Development and evaluation of a robot trilateration localization system using four beacons

Stefany Yumie Kawashima - 54156

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Industrial - Electrotécnica.

Trabalho orientado por:

Prof. M.Sc. José Luís Sousa de Magalhães Lima

Prof. M.Sc. Marcos Banheti Rabello Vallim

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2023

Development and evaluation of a robot trilateration localization system using four beacons

Stefany Yumie Kawashima - 54156

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança no Âmbito da Dupla Diplomação entre Instituto Politécnico de Bragança e Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. M.Sc. José Luís Sousa de Magalhães Lima

Prof. M.Sc. Marcos Banheti Rabello Vallim

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2023

Dedicatória

Dedico este trabalho ao meu pai que ficaria muito orgulhoso de saber que sua filha se tornou engenheira.

Agradecimentos

Agradeço primeiramente à Deus. Agradeço ao meu pai e minha mãe por todo o investimento e dedicação nesses últimos 27 anos. Agradeço todos os meus professores desde o primeiro dia na escola. Agradeço a Universidade Tecnológica Federal do Paraná e ao Instituto Politécnico de Bragança por esta oportunidade incrível. Agradeço ao professor José Luis e ao Arezki e João que estiveram a puxar por mim e me auxiliar ao decorrer de todo o trabalho. Agradeço ao meu noivo por me cobrar para que eu terminasse e todo o apoio que ele me deu. Por último e não menos importante, agradeço a mim mesma por não desistir.

Resumo

KAWASHIMA, S. Y. Esta dissertação tem como objetivo investigar a localização de robôs móveis em um ambiente de *software* virtual (Simtwo) usando quatro beacons e um sensor LiDAR. A localização de robôs desempenha um papel crucial em várias indústrias e aplicações, como robótica, automação e navegação. Uma localização precisa é vital para que os robôs móveis possam navegar e desempenhar tarefas de forma eficaz. Para alcançar a localização, beacons artificiais são usados como pontos de referência. Para fazer isso, um método comumente usado é a trilateração, que requer pelo menos três beacons para serem reconhecidos simultaneamente, juntamente com a distância entre elas e o robô. Usando essa técnica e um algoritmo de identificação de beacons, este trabalho calculará a posição global estimada e erros associados de um robô em uma simulação virtual da *Robot Factory* no Simtwo.

Palavras-chave: Robôs móveis, LiDAR, Trilateração, Localização de Robôs

Abstract

KAWASHIMA, S. Y. This dissertation aims to investigate the localization of robots in a virtual software environment (Simtwo) using four beacons and a LiDAR sensor. The localization of robots plays a crucial role in various industries and applications, such as robotics, automation, and navigation. Accurate localization is vital for mobile robots to navigate and perform tasks effectively. In order to achieve localization, artificial beacons are used as reference points. In order to do that, one commonly used method is trilateration, which requires at least three beacons to be recognized simultaneously along with the distance between them and the robot. Using this technique and a beacon identification algorithm, this work will calculate an estimated global position and respective errors of the robot in a virtual simulation of Robot Factory on Simtwo.

Keywords: Mobile robots, LiDAR, Trilateraion, Robot Localization

Contents

List of Tables

List of Figures

Acronyms

AGV Automated Guided Vehicle. 17

IPB Instituto Politécnico de Bragança. iii

LiDAR Light Detection And Ranging. 3

SLAM Simultaneous Localization and Mapping. 9

UTFPR Universidade Tecnológica Federal do Paraná. iii

Chapter 1

Introduction

Since the creation of the first mobile robot in 1968, called *Shakey* (shown in Figure 1.1), the number of this type of robot is increasing [15].

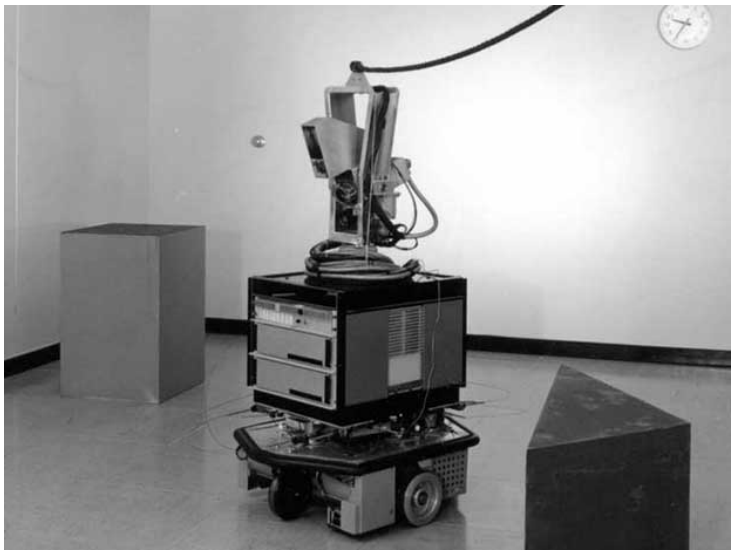


Figure 1.1: Picture of the first mobile robot, known as *Shakey* [5].

Mobile robots are gaining more visibility in applications in robotics. These applications have a wide range, since industrial robots [15], even robots for missions in other planets [9], remote areas or difficult access locations like underwater regions [16] and high radioactivity [12]. In each one of these applications areas mentioned, the robot requires autonomy and mobility. In order to do its tasks, it needs to move inside any type of environment. This

environment is known as Robotics Navigation [12].

The Robotics Navigation deals with three main questions: where the robot is, where should it go and how it should move to reach its destination. In order to do that, robots may be produced in a way to recognize the environment and the reference points to do its tasks.

Besides that, the robots should know how to deal with obstacles to reach their destination safely. To do that, while its navigate, the robots uses sensors to estimate its position and orientation on the environment that it is. However, the use of these sensors may be have measurement errors and uncertainties [2].

1.1 Context

The problem of localization is a fundamental key to the autonomy of mobile robots. If a robot doesn't know where it is, determining its next actions can be challenging. In order to determine its location, a robot has access to relative and absolute measurements of its position, providing feedback to its system about its movement actions and the situation of the environment it is in. Given this information, the robot needs to determine its location with the highest possible accuracy. What makes this task difficult is the presence of uncertainties both in its movement and in the robot's sensor measurements. The information embedded in these uncertainties needs to be optimally combined [12].

Accurate localization is vital for mobile robots to navigate and perform tasks effectively. In order to achieve localization, beacons can be used as reference points. However, the number of beacons that can be placed in the environment is limited. This limitation has led researchers to develop localization algorithms that work with specific beacon placements and numbers. One commonly used method is trilateration, which requires at least three beacons to be recognized simultaneously along with the distance between them and the robot. This method allows the robot to calculate its precise position based on the intersection of the spheres created by the distances from the beacons.[17]

In this work, the SimTwo software will be employed to perform the localization of a

robot using a Light Detection And Ranging (LiDAR) sensor. Four beacons will be used to estimate the location using the trilateration technique within a factory environment.

1.2 Objectives

1.2.1 General Objective

Presenting the SimTwo software and its essential features, elucidating the Localization challenge through the application of trilateration methodology, and formulating an algorithm for the precise positioning of a robot with the aid of four beacons.

1.2.2 Specific Objectives

- To introduce the SimTwo tool and its primary functionalities.
- To present the issue of localization in mobile robots.
- To implement a localization technique (trilateration) for pinpointing the robot's position within the SimTwo software and computing associated errors.

1.3 Document Strucuture

This work is divided in seven chapters, described below.

- Chapter 1: Introduction. This chapter will present a brief general introduction to the work, showing the objectives and the work proposal.
- Chapter 2: State of Art. Concepts related to mobile robots will be reviewed, presenting a brief history, as well as general characteristics. Instances of the localization problem will be addressed, showing examples of where uncertainties are applied in the robotic context. This chapter also will present Simtwo and the trilateration methodology used to solve the localization problem.

- Chapter 3: Development. This chapter will show the algorithm used in SimTwo platform.
- Chapter 4: Results Obtained and Analysis of Results. In this chapter the results obtained will be shown and compared to the ground truth to calculate its errors.
- Chapter 5: Conclusion. Finally, the results obtained will be compared with the existing literature on the subject, discussing the difficulties and proposals for future work.

Chapter 2

State of Art

This chapter will provide a brief theoretical foundation regarding mobile robots, instances of localization problems and trilateration techniques.

2.1 Mobile robots

Navigating purposefully is a fundamental ability to most animals and every intelligent organism. A mobile robot must have this ability in order to do its tasks considering the environment, obstacles and the objectives like space robots as shown in the Figure 2.1 [12].

Mobile robotics is a specialized area of research focused on the control of self-driving and partially autonomous vehicles. What distinguishes mobile robotics from other research domains, like traditional manipulator robotics, artificial intelligence, and computer vision, is its primary focus on addressing challenges related to comprehending extensive spatial environments, surpassing the confines of what can be perceived from a single viewpoint [6].

To behave in large-scale not only needs to deal with incremental acquisition of knowledge, the estimated error, the recognition of important objects or places, and real-time responses, but it also requires the full integration of these functionalities [6].

