Distributed control for mobile robot team coordination

Bernardo Filipe Lapa Marques da Costa

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. João Manuel Lage de Miranda Lemos
Supervisor: Prof. Maria Helena da Costa Matos Sarmento

Examination Committee
Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Members of the Committee: Prof. Miguel José Simões Barão
Prof. Bertinho Manuel D’ Andrade da Costa

April 2014
Entrei numa livraria. Pus-me a contar os livros que há para ler e os anos que terei de vida. Não chegam! Não duro nem para metade da livraria! Deve haver certamente outras maneiras de uma pessoa se salvar, senão... estou perdido.

José de Almada Negreiros
Abstract

The objective of this work is to exploit the appearance of low cost, miniature, mobile robot platforms that can be fitted with sensors and wireless communication systems. This work is based upon the m3pi robot, made by Pololu, that is supported by the mbed platform. In order to be able to explore the possibilities given by this robot, it was developed a system capable of managing the operation of a team of robots in order to achieve a common goal.

The developed system uses a multi-threaded approach to conciliate the pooling of the sensors, feeding and retrieving information to and from the wireless device and the path planning and controller module. The sensors are a sonar and an inertial measurement unit, that includes an accelerometer, a gyroscope and a magnetometer.

After the system is developed, a simple approach to path planning is implemented, where robots can follow each other, by sending information to each other about their position, via the artificial potential field approach.

Keywords

Artificial Potential Fields, Odometry Estimation, Object Oriented Mapping, Operating System
Resumo

O objectivo deste trabalho é de explorar o aparecimento de plataformas de robôs móveis, em miniatura e de baixo custo, capazes de ser equipadas com sensores e sistemas de comunicações móveis. Este trabalho é baseado no robô m3pi, desenvolvido pela Pololu, que é suportado pela plataforma mbed. Para ser possível explorar todas as possibilidades dadas por este robô, foi desenvolvido um sistema capaz de gerir a operação de uma equipa de robôs para atingir um objectivo comum.

O sistema desenvolvido usa uma abordagem de múltiplos fios de execução para conciliar a amostragem dos sensores, a alimentação e a aquisição de informação para e de o dispositivo sem fios e o módulo de controlo e planeamento do caminho. Os sensores são um sonar e uma unidade inercial, que inclui um acelerômetro, um giroscópio e um magnetômetro.

Depois de o sistema ser desenvolvido, uma simples abordagem ao planeamento de caminho é implementada, onde os robôs conseguem seguir uns aos outros, através do envio de informação entre eles sobre as suas posições, por via da abordagem por campos de potenciais artificiais.

Palavras Chave

Campo de Potenciais Artificiais, Estimação de Odometria, Mapeamento Orientado a Objectos, Sistema Operativo
1 Introduction 1
  1.1 Motivation ......................................................................................... 2
  1.2 State of The Art .................................................................................. 2
  1.3 Original Contributions ....................................................................... 3
  1.4 Thesis Outline .................................................................................... 3

2 Description and characterization of the mobile platform used 5
  2.1 The Robot ............................................................................................. 6
    2.1.1 Introduction .................................................................................... 6
    2.1.2 Motors and robot model .................................................................. 6
      2.1.2.A Odometry Estimation ................................................................. 13
    2.1.3 Expansion Board .............................................................................. 14
      2.1.3.A Processor .................................................................................. 15
      2.1.3.B Connecting the sonar, the IMU and the XBee CTS output ............. 16
  2.2 Xbee .................................................................................................... 17
    2.2.1 Introduction ..................................................................................... 17
    2.2.2 ZigBee Network Configuration ......................................................... 18
    2.2.3 Addressing, Transmission and Routing ............................................ 19
    2.2.4 Transparent mode VS Application Programming Interface (API) mode 19
    2.2.5 API frames ..................................................................................... 20
      2.2.5.A ZigBee Transmit Request ............................................................ 21
      2.2.5.B ZigBee Transmit Status ............................................................... 21
      2.2.5.C ZigBee Receive Packet ............................................................... 22
  2.3 Sensors .................................................................................................. 23
    2.3.1 Introduction ..................................................................................... 23
    2.3.2 Sonar ............................................................................................... 23
    2.3.3 Inertial Measurement Unit (IMU) ..................................................... 27
      2.3.3.A L3G4200D 3-axis gyroscope ..................................................... 27
      2.3.3.B LSM303DLM 3-axis accelerometer and 3-axis magnetometer .... 30
3 Available software and implemented system

3.1 Operating System and Libraries

3.1.1 Introduction

3.1.2 mbed Software Development Kit (SDK)

3.1.3 Real Time Operating System (RTOS) Library

3.1.4 m3pi Library

3.1.5 Minimu9 Library

3.2 Communications

3.2.1 Introduction

3.2.2 Transmission of data

3.2.3 Processing the messages

3.2.4 Receiving Data

3.2.5 Implemented API

3.3 Reading the Sensors

3.3.1 Reading the Sonar

3.3.2 Getting readings from IMU

3.4 Routines

3.4.1 Introduction

3.4.1.A Common mode controller

3.4.1.B Differential mode controller

3.4.2 Straight movement

3.4.3 Rotation movement

3.4.4 Calibration of the magnetometer

3.4.5 Search for a robot

3.4.6 Follow

3.4.6.A Potential Fields

4 Experimental Results

4.1 Orientation

4.2 Odometry Estimation

4.3 Search Routine

4.4 Follow Routine

5 Conclusions and Future Work

Bibliography
List of Figures

2.1 Pololu 3pi Robot ......................................................... 6
2.2 Robot local inertial frame of reference ................................. 7
2.3 Robot Model .................................................................. 8
2.4 Robot response to a sinusoidal input to the motors ................. 9
2.5 Frequency response of the robot ....................................... 11
2.6 Speed test at 10% motor input with outliers present and fit line 12
2.7 Robot speed calibration: Gain = 1128.83 mm; Offset = -35.60 mm 12
2.8 Robot Model with discontinuities ...................................... 13
2.9 Estimation of the odometry ............................................. 13
2.10 m3pi Top View .......................................................... 14
2.11 ARM mbed NXP LPC1768 pinout and connections ............... 15
2.12 ARM mbed NXP LPC1768 connections after alteration ........ 17
2.13 XBee Pro Series 2 device ............................................. 17
2.14 XBee Explorer USB by Sparkfun ..................................... 18
2.15 ZigBee Network Configuration example .............................. 19
2.16 XBee API UART data frame structure .............................. 20
2.17 ZigBee Transmit Request frame ..................................... 21
2.18 ZigBee Transmit Status frame ....................................... 22
2.19 ZigBee Receive Packet frame ......................................... 22
2.20 MB1013 pinout .......................................................... 24
2.21 Sonar test environment .................................................. 26
2.22 Sonar output comparison .............................................. 26
2.23 Sonar error comparison (real-measured) ......................... 27
2.24 Gyroscope A and B measurements statistical analysis ....... 29
2.25 Magnetometer A and B readings of X and Y axes ............... 31
2.26 Interrupt Service Routine (ISR) attached to serial transmission interrupt 37
2.27 Data structure of the Transmission Slot (txSlot) used for each transmission 38
2.28 Thread responsible for filling the tx circular buffer ............. 39
2.29 Process Frame Thread - responsible to process each frame ... 40
2.30 ISR attached to serial reception interrupt ......................... 42
3.6 Receiving Thread - responsible for reconstructing messages from received bytes ............... 42
3.7 Magnetometer to compass: calibration and conversion .............................................. 47
3.8 Common mode controller ..................................................................................... 50
3.9 Common mode responses to steps of 500 mm ....................................................... 50
3.10 Differential mode controller ............................................................................. 52
3.11 Differential mode responses to steps of $180^\circ$ ............................................... 52
3.12 Implementation of the straight movement command ............................................. 54
3.13 Implementation of the rotation command ............................................................ 56
3.14 Implementation of the calibration of the sonar ................................................. 57
3.15 Implementation of the search algorithm ............................................................. 60
3.16 Implementation of the follow command ............................................................ 62
3.17 Attractive field with goal at $(0,0)$ and $\zeta = 1$ ..................................................... 63
3.18 Repulsive field with obstacle at $(0,0)$ and $\eta = 2 \times 10^{10}$ and $d_0 = 400$ ............. 64
3.19 Attractive field with goal at $(-300,0)$ with repulsive field with obstacle at $(0,0)$ .... 64
3.20 Path generated if robot starts at $(1000,1)$ ............................................................ 65

4.1 Orientation during a $180^\circ$ rotation with a 50Hz sampling rate ......................... 68
4.2 Search routine results ......................................................................................... 70
4.3 Evolution of the position of the artificial goal ...................................................... 71
4.4 1st Follow routine test ....................................................................................... 72
4.5 2nd Follow routine test ..................................................................................... 73
# List of Tables

2.1 Fitness results for the frequencies used in estimating the model .......................... 10  
2.2 Range and resolution comparison between sonars ............................................. 24  
2.3 Consistency results for the MB1013 sonar ..................................................... 25  
2.4 Consistency results for the MB1300 sonar ..................................................... 25  
2.5 Gyroscope A output when robot rotates counterclockwise ............................... 28  
2.6 Gyroscope B output when robot rotates counterclockwise ............................... 28  
2.7 Magnetometer input range and gain .................................................................. 30  
3.1 Sonar models gain and offset ............................................................................ 46  
3.2 Constants common to all actions ...................................................................... 49  
3.3 Common mode response with 100 ms step and 0.001 mm$^{-1}$ of gain ............. 51  
3.4 Differential mode response with 100 ms step and 0.15 rad$^{-1}$ of gain .............. 53  
4.1 Rotation tests results ....................................................................................... 68
Abbreviations

ADC  Analogue Digital Converter
API  Application Programming Interface
CRC  Cyclic Redundancy Check
CTS  Clear To Send
DAC  Digital Analogue Converter
FIFO  First In, First Out
GPIO  General Purpose Input/Output
IDE  Integrated Development Environment
IMU  Inertial Measurement Unit
ISR  Interrupt Service Routine
LCD  Liquid-Cristal Display
LED  Light Emitting Diode
IO  Input/Output
MTU  Maximum Transmission Unit
PCB  Printed Circuit Board
POI  Points Of Interest
RTOS  Real Time Operating System
RTS  Request To Send
SDK  Software Development Kit
SLAM  Simultaneous Localization and Mapping
THR  Transmitter Holding Register
txSlot  Transmission Slot
UART  Universal Asynchronous Receiver/Transmitter
1

Introduction

Contents

1.1 Motivation .................................................. 2
1.2 State of The Art .............................................. 2
1.3 Original Contributions ...................................... 3
1.4 Thesis Outline .................................................. 3
1.1 Motivation

Many problems of considerable complexity can be solved with an approach of divide and conquer. This methodology consists on dividing the problem into smaller problems, more easily solved. This has been applied in many systems, where several local control agents exchange information with each other in order to accomplish a certain end goal. Some examples of these systems are smart energy grids, where the global goal is to achieve the maximum efficiency and the local goal is to be able to deliver the requested power, large scale water channel systems, where the flow along the network must be controlled to avoid overflows, and teams of robots in various applications, such as search and rescue where two robots, one in the ground and the other in the air, coordinate in order to find possible victims.

This thesis considers a situation of a team of robots, exploring the availability of low cost, miniature, mobile robot platforms that can be fitted with sensors and wireless communication systems. The team must be able to move in a coordinated manner, having to estimate the position of each robot and the obstacles in the world. An operating system and network Application Programming Interface (API) must then be developed to enable the use of complex algorithms, such as of mapping, localization and path planning, so that the team can achieve its goal.

1.2 State of The Art

Mapping has been a field of study in the subject of robotics for several years and has many different approaches to the problem, differing mostly on the way of representation of the map. The maps can be classified by the kind of information they store, being available various classes such as free-space maps\[1\], object oriented maps\[1\] and grid maps\[2\]. The class of free-space maps include various methods to create zones on the map, one of which is to construct the map as a Voronoi Diagram \[1\]. Another possibility is to use Cell Decomposition \[2\] where the free space is decomposed in simple regions, having two possible approaches, one with exact decomposition and another with approximated where the regions have a pre-defined shape. The object oriented map approach consists on having a list of all the objects on the map\[1\]. This method stores the objects descriptions in a list, making it light in memory if the number of obstacles is small, in comparison with, for example, the occupancy grid map. The occupancy grid maps was developed by A. Elfes \[3\] in 1989 and consists in dividing the map in cells of equal dimension. Each cell would then have a value that represents the probability of occupancy. This approach is very demanding in terms of memory, since it increases with the size of the map and the precision.

A field that is closely related to the field of mapping is the localization field, since use of a map relies on the ability in one locating itself. There are three major types of localization, relative, absolute and the previous two combined. In relative localization one can use the odometry of the robot, which localizes the robot through wheel motion evaluation, and inertial navigation, that evaluates the motion state of the robot through velocities and accelerations. These types of localization rely on the robot’s model. In the absolute localization field, the robot can use active beacons or landmarks in known
locations or use a form of model matching. In the first case, if the distance to three or more beacons is available, through the processes of trilateration, the process by which the GPS system works, or triangulation, the robot can achieve absolute localization if the beacons, whether active or passive, position on the map is known. In terms of model matching, two of the most used are the Monte Carlo Localization approach [4], which uses particle filtering [5], and the Markov Localization approach [6], which uses the Extended Kalman Filter[7].

Joining both the mapping and localization, there is another field in robotics called Simultaneous Localization and Mapping (SLAM). Most of the initial development of this fields was done in the last decade of the past century with three different approaches. The first one to appear was the Extended Kalman Filter approach [8], by Smith, Self and Cheeseman in the 80s, followed by the work of Gordon et al. which uses Monte Carlo Filter or Particle Filter [9]. In 1997, Lu and Milios presented their work on SLAM based in graphs [10].

To solve the path planning problem, one can use path finding, or graph search, algorithms such as the A star [11] if the map is organized in connected nodes such as the case with the Voronoi Diagram or cell decomposition mapping approach. Another possible way to solve the path planning problem is with Artificial Potential Fields [12] where the goal is set with an attractive potential and the obstacles are set with repulsive ones. This method is easily used when the map is object oriented.

1.3 Original Contributions

The contributions that this thesis provides rely on the approach to the operating system that manages the robots. With the use of multi threading, a system was created that enables the separation of the communication handling from the controller. Although further development can be made to the operating system, it has sufficient compartmentalization for anyone to change the behaviour of the controller, and even the custom network API without having to worry about managing lower level systems such as the connection with the Xbee device or the Inertial Measurement Unit (IMU).

1.4 Thesis Outline

This thesis is organized in the following way: Chapter 2 presents the hardware aspects of the robot and sensors, the IMU and sonar, used including their models. Chapter 3 presents the software available to start with and the software that was created for managing the communication, sensor pooling and controller subsystems. Chapter 4 shows the tests and results obtain in the various routines implemented. Finally, Chapter 5 sums up this thesis and comments on possible improvements and future work.
Descriptive and characterization of the mobile platform used

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The Robot</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Xbee</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Sensors</td>
<td>23</td>
</tr>
</tbody>
</table>
This chapter focuses on describing the various hardware that is used in this work. The model of the robot is also presented.

2.1 The Robot

2.1.1 Introduction

In this work a platoon of Pololu m3pi robots \[13\] is used. This robot is composed by two parts, the lower part is the Pololu 3pi Robot and the upper part is an expansion board with two sockets, one for an ARM 32-bit mbed development board and the other for networking (Xbee or Wixel module). The expansion board also has prototyping space to include additional sensors to the robot, where a sonar and an IMU device will be installed.

The 3pi Robot is a small size robot (9.5 cm diameter and 83 g without batteries) designed for solving maze and path following problems.

