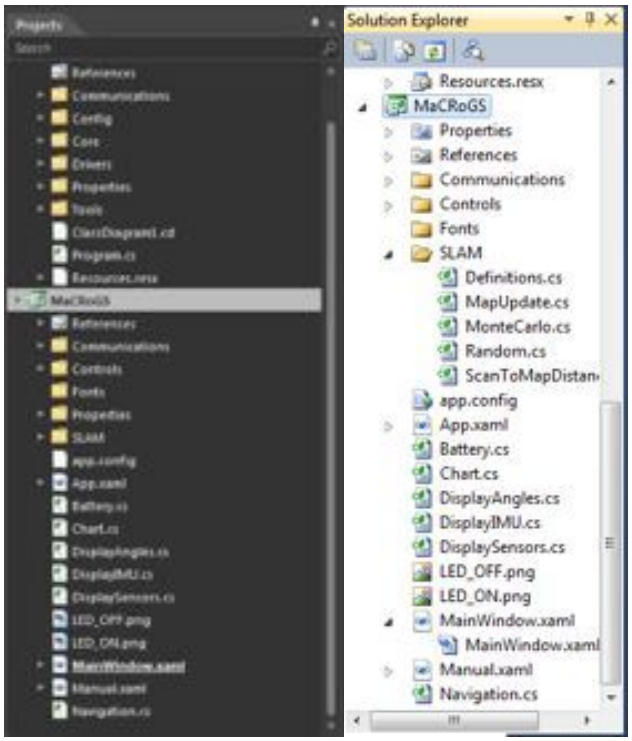


Figure 7.8: Expression Blend and Visual Studio solutions

ANNEX IV: PLATFORM IMAGES

Figure 7.9: Assembly 1

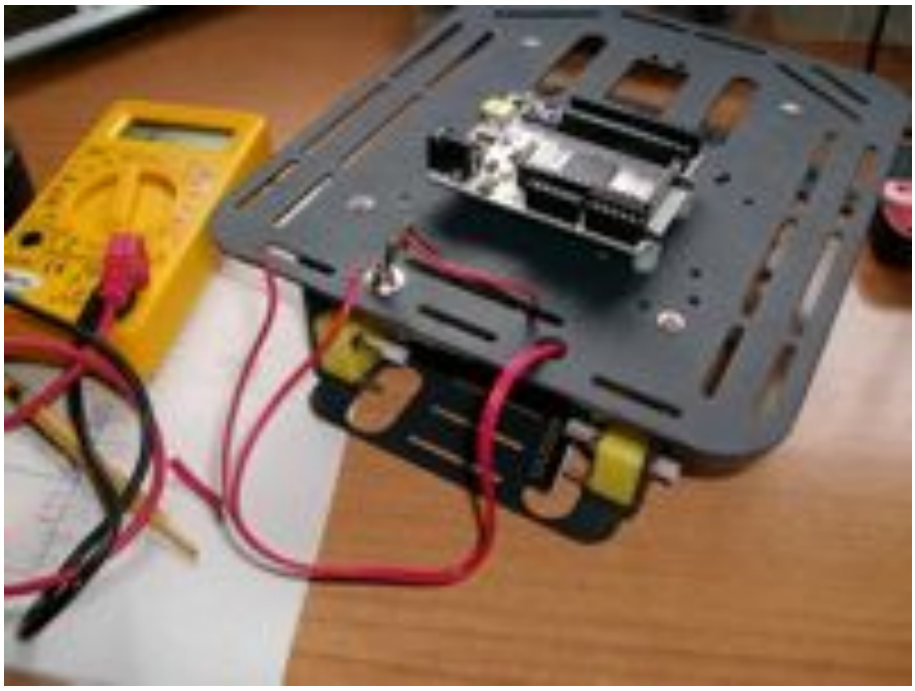


Figure 7.10: Assembly 2

Figure 7.11: Platform 1

Figure 7.12: Platform 2

Figure 7.13: Platform 3

Figure 7.14: Platform 4