The word ***autonomous*** have its roots in the Greek for "self-willed". The term "robot"

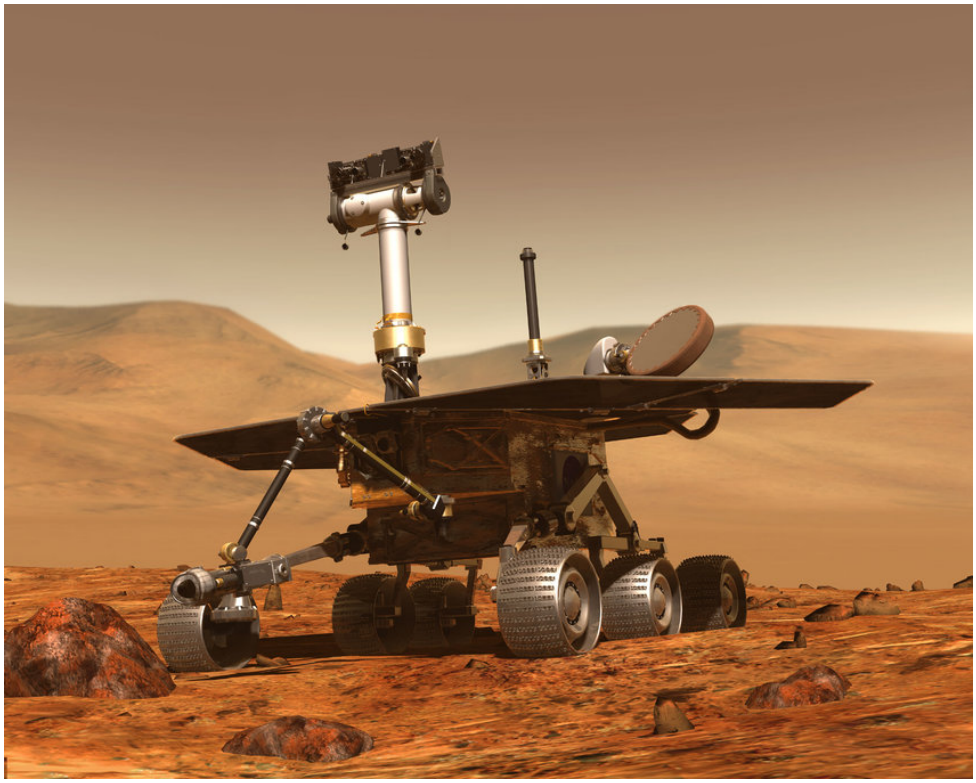


Figure 2.1: Mars Exploration Rover-a semi-autonomous exploration robot from NASA [11].

was introduced by Karel Čapek in 1923 and its derived from the Czech word "robota" that means "labour" and "robotinik", meaning "workman" [6].

2.1.1 Mobile Robot kinematics

Kinematics represents the fundamental examination of mechanical system behavior. In the field of mobile robotics, comprehending the mechanical characteristics of a robot is crucial, serving a dual purpose: aiding in the design of well-suited mobile robots for specific tasks and facilitating the development of control software tailored to a particular mobile robot hardware configuration [15].

Robot position

It is going to be considered the robot as a rigid body on wheels, operating on a horizontal plane as shown in the Figure 2.2 [15].

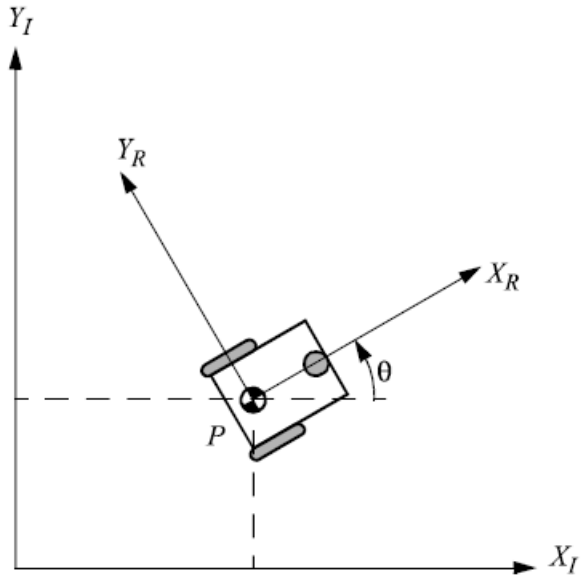


Figure 2.2: The global reference frame and the robot local reference frame [15]

To find the global position of the robot on the plane, it is going to be established the relationship between the global frame I and the robot frame R. To specify the robot position it is going to be set a point named P on the robot chassis and as the reference frame R of the robot. The position P represents the global position of the robot x and y, and the angular difference θ between frames I and R. The pose of the robot is described as:

$$\zeta = [x, y, \theta]^T \tag{2.1}$$

To describe movement, it is needed to map motion along the axes of the global reference frame to motion along the axes of the robot's reference frame. That could be done using

the orthogonal rotation matrix:

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

This matrix can be used to map motion in the global reference frame X_I, Y_I in terms of motion of the robot's reference frame X_R, Y_R.

$$\dot{\zeta}_R = R(\theta)\dot{\zeta} f_I \tag{2.3}$$

2.2 Localization problem

The challenge of robot localization revolves around answering the essential question: 'Where is the robot positioned?'. In other words, it involves the robot's ability to determine its precise coordinates, including the x and y positions and its orientation within a global reference system. The importance of localization plays a critical role in the functionality of numerous autonomous robot systems. Without a clear understanding of its own location, a robot faces significant obstacles in making decisions and executing tasks effectively. Is considered one of the most fundamental problem to provide robots truly autonomous capabilities [12].

To enable a robot to move from one point to another, it needs to know its position and orientation (pose) at any given point in time [13]. Therefore, precise localization is a fundamental requirement when dealing with mobile robot control problems. According to [8], there are two types of localization: relative and absolute. The former is provided through sensors that measure variables related to the robot's internal dynamics. For example, incremental encoders with optical sensors are placed on the wheels or the steering axis of the vehicle. At each sampling instant, the position is estimated based on the encoder increment over a period with multiple samples.

One drawback of this method is that errors accumulate with each measurement, leading to issues of inaccuracy and imprecision in the estimated position and orientation, especially on sinuous trajectories [18].

In the case of absolute position, it is determined by a set of sensors that measure parameters of the environment in which the vehicle is operating. Typically, sonars fixed to the vehicle are used to measure distances while respecting obstacles encountered in the known environment. One of the significant challenges in using sonars is their dependency on environmental characteristics, meaning that potential changes in the scenario's parameters can lead to misinterpretations in measurements by the localization algorithm [10].

2.2.1 Problem instances

The localization have some problem instances. In the context of position tracking, the robot possesses knowledge of its initial location, and the main objective of localization is to continuously monitor and update the robot's position as it moves through its environment. This specific challenge is typically addressed using techniques known as *tracking* or *local methods* [12].

Another problem is the $wake - up$ robot or *global positioning* problem. It is more difficult because the robot does not know its initial position. Methods to solve this type of problem are called *global* techniques.

One harder problem is the *kidnapped robot problem*. The robot does not know where it is because it was moved to another location without the robot being aware of this. The issue is for the robot it is to know that was kidnapped and to find the new location coordinates.

The technique used for these localization problems is known as Simultaneous Localization and Mapping (Simultaneous Localization and Mapping), which deals with simultaneous localization and mapping through four main actions: mapping, perception, localization, and modeling [14].

2.3 Trilateration

One of the solutions to the localization problem is using the trilateration method. Trilateration is a technique to determine the position of an object based on simultaneous range measurements from three stations located at known sites as shown in the figure 2.3. This is a common operation not only in robot localization, but also in other areas [17].

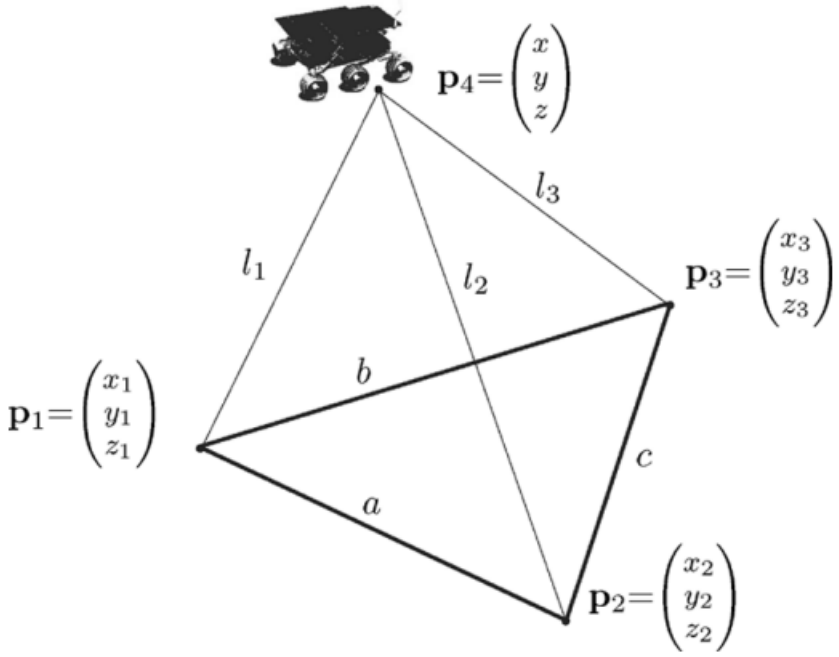


Figure 2.3: Representation of a trilateration technique applied to a robot [17]

It can be expressed as the problem of finding the intersection of three spheres, that is, finding the solutions to the following system of quadratic equations:

$$l_1^2 = (x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2 \tag{2.4}$$

$$l_2^2 = (x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2 \tag{2.5}$$

$$l_3^2 = (x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2 \tag{2.6}$$

Where p_i=(x_i,y_i,z_i) are the coordinates of the known station i and l_i is the distance measured between the robot and the station. Considering a 2D plane, the equations can

be represented as:

$$l_1^2 = (x_1 - x)^2 + (y_1 - y)^2 \tag{2.7}$$

$$l_2^2 = (x_2 - x)^2 + (y_2 - y)^2 \tag{2.8}$$

$$l_3^2 = (x_3 - x)^2 + (y_3 - y)^2 \tag{2.9}$$

For i known stations:

$$l_i^2 = (x_i - x)^2 + (y_i - y)^2 \tag{2.10}$$

These equations are going to be developed in the next chapter.

2.4 SimTwo environment

SimTwo is a realistic simulator that allows the implementation and simulation of various types of robots in a realistic environment. It supports the following types of robots [3]:

1. **Wheeled Mobile Robots:**
    - Differential drive
    - Robots with omnidirectional wheels
    - Manipulators
2. **Legged Mobile Robots:**
    - Bipedal robots
    - Quadrupeds
    - Hexapods
3. **Lighter than Air Vehicles:**
    - These are likely airships or balloons with propellers for propulsion.

4. **Underwater Vehicles:**

    - These are likely underwater drones or submersibles with thrusters for propulsion.