![3pi Top View](image1.png) ![3pi Bottom View](image2.png)

**Figure 2.1:** Pololu 3pi Robot

Since one of the basic functionalities of this robot is the ability to follow lines, there are a few features that will not be used in this work, such as the bottom light sensors. For this work we will use primarily the two DC motors, without encoders, and the Liquid-Crystal Display (LCD) monitor, both will be controlled by the ATmega328P microcontroller, which will be working as a serial slave for the actual microcontroller that is the mbed platform seated on the expansion board (section 2.1.3). The power supply of the robot will be used to power the expansion board and all sensors.

2.1.2 Motors and robot model

The model of the robot has three coordinates: \( x \), \( y \) and \( \theta \). These coordinates are related to the world referential where the X axis points to the north and the Y axis points to the west. The angle increases anticlockwise, starting with 0 degrees when facing north, and 90/-90 degrees when facing west/east. Figure 2.2 shows the local reference frame for each robot that is used by the IMU. For this reason, although not used, the Z axis is represented.
The kinematic model of the robot is given in equation 2.1

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} =
\begin{bmatrix}
\cos(\theta) & 0 \\
\sin(\theta) & 0 \\
0 & 2l
\end{bmatrix}
\begin{bmatrix}
v_c \\
v_d
\end{bmatrix}
\]  

(2.1)

where, as stated before, \(x\), \(y\) and \(\theta\) are the coordinates of the robot in relation to the world, \(l\) is the distance between the wheels, \(v_c\) is the common mode velocity and \(v_d\) is the differential mode velocity. Both of the velocities are in mm.s\(^{-1}\), \(x\) and \(y\) are in mm and \(\theta\) is in radians. The equations 2.2 and 2.3 explain how the velocities mention in the kinematic model, equation 2.1, are related to the left, \(v_l\), and right, \(v_r\), velocities of the DC motors of the robot.

\[v_c = \frac{v_r + v_l}{2}\]  

(2.2)

\[v_d = \frac{v_r - v_l}{2}\]  

(2.3)

For simplicity, the common/differential mode velocities will be used instead of the left/right. Therefore, the robots will have two models, one for a common mode operation, where the input variable represents both motors moving at that speed, and a differential mode where the input variable represents the motors moving at symmetrical speeds. Both systems are linear, therefore, they can be added to achieve composed movements. The input of the motors is normalized between -1 and 1, with a quantization of 8 bits, limiting the sum of the absolute values of both inputs. Although the motor’s operation should be linear, a dead zone was detected roughly between -0.08 and 0.08. Since this is due to static attrition, it was decided to consider that the motor only responds to signals greater than 0.09 in module. A basic model for the robot’s operation, without the dead zone or saturation, is
described in figure 2.3.

\[
\frac{a}{s+b} \quad \frac{1}{s} \quad \text{position}
\]

Figure 2.3: Robot Model

The gain of the first block, \( \frac{a}{s+b} \), is given by \( a/b \) and is related to the maximum speed achieved by the robot. The pole given by \( b \) is related to the time that the robot takes the system to achieve a stationary response.

To determining the model for the robot when it rotates, a test was made by sending a sinusoid to the motors input, positive for one wheel and negative for the other, in order for the robot to rotate on itself. The gyroscope from the IMU (section 2.3.3) was used to acquire the angular speed of the robot. The way that the test was implemented was with the principal thread updating the motor as soon as it could, using a timer that was started in the beginning of the test run. This ensures that the motor input will behave as closely as possible to a true sinusoidal wave. The gyroscope was set to acquire a reading at 800 Hz, every 1.25 ms, which will be stored in its internal buffer. A cyclical thread, with a period of 2 ms, was created to fetch the gyroscope reading, leading to a maximum delay of 0.75 ms, while saving the motor input at that given time. Both of the values are stored in a buffer that is flushed into a text file in the end of the test. A few frequencies were tested, as shown in figures 2.4a and 2.4b, and the data obtained was used to fit a model with only one pole, \( G(s) = \frac{a}{s+b} \).
The fitting was made with the use of Matlab's Identification Toolbox with a recursive procedure. The recursion was necessary due to with the increase of frequency of the input leading to the output, of the gyroscope, having more error. This leaded to a model that would try to fit the data, but would fail, nonetheless, it would decrease the fitness with the lower frequency data. Therefore, after fitting the model with the various frequencies responses concatenated in a single data stream, each of those
responses were tested, individually, with the model. If the fit was less than 85%, that frequency was discarded and the process would start over again. This procedure generated the model represented in equation 2.4 with the fit results shown in table 2.1 and it’s frequency response in figure 2.5.

\[ G(s) = \frac{634.77}{s + 23.40} \text{rad.s}^{-1} \]  

(2.4)

<table>
<thead>
<tr>
<th>Frequency [Hz]</th>
<th>Fitness [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.30</td>
<td>97.60</td>
</tr>
<tr>
<td>0.40</td>
<td>97.25</td>
</tr>
<tr>
<td>0.50</td>
<td>96.85</td>
</tr>
<tr>
<td>0.60</td>
<td>96.47</td>
</tr>
<tr>
<td>0.70</td>
<td>96.09</td>
</tr>
<tr>
<td>0.80</td>
<td>95.61</td>
</tr>
<tr>
<td>0.90</td>
<td>95.19</td>
</tr>
<tr>
<td>1.00</td>
<td>94.91</td>
</tr>
<tr>
<td>1.10</td>
<td>94.59</td>
</tr>
<tr>
<td>1.20</td>
<td>94.26</td>
</tr>
<tr>
<td>1.30</td>
<td>94.09</td>
</tr>
<tr>
<td>1.40</td>
<td>93.94</td>
</tr>
<tr>
<td>1.50</td>
<td>93.94</td>
</tr>
<tr>
<td>1.60</td>
<td>93.79</td>
</tr>
<tr>
<td>1.70</td>
<td>93.76</td>
</tr>
<tr>
<td>1.80</td>
<td>94.14</td>
</tr>
<tr>
<td>1.90</td>
<td>93.93</td>
</tr>
<tr>
<td>2.00</td>
<td>94.09</td>
</tr>
<tr>
<td>2.20</td>
<td>94.54</td>
</tr>
<tr>
<td>2.40</td>
<td>94.78</td>
</tr>
<tr>
<td>2.60</td>
<td>94.01</td>
</tr>
<tr>
<td>2.80</td>
<td>92.90</td>
</tr>
<tr>
<td>3.00</td>
<td>91.02</td>
</tr>
<tr>
<td>3.20</td>
<td>90.05</td>
</tr>
<tr>
<td>3.40</td>
<td>87.02</td>
</tr>
<tr>
<td>3.60</td>
<td>85.70</td>
</tr>
</tbody>
</table>

*Table 2.1:* Fitness results for the frequencies used in estimating the model
To obtain the model for the forward movement, the sonar, section 2.3.2, was used. The forward
movement model is to be used when the robot does not have a differential mode, both wheels are
moving at the same time, and the robot moves in a straight line. The gain of the model was obtained
by making measurements of the distance while the robot moved, with a given velocity, towards a
target. The sonar used was the MB1300. In figure 2.6 the output of the sonar is pictured when the
input of the motors is 10%. As it can be seen, the output has a few outliers that need to be taken
out before the fitting. These outliers are due to the sonar used being very sensitive, being capable of
detecting, sometimes, the ground, since it is closely mounted to it. After the outliers were taken out,
a fit line was calculated. The speed that the robot was moving is indicated by the derivative of the fit
line.
This speed test was made with various motor power inputs in order to be have a decent amount of data points to acquire the relation between the robot speed and the motor input. The results of this linear regression are represented in figure 2.7, where the gain of the system, 1128.83 mm.s\(^{-1}\), was determined with the derivative of the fit line.

Since the sonar's pooling rate is just 10 Hz, it is not possible to acquire the model for the common mode directly. For this reason, the pole from the differential mode was used, resulting in the transfer function represented in equation 2.5.
\[ G(s) = \frac{26414.51}{s + 23.40} \text{mm.s}^{-1} \]  

In addition to the saturation of the input, between -1.0 and 1.0, and the dead zone, between -0.09 and 0.09, there is also another discontinuity in the model. The serial slave program, loaded in the ATmega microcontroller, is configured to brake the motor when the input is zero instead of coasting. The more complete model of the robot is described in figure 2.8 with the input normalized.

![Figure 2.8: Robot Model with discontinuities](image)

The transfer function in the centre of the robot model depends if it is for the differential mode, where the equation would be 2.4, or for the common mode, where the equation would be 2.5.

### 2.1.2.A Odometry Estimation

As stated along this thesis, the robots used do not have access to odometry. For this reason, the robot must solely rely on its model to predict its position. Taking into account the kinematic model in equation 2.1 and considering the robot is at 0° of orientation, in the end of the calculations a rotation is made to correct this, the trajectory that the robot takes can be described as a curve. In order to get the x and y offsets, \( \Delta x \) and \( \Delta y \), of this movement, one has to calculate the radius of the curvature, the distance between the robot and the center of the circumference, and the delta of orientation achieved in the movement, as shown in figure 2.9.

![Figure 2.9: Estimation of the odometry](image)
The angular movement is calculated by integrating the angular velocity given in the kinematic model in 2.1, resulting in

\[ \Delta \theta = \Delta t \cdot \dot{\theta}, \]  

(2.6)

where \( \Delta t \) is the amount of time the robot has travelled with the current motor inputs.

The radius is given with

\[ r = \frac{v_r + v_l}{v_r - v_l} \frac{l}{2}. \]  

(2.7)

The offsets, prior to rotation, can now be calculated with

\[ \Delta x = r \cdot \sin(\Delta \theta) \]  \hspace{1cm}  

(2.8)

\[ \Delta y = r(1 - \cos(\Delta \theta)). \]  \hspace{1cm}  

(2.9)

This calculations do not work if the robot is running purely in common mode, both wheels at the same speed, since the radius, \( r \), would be infinite. In this case, the robot is moving forward, and the offsets, prior to ration, are given by

\[ \Delta x = \Delta t \cdot v_c \]  \hspace{1cm}  

(2.10)

\[ \Delta y = 0. \]  \hspace{1cm}  

(2.11)

After getting both of the offsets, a rotation is needed. The rotation is done with equation 2.12 where the angle given, \( \Delta \theta \), is the orientation of the robot before starting the movement. After the rotation, the final offsets, \( \Delta x' \) and \( \Delta y' \), are now ready to be sent to the other robots and to used to update the landmarks positions in the current robot.

\[
\begin{bmatrix}
\Delta x' \\
\Delta y'
\end{bmatrix} =
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix}
\]

(2.12)

### 2.1.3 Expansion Board

The expansion board for the 3pi Robot will house the heart of the robots used in this work, transforming the 3pi Robot into the m3pi Robot. The name is given due to the mbed socket in the expansion board. The board also has an Xbee/Wixel socket alongside the prototyping space to include the other sensors.
For the mbed socket the possibilities were the interface rich mbed NXP LPC1768 [14] or the less powerful mbed NXP LPC11U24 [15]. The first was chosen due to the increased performance. Since the robot will have the mbed socket populated, the Pololu Wixel [16] module is not necessary since its advantage over the Xbee module is in the embedded microcontroller. For this reason, the socket will be populated by a Xbee Pro 2, section 2.2, module to handle networking. As for the sensors, the robots will use a sonar and an IMU device.

### 2.1.3.1 Processor

As stated previously, the mbed board used is the NXP LPC1768. This board features an ARM mbed NXP LPC1768, which is a 32-bit ARM Cortex-M3 at 96 MHz with 512 KB of flash memory and 32 KB of RAM.

![ARM mbed NXP LPC1768 pinout and connections](image)

The board also has 2 MB of flash storage, external to the processor, that can be accessed by the mini-USB port as a removable storage device. The program is saved in this storage area and it is selected by the processor upon booting, when it loads the most recent file in the external flash storage to the internal flash memory.

The programming for this development board was designed to be made online [17] and all of the online resources made available in the mbed website, including the compiler and the source code, are free even for commercial use. The mbed online compiler is a very simple Integrated Development Environment (IDE) with a source control feature built in. To use the compiler an account, which is free, is necessary and to be able to compile to a certain target it is needed to prove that the user has the device. This proof is made by opening the HTML file present in the flash memory of the board (this file is created, if it does not exist, as the board boots) with the login made in the mbed website. Even though the mbed website advises the use of the online compiler in order to ensure the libraries are up to date, the mbed compiler supports exporting the code to other compilers and IDEs.

The debugging capabilities of the mbed platform for this work are limited, although there is support for a full debugger, the CMSIS DAP MDK [18], it is only possible to use it if the USB cable is connected.
This is not practical when debugging a small mobile robot, as is this case, since the robot cannot carry a laptop on top of it. This issue also limits the use of the printf technique [19], which consists of sending text messages, via the serial connection made by the USB interface. Because of this, debugging is only possible via the 4 Light Emitting Diodes (LEDs) on the mbeb board, the LCD screen of the 3pi robot or via radio messages (if this system is working and not the one with the problem). The mbed system has the functionality of detect runtime errors by flashing the LEDs, although it does not give additional information of what has happen neither where. During the development of this work, the serial printf and the 4 LEDs states where used in the early stages and when the network was configured and tested, the debugging process started to use the network to send messages according to the debugging needs.

The connections between the processor and the expansion board are demonstrated in figure 2.11 where it is shown that the processor connects to the 3pi microcrotroller via one SPI interface (pins 5, 6 and 7), one serial interface (pins 9 and 10) and one output pin (pin 23) that is used to send the reset signal to the microcontroller. For the Xbee connection, the serial interface on pins 27 and 28 is used, alongside with the output pin 26, that is used as the reset signal. The other connections are not used in this work, such as the ones regarding eight LEDs in the expansion board and the ones for the infra-red sensors on the bottom of the 3pi robot.

2.1.3.B Connecting the sonar, the IMU and the XBee CTS output

As stated, two additional sensors will installed and used, they are the sonar and the IMU that are discussed in subsections 2.3.2 and 2.3.3 respectably. The sonar readings will be made via the analogue voltage output, therefore one of the Analogue Digital Converter (ADC) pins will be used. The ADCs channels are of 12-bit and have a maximum pooling rate of 200 kHz [20]. It was decided to use pin 20, that is shared with the first red LED of the expansion board. In addition to the analogue output connection, pin 19 is used to start and stop the sonar. The IMU has to be connected by I2C but, due to the existing connections of the expansion board, there is none available. Therefore, the 3pi serial connection to the processor was diverted to the serial interface on pins 13 and 14 in order to free an I2C interface. The decision to divert the 3pi serial connection and not the Xbee serial connection was solely due to the first change being simpler to accomplished than the latter. An additional connection to the Xbee was made, between pin 12 and the Clear To Send (CTS) output of the XBee, in order to implement flow control from the CPU to the XBee.
In the end, the I2C for the IMU uses the pins 9 and 10, the 3pi serial connection uses pins 13 and 14, the Xbee serial connection maintains the use of pins 27 and 28 with the addition of CTS via pin 12 and the sonar uses the ADC at pin 20 and pin 19 as a digital output to start and stop the sonar.

2.2 Xbee

2.2.1 Introduction

For the networking needs of this work, Xbee Pro 2 devices will be used connected via serial interface.

The XBee Pro 2 device is a module from Digi that allows the creation of complex mesh networks based on Zigbee networks [21]. In this work a simple Zigbee network will be used, as described in section 2.2.2. This device has many features as specified in the following list (the ADCs and digital General Purpose Input/Output (GPIO) pins will not be used):

- 3.3 V @ 295 mA
- 250 kbps Max data rate
- 63 mW output (+17 dBm)
To configure the device a board from Sparkfun will be used (see figure 2.14) to convert the serial interface to a slave USB interface accessible via a mini-USB cable.

