Distributed control for mobile robot team coordination

Bernardo Filipe Lapa Marques da Costa
MSc Student of Electrical and Computer Engineering
Instituto Superior Técnico
Portugal
Email: bernardo.m.costa@tecnico.ulisboa.pt

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, by developing an operating system for them that is capable of distributively managing the operation of a team of robots 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.

I. INTRODUCTION

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 paper 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 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.

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 os 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 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.

II. THE HARDWARE

A. The Robot

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 a 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. 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 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. The power supply of the robot will be used to power the expansion board and all sensors.

1) 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 antclockwise, starting with 0 degrees when facing north, and 90/-90 degrees when facing west/east.

The kinematic model of the robot is given in equation 1

$$
\begin{bmatrix}
x' \\
y' \\
\theta'
\end{bmatrix} =
\begin{bmatrix}
\sin(\theta) & 0 & 0 \\
\cos(\theta) & 0 & -l \\
0 & 2 & 0
\end{bmatrix}
\begin{bmatrix}
v_c \\
v_d \\
v_r
\end{bmatrix}
$$

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 and 3 explain how the velocities mention in the kinematic model, equation 1, are related to the left, $v_l$, and right, $v_r$, velocities of the DC motors of the robot.

$$
v_c = \frac{v_r + v_l}{2} \quad (2)
$$

$$
v_d = \frac{v_r - v_l}{2} \quad (3)
$$

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.

The robot model is given in figure 1, where the gain of the fist block, $\frac{a}{s^2 + b}$, 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 frequency response test was made and the data obtained was used to fit a model with only one pole, $G(s) = \frac{a}{s + \tau}$. The fit generated the model represented in equation 4.

$$
G(s) = \frac{634.77}{s + 23.40} \quad rad.s^{-1}
$$

To obtain the model for the forward movement, the sonar, section II-D, 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. The speed that the robot was moving is indicated by the derivative of the fit line. The gain of the system, 1128.83 mm.s$^{-1}$, was determined with the derivative of the fit line. The offset, due to the dead zone, is -35.60 mm.

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 5.
\[ G(s) = \frac{26414.51}{s + 23.40} \text{mm.s}^{-1} \] (5)

The transfer function in the center of the robot model depends if it is for the differential mode, where the equation would be 4, or for the common mode, where the equation would be 5.

2) Odometry Estimation: 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 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 curvature, the distance between the robot and the centre of the circumference, and the delta of orientation achieved in the movement, as shown in figure 2.

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.

\[
\begin{bmatrix}
\Delta x' \\
\Delta y'
\end{bmatrix} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix} \begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix}
\] (12)

B. 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 that is populated with the interface rich mbed NXP LPC1768 [14]. The board also has an Xbee/Wixel socket, populated by a Xbee Pro 2, alongside the prototyping space to include the other sensors.

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.
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. An additional connection to the XBee was made, between pin 12 and the CTS output of the XBee, in order to implement flow control from the CPU to the XBee.

![Image of ARM mbed NXP LPC1768 connections after alteration](image)

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.

**C. Xbee**

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 [16]. In this work a simple Zigbee network will be used where the PC will be configured as the coordinator and the robots as routers.

The system for addressing in 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 its 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 0xFFE.

Due to the API mode having error checking built in, this mode was chosen in depreciation of the transparent. It also gives information about the status of the messages sent such as if they were delivered or not. In the API mode there are two types of configuration, one with escape characters support and another without. The mode with the escape characters is used to avoid problems when the message includes one of the control bytes.

**D. Sonar**

In this work two sonars from MaxBotix [17] are used. They are the MB1300 [18] and the MB1013 [19] 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 sonars will be read via the analog output and in trigger mode, where the sonar only starts a measurement when commanded. 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 measured resolution of the analogue reading is greater than the one in Table I due to the use of a 10-bit DAC to generate the outputs, that are read by a 12-bit ADC.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>MB1300</th>
<th>MB1013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog</td>
<td>10 mm</td>
<td>1 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>

The sonar beam diagrams for the MB1013 and MB1300 are available in the respective datasheets [20], [21].