5. **Custom Robots:**

    - SimTwo allows the creation of custom robots that consist of rigid bodies connected by hinge or prismatic joints, which can be optionally actuated by electric motors. These custom robots may use classical wheels, omnidirectional wheels, or propellers for movement.

The drive and motor system can include the following components:

- **DC Motor:** The system may consist of a DC motor, optionally equipped with a gearbox and a controller.
- **Controllers:** Several types of controllers can be used in SimTwo:
    - A PID controller applied to position or velocity signals.
    - A state feedback controller.
- **DC Motor Model:** The DC motor model within SimTwo accounts for various nonlinear elements, including voltage saturation, current limitation, and Coulomb friction.

  By simulating robots in a realistic environment and modeling them as systems of rigid bodies with various joint types, SimTwo provides a valuable tool for researchers, engineers, and robotics enthusiasts to test and refine their robot designs and control algorithms without the need for physical prototypes. This can save time and resources in the development and testing of robotic systems. SimTwo offers a comprehensive set of drive and control systems for simulating various robots.

In addition to the low-level control system, SimTwo provides the flexibility to feed reference signals to these controllers from a higher-level controller implemented by the user. This higher-level controller can be implemented using the following methods:

- A scripting language, which is editable and compilable within SimTwo.
- A remote program that communicates via UDP packets or over the serial port.

This modular approach to drive and control systems enables users to fine-tune and customize the behavior of their robots within the SimTwo simulator, making it a powerful tool for robotic development and testing.

2.4.1 Simtwo overview

When opened, Simtwo have several windows [7], explained below .

- Scene visual: The real time 3D visualization of the robot shown in Figure 2.4.

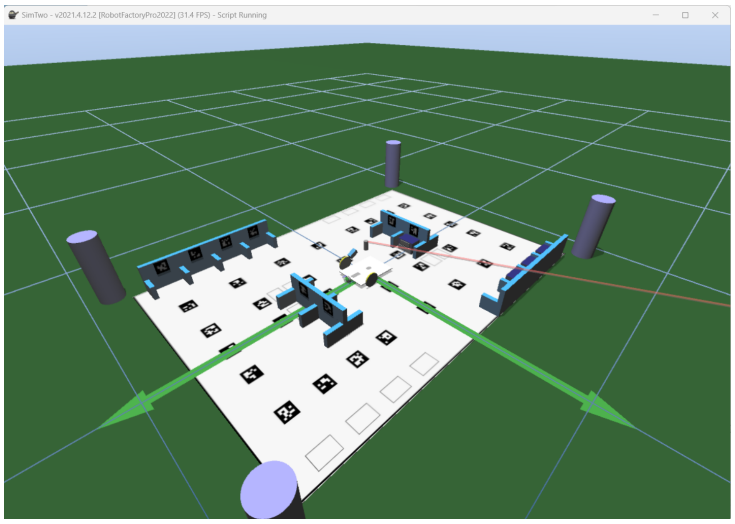


Figure 2.4: Scene Window

- Scene editor: It is the XML editor. The scene tab contains information on the scene. The track tab defines the track, including sensors. It is possible to add obstacles. The AGV tab contains the robot's editor (Figure 2.5).

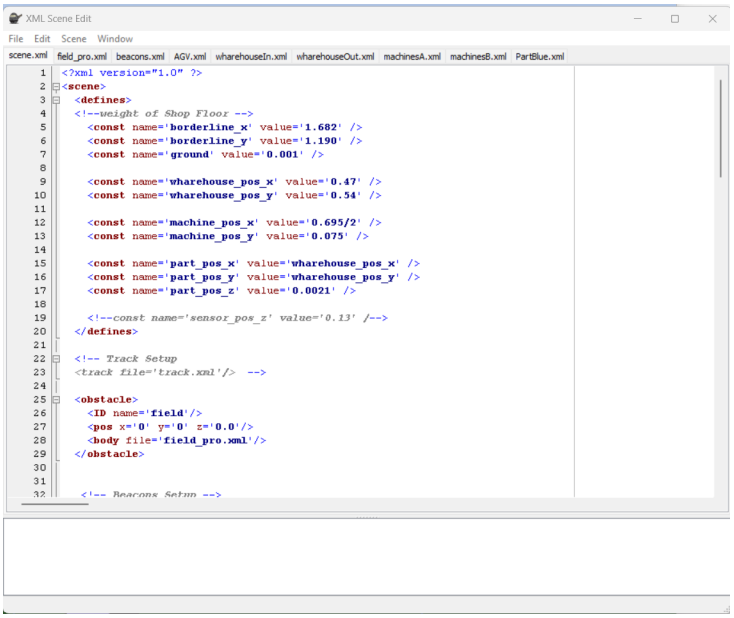


Figure 2.5: XML editor Window

- Control editor: It is the Pascal editor (Figure 2.6) to control procedure defined for the robot (speed, logic, for example).

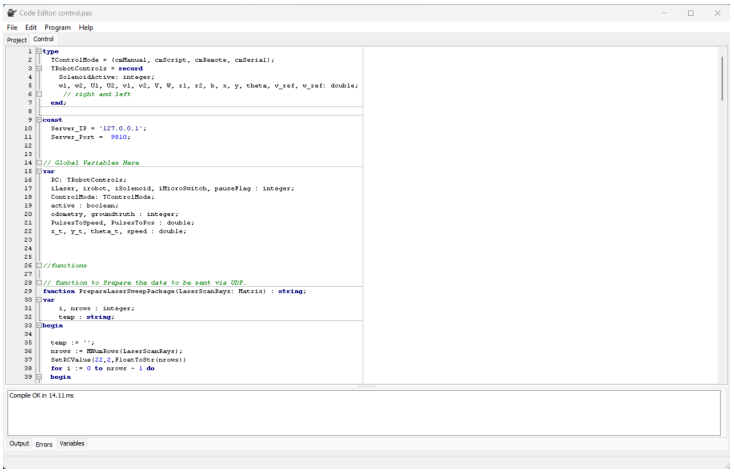


Figure 2.6: Editor Window

- Config: It is used to control the camera and scene as well as simulation speed and debugging (Figure 2.7).

- Chart: It is used to plot simulation variables (Figure 2.8).

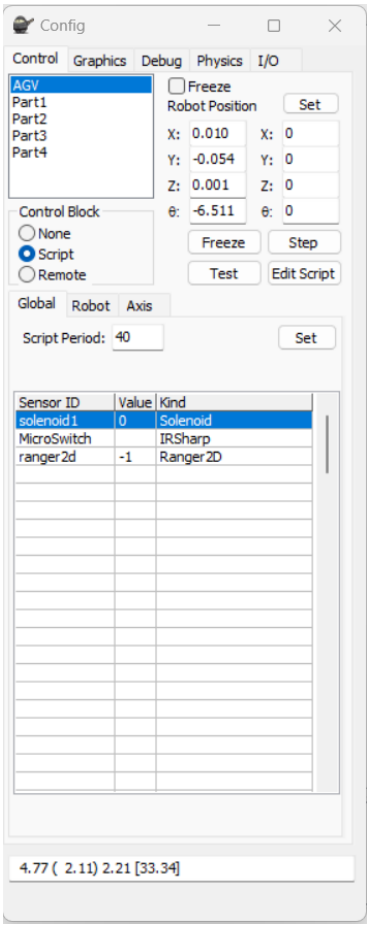


Figure 2.7: Config Window

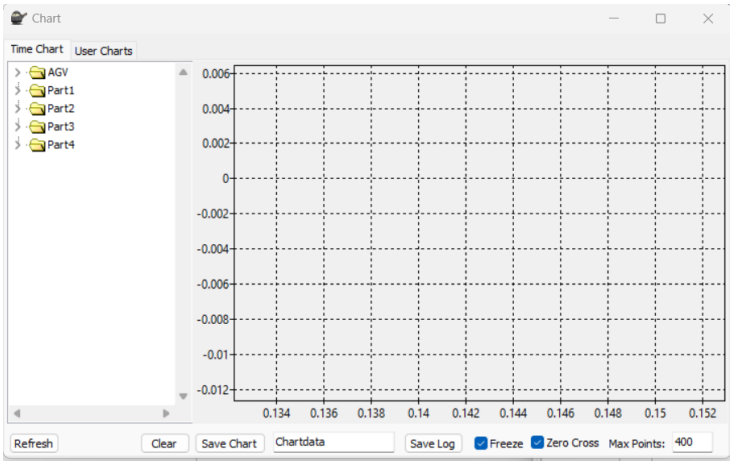


Figure 2.8: Chart Window

- Sheets: Displays real time values and it is possible to set buttons (Figure 2.9).

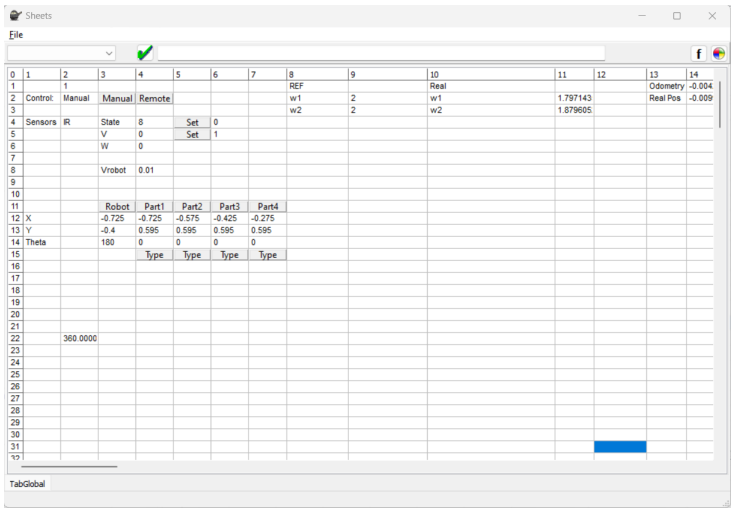


Figure 2.9: Sheets Window

Chapter 3

Development

In this work it was used the SimTwo Robot Factory 2022 shown in Figure 3.1 that simulates a factory plant. In this scene, a differential Automated Guided Vehicle (AGV) robot is inserted in the plant. The AGV have the LiDAR sensor that can returns the distance between an object, for example a beacon, and the angle between its frame reference (robot's frame) and the beacon's angle. The Figure 3.1 is the Scene of the work's simulation environment in SimTwo.