Figure 2.14: XBee Explorer USB by Sparkfun

With this board (figure 2.14), the device can be connected to a PC Windows machine where the configuration can take place with the X-CTU software [22].

2.2.2 ZigBee Network Configuration

In a ZigBee network there are three types of devices:

- Coordinator - There has to be one, and only one, coordinator in each network. This is the device that creates the network by setting the channel and PAN ID. The coordinator can allow routers and end devices to join the network, assists in routing data, can buffer RF data packets for sleeping end devices and cannot sleep (since all children devices would lose the network).

- Router - This type of device must first connect to the ZigBee network and after that is identical to the coordinator except it does not create networks.

- End device - This type of device is the most basic, it must first join a network before being able to transmit and receive RF data through its parent. It cannot route data but can enter low power modes to reduce power consumption.

[1] ZigBee network identifier composed by a 64-bit and a 16-bit ID. The 64-bit is used in joining and in 16-bit conflict resolution
The configuration of the network for this work will be a PC as the coordinator and the robots as routers since the sleep mode will not be implemented.

2.2.3 Addressing, Transmission and Routing

For the addressing problem the ZigBee network uses two addresses to identify a given device. One of 64-bit (that is unique for each device) and a 16-bit address that is given upon joining the network. The routing is made with the 16-bit address but since that address is not static, the 64-bit address is used to confirm the destination. The coordinator can be addressed by it’s 64-bit address and the reserved 16-bit address of all zeros or it can be address with the 64-bit address all zeros and the 16-bit address as 0xFFF.

To make unicast transmissions, both of the addresses must be used to identify the destination device. If one does not know the 16-bit address, it can be set to all zeros and the transmitting device will automatically ask the network for the 16-bit address associated with the 64-bit address given. If this occurs, some overhead is added to the communication leading to a small delay of the transmission due to the need of searching the network for the destination address.

Broadcast transmissions are made by setting the destination 64-bit address to 0x0000 0000 0000 FFFF and the 16-bit address to 0xFFF. In this type of transmission, each device sends the packet 3 times upon receiving it for the first time. This is achieved by saving each packet for 8 seconds and compare each broadcast packet received to the ones still in the broadcast buffer, thus avoiding an infinite loop of resends. This method of dealing with the infinite retransmission problem leads to an increase of the buffer usage, which along with the flooding of the network with retransmission packets turns this transmission method into one that should be used sparingly.

2.2.4 Transparent mode VS [API] mode

These devices have two modes for operation as a ZigBee network:

- Transparent mode - In this mode, the modem acts as a serial line, in which, everything that is sent to the Xbee will be sent to the network as soon as there is enough data for a packet (the packet size is determined in the configuration) or a timeout period has passed without data. The network can be configured as a broadcast network or a unicast network. To change the
destination device, it is needed to enter the command mode, leading to additional delay on the transmission.

- **API mode** - In this mode, to interact with the modem one has to use the API provided by the manufacturer. In this mode the Xbee does not support fragmentation, therefore, all communications must fit in a single packet, the size is given by the Maximum Transmission Unit (MTU), and each packet that conforms with the API is called a frame. This leads to some additional overhead that needs to be sent through the serial line, but it also gives more control over the destination and transmission of the packet. When a device receives a frame, it gives, beside the packet, information about what device sent it. For this reason, when sending a frame in this mode, the frame can ask the destination device to send an acknowledgement and if the first device does not receive the acknowledgement in a specified time frame, a retransmission occurs. Every frame has, in the end, a byte with the Cyclic Redundancy Check (CRC). If the CRC fails the frame is silently discarded and, if the frame was one with retransmission, a retransmission is made due to timeout (this process is done by the modem, independent of the master).

For this work, the API mode will be used due to the automatic retransmissions and easier unicast operation. In the API mode there are two types of configuration, one with escape characters support and another without. It will be used the mode with the escape characters to avoid problems when the message includes one of the control bytes (0x7E – Frame Delimiter, 0x7D - Escape, 0x11 - XON and 0x13 - XOFF). The masking is done by adding the byte 0x7D before the conflicting byte and then masking the byte by replacing it with a XOR'd with 0x20 version of it.

### 2.2.5 API frames

All the API frames will adopt the structure represented in the next figure, and all will have escaped characters.

![UART Data Frame Structure - with escape control characters](image)

**Figure 2.16:** XBee API UART data frame structure

The frames begin with an one byte Start Delimiter (0x7E) and end with a one byte checksum. After the start delimiter a 16bit length is introduced in Big-endian format that represents the length of the frame data (does not include the checksum).

The checksum is calculated by summing all the bytes in the data structure, keeping only the last 8 bits of the result and subtracting the result from 0xFF. To verify the checksum, the bytes from the frame data structure and the checksum are added and the sum must be equal to 0xFF.
The Xbee modules used support a large number of API frames (at this moment 18) but in this work only 3 will be used extensively. Those 3 are the ZigBee Transmit Request (section 2.2.5.A), ZigBee Transmit Status (section 2.2.5.B) and ZigBee Receive Packet (section 2.2.5.C).

### 2.2.5.A ZigBee Transmit Request

The Transmit Request frame (with 0x10 as the frame type byte) is the one used to send data to other devices. It can be used as a unicast transmission, where a specific destiny module is defined, or as a multicast.

![ZigBee Transmit Request frame](image)

In figure 2.17 the frame is specified, do note that the number and position of bytes can be altered if the masking of bytes is enabled. This does not alter the order.

- The **Frame ID** byte identifies the UART data frame and it is used to correlate with a future acknowledgement sent from the destination. If set to 0 no ACK is sent from the destination.

- The **64-bit Destination Address** has to be set to the destination address. If set to all zero bits it will address the coordinator (one can also address the coordinator with it’s device address) and if set to 0x0000 0000 0000 FFFF the transmission will be of broadcast nature.

- The **16-bit Destination Address** is the network address, given upon joining the network, and can be set to 0xFFFE if the address is unknown, or if sending a broadcast.

- The **Broadcast Radius** is the maximum number of hops a broadcast transmission can do. If set to 0, it will be set to the maximum hops value.

- The **Options** byte is a bitfield of supported transmission options. All unused and unsupported bits must be set to 0. The supported bits are: 0x01 - Disable ACK; 0x20 - Enable APS encryption and 0x40 to use the extended transmission timeout.

- **Data** are the bytes that are going to be sent to the destination device.

### 2.2.5.B ZigBee Transmit Status

The Transmit Status frame (with 0x8B as the frame type byte) is the one that reports if the packet was transmitted successfully or if there was a failure. It is only generated if the transmit request had a frame ID different from 0 (has ACK enabled).
Figure 2.18: ZigBee Transmit Status frame

In figure 2.18 the frame is specified, do note that the number and position of bytes can be altered if the masking of bytes is enabled. This does not alter the order.

- The **Frame ID** byte identifies the UART data frame and it is used to correlate with the transmit request frame that this status refers to.

- The **16-bit Destination Address** is the network address that is attributed to the destination device. If the packet was not delivered, it is the value provided in the Transmit Request frame.

- The **Transmit Retry Count** is the number of application transmission retries that took place.

- The **Delivery Status** byte indicates if the transmission was successful or, if unsuccessful, the reason for the failure. The codification of the byte is on the Xbee Manual [23].

- The **Discovery Status** byte indicates if there was a discovery, and of there was what kind was it. Discovery can for address (in case the 16-bit network address is not indicated) or route.

2.2.5.C ZigBee Receive Packet

The Receive Packet frame (with 0x90 as the frame type byte) is the message that is sent out the UART when the module receives an RF packet.

Figure 2.19: ZigBee Receive Packet frame

In figure 2.19 the frame is specified, do note that the number and position of bytes can be altered if the masking of bytes is enabled. This does not alter the order.

- The **64-bit Destination Address** indicates the sender’s 64-bit address, but can be set to all one bits if the sender’s address is unknown.

- The **16-bit Destination Address** indicates the network address of the sender.

- The **Receive Options** byte is a bitfield that relates additional information such as if a packet was acknowledged (0x01) of if it was a broadcast packet (0x02).

- **Received Data** are the bytes that were received.
2.3 Sensors

2.3.1 Introduction

A key part of this Thesis is the use of sensors since the Pololu robot is designed to solve line following, therefore, it does not have, natively, the capability to solve free movement problems due to the lack of odometry. To solve part of the problem it was decided to include a sonar and an [IMU] to the robots. The [IMU] solves the problem regarding the orientation and the sonar is used to find other robots or obstacles. Without changing the motors there it is not possible to have odometry or substitute the need of.

2.3.2 Sonar

In this work two sonars from MaxBotix [24] are used. They are the MB1300 [25] and the MB1013 [26] models. Two different models were purchased to test which of them was best for this application, being the beam detection angle and the minimum distance the most significant differences for this kind of application. The main characteristics, common to both of these sonars, are the following:

- 10 Hz parsed reading rate
- 42 kHz ultrasonic sensor measures distance to objects
- Read from all 3 sensor outputs: Analog Voltage, Serial, Analog Envelope/Pulse Width
- Virtually no dead zone
- Firmware filtering for better noise tolerance and clutter rejection

The reading rate cannot be changed as the sonars are hard coded to do some filtering to increase the noise tolerance. This process begins by making some calibrations in the beginning of each cycle, after which twenty measurements are made that are then processed in order to filter noise sources. The latter calculations are the reason of the hundred milliseconds wait for new measurements, as the output is only updated in the end of the cycle. The sonar can function in a free run mode or a real time mode. The difference relies on the free run mode having an additional 2 Hz filter that further enhances the noise rejection, although this means that the sonar is always emitting. The real time mode, besides not having the additional filter, only makes a reading, that consists of 20 measurements, when the sonar is triggered. The triggering is made through pin 4, that is internally pulled high, that initiates a reading if it is brought high for at least 20 microseconds. If left unconnected or held high, the sonar enters free run mode, therefore the pin must be held low before the end of the 100 millisecond cycle if the real time mode is wanted. The statement of the supplier of virtually no dead zone translates on the sonar outputting values lower than the minimum range as minimum range and the same with the maximum range with values farther.

The MB1013 back is depicted in figure [2.20] with the identification of the pins. Although the MB1300 does not have the TTL jumper, the position of the pins is the same. The colour of the Printed Circuit Board (PCB) of the MB1300 is green instead of black.
The reading of both models can be made via analog voltage (pin 3), the one used in this thesis, by a serial output (pin 5), that can be set to TTL mode in the MB1013 via a jumper, or via an analog envelope, in the MB1300, or a pulse width signal in the MB1013 (pin 2). Pin 1 is used for an external temperature sensor, in case of the MB1013, or to control the output of pin 5, in the case of the MB1300, that can be either be set to normal serial use or to send a single pulse without serial data.

As stated previously, the sonars will be read via the analog output. This decision was due to the lack of enough serial ports to use the serial output and both the analog envelope in the MB1300 as the pulse width impulse in the MB1013 need polling. The analogue output does not need pooling because it buffers the last output until a new one is calculated.

One disadvantage of the analogue output is that the precision is lesser than the other methods, being the serial output the most accurate according to the value calculated. The specifications state that a +/-10 mm drift, from the calculated value, can be expected. The difference of the resolution of the analogue reading (table 2.2 has the actual resolution and tables 2.3 and 2.4 the measured) is due to the use of a 10-bit Digital Analogue Converter (DAC) to generate the outputs, that are read by a 12-bit ADC.

<table>
<thead>
<tr>
<th></th>
<th>MB1300</th>
<th>MB1013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>10 mm</td>
<td>1 mm</td>
</tr>
<tr>
<td>Analogue resolution</td>
<td>10 mm</td>
<td>5 mm</td>
</tr>
<tr>
<td>Minimum range</td>
<td>200 mm</td>
<td>300 mm</td>
</tr>
<tr>
<td>Maximum range</td>
<td>7650 mm</td>
<td>5000 mm</td>
</tr>
</tbody>
</table>

Table 2.2: Range and resolution comparison between sonars

The performance of the two types of sonar are very different, while the MB1013 has a better resolution the MB1300 has a greater sensitivity. This increased sensitivity turned out to be a problem due to the low height of the robot and, therefore, of the sonar. This lead to the sonar being able to detect inconsistencies on the surface in which the robot moves, classifying them as targets even after the filtering. The sonar beam diagrams for the MB1013 and MB1300 are available in the respective datasheets [27] [28].

The consistency and accuracy of the results were tested (tables 2.3 and 2.4) and were deemed consistent with the specifications in the case of the MB1013 sonar but with the MB1300 the measured distances was a bit off, about 33 mm. Since this was a static offset, it was easily correct as shown

Figure 2.20: MB1013 pinout
in figure 2.23 with the 33 mm offset, having only negative effect in the readings of distances less than 250 mm. The MB1013 does show an error when the distance is greater than 400 mm but it was deemed acceptable, less than 2 cm, and no corrective action was taken. The comparison of the measured values is depicted in figure 2.22.

<table>
<thead>
<tr>
<th>Distance [mm]</th>
<th>Measurements (10 ms) [mm]</th>
<th>Measurements (100 ms) [mm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>275 +/- 2</td>
<td>275 +/- 1</td>
</tr>
<tr>
<td>250</td>
<td>276 +/- 2</td>
<td>276 +/- 2</td>
</tr>
<tr>
<td>300</td>
<td>299 +/- 2</td>
<td>299 +/- 2</td>
</tr>
<tr>
<td>350</td>
<td>347 +/- 3</td>
<td>348 +/- 3</td>
</tr>
<tr>
<td>400</td>
<td>400 +/- 2</td>
<td>400 +/- 2</td>
</tr>
<tr>
<td>450</td>
<td>441 +/- 3</td>
<td>441 +/- 3</td>
</tr>
<tr>
<td>500</td>
<td>486 +/- 2</td>
<td>486 +/- 2</td>
</tr>
<tr>
<td>550</td>
<td>530 +/- 2</td>
<td>530 +/- 2</td>
</tr>
<tr>
<td>600</td>
<td>586 +/- 2</td>
<td>586 +/- 2</td>
</tr>
<tr>
<td>650</td>
<td>644 +/- 2</td>
<td>644 +/- 2</td>
</tr>
<tr>
<td>700</td>
<td>692 +/- 2</td>
<td>692 +/- 2</td>
</tr>
</tbody>
</table>

Table 2.3: Consistency results for the MB1013 sonar

<table>
<thead>
<tr>
<th>Distance [mm]</th>
<th>Measurements (10 ms) [mm]</th>
<th>Measurements (100 ms) [mm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>206 +/- 2</td>
<td>206 +/- 1</td>
</tr>
<tr>
<td>250</td>
<td>217 +/- 1</td>
<td>216 +/- 1</td>
</tr>
<tr>
<td>300</td>
<td>275 +/- 1</td>
<td>275 +/- 2</td>
</tr>
<tr>
<td>350</td>
<td>316 +/- 2</td>
<td>316 +/- 2</td>
</tr>
<tr>
<td>400</td>
<td>366 +/- 2</td>
<td>366 +/- 2</td>
</tr>
<tr>
<td>450</td>
<td>418 +/- 1</td>
<td>418 +/- 1</td>
</tr>
<tr>
<td>500</td>
<td>467 +/- 2</td>
<td>467 +/- 2</td>
</tr>
<tr>
<td>550</td>
<td>518 +/- 2</td>
<td>518 +/- 1</td>
</tr>
<tr>
<td>600</td>
<td>567 +/- 2</td>
<td>567 +/- 1</td>
</tr>
<tr>
<td>650</td>
<td>615 +/- 2</td>
<td>615 +/- 2</td>
</tr>
<tr>
<td>700</td>
<td>667 +/- 2</td>
<td>667 +/- 2</td>
</tr>
</tbody>
</table>

Table 2.4: Consistency results for the MB1300 sonar

For each type of sonar, two tests were made. Both tests were with 1000 samples but with different sampling times. This was done to check if the readings were stable. The faster rate was meant to test the stability of the ADC and the slower, that is equal to the sonar’s refresh rate, to test the stability of the sonar output. All of the tests were made with a cardboard box as the target and the distance measured from the end of the sonar casing (figure 2.21). In the case of the MB1300, the distance is calculated from the PCB therefore the offset in the datasheet [27] was used to correct the reading.
It must be noted that the NXP LPC1768 processor has 12-bit ADC [20], that is greater than the 10-bit DAC from the sonar, and that leads to a virtual increase in the sonar resolution due to greater sensitivity to noise.