**E. IMU**

The IMU used is the Pololu MinIMU-9 [22] that packs an L3G4200D [23] 3-axis gyroscope and an LSM303DLM [24] 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, as explained in section III-C2.

1) L3G4200D 3-axis gyroscope: The L3G4200D 3-axis gyroscope sampling is set to 200 Hz with an output, on each axis, of 16 bits. The range of the device is set to the maximum, +/-2000 degrees per second, due to the speeds achieved by the robots. At this range, the sensitivity is of 70 millidegrees per second per digit.

Due to the high speed, 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, on two different sensors, was of 768.01 deg.s\(^{-1}\) and 769.47 deg.s\(^{-1}\) resulting in a relative error of 7% on both occasions.

2) LSM303DLM 3-axis accelerometer and 3-axis magnetometer: The range of the magnetometer can be set to one of seven predefined values by changing the input gain applied by the sensor. The output of the magnetometer 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. For this work, the range used for the magnetometer is the smallest, +/- 1.3 Gauss, since it’s purpose is to measure the earth’s magnetic field, which is around half a Gauss [25]. The data rate of the sensor is set to the maximum, 220 Hz.

Since the actual geographic orientation is not important for this work, no correction was made to the readings (the magnetic poles are used as reference).

III. THE SOFTWARE

A. Operating System and Libraries

The programming of the mbed board is done online and the supported language of the compiler is C/C++. The mbed platform is supported by an open-source SDK[26] 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 for each of them, such as for the ADC, 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 RTOS[27] 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 and it is called minimu library [28].

B. Communications

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. A communications 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. In order to avoid pooling, ISRs were created to handle the UART buffer.

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, or retrieve, the bytes to be sent to, or received from, the Xbee. The circular buffer has an 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 possible to have a routine running, that may last a few seconds or more, alongside with a query for a robot status that can be processed in a few milliseconds. The thread that reads from the receiving circular buffer is responsible for assembling the bytes to from a complete packet, that is called a frame, and sends it to a free thread from the mentioned pool.

The ISR responsible for feeding the Xbee has, besides the flow control to the UART, the additional flow control to the Xbee given by the CTS pin. This method of flow control ensures that the Xbee’s internal buffer is not overrun, leading to discarded bytes.

After a frame is received, it is processed, by one thread of the pool of threads, and has two possible outcomes. If the frame is one of transmission status, its information is processed so that the acknowledgement, or lack of it, is used to unlock resources and, if asked by a thread, to warn a said thread if the frame that it send reached the destination. If the frame has a command, it is processed accordantly, freeing the thread only at the end.

For transmitting frames, a structure to store the frame is used. When a thread needs to send a message, it uses on of those structures to save the message. After the message is constructed, the structure is sent to the transmitting thread, that is responsible to consume those structures by copying their bytes to the circular buffer reserved for transmission, from which the ISR that feeds the Xbee fetches bytes.

C. Reading the Sensors

1) Reading the Sonar: The reading of the sonar measurement is made thought pin 20, that must be configured as an ADC input. To control the readings, pin 19 will be configured as a digital output. 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.

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 minimu9 library [28]. 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].

The filter will be updated inside a RTOS 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.
D. Movement

In this work, a movement, or a series of movements, that must be executed in order to achieve a certain goal is called a routine and is implemented via a thread. Only one routine can be active at each time.

Supporting all of the available routines of the robot, there is a system that tracks the relative position of the other robots. 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 world reference, the robot will only store the positions, x and y coordinates, and the orientation of relevant POI.

1) Controller: The control that must be given to the motors is composed by the common and differential mode. The common mode control variable, \( u_c \), is given by equation 13

\[ u_c = \text{TargetDistance} \cdot K_c \]  

where \( \text{TargetDistance} \) is the distance to the target and \( K_c \) is the gain (0.001 mm\(^{-1}\)). The differential control variable, \( u_d \), is given by equation 14

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

where \( \text{TargetOrientation} \) is the initial orientation, \( \text{CurrentOrientation} \) is the current orientation and \( K_d \) is the gain (0.15 rad\(^{-1}\)). 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 15 and 16).

\[ u_l = u_c - u_d \]  
\[ u_r = u_c + u_d \]

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. 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 of 100 ms. 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.