In this work, in order to simplify the problem, it is going to be considered the following system shown in Figure 3.2:

The main objective is to locate the robot's global position using these four beacons. Basicaly, the algorithm has 6 phases: initialization, data acquisition from the LiDAR, average distances and angles among robot and beacons, the θ value, the beacon identification and finally, the estimated robot position and respective errors. This chapter will show the core of the work, explaining the code.

3.1 Initialization

As a first step is necessary to set some variables. The initial position of the robot, according to Equation (2.1) is set as $\zeta_{initial} = [0, 0, rad(0)]^T$.

Simtwo have some control modes but in this work the manual mode is used.

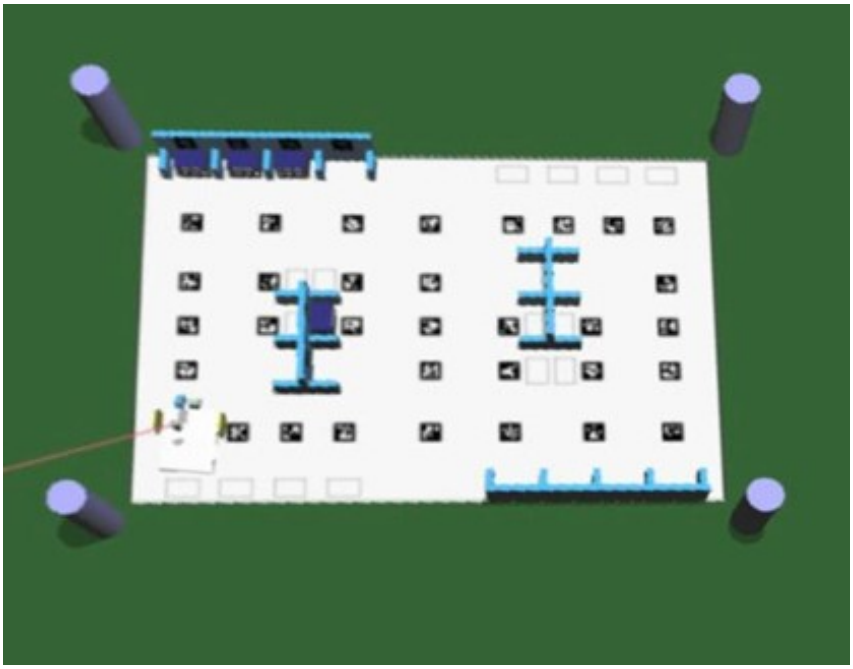


Figure 3.1: Robot Factory

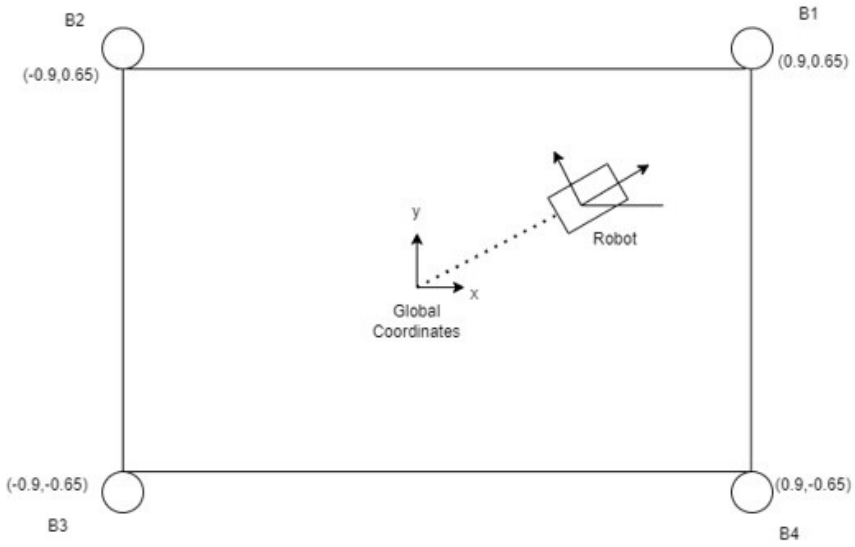


Figure 3.2: Representation of the system

```
    procedure Initialize;
begin
  irobot := GetRobotIndex('AGV');
```

```
  iSolenoid := GetSensorIndex(irobot, 'solenoid1');
  iLaser := GetSensorIndex(irobot, 'ranger2d');
  iMicroSwitch := GetSensorIndex(irobot, 'MicroSwitch');
  SetRobotPos(irobot, 0, 0, 0.001, rad(0));
  RC.x := GetRobotPos2D(irobot).x;
  RC.y := GetRobotPos2D(irobot).y;
  RC.theta := GetRobotPos2D(irobot).angle;
  active := True;
  RC.r1 := 0.06884/2;
  RC.r2 := 0.06884/2;
  RC.b := 0.186;
  pauseFlag := 0;
  odometry := 0;
  PulsesToSpeed := 2 * pi / (3840*0.04);
  PulsesToPos := 2 * pi / (3840);
  groundtruth := 0;
  ControlMode := cmManual;
    active := False;
    SetMotorControllerState(irobot, 0, false);
    SetMotorControllerState(irobot, 1, false);
end;
```

In this procedure, the robot is defined as 'AGV' that is the XML file, the sensors are defined and the procedure "SetRobotPos()" defines the initial position of the robot and other constants are set.

3.2 LIDAR measaments

The YDLiDAR x4 LIDAR (Figure 3.3) sensor performs 360 measurements each cycle counterclockwise, corresponding to 360 degrees. It has a range frequency of 5 kHz, a range distance between 0.12 m to 10 m and an angle resolution of 0.43^o to 0.86^o. Considering that the model dimensions are (1,68m x 1,18m), this LiDAR is applicable. If an object or obstacle is detected, the LiDAR returns the distance and angle value, if there is no detection, a value of -1 is returned [1].

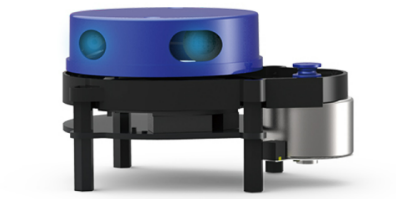


Figure 3.3: YDLiDAR X4 [1]

In the XML file the LiDAR has been previously set considering its datasheet and in the code, two arrays distance and angle were created, which store the values of distance and i (angle), respectively.

```
1     for i:= 1 to (length(distance)-1) do
2             begin
3             distance[i-1]:= Mgetv(LaserScanRays, i, 0);
4             angle[i-1]:=i;
5             j:=j+1;
6             end;
```

3.3 Average distances and angles

The main idea is to identify certain points with measurements and beginnings and ends of each beacon found. From the LIDAR readings, the points corresponding to each beacon found will be added in order to find the average distance and angle value for each beacon. The table 3.1 below shows an example of when a beacon is found.

Table 3.1: Example of beacon found

Angle	1	2	3	4	5	6
LiDAR Measurement	-1	0.1	0.12	0.11	0.1	-1

One of the problems encountered during the development of the code was the existence of a "fifth beacon" as shown in table 3.2. This occurs in the scenario where the robot starts measurements in the middle of a beacon and ends up having more measurements associated with it.

Table 3.2: Example of a middle of a beacon found

Angle	1	2	3	4	....	358	359	360
LiDAR Measurement	0.2	0.2	0.23	-1	....	-1	0.2	0.2

The developed code checks each beacon start situation:

1. When the previous value, i_{previous}, is negative and the current value, i_{current} , is positive, it indicates the beginning of a beacon, and the count of beacons found must be increased.

2. If the first reading is a positive number, there are two possible scenarios:

   (a) Measurements started at the beginning of a beacon.

   (b) Measurements started in the middle of a beacon.

   If the second scenario occurs, at the end, the LiDAR will have found 5 beacons. In this case, it is going to be added the measurements of this fifth beacon to the first one and set the count of beacons found as 4.

3. If the LiDAR starts reading in the middle of a beacon (when both the current and previous distances are positive), the code must sum the distances until it identifies a situation where the current distance is positive, and the next one is negative.

4. After that, it is calculated the average distances for each beacon found.

The code is shown above:

```
for i:=1 to (length(distance)) do
      begin

          if ((((distance[i-1])<0) and ((distance[i])>0)) or
             ((distance[0])> 0) and (i =1)) then
          begin

              bf:=bf+1;
              if (i = 1) and (distance[0]> 0) then begin
              sumdistance[bf-1]:=sumdistance[bf-1]+distance
                 [0];
              sumangle[bf-1]:=sumangle[bf-1]+angle[0];
              numberofpoints[bf-1]:=numberofpoints[bf-1]+1;
              end;

              sumangle[bf-1]:=sumangle[bf-1]+angle[i];
              sumdistance[bf-1]:=sumdistance[bf-1]+distance[i
                 ];
              numberofpoints[bf-1]:=numberofpoints[bf-1]+1;

          end

```

```
20          else if  (((distance[i-1]) > 0) and ((distance[i])
               > 0)) then
21          begin
22
23              sumdistance[bf-1]:=sumdistance[bf-1]+distance[i
                   ];
24              sumangle[bf-1]:=sumangle[bf-1]+angle[i];
25              numberofpoints[bf-1]:=numberofpoints[bf-1]+1;
26
27          end
28
29      end;
30
31
32      if (bf=5) then
33      begin
34
35          sumdistance[0]:=sumdistance[0]+ sumdistance[4];
36          sumangle[0]:=sumangle[0]+ sumangle[4] - 360*
               numberofpoints[4];
37          numberofpoints[0]:=numberofpoints[0]+ numberofpoints
               [4];
38          bf := 4;
39
40      end;
```

3.4 θ value

The aim is to determine the current value of θ. The initial value of θ is known, and when a rotation of γ degrees occurs, it can be expressed as follows:

$$\theta_{\text{current}} = \theta_{\text{previous}} + \gamma \tag{3.1}$$

However, the value of γ is not known; instead, the angle between the robot frame and the beacon is known. In this scenario, γ can be determined as the difference between the previous ϕ and the current ϕ:

$$\gamma = \phi_{\text{previous}} - \phi_{\text{current}} \tag{3.2}$$

By substituting these values, the following equation can be established:

$$\theta_{\text{current}} = \theta_{\text{previous}} + \phi_{\text{previous}} - \phi_{\text{current}} \tag{3.3}$$