Figure 2.21: Sonar test environment

Figure 2.22: Sonar output comparison
2.3.3 IMU

The IMU used is the Pololu MinIMU-9 \[29\] that packs an L3G4200D \[30\] 3-axis gyroscope and an LSM303DLM \[31\] 3-axis accelerometer and 3-axis magnetometer, both accessible via an I2C interface. The purpose of this unit is to get the robot's orientation via the magnetometer and the gyroscope. The accelerometer was not used because it would need a double integration in order to get position information since those calculations are very sensitive to small errors in the readings, but, it could be used to compensate the odometry prediction in case of the robot being on a plane with a slope. The compensation is required since the control variable of the motors is power, therefore, the torque needed to achieve the same speed is different if the robot is inclined. Since this thesis was developed around the idea of using the robots in a flat plane without a slope, this was not implemented.

As stated in subsection 2.1.3.B the access to this IMU is made via I2C. This interface enables the communication, between the host and the two sensors in the IMU chip, to be made via a single interface. This is possible because the I2C protocol enables several slaves in the same connection with one master, such would not be possible with a serial line. The communication with the IMU is done with a third party library that handles the I2C protocol, the library is called minimu9 and is explained in section 3.1.5.

2.3.3.A L3G4200D 3-axis gyroscope

The L3G4200D gyroscope's sampling rate can be set to one of four values, 100, 200, 400 and 800 Hz, and all of them output each axis at a 16 bit-rate. The output rate chosen is 200 Hz in order to have the same rate as the magnetometer. This is advantageous because the way that the I2C protocol is implemented in the mbed library uses the wait calls that is active wait and, therefore, leads to the system being locked during the communication with the device. The process of communicating with the IMU and the cooperation between both of the sensors, in order to obtain a more precise
Due to the high update speed that this sensor can achieve, a First In, First Out (FIFO) system is implemented in the sensor that can store up to 32 samples for each of the 3 axis. This feature, although being attractive in terms of power efficiency, is not used in this project since the high refresh rate is not needed. When defining the sampling rate of the sensor, the device also sets the cut-off frequency, that can be chosen from four predefined values, of the low pass filter. For this thesis, the maximum frequency allowed was selected, 70 Hz. A high pass filtering can be enabled also, but this was not done since the output is filtered when combined with the magnetometer readings.

The range of the device can also be defined between three values, being those +/-200, +/-500 and +/-2000 degrees per second. Due to the speeds achieved by the robots, theoretical maximum of 720 degrees per second at half speed \[32\], the maximum range was chosen, which gives a typical sensitivity of 70 millidegrees per second per digit. Unfortunately, the speeds that the robots can achieve turn the calibration task difficult and it was not possible to overcome this difficulty with the available resources. One possible way to solve the calibration problem is to use a high speed camera and measure the number of rotations made in a given time frame.

Despite the lack of calibration, it was decided that the typical values presented in the datasheet would suffice given that these readings will be used in conjunction with the magnetometer readings. The following tables \[2.5\] and \[2.6\] have the readings of two different samples of the IMU attached to a robot while it was spinning counterclockwise. For each power level, 3000 samples were captured.

<table>
<thead>
<tr>
<th>Motor Power (%)</th>
<th>X axis (degrees.s(^{-1}))</th>
<th>Y axis (degrees.s(^{-1}))</th>
<th>Z axis (degrees.s(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.11 +/- 1.28</td>
<td>-0.17 +/- 1.50</td>
<td>-1.10 +/- 0.25</td>
</tr>
<tr>
<td>10</td>
<td>7.26 +/- 1.47</td>
<td>-0.74 +/- 3.85</td>
<td>107.71 +/- 5.63</td>
</tr>
<tr>
<td>15</td>
<td>13.68 +/- 2.80</td>
<td>-1.34 +/- 5.23</td>
<td>200.35 +/- 4.63</td>
</tr>
<tr>
<td>25</td>
<td>24.42 +/- 5.08</td>
<td>-2.23 +/- 7.91</td>
<td>354.57 +/- 4.85</td>
</tr>
<tr>
<td>50</td>
<td>52.37 +/- 10.34</td>
<td>-3.93 +/- 17.56</td>
<td>768.01 +/- 6.41</td>
</tr>
<tr>
<td>100</td>
<td>81.71 +/- 25.4858</td>
<td>5.48 +/- 38.53</td>
<td>1554.42 +/- 9.41</td>
</tr>
</tbody>
</table>

**Table 2.5:** Gyroscope A output when robot rotates counterclockwise

<table>
<thead>
<tr>
<th>Motor Power (%)</th>
<th>X axis (degrees.s(^{-1}))</th>
<th>Y axis (degrees.s(^{-1}))</th>
<th>Z axis (degrees.s(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.90 +/- 0.23</td>
<td>0.92 +/- 0.22</td>
<td>-0.75 +/- 0.39</td>
</tr>
<tr>
<td>10</td>
<td>-3.11 +/- 1.26</td>
<td>3.87 +/- 3.95</td>
<td>108.00 +/- 5.75</td>
</tr>
<tr>
<td>15</td>
<td>-5.10 +/- 2.35</td>
<td>6.32 +/- 5.25</td>
<td>200.96 +/- 4.82</td>
</tr>
<tr>
<td>25</td>
<td>-8.46 +/- 4.33</td>
<td>10.67 +/- 7.84</td>
<td>355.43 +/- 4.64</td>
</tr>
<tr>
<td>50</td>
<td>-18.64 +/- 10.32</td>
<td>22.41 +/- 16.99</td>
<td>769.48 +/- 5.47</td>
</tr>
<tr>
<td>100</td>
<td>-62.86 +/- 26.59</td>
<td>51.59 +/- 41.47</td>
<td>1558.17 +/- 9.74</td>
</tr>
</tbody>
</table>

**Table 2.6:** Gyroscope B output when robot rotates counterclockwise

The axis that is important for this thesis is the Z. The other two are not used and, in an ideally situation, should be zero. This does not occur because of the way the IMU is installed that uses the connectors as support so that the IMU can be easily changed. This feature results in some slack being present in the connectors that act as support leading to the ability to have free movement on those axes, which explains the increase of the mean alongside with the motor power. The variance
(figure 2.24b) of those axes also has a significant increase which can also be explained by the free movement since it is erratic due to being highly dependable of the surface, on which the robot is rotating, imperfections. Such effect is not noticed in the Z axis that has an almost static variance with a very small increase with the motor power. This variance is also small when compared with the actual measurement of speed.

(a) Gyroscope A and B measurement means

(b) Gyroscope A and B measurement variances

**Figure 2.24:** Gyroscope A and B measurements statistical analysis
As said previously, the gyroscope calibration was not possible, therefore the analysis of the output can only regard known factors such as the DC motor linearity and the theoretical half speed rotation of 720 degrees per second. The half speed measured can be retrieved from the tables 2.5 and 2.6, being 768.01 deg.s\(^{-1}\) and 769.47 deg.s\(^{-1}\) the values for the gyroscope A and B respectively, resulting in a relative error of 7% on both occasions. Since the actual value for the robot speed cannot be ascertained, no correction was made to the measurements of the gyroscope. The linearity can be confirmed with figure 2.24a where both of the Z axes responses can be observed having a linear response after the 10% motor power. The line does not start at zero because the robot can only start movement at a certain motor power, as discussed in section 2.1.2.

2.3.3.B LSM303DLM 3-axis accelerometer and 3-axis magnetometer

As stated before, the accelerometer present in the LSM303DLM is not used in this thesis, therefore, no analysis or tests were made with this sensor part. The range of the magnetometer can be set to one of seven predefined values by changing the input gain applied by the sensor. The sensitivity of the axes differ, having the X and Y axes the same sensitivity but differing from the Z axis as shown in the following table.

<table>
<thead>
<tr>
<th>Sensors input range [Gauss]</th>
<th>Gain for the X/Y axes [Gauss(^{-1})]</th>
<th>Gain for the Z axis [Gauss(^{-1})]</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/-1.3</td>
<td>1100</td>
<td>980</td>
</tr>
<tr>
<td>+/-1.9</td>
<td>855</td>
<td>760</td>
</tr>
<tr>
<td>+/-2.5</td>
<td>670</td>
<td>600</td>
</tr>
<tr>
<td>+/-4.0</td>
<td>450</td>
<td>400</td>
</tr>
<tr>
<td>+/-4.7</td>
<td>400</td>
<td>355</td>
</tr>
<tr>
<td>+/-5.6</td>
<td>330</td>
<td>295</td>
</tr>
<tr>
<td>+/-8.1</td>
<td>230</td>
<td>205</td>
</tr>
</tbody>
</table>

Table 2.7: Magnetometer input range and gain

The output of the sensor is a 12-bit signed integer, range from -2048 to 2047, that must be divided by the gain in order to get the output in Gauss within the sensor range defined. In this thesis, the range used for the magnetometer is the smallest since it’s purpose is to measure the earth’s magnetic field, which is around half a Gauss [33].

The data rate of the sensors can be configured between eight minimum output rates, being those 0.75, 1.5, 3.0, 7.5, 15, 30, 75 and 220 Hz. The highest frequency, 220 Hz, was chosen in order to have the lowest possible delay when retrieving the orientation. At this rate, the orientation of the robot, if asked directly to the sensor, will have a maximum delay of 4.5 ms if the I2C communication delay is discarded.

Regarding the accuracy of the values given by the magnetometer, nothing can be ascertained since it was not possible to access a calibrated magnetometer. Although, for this thesis, the actual value of the magnetic field is not important since the orientation is given by the relation of the X and Y values as explained in section 3.3.2 (the Z axis is not be used). Despite not knowing if the values were correct, some tests were made to ascertain the quality of the readings in terms of noise presence. As with the gyroscope, since both of the sensors are in the same [MU] package, two different samples
were tested.

For testing the magnetometer, a full rotation composed by 148 steps with 100 measurements per step was executed by the robot with the IMU installed (figure 2.25). This procedure was performed in four different points, two being on the floor and the other two on top of a table. All of the four tests were made inside a building.

(a) Magnetometer A readings

(b) Magnetometer B readings

Figure 2.25: Magnetometer A and B readings of X and Y axes
When testing the magnetometers, the same starting position was used in order to have a better comparison between both of the samples, even though this is not necessary to test the presence of noise since the results are without any calibration at this stage. To obtain orientation information from this readings, some calibration is needed, as it is explained in section 3.3.2. By comparing the situations of magnetometer A and B one can conclude that the phase of the magnetic field waves are very similar but have different offsets and amplitudes. Only on the last case, the second one on the table, the readings show a large presence of noise and suffer distortion on the end of the rotation. This is explained by the presence of iron [34] on that section of the table, being this the reason for choosing that spot for testing. It is for this reason, the distortion in the presence of metal objects, that the gyroscope is used. Although it does not give the direct orientation, it can be used to detect when there is distortion, due to iron, if the outputs are not coherent.
This chapter is focused in explaining the software that was available to start with and the one that was developed to create the system for the robots. It also explains the way that the sensors are read.

3.1 Operating System and Libraries

3.1.1 Introduction

The programming of the mbed board is done online, as stated in 2.1.3.A, and the supported language of the compiler is C/C++. The mbed platform is supported by an open-source Software Development Kit ([SDK][35]) that handles the startup code, C runtime and libraries needed to compile the code. This [SDK] simplifies access to the hardware peripherals by creating a high level [API][35] for each of them, such as for the [ADC][35], serial and i2C interfaces. Since the mbed [SDK] only, directly, supports the basic access to the mbed hardware, additional libraries were created such as the mbed Real Time Operating System ([RTOS][35]) to give multiprogramming capabilities and the m3pi library to control the robot. Thanks to the mbed platform being open source, several other libraries were created by the community in order to offer a high level access to additional hardware such as sensors. One of these libraries is used in this work to handle the access to the [IMU][35] and it is called minimu library [36].

3.1.2 mbed [SDK]

As said in the introduction, the mbed [SDK] gives the programmer a few libraries that simplifies the access to the hardware. There is also a pin variable (called PinName) that enables access to the pins by name, for example, to reference the first LED on the mbed board one uses the variable `LED1` and to reference pin 20, `pin20` is used. The libraries documentation can be accessed in the mbed handbook [37].

One of the most used libraries is the one that controls a pin as a digital output. This is used to send reset signals to devices and control the LED's on the mbed board. To control a pin as a digital output, the pin must be declared as such (via the DigitalOut library) and after that the variable created can be used as an integer in operations like setting the value and comparisons since the `=` operation is overloaded. This variable only accepts the values 0, for setting the output to low, and 1, for setting the output to high.

Another library used is the one that controls the [ADC][35], one of these will be used to read the sonar values. After setting a pin, the pin must support analogue input, to act as an analogue input, it can be read as a single precision float normalized by the overloaded `=` operator. The variable cannot be set and the normalized value is between 0 and 1.0. There is also a function that can be used to read the variable as a 16-bit unsigned integer, although only the first 12 bits will matter since that is the [ADC][35]'s precision.

The serial interface simplifies the setup of the link by asking only for the transmit and receive pins. By use of functions the baud rate, packet length, parity and stop bits can be set. The use of a flow control method can also be configured by a function where it is indicated the type of flow control and pins used. The flow control methods supported are Request To Send ([RTS][35]), [CTS][35] or
both. The variable created for the serial interface behaves like a file descriptor, enabling the use of Input/Output (IO) functions such as `printf` and `getc/putc`. It is also possible to query the variable if there are bytes available to be read or space available to write bytes into the interface, since the Universal Asynchronous Receiver/Transmitter (UART) buffer is limited. The best way to ensure that the UART buffer is not overrun is to implement Interrupt Service Routines (ISRs) for receiving and sending through the serial line that are launch as soon as there are bytes to read or space to write the bytes to. It must be noted that if the RTOS library is used, the `printf` and `getc/putc` functions cannot be used inside the ISRs due to them using synchronization primitives that lead to a lock out. Therefore, for this case, the reading and writing to the UART must be made directly through the accordingly registers.

### 3.1.3 RTOS Library

The RTOS library [38] gives multitasking capabilities to the mbed platform at the expense of losing some of the available ram. The exact amount of ram needed is dependent of the version of the library since it depends on the maximum number of threads and their stack size and those numbers may change between versions (the maximum number of threads increased at least once during the development of this work).

The implementation is based on the CMSIS-RTOS API open standard [39]. Thanks to this library the mbed platform gains access to a maximum of 14 threads (in the case of the board used, with the LPC1768 processor) where one is the main thread. There is an additional thread, that is not included in the limit, called `osTimerThread` that is responsible for controlling the timers. The thread pool is managed with a round-robin scheduler with a 5ms time slice.

The library has the most used synchronization primitives such as mutexes and semaphores, along side the support for thread signals. Another synchronization system offered by the library is a queue that has a maximum size defined upon creation. This size cannot be altered and there are only two operations on the queue, one to put an object and one to retrieve. Both of this methods can have a timeout since the operations will lock the thread if the queue is full or empty if putting or getting from the queue.