This is the base controller used along this work, having minimal differences between the applications, such as how to respond to the dead zone and when to manually stop the controller.

The first step, in a new iteration of the controller, is to calculate the trajectory. This is done via potential fields.

2) 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 path planning is inserted in the beginning of each controller step, where the goal is set as an 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.

The artificial potential present in the map, \( U_{art} \), is given by equation 17

\[ U_{art}(x) = U_{goal}(x) + U_{obs}(x) \]  

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 work 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 18

\[ F_{art}(x) = F_{goal}(x) + F_{obs}(x), \]  

with

\[ F_{goal}(x) = -\nabla U_{goal}(x), \]
\[ F_{obs}(x) = -\nabla U_{obs}(x). \]

\( F_{goal} \) and \( F_{obs} \) 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

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

where \( x - x_{goal} \) 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 5 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.

The repulsive potentials used in this work will form an obstacle in the form of a cylinder (equation 22) 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 6 has an example of this type of fields.

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

(22)

A combination of both types of potentials is given in figure 7 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 \(d_0 = 400\).

This example, the one in figure 7, is one relevant to this work 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 work, 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 8 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.

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 [29].

IV. EXPERIMENTAL RESULTS

A. Orientation

For testing the performance of the controller responsible for setting the robot with a given orientation, 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, a separate thread was running, collecting the orientation from the Kalman filter, section III-C2, 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. In table II the results from the 5 runs are presented.

<table>
<thead>
<tr>
<th>Start Orientation [deg]</th>
<th>End Orientation [deg]</th>
<th>Rotation Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.25 +/- 0.15</td>
<td>176.29 +/- 0.16</td>
<td>0.88</td>
</tr>
<tr>
<td>-1.30 +/- 0.16</td>
<td>179.13 +/- 0.13</td>
<td>0.98</td>
</tr>
<tr>
<td>-0.73 +/- 0.13</td>
<td>178.00 +/- 0.16</td>
<td>1.18</td>
</tr>
<tr>
<td>2.04 +/- 0.16</td>
<td>183.74 +/- 0.16</td>
<td>0.78</td>
</tr>
<tr>
<td>-0.88 +/- 0.15</td>
<td>181.65 +/- 0.23</td>
<td>0.98</td>
</tr>
</tbody>
</table>

TABLE II

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 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. 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.

B. Search Routine

The search routine, section IV-B, is the operation responsible for locating other robots, therefore, it’s accuracy influences the results of the follow routine, section V. For testing the performance of this routine, the search command, responsible to find the coordinates of the target robot, was issue, followed by a follow command, in order to check the accuracy of the coordinates. The main issue with the search common, is with the sonar beam angle, section II-D, since the coordinates are calculated with both the distance from the sonar and the direction that the robot is facing, equations 23 and 24.

\[ x = \text{distance} \cdot \cos(\text{orientation}) \]  \hspace{1cm} (23)
\[ y = -\text{distance} \cdot \sin(\text{orientation}) \]  \hspace{1cm} (24)

The results presented here were achieved from 10 tests and the best, figure 9a, worst, figure 9b, and average, figure 9c, 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.

V. FOLLOW ROUTINE

To test the follow routine, a simple path was generated for one of the robots to follow, the followee. The other robot, the follower, is then commanded to follow the followee. 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 9a).

The parameters for the follow command were to follow the target from behind it, at 300 mm of distance. One run is shown in figure 10 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 test, 10, evolution of the robots position show that the odometry estimation has a few flaws, since the distance between robots is not constant along time. One possible cause for the error propagation in this tests, is the initial position of the robots, that lead to the green robot, the followee, to take avoiding actions (frames 1 to 3). 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 its minimum range, section II-D, 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 its 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 IV-B.

VI. CONCLUSION

Most of the work 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 work, 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 meters, 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 through the robot model. The pole of the system is at 3.72 Hz, equation 4, leading to a rise time slightly higher 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 through the accelerometer readings. Nevertheless, if the robot avoids an erratic acceleration, in other words, having smoother 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 patterns 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.

REFERENCES