In the first iteration, the current ϕ is computed as the average of the beacon angles. The previous ϕ is set to be equal to the current ϕ, especially in the first iteration, and the previous θ is based on a measurement provided by the software as shown in the code below.

```
1 begin
2 for m := 0 to 3 do
3  begin
4    average[m] := ((sumdistance[m]) / numberofpoints[m]) + rb
       ;
5    averageangle[m] := (sumangle[m]) / numberofpoints[m];
6      actual_phi[m] := averageangle[m]
7    if (gama = 1) then
8    begin
9      previous_phi[m] := actual_phi[m];
```

```
10        previous_theta := (180 * GetRobotTheta(GetRobotIndex('
              AGV'))) / pi;
11        gama := 0;
12      end;
13    end;
14    for n := 0 to 3 do
15     begin
16       if actual_phi[n] <> previous_phi[n] then
17       begin
18        actual_theta := previous_theta + previous_phi[0] -
              actual_phi[0];
19        previous_phi[n] := actual_phi[n];
20        previous_theta := actual_theta;
21       end;
22    end;
```

First the code sum the distances and angles of each beacon and finds its average to each beacon and defines the current ϕ as the value of the average angle. There is one first time exception that is to set the values of previous ϕ as the current ϕ and the previous θ as the initial θ of the robot, that is got by the procedure GetRobotTheta().

After that, if the current ϕ is different from the previous one, then the value is calculated as shown before.

3.5 Beacon identification

Considering the general transformation of a vector shown in Figure 3.4.

The general equation for translation and rotation is defined as [4]:

$$^{A}P = {}^{A}_{B}R\,{}^{B}P + {}^{A}P_{BORG} \tag{3.4}$$

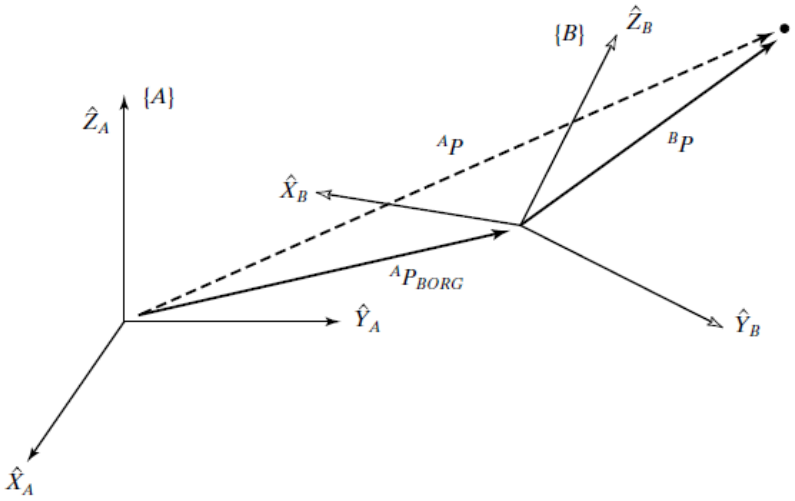


Figure 3.4: General transformation of a vector [4]

Where ${}^AP$ is the position of P regarding to frame A, ${}^A_BR$ is the Rotation Matrix in z axis, ${}^BP$ is the position of P regarding to frame B and ${}^AP_{BORG}$ is the position of the origin of frame B regarding frame A. When applied to the problem context, it is given this equation:

$$ {}^G_BP = {}^G_R\, R^R_BP + {}^RP_{RORG} \tag{3.5} $$

Where ${}^G_BP$ is the global position of the beacon, ${}^G_RR$ is the rotation matrix in z axis, ${}^R_BP$ is the position of the beacon regarding the robot's position and ${}^RP_{RORG}$ is the global position of the robot. In other words:

$$ \begin{bmatrix} X^G_{bfn} \\ Y^G_{bfn} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X^R_{bfn} \\ Y^R_{bfn} \end{bmatrix} + \begin{bmatrix} X_{RG} \\ Y_{RG} \end{bmatrix} \tag{3.6} $$

Where X^G_{bfn} and Y^G_{bfn} are the coordinates of Beacon found n regarding the global frame, X^R_{bfn} and Y^R_{bfn} are the coordinates of the beacon found n regarding the robot's frame and X_{RG} and Y_{RG} are the global coordinates of the robot. If X^R_{bfn} and Y^R_{bfn} are decomposed:

$$ \begin{bmatrix} X^R_{bfn} \\ Y^R_{bfn} \end{bmatrix} = \begin{bmatrix} d_n \cdot \cos(\beta) \\ d_n \cdot \sin(\beta) \end{bmatrix} \tag{3.7} $$

By substituting Equation 3.7 in Equation 3.6 , it is obtained the following:

$$\begin{bmatrix} X_{bfn}^{G} \\ Y_{bfn}^{G} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} d_n \cdot \cos(\beta) \\ d_n \cdot \sin(\beta) \end{bmatrix} + \begin{bmatrix} X_{RG} \\ Y_{RG} \end{bmatrix} \tag{3.8}$$

In order to consider the first iteration, a variable *fi* was created to set the current value of x and y as 0,0 once, as defined in the initialization. In the next iteration, the pose used is going to be the global position calculated in the next section (Section 3.6). The code was written as following:

```
1  if fi=1 then begin
2    xatual:=0;
3    yatual:=0;
4    fi:=0;
5  end;
6       for j:=0 to 3 do
7          begin
8              xb[j]:=average[j]*(cos(rad(actual_theta))*cos(rad
                  (averageangle[j]))-sin(rad(actual_theta))*sin(
                  rad(averageangle[j])))+xatual;
9              yb[j]:=average[j]*(sin(rad(actual_theta))*cos(rad
                  (averageangle[j]))+cos(rad(actual_theta))*sin(
                  rad(averageangle[j])))+yatual;
10         end;
```

To organize the code, each beacon is going to be defined as: Beacon 4 refers to B1, Beacon 2 refers to B2, Beacon 1 refers to B3 and Beacon 3 refers to B4.

After calculating the value of each beacon found, the algorithm it is going to be compare the value of the global coordinates of the n beacons found with the nominal global value of the beacons, using quadrants. It is possible to have 4 valid scenarios:

- If the first beacon found coordinates are both positives, that means $x_{bf1} > 0$ and

$y_{bf1} > 0$, then the sequence of measurements are B1, B2, B3 and B4.

- If the first beacon found x coordinate is negative and the y is positive, that means $x_{bf1} < 0$ and $y_{bf1} > 0$, then the sequence of measurements are B2, B3, B4 and B1.
- If the first beacon found coordinates are both negatives, that means $x_{bf1} < 0$ and $y_{bf1} < 0$, then the sequence of measurements are B3, B4, B1 and B2.
- If the first beacon found x coordinate is positive and the y is negative, that means $x_{bf1} > 0$ and $y_{bf1} < 0$, then the sequence of measurements are B4, B1, B2 and B3.

The integral code is in Appendix C.

3.6 Robot estimated global position

Once the beacons have been correctly identified, a system of equations will be set up to determine the values of $X_{RG,estimated}$ and $Y_{RG,estimated}$. In this simulated plant were placed four beacons (B1, B2, B3 and B4), as shown in the Figure 3.2. The global coordinates x_{Bn}, y_{Bn} (for n = 1,...,4) of each beacon are known, the relative distance d_{Bn} between the robot's sensor and the beacon found B_{fn} and the the angle ϕ_{Bn} between the beacon and the robot's frame reference can be calculated from the LiDAR data.

The coordinates of each beacon are given as following:

B1 = (0.90, 0.65)

B2 = (-0.90, 0.65)

B3 = (-0.90, -0.65)

B4 = (0.90, 0.65)

The AGV is differential with two drive wheels and a castor wheel, as shown in the Figure 3.5.

The objective is to know the global pose $\zeta_{current}$ of the robot using the coordinates and measurements from the robot's sensor that is defined as the Equation 2.1:

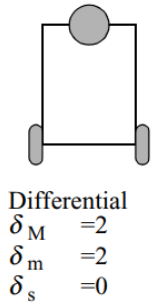


Figure 3.5: Differential AGV [15]

$$\zeta_{current} = [x, y, \theta]^T \tag{3.9}$$

In order to do that, the trilateration equations are going to be applied.

As seen in Chapter 2, the distance between the robot and each beacon is defined as the equation 2.10, when applied to the problem context:

$$d_{Bn}^2 = (x_{Bn} - x)^2 + (y_{Bn} - y)^2 \tag{3.10}$$

Where x and y are the global coordinates of the robot. Expanding Equation 3.10:

$$d_{Bn}^2 = x_{Bn}^2 - 2.x_{Bn}.x + x^2 + y_{Bn}^2 - 2.y_{Bn}.y + y^2 \tag{3.11}$$

By isolating x^2+y^2, it is obtained:

$$x^2 + y^2 = 2.x_{Bn}.x + 2.y_{Bn}.y + d_{Bn}^2 - y_{Bn}^2 - x_{Bn}^2 \tag{3.12}$$

To simplify the problem, it is going to set the constants as:

$$a_{Bn} = 2.x_{Bn} \tag{3.13}$$

$$b_{Bn} = 2.y_{Bn} \tag{3.14}$$

$$c_{Bn} = d_{Bn}^2 - y_{Bn}^2 - x_{Bn}^2 \tag{3.15}$$

Replacing equations 3.13, 3.14 and 3.15 in equation 3.9:

$$x^2 + y^2 = a_{Bn}.x + b_{Bn}.y + c_{Bn} \tag{3.16}$$

For n=1,2,3,4:

$$x^2 + y^2 = a_{B1}.x + b_{B1}.y + c_{B1} \tag{3.17}$$

$$x^2 + y^2 = a_{B2}.x + b_{B2}.y + c_{B2} \tag{3.18}$$

$$x^2 + y^2 = a_{B3}.x + b_{B3}.y + c_{B3} \tag{3.19}$$

$$x^2 + y^2 = a_{B4}.x + b_{B4}.y + c_{B4} \tag{3.20}$$

As there are four beacons positioned in a mirrored manner, there will be some ambiguities. When is used two horizontal beacons in a row, the value of y_{Bn} are the same, so, $b_{Bn} = b_{Bn+1}$ so in this case, it is just possible to calculate $x_{n,n+1}$. On the other hand, when it is used two vertical beacons in a row, the value of x_{Bn} are the same, so $a_{Bn} = a_{Bn+1}$, so it is only possible to calculate $y_{n,n+1}$. It is going to be used Beacons (4,2), (2,1), (1,3), and (3,4) with bilaterations, and along with the diagonals (4,1) and (2,3) with two bilaterations combined.