The library also has a wait function that yields the execution of the thread for the time passed through the parameters in milliseconds. This method is preferred over the normal wait functions as those use active wait, as in wasting processor cycles until the time has passed, keeping the thread in active state instead of enabling other threads to run.

Another functionality given by this library is the use of timer functions that execute outside the scope of an ISR they run in the `osTimerThread`, enabling the use of the functions that cannot be run in an ISR. The timer functions can be configured to run cyclical or only once after a delay.

### 3.1.4 m3pi Library

The m3pi library [40] enables the control of the robot via serial interface with the microcontroller in the lower part. The library has functions to control the wheels individually or simultaneously if the
desired movement is a straight line or on spot rotation. All of these commands are given with normalized values between -1.0 and 1.0. In the case of the simultaneous wheel control functions, the library simply sends two single wheel control commands. Therefore, there is not a significant advantage in terms of speed that might affect the orientation of the robot since one wheel will get the command before the other. There is also a command that will stop the robot, this is once again two control commands, one for each wheel, but set to 0. The other commands used are the ones regarding the LCD. There is a command to set the column and line where the following print command, that receives a string, will apply.

3.1.5 Minimu9 Library

The minimu9 library [36] acts as middleware to two sensors connected via the same I2C channel. The supported sensors are the LSM303DLM[31], a three-axis accelerometer and three-axis magnetometer, and the L3G4200D[30], a three-axis digital output gyroscope, that are present in the Pololu MinIMU-9 package [29].

The library handles the I2C communication and offers functions that return the values in raw, as given by the sensor, or converted, as specified in the datasheet, to the correspondent units such as gauss [G] for the magnetic field, meters per squared second [ms\(^{-2}\)] for the acceleration and degrees per second [\(\text{deg.s}^{-1}\)] for the rotation speed. It is also possible to read the temperature value given by the L3G4200D.

Another functionality of the library is the simplification of configuration of the sensors. There is an object for each of the two sensors and within that object there is a function that accepts a register and a value. The function, when called, manages the I2C channel communication, therefore there is no need for the developer to know the protocol, and since the function is specific for one of the sensors the I2C address is already known, and thus, all the data necessary for changing the value of the register is available. In the header file regarding each sensor object, there is a list with each of the registers name and addresses. The developer only needs to know the name of the register he wants to change and what value corresponds to the behaviour the sensor should have.

3.2 Communications

3.2.1 Introduction

To handle the communications, it was decided to separate the system in three parts. One to handle the reception of data, one to handle the transmission and another to process the messages. An [API] was also created according to the project needs.

The communication is done via a Xbee module connected via a serial line at 38400 bits per second. The format used is eight bits for data and one for the stop bit, leading to a data rate of 4.3 bytes per millisecond. Since the Zigbee receive packet [2.2.5.C] is at least 16 bytes long, the reception of the packet in the UART buffer will take at least 3.72 milliseconds. With the processor at 96 MHz, it was decided to use an ISR to handle the UART buffer in order to avoid wasting processor time.
cycles.

The parts regarding the reception and the transmission are handled almost in the same way, one ISR and one thread with a circular buffer to store the bytes being exchanged between them. The circular buffer has a semaphore that holds the same number of tokens as there are bytes. This was done in order to lock the thread, in a waiting state, if the buffer is full or empty whether the thread is the one responsible for sending or transmitting. The processing of the messages is done by a pool of threads in order to be possible to have an action, that may last a few seconds or more with wait periods, alongside with a query for a robot status that can be processed in a few milliseconds.

The pool of threads, and other things discussed in the following subsections, use a queue object that is available in the RTOS library (discussed in 3.1.3). In each case, the queue object will be initialized with the maximum number in order to the method of inserting something to the queue not being a blocking call.

### 3.2.2 Transmission of data

The transmission of data is done via an ISR that is attached to the UART of the serial line connected to the XBee, transmission interrupt that is activated when the transmit buffer is empty. The bytes waiting to be transmitted are stored in a circular buffer that is consumed by the transmit ISR and is filled by the transmission thread.

![ISR attached to serial transmission interrupt](image)

Figure 3.1: ISR attached to serial transmission interrupt

The ISR was implemented in order to avoid an active wait when the UART buffer is full. In this way, when the buffer is full the system is free to use the processor for other things while the buffer is being emptied. As figure 3.1 shows, while there are bytes available in the circular buffer and the UART buffer is empty, the ISR continues to move bytes from one buffer to the other.

In addition to the flow control into the UART buffer, an additional one between the CPU and the Xbee was implemented. The purpose was to avoid the discarding of incoming bytes by the XBee when it’s buffer is full due to having to store each frame until it receives the respective acknowledgement or the maximum retransmissions have occurred. Since it’s the incoming of bytes to the XBee that
must be controlled, a CTS control as implement. Despite being implemented in the mbed library, the control flow is not yet fully compatible with the RTOS library, leading to the necessity of manually implementing this feature. This was done via pooling the CTS output from the XBee after checking if the UART buffer is full but before transferring the byte to it. If the CTS output is held high, the transmission ends. In order to be able to start again the transmission, an ISR was created in order to manually start the ISR depicted in figure 3.1 when the CTS output transits from high state to a low state.

As mentioned before, the circular buffer is filled by a thread. This thread receives the frames from a queue of Transmission Slots (txSlots). The txSlot is an object that stores a ready to be transmitted frame, that is stored in a dynamically allocated buffer, the current length of the frame in the buffer, a pointer to the thread that awaits confirmation, the number of retransmits and an ID.

![Figure 3.2: Data structure of the txSlot used for each transmission](image)

The maximum size, without taking in account masking, the buffer can achieve is determined by the Xbee MTU that with the configuration used is 75 bytes, and the frame overhead of 18 bytes, giving a total of 93 bytes as the maximum packet size in the API mode. Since masking is enabled, it is impossible to know the actual size of the buffer before the creation of the frame, leading to a theoretical maximum of 184 bytes (only the first and last byte of the frame are free from masking, and masking adds an additional byte for each masked). For this reason, it was decided to do the masking when copying the frame to the circular buffer, enabling the possibility of dynamically allocate the buffer, in order to save RAM, since it is now possible to know the frame size before hand. The actual amount of memory used, therefore allocated, by the buffer for a frame is indicated in the length variable. The confirmation thread pointer is used to signal threads about the state of the packet such as if it was received by the target or if the target responded. It's use is discussed in section 3.2.4. The retransmits byte stores the number of retransmits possible, being incremented in each retransmission. If it reaches 3 the frame is discarded and if it is -1 retransmissions are not possible. This is also discussed in section 3.2.4. The ID byte has the identification of the slot according to the position in the txSlot's array.

The txSlot's array has 10 elements and the ID starts at 1 instead of 0 as used by the programming language. This is due so that the ID can be used as the ZigBee frame ID when an ACK is desired (as explained before in section 2.2). The correspondence, alongside with the thread pointer, is used to signal waiting threads if the transmission was successful as explained, again, further in section 3.2.4. In order to distinguish the txSlots that are free from the ones that are in use, two separate queues were created. One for the free ones and another for the in use. The thread responsible for filling the circular buffer will acquire the txSlots from the in use queue. The txSlots are created by calling a function where it is indicated the destination, with the two addresses, the payload and it's length, if an
ack is required and, optionally, the thread to signal if that is the case. This function creates a transmit request (section 2.2.5.A) frame with the given payload, handling the masking of the bytes, if needed, and, in the end, inserts the txSlot in the in use queue.

Figure 3.3: Thread responsible for filling the tx circular buffer

As demonstrated by figure 3.3, the thread responsible for filling the transmission circular buffer is at a wait state until there is a txSlot in the queue that stores the in use slots. This is done so that no processors cycles are wasted when there is nothing to transmit. When a txSlot becomes available, the first thing that is done is to check if the circular buffer is empty. This is done because the ISR only fires when the buffer becomes empty. Therefore, if the buffer is already empty, the interrupt will not be fired and a manual transmission, of one byte, must be made. It is for this reason that the check is made before copying the frame to the buffer but the reaction is only after the frame is in the buffer. The filling of the circular buffer is made with the respective semaphore representing the number of free bytes. In this way, the thread consumes the semaphore, therefore is liable to be blocked and enter a wait state, while the ISR acts like a producer, never blocking.

Before coping each byte, a token must be consume from the semaphore to avoid overrunning the circular buffer. During the copying of the frame, each of the bytes is checked if they need masking, being dealt with accordantly, except for the start of frame byte and the CRC byte. After the frame is copied, the thread reacts if the buffer was previously empty, no action is done if it was not. The reaction is a manual jump start of the ISR responsible for the transmission. This is done by disabling the interrupts and take one byte, and updating the semaphore accordantly, and put it in the Transmitter Holding Register (THR) of the UART in which the Xbee is connected. After this, the interrupts are enabled and, because of the transmission of that one byte, the interrupt flag for transmission is active,
leading to the circular buffer being consumed. After the manual jump start, or the lack of it, the retransmits counter is incremented or, if the retransmissions were not enabled, the txSlot is discarded and put in the queue that stores the free ones.

### 3.2.3 Processing the messages

For processing the messages sent from other devices, a pool of four threads, called *ProcessFrames*, was created. The pool is managed via a queue object that stores the threads that are free at the given time. The data structure that is used by these threads has two additional data fields beside the thread pointer and is called *ProcessThread*. One is the buffer field and has the same length as the one in the txSlots. The other is the length field that will store, as it is with the txSlots, the actual amount of bytes in use. The pool is consumed by the receiving thread, discussed in 3.2.4, where the buffer will be filled and the length data field will store the actual length of the reconstructed and unmasked frame.

![Figure 3.4: Process Frame Thread - responsible to process each frame](image)

The thread, represented in figure 3.4, is an infinite loop thread that is locked, by waiting for a signal, at the start. The thread has three main states: the start, in which it waits for a signal; the processing of the frame and the release of the thread by inserting it in the free queue once again. The start
signal is sent only by the receiving thread after the *ProcessThread* data structure, corresponding to this particular thread, is removed from the pool. Since the structure is only inserted in the pool during the initialization and before being locked again in the end of each cycle, it can be guaranteed that the pool only has free threads in it.

The processing of the frame starts with identification of the frame type. In this work only two frames will be processed while the others will be ignored due to not being relevant. The two types that are handled are the *TxStatus*, that corresponds to the ZigBee Transmit Status frame (2.2.5.B), and the *RxPacket*, that corresponds to the ZigBee Receive Packet frame (2.2.5.C).

The *TxStatus* type of frame has the information regarding the delivery of a previously sent ZigBee Transmit Request frame (2.2.5.A). The status of the transmission is given by a one byte code and two of the possible states were singled out. Those were the Network ACK Failure and the Success states. All other states were treated as errors without specific handling. Both in the Success and the default state, regarding the other errors, the handling is almost the same, deferring only on the type of signal sent, if any.

In the status frame, besides the status code, there is the frame ID that correlates to an ID of a sent frame (2.18). As said earlier, the ID is the position of the *TxSlot* (3.2) in the respective array, and so, it can be directly accessed in order to check if there is the need to send a signal. The check is made by verifying if the thread pointer is in use, in which case a signal is sent being one of success if that was the case, or one of error otherwise.

The difference of the Network ACK failure case relies in the possibility of putting the *txSlot* once again in the queue that stores the *txSlots* waiting to be transmitted. This is done by checking if the number of retransmissions, the number is in the *txSlot* data structure, is over the retransmission limit (the limit used is 3). If it is not over, the *txSlot* is inserted in the in use queue, otherwise, the handling is the same as the one described earlier for the error case.

The other type of frame that is handled is of the *RxPacket* type. This type of frame (2.19) is the recipient equivalent of a transmit request (2.2.5.A) and is processed according to the API described in 3.2.5. The information retrieved from this type of frame, besides the data transmitted, is the address of the sender.

### 3.2.4 Receiving Data

The system for receiving data is composed by an [ISR](#) that is fired when a byte is available in the UART buffer, and a thread. The [ISR](#) is responsible to move the bytes from the UART buffer to a circular one in the program memory. This buffer is then consumed by the thread.
The semaphore linked to this circular buffer represents the number of bytes available, to be read by the thread, in the circular buffer. This contrasts with the previous case, since it is again needed that the ISR acts as a producer in order to avoid a dead lock by locking inside an interrupt. The ISR routine is demonstrated in figure 3.5.

The receiving thread is the one that is responsible to transform the received bytes into a message and dispatch one of the idle working threads available in the pool. As described in 3.2.3, each of the ProcessThread structure has a static buffer that will be used by the receiving thread to store the reconstructed frame.

Figure 3.5: ISR attached to serial reception interrupt

The receiving thread - responsible for reconstructing messages from received bytes

Figure 3.6: Receiving Thread - responsible for reconstructing messages from received bytes
After getting a free ProcessThread, the thread tries to get a Start byte (0x7E) from the circular buffer. This, the discarding of bytes until a Start byte is read, is done in case there is a buffer overrun, whether it is the UART or the circular buffer, and the next available byte after a full frame is read is not the Start byte. This is the only way this might happen, since there is only one consumer, this thread, and one producer, the ISR, therefore, no synchronization problems can occur. After getting the Start byte, the rest of the frame is read. During the reading, the unmasking of the frame is done, storing in the buffer the unmasked frame and the CRC is calculated. The unmasking is not needed when waiting for the start byte because that byte is one of the protected by the masking. After the frame is complete, the CRC is checked by adding the local CRC value with the CRC value in the frame. If the result is different than 0, there as a corruption of the message. Since the Xbee is working in API mode, the only way that the CRC fails is when there is a buffer overrun. During the testing, there was no case of buffer overrun. If there was, one solution would be the increase of the circular buffer, if that was the overran buffer, or an increase in the serial connection speed between the processor and the Xbee. If the CRC checks out, the ProcessThread is started via a signal and the thread goes back to it's initial state.

3.2.5 Implemented API

As stated in the introduction, section 3.2.1, an API was created according to this thesis needs. Since the Zigbee network already solves the problem of the transmission including the addressing and error checking, as discussed in section 2.2. For this reason, the API created only needed to include the actual data since the overhead is already present in the frame. This implemented protocol only has two components, the identification of the command, one byte, and the additional parameters that depend from the command, such as distance and orientation angle. Although is common practice to use big endian notation when networking, this protocol uses little endian since this is the endianness of both the robots and the computers used. Even though a computer is used to control the robots, all of the commands can be sent directly by a robot, however, not all of them have been implemented in the robots due to not being used.

The most important messages of this protocol are described in the following lines, existing other messages that are not described here due to not being used in the final version of this work or they are solely used for debugging purposes. Although the value is not specified, all of the messages, being them queries or responses, have a unique byte in the beginning of the payload to identify it. When orientation is specified it is in radians, when distance is specified it is in mm and when speeds are specified they are in rad.s⁻¹ or mm.s⁻¹, if they are rotation or linear speed accordingly.

- **Orientation Query** - Only the ID is sent. This will generate a response from the target with it's current orientation.
- **Orientation Response** - This message has the current orientation of the robot as a single precision float (4 bytes) besides the message ID.
- **Sonar Query** - Only the ID is sent. This will generate a response from the target with it's
current sonar reading. This reading will be taken with the trigger mode, therefore, the response
generation will take at least 100 milliseconds.

• **Sonar Response** - This message has the current sonar reading of the robot as a single precision float (4 bytes) besides the message ID.

• **Following Command** - This message warns the target robot that we, the entity that send the message, are following it. This will make the target robot warn this entity when and how it moves. Besides the ID, the payload includes another byte that activates (with value 1) or deactivates (with value 0) this feature.