3.6.1 Robot global position based on beacons B1 and B2

In this particular case, b_{B1}= b_{B2}, so no $y_{1,2}$ can be calculated. Equating Equation (3.17) to Equation (3.18):

$$a_{B1}.x + b_{B1}.y + c_{B1} = a_{B2}.x + b_{B2}.y + c_{B2} \tag{3.21}$$

Isolating x, and calling as $x_{1,2}$:

$$x_{1,2} = \frac{c_{B2} - c_{B2}}{a_{B1} - a_{B2}} \tag{3.22}$$

To $y_{1,2}$ the value is going to be undefined, so:

$$y_{1,2} = NaN \tag{3.23}$$

3.6.2 Robot global position based on beacons B2 and B3

In this particular case, $a_{B2} = a_{B3}$, so no $x_{2,3}$ can be calculated. Equating Equation (3.18) to Equation (3.19):

$$a_{B2}.x + b_{B2}.y + c_{B2} = a_{B3}.x + b_{B3}.y + c_{B3} \tag{3.24}$$

So:

$$x_{2,3} = NaN \tag{3.25}$$

Isolating $y_{2,3}$:

$$y_{2,3} = \frac{c_{B3} - c_{B2}}{b_{B2} - b_{B3}} \tag{3.26}$$

3.6.3 Robot global position based on beacons B3 and B4

Similar to B1 and B2, b_{B3}= b_{B4}, so no $y_{3,4}$ can be calculated. Equating Equation (3.19) to Equation (3.20):

$$a_{B3}.x + b_{B3}.y + c_{B3} = a_{B4}.x + b_{B4}.y + c_{B4} \tag{3.27}$$

Isolating x, and calling as $x_{3,4}$:

$$x_{3,4} = \frac{c_{B4} - c_{B3}}{a_{B3} - a_{B4}} \tag{3.28}$$

To $y_{3,4}$ the value is going to be undefined, so:

$$y_{3,4} = NaN \tag{3.29}$$

3.6.4 Robot global position based on beacons B4 and B1

Similar to B2 and B3, $a_{B4} = a_{B1}$, so no $x_{4,1}$ can be calculated. Equating Equation (3.20) to Equation (3.17):

$$a_{B4}.x + b_{B4}.y + c_{B4} = a_{B1}.x + b_{B1}.y + c_{B1} \tag{3.30}$$

So:

$$x_{4,1} = NaN \tag{3.31}$$

Isolating $y_{4,1}$:

$$y_{4,1} = \frac{c_{B1} - c_{B4}}{b_{B4} - b_{B1}} \tag{3.32}$$

3.6.5 Robot global position based on diagonal beacons B1,B3 and B2,B4

To diagonal B1,B3:

$$a_{B1}.x + b_{B1}.y + c_{B1} = a_{B3}.x + b_{B3}.y + c_{B3} \tag{3.33}$$

Isolating x:

$$x = \frac{c_{B3} - c_{B1}}{a_{B1} - a_{B3}} + \frac{b_{B3} - b_{B1}}{a_{B1} - a_{B3}} y \tag{3.34}$$

Isolating y:

$$y = \frac{c_{B3} - c_{B1}}{b_{B1} - b_{B3}} + \frac{a_{B3} - a_{B1}}{b_{B1} - b_{B3}} x \tag{3.35}$$

To diagonal B2, B4:

$$a_{B2}.x + b_{B2}.y + c_{B2} = a_{B4}.x + b_{B4}.y + c_{B4} \tag{3.36}$$

Similarly to x and y:

$$x = \frac{c_{B4} - c_{B2}}{a_{B2} - a_{B4}} + \frac{b_{B4} - b_{B2}}{a_{B2} - a_{B4}} y \tag{3.37}$$

$$y = \frac{c_{B4} - c_{B2}}{b_{B2} - b_{B4}} + \frac{a_{B4} - a_{B2}}{b_{B2} - b_{B4}} x \tag{3.38}$$

From Equation (3.35) = (3.38):

$$\frac{c_{B3} - c_{B1}}{b_{B1} - b_{B3}} + \frac{a_{B3} - a_{B1}}{b_{B1} - b_{B3}} x = \frac{c_{B4} - c_{B2}}{b_{B2} - b_{B4}} + \frac{a_{B4} - a_{B2}}{b_{B2} - b_{B4}} x \tag{3.39}$$

$$x[\frac{a_{B3} - a_{B1}}{b_{B1} - b_{B3}} - \frac{a_{B4} - a_{B2}}{b_{B2} - b_{B4}}] = \frac{c_{B4} - c_{B2}}{b_{B2} - b_{B4}} - \frac{c_{B3} - c_{B1}}{b_{B1} - b_{B3}} \tag{3.40}$$

Isolating x and calling as $x_{d(1,3),d(2,4)}$:

$$x_{d(1,3),d(2,4)} = \frac{(c_{B4} - c_{B2})(b_{B1} - b_{B3}) - (c_{B3} - c_{B1})(b_{B2} - b_{B4})}{(a_{B3} - a_{B1})(b_{B2} - b_{B4}) - (a_{B4} - a_{B2})(b_{B1} - b_{B3})} \tag{3.41}$$

From Equation (3.34) = (3.37):

$$\frac{c_{B3} - c_{B1}}{a_{B1} - a_{B3}} + \frac{b_{B3} - b_{B1}}{a_{B1} - a_{B3}} y = \frac{c_{B4} - c_{B2}}{a_{B2} - a_{B4}} + \frac{b_{B4} - b_{B2}}{a_{B2} - a_{B4}} y \tag{3.42}$$

$$y[\frac{b_{B3} - b_{B1}}{a_{B1} - a_{B3}} - \frac{b_{B4} - b_{B2}}{a_{B2} - a_{B4}}] = \frac{c_{B4} - c_{B2}}{a_{B2} - a_{B4}} - \frac{c_{B3} - c_{B1}}{a_{B1} - a_{B3}} \tag{3.43}$$

Finally:

$$y_{d(1,3),d(2,4)} = \frac{(c_{B4} - c_{B2})(a_{B1} - a_{B3} - (c_{B3} - c_{B1})(a_{B2} - a_{B4})}{(b_{B3} - b_{B1})(a_{B2} - a_{B4}) - (b_{B4} - b_{B2})(a_{B1} - a_{B3})} \tag{3.44}$$

To have a best global position estimate of the robot, it is going to be used the average value, as shown in the next equations:

$$x_{RG,estimated} = \frac{x_{1,2} + x_{2,3} + x_{3,4} + x_{4,1} + x_{d(1,3),d(2,4)}}{5} \tag{3.45}$$

$$y_{RG,estimated} = \frac{y_{1,2} + y_{2,3} + y_{3,4} + y_{4,1} + y_{d(1,3),d(2,4)}}{5} \tag{3.46}$$

SimTwo provides the global position of the robot, so it is possible to calculate the absolute error in order to compare the accuracy of the results:

$$E_x = x - x_{RG} \tag{3.47}$$

$$E_y = y - y_{RG} \tag{3.48}$$

$$E_\theta = \theta - \theta_{estimated} \tag{3.49}$$

In the code, after identifying each beacon, the constants a, b and c are calculated and the equations are applied. The algorithm fluxogram is shown in the Appendix B and the code is shown in the Appendix C.

In the next chapter it is going to be shown the results and errors obtained.

Chapter 4

Results Obtained and Analysis of Results

This chapter presents the tests carried out to verify that the developed project meets the assumed objectives and, if solves the problem described in the previous chapter. The first part is to validate the beacon identification algorithm in order to see its problems and limitations and to test the robot estimated position algorithm, testing some poses and calculating respective errors.

4.1 Testing the Beacon Identification algorithm

In order to calculate the global position of the robot it is first needed to identify each beacon. To do that, the beacon identification algorithm is tested for 4 known scenarios, for θ=[0,90, 180, 270] degrees set in Config tab on SimTwo. In each of these θ values is expected that one of each beacons be the first detected. For instance, for $\theta = 0$ degrees, is expected that the first beacon detected be Beacon 4, if $\theta = 90$ degrees, the first one should be Beacon 2, if $\theta = 180$ degrees, the first one should be Beacon 1 and lastly, if $\theta = 270$ degrees, the first one should be Beacon 3. The subsections below will show the coordinates found and errors.

4.1.1 Sequence of Beacon detected: 4, 2, 1 and 3

The angle was set to 0 degrees in the config tab as shown in Figure 4.1 . The Table 4.1 shows the coordinates of each beacon found and its errors.

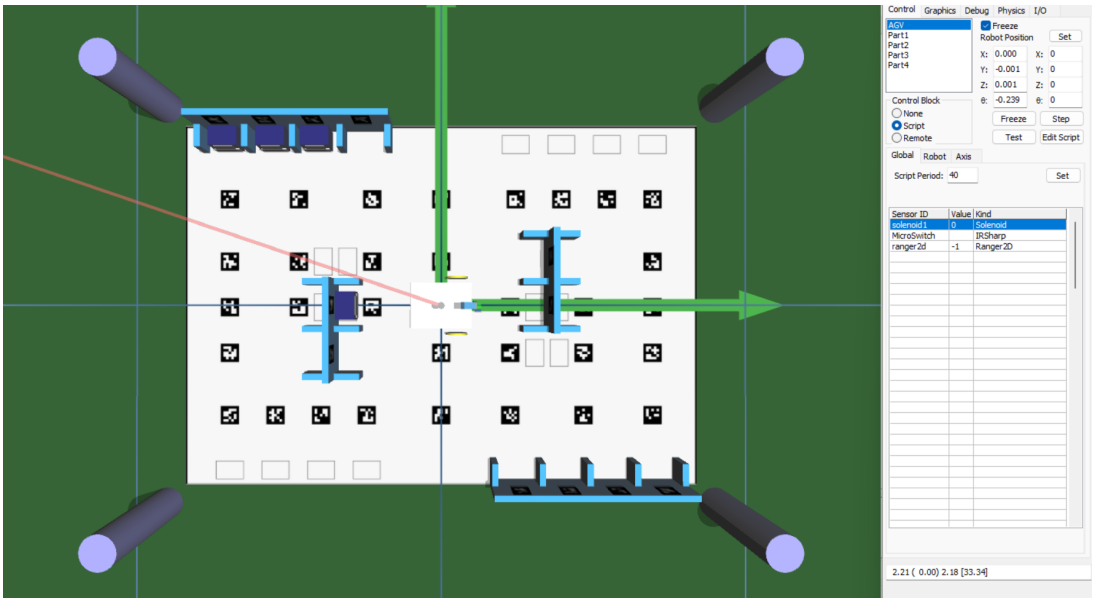


Figure 4.1: Beacon identification for $\theta = 0$ degrees

Table 4.1: Calculated coordinates of Beacons 4, 2, 1 and 3

	B_{f1}	**Error**	B_{f2}	**Error**	B_{f3}	**Error**	B_{f4}	**Error**
x_{bfn}	0,907883	-0,00788	-0,89861	-0,00139	-0,91118	0,01118	0,898925	0,001075
y_{bfn}	0,653851	-0,00385	0,658621	-0,00862	-0,63237	-0,01763	-0,65885	0,00885

4.1.2 Sequence of Beacon detected: 2, 1, 3 and 4

The angle was set to 90 degrees in the config tab as shown in Figure 4.2 . The Table 4.2 shows the coordinates of each beacon found and its errors.

Table 4.2: Calculated coordinates of Beacons 2, 1, 3 and 4

	B_{f1}	**Error**	B_{f2}	**Error**	B_{f3}	**Error**	B_{f4}	**Error**
xbfn	-0,90214	0,00214	-0,90992	0,00992	0,900571	-0,00057	0,916214	-0,01621
ybfn	0,656396	-0,0064	-0,63619	-0,01381	-0,65525	0,00525	0,640596	0,009404

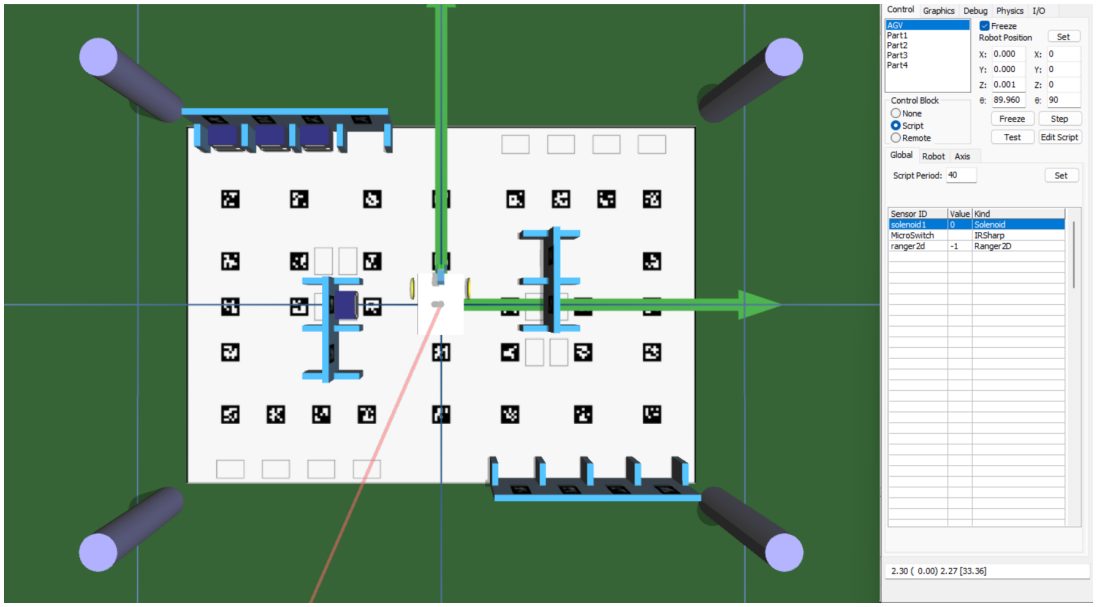


Figure 4.2: Beacon identification for $\theta = 90$ degrees

4.1.3 Sequence of Beacon detected: 1, 3, 4 and 2

The angle was set to 180 degrees in the config tab as shown in Figure 4.3 . The Table 4.3 shows the coordinates of each beacon found and its errors.

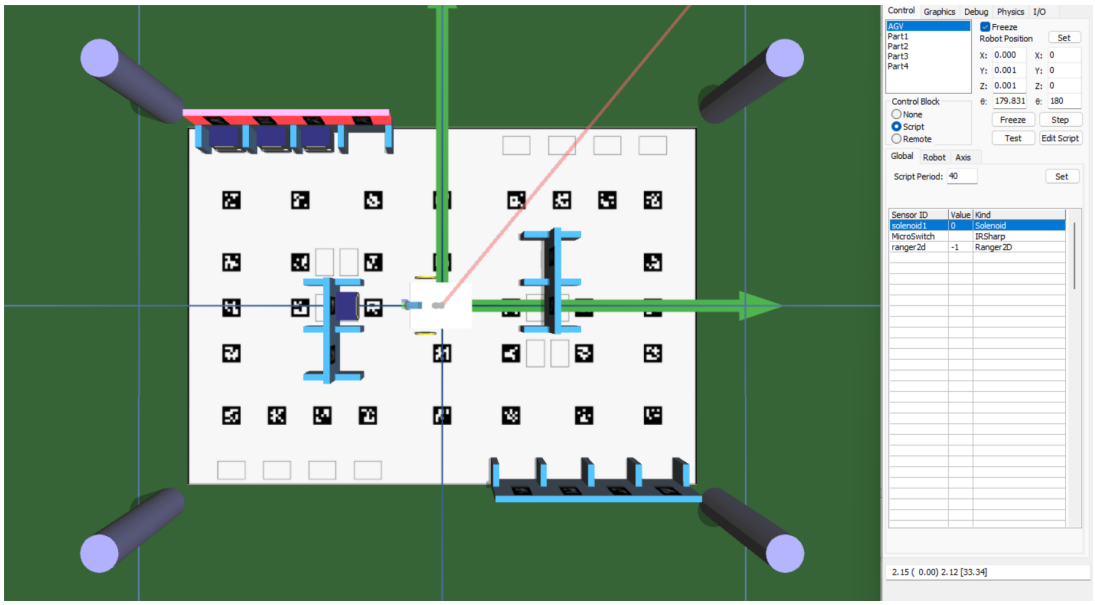


Figure 4.3: Beacon identification for $\theta = 180$ degrees

Table 4.3: Calculated coordinates of Beacons 1, 3, 4 and 2

	B_{f1}	**Error**	B_{f2}	**Error**	B_{f3}	**Error**	B_{f4}	**Error**
xbfn	-0,91122	0,01122	0,901195	-0,00119	0,914454	-0,01445	-0,8967	-0,0033
ybfn	-0,64591	-0,00409	-0,65883	0,00883	0,636286	0,013714	0,65555	-0,00555

4.1.4 Sequence of Beacon detected: 3, 4, 2 and 1

The angle was set to 270 degrees in the config tab as shown in Figure 4.4 . The Table 4.4 shows the coordinates of each beacon found and its errors.

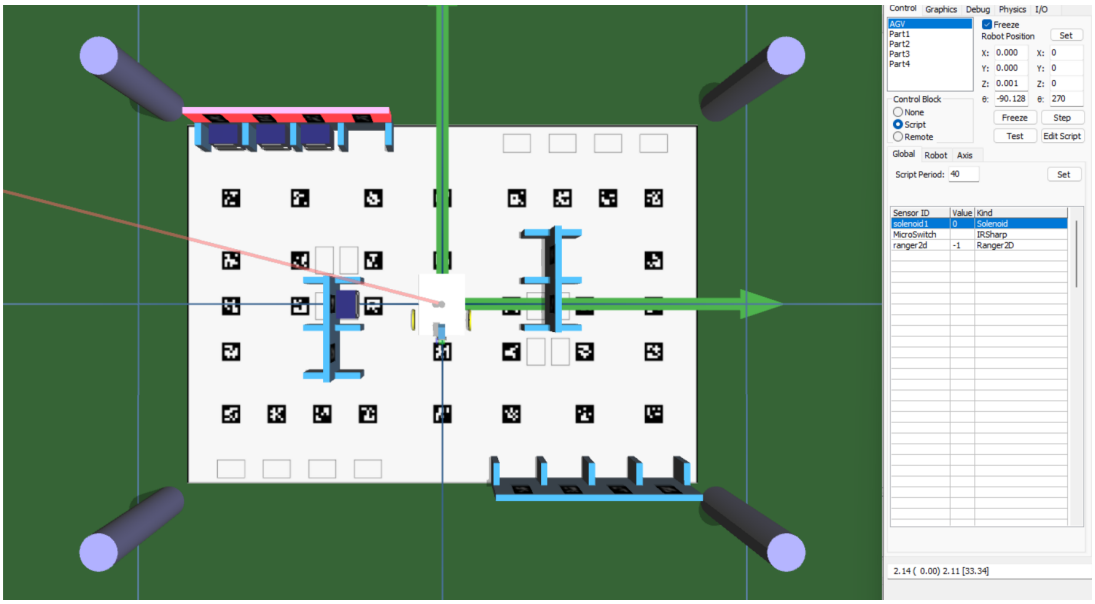


Figure 4.4: Beacon identification for $\theta = 270$ degrees

Table 4.4: Calculated coordinates of Beacons 3, 4, 2 and 1

	B_{f1}	**Error**	B_{f2}	**Error**	B_{f3}	**Error**	B_{f4}	**Error**
xbfn	-0,91122	0,01122	0,901195	-0,00119	0,914454	-0,01445	-0,8967	-0,0033
ybfn	-0,64591	-0,00409	-0,65883	0,00883	0,636286	0,013714	0,65555	-0,00555

While testing various positions, some limitations were found:

- If the robot is moved outside of the factory this algorithm looses its function
- If the robot approaches one of the beacons, the error increases significantly, but it does not invalidate the algorithm used (quadrants).