• **Following Response** - This is the message that is generated, and sent to all the following entities, when a robot moves. The payload consists, besides the ID byte, three single precision floats, with 4 bytes each, with the x offset, the y offset and the current orientation of the robot. The offsets are calculated according to distance traveled and orientation. Although each robot has it's own referential, the orientation of all of them is the same.

• **Rotate Command** - This message asks the target robot to rotate to a given orientation. The payload includes, besides the ID byte, two 4 byte single precision floats with the intended orientation and maximum speed to use during the rotation.

• **Rotate Response** - This is the message sent after the rotation is completed. Besides the ID byte, the payload includes a 4 byte single precision float with the robot's orientation after the rotation.

• **Move Command** - This message asks the target robot to move in a straight line. Besides the ID byte, the payload includes two 4 byte single precision floats with the distance that the robot is expected to cover and the maximum speed that it can achieve. If the speed is negative, the robot will move backwards.

• **Move Response** - This is the message sent after the robot moves. Besides the ID byte, the payload includes a 4 byte single precision float with the distance traveled. This is an estimate value due to not having odometry on the wheels.

• **Follow Command** - This message is used to ask the target robot to follow another one. For this to work properly, the following command must be issue by the target robot to the robot to follow, this is done in the end of the search command. The payload includes, beside the ID byte, the identification of the robot to follow, via the 16-bit unique address, and three 4 byte single precision floats with the speed, distance and orientation at which to follow the robot. The orientation is such that at 0 degrees the follower robot will be in front of the followee, 90 degrees the follower will be at the left of the followee and at 180 degrees the follower will be at the back of the followee.

• **Follow Response** - This message is sent in the end of the follow command. Besides the ID byte, the payload also includes a 4 byte integer with a status code.
• **Search Command** - This message is used to ask the target robot to search for another robot. The payload includes, beside the ID byte, 10 bytes with the identification of the robot to follow, the first 8 bytes correspond to the 64-bit address and the other 2 bytes to the 16-bit address.

• **Search Response** - This message is sent in the end of the search command. Besides the ID byte, the payload also includes a 4 byte integer with a status code.

• **Stop Command** - Stops the target robot.

• **Calibrate Command** - Asks the robot to calibrate the compass.

• **Calibrate Response** - This message is sent in the end of the calibrate command. Besides the ID byte, the payload also includes four 4 byte integer with the calibration constants (AMP_X, AMP_Y, OFFSET_X and OFFSET_Y).

• **Set Calibration Command** - This message sets the four calibration constants (AMP_X, AMP_Y, OFFSET_X and OFFSET_Y). The payload, besides the ID byte, includes four 4 byte single precision float with the aforementioned constants.

• **Target Position Query** - This message asks the position of a robot. The payload, besides the ID byte, includes the 16-bit unique address as the identification of the robot.

• **Target Position Response** - This message has the position of the robot asked in the query. Besides the ID byte, the payload includes three 4 byte single precision floats with the x and y position and the orientation of the asked robot. If unknown, each of the byte fields will be at 0xFF.

• **Set Target Position Command** - This message sets the position of a robot in the target robot. Besides the ID byte, the payload includes the 16-bit unique address as the identification of the robot and three 4 byte single precision floats with the x and y position and the orientation of the robot.

• **Set Target Position Response** - This message is sent after setting a target robot position. Besides the ID byte, the payload also includes a 4 byte integer with a status code.

Some of the responses return a status code. This code can represent several errors, besides the success status, such as a system error, communication error or an error in finding the target.

### 3.3 Reading the Sensors

#### 3.3.1 Reading the Sonar

The reading of the sonar measurement is very straightforward since it’s analogue output will be used, as stated in section 2.1.3. The sonar has two connections with the processor, one of which is connected to the analogue output. To read it’s value, that port, pin 20, must be configured as an ADC input. To do this, only must initiate the pin as one and ask for a reading when needed. The
other connection is used to control if the sonar acts in free run mode or in trigger mode as explained in section 2.3.2. This pin, pin 19, will be configured as a digital output and be set to low after being configured. This will disable the sonar.

To make an actual reading, the sonar control pin is set to high for 10 ms and then it is set to low. After 90 ms, adding to the 100 ms delay necessary for a new reading, the actual reading is made through the ADC and it will be in the form of a normalized floating number. For this reason, the actual distance is obtained by multiplying the measurement by a gain and, if needed, adding an offset. In the case of the MB1013, the offset used is the one from the datasheet and is needed because of the reference for the reading for this sonar being its PCB and not the front of the sonar. In the case of the MB1300, the offset is due to a constant drift observed while testing the sonar. It must be noted that all readings in trigger mode will block for at least 100 ms. All readings mention in this thesis must be considered to be in trigger mode unless stated otherwise.

<table>
<thead>
<tr>
<th>Sonar Model</th>
<th>Gain [mm]</th>
<th>Offset [mm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB1013</td>
<td>512</td>
<td>-15.5</td>
</tr>
<tr>
<td>MB1300</td>
<td>1024</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 3.1: Sonar models gain and offset

### 3.3.2 Getting readings from IMU

Both of the sensors that are used in this work, the magnetometer and the gyroscope, are accessed via a single I2C link via the miniumu9 library (section 3.1.5). The purpose of using both the magnetometer and the gyroscope is to increase the robustness to magnetic interference, that affects the magnetometer, by trying to estimate the rotation of the robot with the gyroscope. This will be done with a Kalman filter [7] as described later in this section.

The measurement of the Z axis given by the gyroscope is already in an usable format, degrees per second, and it only needs to be converted to radians per second. This conversion is necessary since all of the trigonometric functions, implemented in the C language, use radians as their unit. A small correction to the value obtained via I2C is made, the zero value is corrected by adding an offset calculated by measuring the output when the robot is still.

The magnetometer readings, on the contrary to the gyroscope readings, are not in an usable format to estimate the orientation. Despite this fact, the orientation can be estimated by comparing the relative strength of the magnetic field in each of the axes since the two axes, X and Y, are perpendicular to each other. If considered that the magnetic field measured is the earth's, as in no distortion or electromagnetic noise is present, when one axis is at a peak, maximum or minimum, the other one will be zero since the first axis will be completely aligned with the earth's magnetic field. Difference between the maximum and minimum situation lies with the direction of the field, therefore, if the axis is facing the north or south magnetic pole. An important remark is that the magnetic poles are different than the actual geographic poles. In this work, this discrepancy is not corrected since the importance of heading in that all robots have the same calibration. As long the north is the same for all robots, the difference between the magnetic and geographic poles can be ignored.
The first step to accomplish the conversion from magnetic field readings to a heading is the calibration of the sensor in order to normalize and zero centre the readings. This is done, in a not so quick but still dirty way, by making readings while the robot does a full rotation in 148 steps, averaging five samples each step in order to get a reading (figure 3.7a). After the full rotation is made, the mean value of each of the axes curves is calculated. This generates one offset for each axis that is used to correct the zero value of the curves. The next step is the normalization process, which is made by searching for the amplitude of the sinusoids and using those values to normalize the readings. In the end, we have normalized and zero centred sinusoids (figure 3.7b) with each one having two calibration values: one for offset and the other for amplitude. After this process, if no magnetic interference is present, the model state Y/X should be approximated to a circumference where, if in presence of interference, the result would be an ellipse (figure 3.7c).

The conversion to heading can now be made with the use of the arctangent function with two arguments in order to obtain the result in the proper quadrant. In computer languages this function is called atan2 [41] and returns positive angles when the y argument is positive. The thesis uses the anticlockwise notation, therefore, due to the placement of the inertial axis (figure 2.2), the y value used in the atan2 function must be inverted, after being zero centred and normalized, in order to preserve the anticlockwise nomenclature (figure 3.7d).

![Figure 3.7: Magnetometer to compass: calibration and conversion](image)

As previously stated, a Kalman will be used to agglomerate the IMU readings in order to improve
the resistance to noise sources due to filtering. The filter will be updated inside a library timer, acting as an interrupt without the inconveniences of an ISR. The period of this timer will be of 5 ms. The Kalman filter models are \( x_k = A x_{k-1} + B u_{k-1} + w_k \) for the state variable and \( z_k = H x_k + v_k \) for the measurement with the following parameters:

- \( A = 1 \) because the state is a scalar (the heading) and it does not change from step to step without an outside influence (robot moving).
- \( B = 5\text{ms} \) because the control variable is the gyroscope value in degrees per second. The 5ms value is the interrupt period, therefore, the amount of time the robot rotated at that speed (approximately).
- \( H = 1 \) because the measurement, given by processing the magnetometer readings, is directly mapped to the actual value.
- The variances of the gyroscope and the compass readings are used in the filter, being multiplied with \( 25\mu\text{s}^2 \) to convert to the interrupt period. They are considered constant along time.

### 3.4 Routines

#### 3.4.1 Introduction

In this thesis, a movement, or a series of movements, that must be executed in order to achieve a certain goal is called a routine. For this reason, only one routine can be executed at each time. Therefore, if a routine is started when another one is running, the older one is terminated since it is considered that the most recent takes precedence. The routines that can be executed by the robot will be discussed along this section and all of them can be executed directly via a wireless command. The more basic ones, move and rotate for example, are also used in the more complex routines, such as the search, where they are not considered as a separate action.

The way that the routines are managed is by the form of a thread. There is a global thread pointer that stores the current routine’s thread, if any, that is cleared when the routine terminates. When a new routine is request, the system checks if the thread pointer is empty before creating the new routine. If it is not empty, the system terminates the current routine and enforces a mandatory motor stop. As stated before, there are some movements that are not executed as a threaded routine. For those, there is the possibility to execute said movement without the need to use a thread, they use a simple function call.

Supporting all of the available routines of the robot, there is a system that tracks the relative position of the other robots. The decision of storing relative positions, instead of global, was due to the lack of a map of the region and the organization of the robots being decentralized. Without a map common to all of the robots, the centre of each local map is irrelevant. Although one of the robots can be the leader of the platoon, it was decided that the information should be spread so that in case of a malfunction or a change of the leading robot, the remaining robots could continue to operate. Therefore, each of the robots will store a local map with their coordinates as the map centre, although
the axis of the coordinate system will still be common to all of the robots, and equal to the world reference.

Due to the lack of sufficient memory, the implementation of the map was made through landmarks. Instead of storing the map, the robot will only store the positions, x and y coordinates, and the orientation of relevant Points Of Interest (POI). At this stage, the landmarks stored will only be other robots, that will be identified with their unique 16-bit address from the Xbee module. Since the other robots are mobile, it is necessary to update their positions when they move, leading to the need of robots to start sharing information of their movements, as explained in section 3.2.5 after the first localization of a robot, as explained in section [3.4.5]. The storage of the orientation in the [POI] is important, so that when following a robot, the follower can decide not only how far to follow, but also from which direction.

In the following table there are some constant values that are used in this work. If not stated otherwise, these are the used values.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default forward speed</td>
<td>200 mm.s⁻¹</td>
</tr>
<tr>
<td>Default rotation speed</td>
<td>200 deg.s⁻¹</td>
</tr>
<tr>
<td>Gain used for differential wheel movement</td>
<td>0.15 rad⁻¹</td>
</tr>
<tr>
<td>Gain used for common mode movement</td>
<td>0.001 mm⁻¹</td>
</tr>
<tr>
<td>Step time</td>
<td>100 ms</td>
</tr>
<tr>
<td>Default timeout for communication operations</td>
<td>1000 ms</td>
</tr>
<tr>
<td>Maximum forward speed (estimate)</td>
<td>1128.83 mm.s⁻¹</td>
</tr>
<tr>
<td>Forward speed offset</td>
<td>-35.60 mm.s⁻¹</td>
</tr>
<tr>
<td>Half rotation speed (from specifications)</td>
<td>720 deg.s⁻¹</td>
</tr>
<tr>
<td>Minimum control the motors need to surpass attrition</td>
<td>0.09</td>
</tr>
<tr>
<td>Acceptable Orientation error</td>
<td>4 deg</td>
</tr>
</tbody>
</table>

Table 3.2: Constants common to all actions

The speed values in the end of table 3.2 are used primarily to limit the value of the control, that is normalized, that is given to the motors. This is why the rotation speed is in deg.s⁻¹ instead of rad.s⁻¹, since its value is only used in a normalization calculation and it is simpler for a person to limit the rotation speed in deg.s⁻¹ than rad.s⁻¹. Although, due to the lack of odometry, the amount of energy given to the robot must be used to estimate the travel distance. For this reason, the forward speed was estimated and has an additional parameter, the offset, in order to account for the minimum power necessary to overcome the static attrition, as explained in section 2.1.2. The communication timeout is used when waiting for a response of a target robot. If the robot does not reply, the system will try to recover from the error instead of locking up.

In table 3.2 there are two gains, one for differential wheel movement and another for common mode, that are used in proportional controllers. The value of these gains was chosen with regard to the minimum error acceptable and in order to have a stable response, since the step time is fixed, the robot cannot move more in one step than the distance/angle of the error that generated the output. These controllers, the common mode and differential mode, are discussed in sections 3.4.1.A and 3.4.1.B, respectively.
3.4.1.A Common mode controller

The controller for the common mode movement is described in figure 3.8 where the zero-order hold sets the 100 ms step time and the minimum control system defines how we handle the dead zone. Is due to this dead zone that the speed given by the transfer function must be corrected by a constant, since the dead zone creates an offset in the relation between the control given to the motor and the speed (relation in the form \( y=ax+b \)). This system can be configured to enforce the minimum control if the absolute value of the control is inferior, or to set it at zero. In the first option, the system will, almost, never stabilize since the controller only updates every 100 ms. The other option, the controller stops before achieving zero error, and the distance that this occurs depends on the gain. In this simulation, it was considered that the position is observable. This is not the case in the robot, since it does not have odometry and the movement is estimated. The response to a 500 mm step as input is given in figures 3.9a and 3.9b where in the first run the controller enforced the minimum control and in the second the controller set the control variable to zero.

Figure 3.8: Common mode controller

(a) Common mode controller response to a 500 mm step with minimum control enforced
(b) Common mode controller response to a 500 mm step with dead zone set to zero control

Figure 3.9: Common mode responses to steps of 500 mm

The differences in the responses in figure 3.9 are solely caused by the different way that the dead zone of the robot is handled. The first case is acceptable if the objective is to travel a certain
distance, like the move command in section 3.4.2, since the oscillation is between 492 and 506 mm. In this case, the controller can stop when the error achieves a certain threshold in order to avoid the oscillation. The second approach leads to controller being unable to correct the error, stabilizing at 410 mm, due to the motors being fed zero as the control when in the dead zone. This type of control is more appropriate when the controller is left active, like in the follow routine in section 3.4.6, where the error is not that important and can be corrected via software by putting the target further away, with the additional distance accordingly to the expect error. This error can be estimated with table 3.3 by checking the line corresponding to the minimum control (the first in the table). The first column has the error, that multiplied by the gain will generate the control for the motors, with the value in the second column. This generates a thrust that will move the robot at the speed in the last column, that leads to the assumption that the robot will move the distance in the third column in a single step, however, this is an approximate value, due to the existence of delay in the response. With this gain, any error lower than 90 mm will put the motor in the dead zone.

<table>
<thead>
<tr>
<th>Error [mm]</th>
<th>Control</th>
<th>Distance Travelled [mm. Step⁻¹]</th>
<th>Speed [mm. s⁻¹]</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>0.09</td>
<td>6.60</td>
<td>65.99</td>
</tr>
<tr>
<td>100</td>
<td>0.10</td>
<td>7.73</td>
<td>77.28</td>
</tr>
<tr>
<td>200</td>
<td>0.20</td>
<td>19.02</td>
<td>190.17</td>
</tr>
<tr>
<td>300</td>
<td>0.30</td>
<td>30.30</td>
<td>303.05</td>
</tr>
<tr>
<td>400</td>
<td>0.40</td>
<td>41.59</td>
<td>415.93</td>
</tr>
<tr>
<td>500</td>
<td>0.50</td>
<td>52.88</td>
<td>528.82</td>
</tr>
<tr>
<td>600</td>
<td>0.60</td>
<td>64.17</td>
<td>641.70</td>
</tr>
<tr>
<td>700</td>
<td>0.70</td>
<td>75.46</td>
<td>754.58</td>
</tr>
<tr>
<td>800</td>
<td>0.80</td>
<td>86.75</td>
<td>867.46</td>
</tr>
<tr>
<td>900</td>
<td>0.90</td>
<td>98.03</td>
<td>980.35</td>
</tr>
<tr>
<td>1000</td>
<td>1.00</td>
<td>109.32</td>
<td>1093.23</td>
</tr>
</tbody>
</table>

Table 3.3: Common mode response with 100 ms step and 0.001 mm⁻¹ of gain

To test the delay, important since the odometry will be estimated, a simulation with the motors at full speed was made. In that simulation, the position given by the system with a pole and the system without the pole (just a gain) was of 49 mm. If the motors are set to half speed, the error is 24 mm.

3.4.1.B Differential mode controller

The controller for the differential mode is similar to the one for the common mode, section 3.4.1.A, differing only in the transfer function (figure 3.10) where both the gain and the offset is different. Another difference, but this time regarding the implementation, is that with the differential operation, the control can access the speed, through the gyroscope, and the position, through the magnetometer. The minimum control has the same problem with the dead zone, but this is only a problem in pure differential movement. If the movement is a composite one, the dead zone might not apply since the differential control is used alongside the common mode control (the dead zone is regarding the control sent to the wheels, equations 3.3 and 3.4). For this reason, the second subfigure in figure 3.11 does not force the control to zero if it’s lower than the minimum control, in order to observe the differential response when the dead zone does not apply. This is also the reason why the offset, of the speed, is
zero in this case (dead zone is ignored).

Figure 3.10: Differential mode controller

(a) Differential mode controller response to a 180° step with minimum control enforced
(b) Differential mode controller response to a 180° step without dead zone

Figure 3.11: Differential mode responses to steps of 180°

In figure 3.11b, the minimum control is enforced, like with the common mode, and the oscillation is between 184° and 177°. Because of this, this approach is only acceptable in pure rotations where the controller can manually stop the robot. The second figure, 3.11, the dead zone was eliminated, in contrast with the common mode where the second figure had the control set to zero if in the dead zone. This was made, was stated previously, to observe the response when used alongside the common mode. In this case, the controller achieves zero error with a half degree overshoot.

In table 3.4 there’s the equivalent of table 3.3 but for the differential mode. The maximum speed used in this calculations was the mean of the one reported by the gyroscopes (tables 2.5 and 2.6), 1556 deg.seg⁻¹. The first two columns have the error to be used in calculating the control value. To be used in the calculation, the error must be in radians, in order for the control being the result of the multiplication of the error by the gain, but the values in the table are in degrees to be easier to understand.
Table 3.4: Differential mode response with 100 ms step and 0.15 rad\(^{-1}\) of gain

<table>
<thead>
<tr>
<th>Error [deg]</th>
<th>Error [rad]</th>
<th>Control</th>
<th>Rotated [deg.Step(^{-1})]</th>
<th>Speed [deg.s(^{-1})]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.09</td>
<td>0.01</td>
<td>2.03</td>
<td>20.34</td>
</tr>
<tr>
<td>10</td>
<td>0.17</td>
<td>0.03</td>
<td>4.07</td>
<td>40.68</td>
</tr>
<tr>
<td>15</td>
<td>0.26</td>
<td>0.04</td>
<td>6.10</td>
<td>61.03</td>
</tr>
<tr>
<td>20</td>
<td>0.35</td>
<td>0.05</td>
<td>8.14</td>
<td>81.37</td>
</tr>
<tr>
<td>30</td>
<td>0.52</td>
<td>0.08</td>
<td>12.21</td>
<td>122.05</td>
</tr>
<tr>
<td>40</td>
<td>0.70</td>
<td>0.10</td>
<td>16.27</td>
<td>162.73</td>
</tr>
<tr>
<td>50</td>
<td>0.87</td>
<td>0.13</td>
<td>20.34</td>
<td>203.42</td>
</tr>
<tr>
<td>60</td>
<td>1.05</td>
<td>0.16</td>
<td>24.41</td>
<td>244.10</td>
</tr>
<tr>
<td>120</td>
<td>2.09</td>
<td>0.31</td>
<td>48.82</td>
<td>488.20</td>
</tr>
<tr>
<td>180</td>
<td>3.14</td>
<td>0.47</td>
<td>73.23</td>
<td>732.31</td>
</tr>
</tbody>
</table>

3.4.2 Straight movement

The straight movement routine is the one responsible for moving the robot in a straight line for a given distance. This movement only has two variables that can be changed, being those the actual distance to cover and the maximum speed the robot can achieve. In case the maximum speed is negative, the robot will move backwards. Due to being a basic action, this movement is available both in the form of a thread, in order to be executed directly as an action, and in the form of a function. The thread form consists only on a call to the function mode that uses it’s return in the response as stated in section 3.2.5.

The implementation of this routine is presented in figure 3.12 and starts by saving the current orientation of the robot. Since the robot will move along side a straight line, this value will be used as the target orientation for the movement. The main loop consists of the controller and has two stop conditions. The first is to check a global variable that indicates if the robot should be moving. This is set to true at the beginning of the execution and is set to false if a stop command is issued. The other stop condition is the one related to the controller objective, while it isn’t reached, the controller continues to run. The next step is to calculate the control that must be given to the motors. The common mode control variable, \(u_c\), is given by equation 3.1

\[
u_c = (\text{TargetDistance} - \text{Distance}) \cdot K_c
\]  

(3.1)

where \(\text{TargetDistance}\) is the distance that the robot must travel, \(\text{Distance}\) is the distance traveled and \(K_c\) is the gain. The differential control variable, \(u_d\), is given by equation 3.2

\[
u_c = (\text{TargetOrientation} - \text{CurrentOrientation}) \cdot K_d
\]  

(3.2)

where \(\text{TargetOrientation}\) is the initial orientation, \(\text{CurrentOrientation}\) is the current orientation and \(K_d\) is the gain. Although not represented, an additional processing is made to the error calculation in order to ensure that the value is between -\(\pi\) and \(\pi\). This is made so that the robot always rotates in direction in which the distance to the goal is smaller since the maximum error is always half of a complete rotation to either side. According to the defined inertial reference, the control given to the wheels must make the robot turn anticlockwise when the error is positive (equations 3.3 and 3.4).
If the addition of the differential mode control leads to the expected speed be greater than the maximum, that was set alongside the target distance when calling the function, the common mode control is the one that is pruned in order to conform to the limit. On the other hand, if the common mode control absolute values is lesser than the minimum control, as specified in table 3.2 the controller forces the value to the minimum control with the same sign as the supposed value. After this check, the wheels are set to the calculated control and the control waits for the remaining of the time left in this step. The step duration is given in table 3.2. On exiting, the function returns the estimated distance travelled, that is return to the entity that issued the command (via wireless communication if that was the case).
3.4.3 Rotation movement

This movement has some similarities with the straight movement, as it can also be run as a thread or an isolated function called from a more complex routine. It also has only two arguments, one for intended orientation to act as goal and the other to set the maximum rotation speed that the robot can achieve in this particular rotation. The way that the rotation is handled by the thread is discussed in section 3.2.5.

The implementation of the rotation movement is presented in figure 3.13 and it starts with the calculation of the error. Although not represented in the schematic, before calculating the initial error, the system checks if the inputted orientation is between -pi and pi, being that the acceptable orientation range. If the value given is not in this range, the function terminates. The error is calculated in the same way as described in subsection 3.4.2 by subtraction the robot orientation from the target one.

After the initial error is calculated, the system enters the controller loop that has once again two stop conditions, one related to the permission to move and the other related to the goal. The goal in this movement is to reach an error that is less to the minimum acceptable error as stated in table 3.2. If none of the stop conditions is met, the next step is to calculate the control by multiplying the error, that was calculated in the beginning of the rotation command or in the end of the previous step, by the differential gain, as described in equation 3.2 with the difference on the error having the additional processing as described earlier.
### 3.4.4 Calibration of the magnetometer

The calibration of the magnetometer does not need any argument, having the the only two variables hardcoded. These variables are the number of steps needed to execute a full turn and the number of readings needed to calculate the mean in each step reading.

The figure 3.14 shows the steps taken in the compass calibration, that as stated in section 3.3.2, consists on making a full circle, in 148 steps, while obtaining an average of 5 readings in each step. These readings will then be compared with the current limits, the maximum and minimum limits for the x and y axes, updating them if needed. The values will be then cumulated in separate variables, one for the x axis and the other for the y axis, in order to calculate the mean of each axes in the end. The rotation of a step is made with a combination of a short pulse, of 10 ms, with the adequate control value in order to take 148 steps to make a full rotation.

In the end, the four necessary calibration variables are calculated, first is the offset by diving the variables that accumulated the readings for each of the axes by the number of steps. This offset is
then used to correct the stored limits and estimate the amplitude from them. The calibration values are then stored in the correspondent global variables.

![Diagram of the calibration process](image)

**Figure 3.14:** Implementation of the calibration of the sonar

### 3.4.5 Search for a robot

This algorithm is one of the most important in this thesis since it is the only way for a robot to know the position of the others. The input of this operation is the 16-bit address of the target robot, that acts as the identification as stated in the beginning of this section (3.4.1). The way that this location problem was tackled consists on making a full rotation, in small steps, of both robots while comparing the sonar readings. The rotation of the target robot is synchronized with the one that initiated the search, called master from now on. The only difference is that the target's orientation must be opposite to the one of the master, in order for the sonars be facing each other when they are with the same orientation, albeit not the same direction.

The algorithm is summarized in figure 3.15 and is divided in three stages, represented by the three columns in the figure.

The main stage, corresponding to the fist column, is responsible for performing the loop that will
lead to a full rotation, alongside with the sonar reading at each step. Before the loop starts, the master has to make sure that the target is stopped, this is done by sending a stop command that will terminate any movement action. After this is done, the loop has once again the common two stop conditions. One for the permission to move and the other stops the algorithm if a rotation and a half (540°) was already made. If any of these stop conditions are triggered, the reply that will be sent will be negative, as in the master has failed to acquire the target's position. The first thing that is made in each new step, a new step begins when the master rotates five degrees, is to set the target's orientation to the opposite of the master (master's orientation plus 180 degrees). This is done so that the target is able to face the master sonar to sonar, so that both of the sonars measure the same distance. If the 180 degree offset to the orientation was not performed, the sonars would read in opposing phases, never being able to match, correctly, the measurements. When the error of the measurement, given by subtracting the measurement of the target with the measurement of the master, is within an acceptable range, given in table 3.2, the algorithm accepts this orientation as a possible one for the target to be in. To avoid false positives, a confirmation procedure was made.

Before starting the confirmation itself, one of the robots must move. This has to be done because the confirmation is not needed due to false data from the sonars but from both of the robots facing an obstacle that is at the same distance in both cases. Therefore, one of the robots must move and this is explained in the second column. At first the algorithm tries to move the master forward if there is enough space. At this stage, distances greater than 400 mm are considered enough since after moving, the robot would move 10 cm, the remaining distance would be enough to be picked up by the sonars (as state in section 2.3.2, the sonars have a minimum range). If the master cannot move forward, half of a full rotation is made and the sonar distance is measured. If this distance is considered enough, the robot moves forward, but backwards if the initial state is considered. If neither options are available to the master robot, the movement operation is given to the target. Since the target is considered to be in front of the master, there is only one possible direction to move since the other was already discarded by the master. Therefore, the target rotates half a circle in order to have the same direction as the master. A sonar reading is made and, if the distance is enough, the target moves. If neither the master or the target can move, the orientation that generated the confirmation is considered correct and the algorithm jumps to the end of the confirmation step, detailed in the third column. In the other cases, where one of the robot moves, the algorithm passes to the beginning of the confirmation procedure.

In the third column, the confirmation procedure is displayed, consisting in rotating the target to the presumed correct orientation but, once again, with the 180° offset in order to put the sonar facing the master. The same is done with the master, but without the offset. After both of the robots are aligned, the sonar measurements are compared and if the error is once again inside the acceptable range, the target robot has been found. If the error is not acceptable, the algorithm returns to the main loop, with the rotation counter at zero. The counter is reset due to the movement of the robots, since this invalidates the readings made in the previous steps due to the change of the relative position of the robots.
After confirming the orientation of the target, its position is saved by calculating the x and y coordinates, relative to the master, by the means of the orientation and distance (equations 3.5 and 3.6). Since the centre of the map in each robot is the robot’s position, the master can send its position by calculating the coordinates with 180° added to the angle. After saving the coordinates, both the target robot and the master start to send information of their movements to the other. This will lead to both robots being aware of each other’s position. Before terminating, the algorithm replies to the entity that issued the search command to notify it of the success.

\[
\begin{align*}
    x &= \text{distance} \cdot \cos(\text{orientation}) \\
    y &= -\text{distance} \cdot \sin(\text{orientation})
\end{align*}
\]
3.4.6 Follow

This routine handles the control of the robot when the purpose is to follow a moving target, therefore, the controller is very similar to the one in the move routine, section 3.4.2, where the target is fixed. In this case, the target distance and orientation are updated in each step of the controller. For this reason, this controller does not have an end state, being ever active until it is manually stopped. The inputs for this routine are the target robot to follow, the maximum speed that the robot can achieve, the distance at which to follow and at what orientation to follow. As stated in section 3.2.5, the orientation
is the relation between the followee and the follower, where at 0° the followee is in front of the follower and at 90° it is on the left of the follower. It must be noted that this routine only works as expected if the target robot was searched before. This is to ensure that the target's position is updated, since the search routine, section 3.4.5 sets the target position and asks the target to keep that position updated by issuing the following command, as described in section 3.2.5.

The main process of this routine is composed by just four steps as shown in figure 3.16. The process is in a loop where the only stop condition is by checking if the controller can continue to run, and if not, the robot stops its motors and exists the routine. The first step, in a new iteration of the controller, is to calculate the trajectory. This is done via potential fields by setting the goal as a attractive force and the obstacles, as in other robots, as repulsive forces. The distance to cover is a simple arithmetic distance to goal and the potential fields only serve to calculate the direction to follow, in order to avoid obstacles, by following the direction of the vector given by the sum of all the attractive and repulsive forces. This algorithm is further explained in sub section 3.4.6.A. The way that the control is calculated is the same as in the move routine, section 3.4.2, with the difference in the calculation of the common mode, equation 3.1 where the error distance is given by the distance between the goal and the robot.

The way that the minimum control is handled is different from the other routines, since in this case the minimum control is not enforced, but put to zero. Although, this check, for absolute value greater than the minimum control, is made on the wheels control (equations 3.3 and 3.4). The reason for this is because the differential control can achieve greater values and counteract the common mode. For example, if the common mode has 0.50 as control and the differential has 0.45, one wheel would be at 0.05 and would not spin. Since the odometry is based in estimation, this behaviour is not desired, therefore, that wheel would be set to 0 but the other would still spin, correcting the direction.

After the wait period to set the 100 ms step time, the robot updates the positions of all POI such as other robots. Since the map is centred in the robot, the update operation is made by going through the list of the landmarks and subtract the X and Y offset estimated by taking in account the robot model, see section 2.1.2 and the control given to the wheels.
3.4.6. A Potential Fields

For path planning and obstacle avoidance, an approach with artificial potential fields is used. In [12] Oussama Khatib presented this algorithm and states “The manipulator moves in a field of forces. The position to be reached is an attractive pole for the end effector and obstacles are repulsive surfaces for the manipulator parts.”. Although the algorithm was tested in a robotic arm, the PUMA 560, it can also be applied to mobile robots path planning where the robot that is calculating the path is both attract and repulsed by the goal and obstacles respectively.

The artificial potential present in the map, $U_{art}$, is given by equation 3.7

$$U_{art}(x) = U_{goal}(x) + U_{obs}(x) \quad (3.7)$$

where $U_{goal}$ is the attractive potential generated by the goal position and $U_{obs}$ is the sum of the potential fields generated by the obstacles. Although in this thesis the robot only has one possible goal, the algorithm can work with multiple attractive fields, possibly leading to multiple local minima. The effect generated by the artificial potential $U_{goal}$ is represented by the force in equation 3.8.
\[ F_{\text{art}}(x) = F_{\text{goal}}(x) + F_{\text{obs}}(x), \]  
\hspace{1cm} (3.8)

with

\[ F_{\text{goal}}(x) = -\nabla U_{\text{goal}}(x), \]  
\hspace{1cm} (3.9)

\[ F_{\text{obs}}(x) = -\nabla U_{\text{obs}}(x). \]  
\hspace{1cm} (3.10)

\( F_{\text{goal}} \) and \( F_{\text{obs}} \) are the forces generated by the goal and the obstacles respectively and are given by the gradient of the potential field. The field generated by the goal is expressed by the equation

\[ U_{\text{goal}}(x) = \frac{1}{2} \xi (x - x_{\text{goal}})^2 \]  
\hspace{1cm} (3.11)

where \( x - x_{\text{goal}} \) is the distance between the current position on the map and the goal and \( \xi \) is a positive gain used to fine tune the effect of the field. Figure 3.17 shows the form of the field when \( \xi \) is equal to one and the goal is at (0,0), and can be easily observed that the curve is always positive and has a smooth gradient to the goal.

\[ U_{\text{obs}}(x) = \begin{cases} 
\frac{1}{2} \eta \left( \frac{1}{d(x)} - \frac{1}{d_0} \right)^2 & \text{if } d(x) \leq d_0 \\
0 & \text{if } d(x) > d_0 
\end{cases} \]  
\hspace{1cm} (3.12)

Figure 3.17: Attractive field with goal at (0,0) and \( \xi = 1 \)

The repulsive potentials used in this thesis will form an obstacle in the form of a cylinder (equation 3.12) where, as with the attractive potential, \( \eta \) is a positive gain, \( d \) is the distance to the obstacle and \( d_0 \) is the distance at which the field starts to have effect. Figure 3.18 has an example of this type of fields.
Figure 3.18: Repulsive field with obstacle at (0,0) and $\eta = 2 \times 10^{10}$ and $d_0 = 400$

A combination of both types of potentials is given in figure 3.19 where the attractive potential has the goal at (-300,0) and $\xi = 1$ and the repulsive potential has the obstacle at (0,0), $\eta = 2 \times 10^{10}$ and $d_0 = 400$.

Figure 3.19: Attractive field with goal at (-300,0) with repulsive field with obstacle at (0,0)

This example, the one in figure 3.19, is one relevant to this thesis since it can represent the problem of following another robot. The target robot is an obstacle and the goal is near it. In this
case, the objective is to follow at 300 mm of distance. However, due to the goal being in the region of influence of the repulsive field, the distance between goal and obstacle is 300 that is lesser than \( d_0 \), the minimum is located at \((-400,0)\). This example also serves the purpose of demonstrating a problem with this type of approach, there is a local minimum at \((400,0)\). This is due to the obstacle being right in front of the objective. This problem could also occur if the current position of the robot is inside a room with the objective in the other side of the wall that is opposite to the exit. Since no obstacle detection was implemented in this thesis, the problem of local minima was not solved since if they are a single point, as is in this case, the probability of falling into it is very low since the map is in mm, therefore, a small error can take the robot out of the minimum. The possible ways of the robot falling in the minimum is to start in a position with the y coordinate at 0 and the x coordinate greater than 0. For this reason, figure 3.20 represents the path the robot should take if it starts at \((1000, 1)\). The black dots are the path and the red dot is the goal.

![Diagram](image)

**Figure 3.20:** Path generated if robot starts at \((1000, 1)\)

One possible way of avoiding local minima is to calculate the full path, instead of just the next step, and when the path stalls in a local minimum, the algorithm can backtrack and try to correct it. One way of doing this is with the use of navigational functions \[42\].
4
Experimental Results

Contents

4.1 Orientation .................................................. 68
4.2 Odometry Estimation ........................................ 69
4.3 Search Routine ............................................... 69
4.4 Follow Routine ............................................... 70
This chapter is focused on showing results of the controllers implemented. The performance of the work on the operating system is integrated in all of the tests, since it’s due to it that the robots can handle the sensors, wireless communication and control generation seamlessly at the same time.

4.1 Orientation

For testing the performance of the controller responsible for setting the robot with a given orientation, section 3.4.3, a simple test where the robot must rotate 180 degrees, the maximum value, was made. The robot was configured to set it’s orientation at 0° and, after 20 seconds, change the orientation to 180°. At the same time, thanks to the multithreading capabilities of the system, a separate thread was running, collecting the orientation from the Kalman filter, section 3.3.2, every 20 ms. For comparison, the Kalman filter updates it’s value every 5 ms and the controller updates every 100 ms.

Five tests were made were the robot was put in a random position, at least 90° from the 0° orientation, so that, on each run, two separate end values of the rotation algorithm can be observed. The evolution of the orientation of the robot while the control is active is shown for the movement from 0° to 180°. In figure 4.1 is one of the runs, test number 1, cropped between 1 second before the rotation and 1 second after finishing it. In table 4.1 the results from the 5 runs are presented.

![Figure 4.1: Orientation during a 180° rotation with a 50Hz sampling rate](image)

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Start Orientation [deg]</th>
<th>End Orientation [deg]</th>
<th>Rotation Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.23 +/- 0.15</td>
<td>176.29 +/- 0.16</td>
<td>0.88</td>
</tr>
<tr>
<td>2</td>
<td>-1.30 +/- 0.16</td>
<td>179.13 +/- 0.13</td>
<td>0.98</td>
</tr>
<tr>
<td>3</td>
<td>-0.73 +/- 0.13</td>
<td>178.00 +/- 0.16</td>
<td>1.18</td>
</tr>
<tr>
<td>4</td>
<td>2.04 +/- 0.16</td>
<td>183.74 +/- 0.16</td>
<td>0.78</td>
</tr>
<tr>
<td>5</td>
<td>-0.88 +/- 0.15</td>
<td>181.65 +/- 0.23</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 4.1: Rotation tests results

In terms of the performance of the controller, the tests show that the robot can always accomplish to reduce the error to the one specified in table 3.2 (4 degrees). Although, due to the dead zone of
the motors, it is not possible to achieve zero error with this approach, were the controller stops when the error is within range, this problem does not occur when the controller is ever running, such as in the follow routine (section 3.4.6). In terms of the stability of the orientation reading, the standard deviation measured was during one second, equivalent to 50 measurements, and it was never higher than a quarter of a degree.

4.2 Odometry Estimation

During the testing of the various routines that involved movement, a qualitative analysis was made to the performance of the odometry estimation. It was noted that during the follow routine, section 3.4.6, the odometry estimation works relatively well when the movements are not too rash, few high value accelerations. If the robots follow a path where the change of rotation is made smoothly, the odometry estimation is acceptable for a longer time. Since this is due to not having actual odometry from the wheels, this is a problem not easily fixed, since the more tightly the robot kinematic model is described, the more it is acceptable to variances on the real robot, such as difference in weight of the robot or the ground surface inclination.

A simple test to confirm the acceleration error was made by setting the robot to move forward 500 mm with the move routine, section 3.4.2. The actual distance travelled was measured with a metric tape, but since the floor didn’t have a grid to help with setting the start and end points, the obtained values are not very precise. Nonetheless, the error measured in the five runs made was between 0.5 cm and 1 cm, where the robot stopped before covering the given distance (500 mm). If the gain in table 3.2 is taking into account, 0.001 mm.s$^{-1}$, the initial control value is 0.5, leading to half of the maximum power being delivered to the motors after the robot had been standing still. This leads to the conclusion that, when accelerating from zero to half speed, the error in the estimation is of around 1 cm, since the model used in the estimation does not take into account the time needed to accelerate to a given speed.

Since the generated maps are relative to each robot’s position, one way to mitigate this problem is to make periodical searches, for the other robot, to correct their positions.

4.3 Search Routine

The search routine, section 3.4.5, is the operation responsible for locating other robots, therefore, it’s accuracy influences the results of the follow routine, section 4.4. For testing the performance of this routine, the search command, responsible to find the coordinates of the target robot, was issued, followed by a follow command, in order to check the accuracy of the coordinates. The main issue with the search command, is with the sonar beam angle, section 2.3.2, since the coordinates are calculated with both the distance from the sonar and the direction that the robot is facing, equations 3.5 and 3.6.

Along the development of this thesis, several instances of the search routine were executed. The results presented here were achieved from 10 tests and the best, figure 4.2a, worst, figure 4.2b, and average, figure 4.2c case are presented.
The best case was achieved with the robot already facing the target, at 90 cm. The search for the average and worst case was done at 120 cm from the target. In all the tests made, the robot was able to find the target, although, in one case, it only managed to find the target after completing a full turn. Since the robot has 9.5 cm diameter, the average error is about half that distance, 5 cm, and the worst is equal to that, 10 cm. Since it is impractical to cover all orientations, in steps of one degree, in the search routine, one solution to this problem is to fit two addition sonars, one on each side of the current one, so that the overlap of the sonars’ model can reduce the uncertainty. The time that it takes for the search algorithm to finish is about 25 seconds when the has an initial orientation offset 180° and it was of about 50 seconds in the case that the robot did not find the target at the first pass.

4.4 Follow Routine

To test the follow routine, a simple chaotic path planner was used, where the followee robot would follow this artificial goal. This planer uses the rand function of the C library to generate random values that will be used to choose an orientation for the goal to move along. The output of the function is normalized between -1.0 and 1.0 and then multiplied by an angle, resulting the offset for the goal’s orientation. This value is then added to the last orientation followed, to be used in calculating the x
and y offsets with

\begin{align}
\Delta x &= \text{Distance} \cdot \cos(\theta) \\
\Delta y &= \text{Distance} \cdot \sin(\theta),
\end{align}

\hspace{1cm} (4.1) \hspace{1cm} (4.2)

where \( \theta \) is the orientation after the offset is added and \( \text{Distance} \) is a configurable value that represents the distance the goal travels in each step. To ensure that the robots would not move beyond the available space, the maximum accumulative offset, for both axes, was set to 500 mm.

For testing, the update of the goal’s position was made every half a second, with the step distance set to 50 mm and the angle set to 10 degrees, resulting in the possible range of -10 to 10 degrees. All of the tests were made with the same initial seed, resulting in the evolution represented in figure 4.3. The green spot is the start position and the red spot is the end position at 60 seconds. Each step equals to half a second.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4.3.png}
\caption{Evolution of the position of the artificial goal}
\end{figure}

To achieve the most similar conditions on each run, the robot that will follow, the follower, was set facing the target, the followee. This ensures that the search result, the search operation must be executed at least one time before the follow operation, has always the best case (figure 4.2a).

After the follower has searched for the followee, the follow command, section 3.4.6, was issued. The parameters were to follow the target from behind it, at 300 mm of distance. Two runs of this test are shown in figures 4.4 and 4.5 through a timelapse of 5 seconds per frame where the followee is pointed by a red arrow and the follower is pointed by a green one.
In the first test, figure 4.4 the robots seem to collide in frame 12. This does happen and is due to error in the prediction of the followee position by the follower. This is observed since frame 4, where the robots are much closer than they should be.
In the second test, the robots managed to avoid collision, the error can again be seen by frame 4, by a few centimetres. This lack of collision results in a greater error in the last frame, since in the previous test, the collision happened to correct a bit the estimation of the position of the followee.

One possible cause for the error propagation in this tests, is the initial position of the robots. The chaotic path planner had the initial orientation, for the movement of the goal, set to 0\(^\circ\) instead of the orientation that the followee, the robot that follows the artificial goal. This leads to the first movement of the followee to be of a rotation of nearly 180\(^\circ\), resulting in and avoidance manoeuvre from the follower.

Although not being evident in the timelapse, when avoidance manoeuvres happen, the follower does not have a smooth behaviour, changing it's orientation, by a considered amount of degrees, almost every step. The effect can be visualized as trying to go around the board of a circle trough...
steps. This effect could be correct by implementing a path planner that plans ahead, instead of just the next step, avoiding unnecessary movements when facing repulsive fields. Reducing the gain for the differential mode could be a solution, but it would slow the response of the robot, making it more difficult for it to avoid another robot that may be travelling in a collision course.

The sonar, although not being used in this occasion, cannot be used to avoid collisions because of it’s minimum range, section 2.3.2, leading to the sonar reporting 20 or 30 cm, depending on the sonar model, even when an obstacle is right in front of it. One possible application of the sonar would be to use it to correct the estimated position of the target robot. This was not implemented due to it’s use being highly limited due to the sonar’s wide beam aperture, since it would only be useful at very close range (in the limit of the minimum distance). If more sonars were used, the conjunction of all the readings would lead to a more precise possible location of the target robot. This would not completely substitute the advantage of another search, after a certain time, as stated in section 4.3.
Conclusions and Future Work
Most of the work in this thesis was in the development of the system that supports the reading of the sensors, communications, path planning and controllers. The accomplished system managed to coordinate all these features in a multithreading environment. In terms of communication, no bottleneck was experienced during the development of this thesis, although, this is highly dependent of the amount of entities on the network and the amount of packet loss. The possibility of deadlocking the communication system exists, happening if the outgoing circular buffer is full. Since the network throughput cannot be increased, the only way to solve this is to increase the size of said buffer.

The performance of the IMU in regards to the orientation estimation has a decent amount of precision for this type of application, since the standard deviation lesser than a quarter of a degree. This ensures that the performance of the search routine is satisfactory, having an error of 10 cm in the worst case if initial distance is 120 cm, even though only having one sonar, therefore no way to reduce the uncertainty, to make the distance measurements. One aspect of this work that did not work as intended was the estimation of the odometry trough the robot model. The pole of the system is at 3.72 Hz, equation 2.4 leading to a rise time slightly hight than the step time of the controllers. The step time is not possible to increase without affecting more the precision of the estimation due to the dead zone of the motors. A more complex kinematic model could be implemented on the robot, having into account the inclination of the robot thought the accelerometer readings. Nevertheless, if the robot avoids an erratic acceleration, in other words, having smother transitions, the estimation problem would be minimized.

A few modifications can increase the potential of this work. One of those is the increase of the number of sonars in the robot, in order to create an array. The conjunction of the readings of this array would lead to a reduction of the uncertainty of the sonar model, since some of the beam patters would superimpose. Although, since the sonars in that array cannot make the measurements at the same time, another model, with a update frequency greater than 10 Hz, would be optimal. Another modification for this work would be to investigate the ability to pass the orientation estimation, the Kalman filter and the IMU pooling, to the ATmega on the base of the 3pi robot. This is very attractive since the pooling rate of the IMU is much higher, 200 Hz, than that of the sonar and controller, 10 Hz. If this could be accomplished, the controller could also be passed to the ATmega, increasing the sleep time of the main processor, leading to a more autonomy for the robot. This way, only the path planning and communications would be on the main processor, that would feed the microcontroller with the intended orientation and speed while the microcontroller feeds the processor with the estimation of the odometry.
Bibliography