- The solution used is very specific because of the mirrored manner, so can not be applied in different beacon's layouts.

4.2 Testing the Robot Estimated Position algorithm

To test the algorithm, it was simulated 8 random points in order to test if the equations work properly and the values are consistent. These are shown in table 4.5 below.

Table 4.5: Positions estimated versus Real positions

i	$x_{RG,i,estimated}$	$x_{RG,i,groundtruth}$	$E_{x,i}$	$y_{RG,i,estimated}$	$y_{RG,i,groundtruth}$	$E_{y,i}$	$\theta_{i,estimated}$	$\theta_{i,groundtruth}$	$E_{i,\theta}$
1	-0,002807443	-0,001309687	0,00149775583	-0,00369	-0,00122	0,002473013	-0,004711633	-0,004711633	0,00
2	0,217148249	0,21541205	-0,00173619826	0,392511	0,401497	0,008985478	0,232106514	0,232106514	0,00
3	0,651283112	0,635270596	-0,01601251687	0,306509	0,321843	0,015334638	-0,284538315	-0,284538315	0,00
4	0,492428655	0,478093535	-0,01433511997	-0,29024	-0,3042	-0,013962186	-0,343633163	-0,343633163	0,00
5	-0,406991609	-0,395926952	0,01106465654	-0,44061	-0,4586	-0,017998068	2,829766579612	2,829766579612	0,00
6	-0,634434132	-0,616434634	0,01799949841	-0,12655	-0,13469	-0,008137588	1,320969797123	1,320969797123	0,00
7	-0,691740357	-0,667012274	0,02472808308	-0,30572	-0,32189	-0,016169444	2,013420540873	2,013420540873	0,00
8	-0,615873271	-0,597346663	0,01852660750	0,151066	0,155716	0,004650174	1,504164717567	1,504164717567	0,00

The Figure 4.5 below shows the real position of the robot in the global frame versus the estimated robot's position.

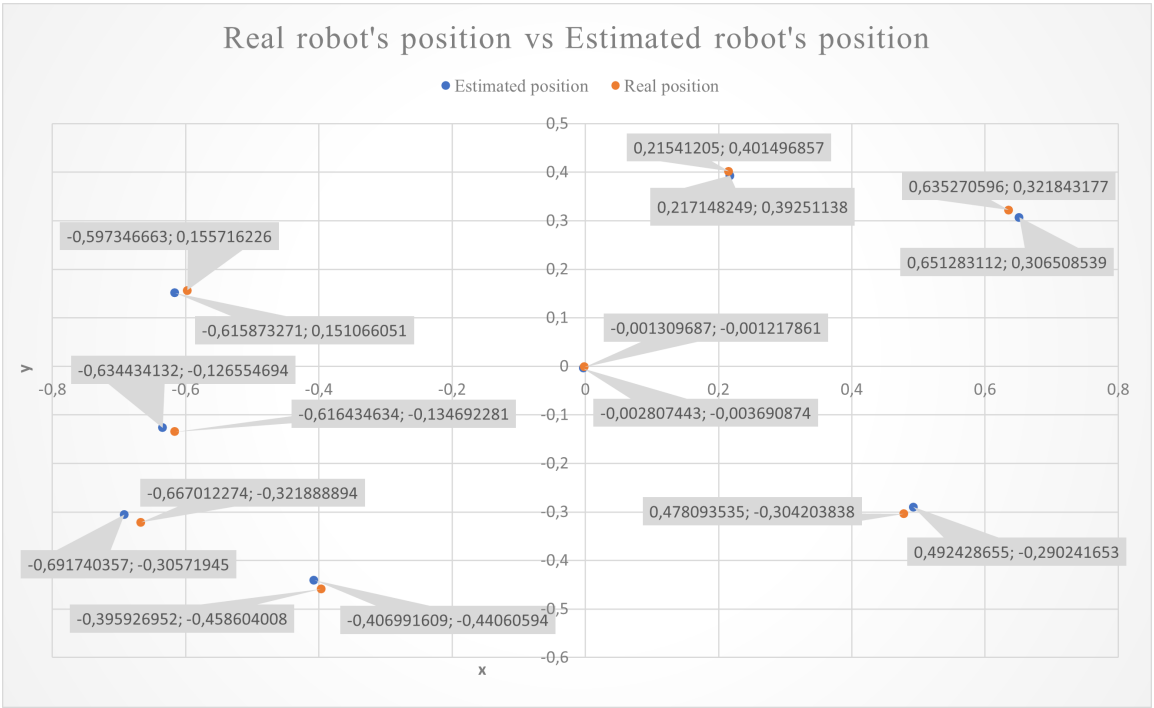


Figure 4.5: Real robot's position versus Estimated Robot's position

Analysing the results obtained, it is possible to see that:

- There is some issue calculating the θ value, because the error is "zero" to all points.
- On the other hand, the error found regarding the x_{RG} and y_{RG} coordinates were pretty low, considering the dimension of the model (1,68m x 1,18m). In these points, the maximum error was 0,02472808308, that it is approximately 2,47 cm. These errors can be associated to the approximations used when the average value is considered. Other point is that in this particular work trilateration was not properly used. Considering the layout of the beacons, bilateration was enough to solve the problem of sequential beacons, but with two reference points the consequence is increasing the error.
- Another point that was not taken in consideration was the angle of the LiDAR and the center of the beacon. The beacon has a significant radius that can interfere in the final calculation of the robot's global position, in this work, instead of calculating this angle, the value of the radius was just summed to the average distance in order to simplify the problem.

Each position is shown in the Figures 4.6, 4.7, 4.8, 4.9, 4.10, 4.11,4.12 and 4.13 below.

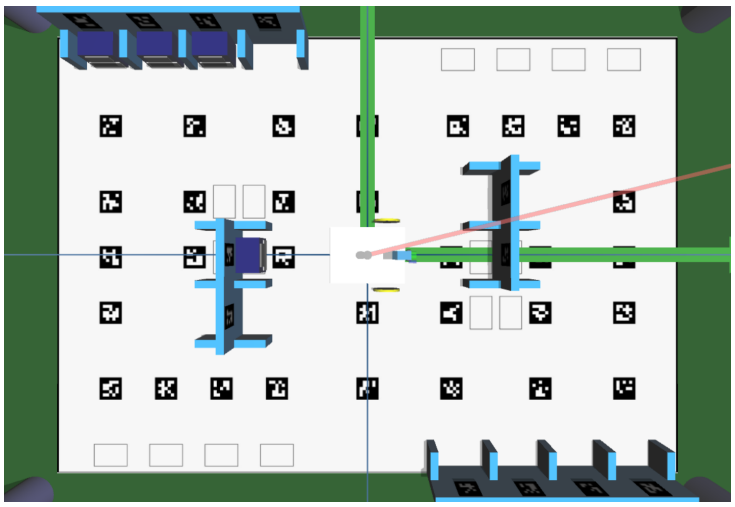


Figure 4.6: Position i=1

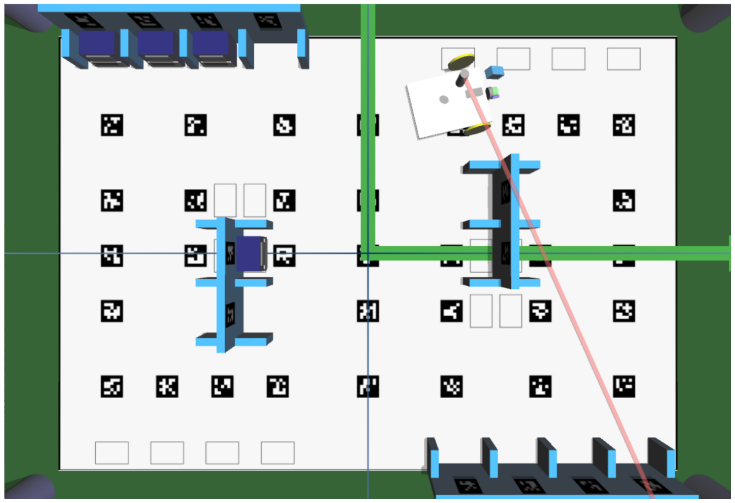


Figure 4.7: Position i=2

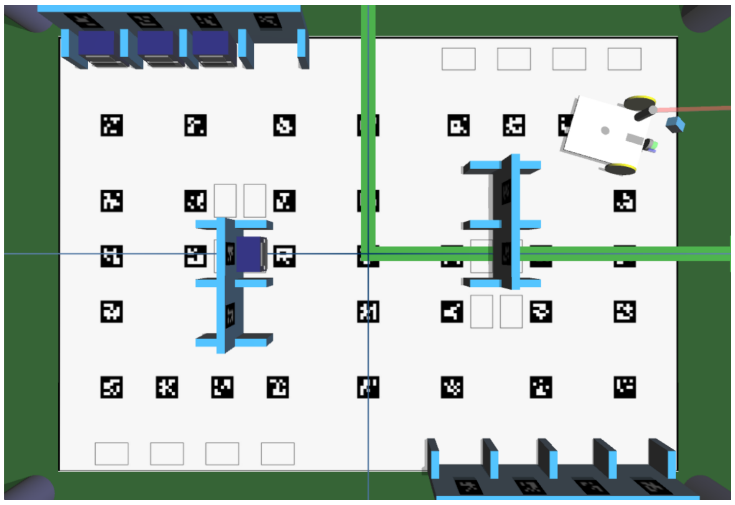


Figure 4.8: Position i=3

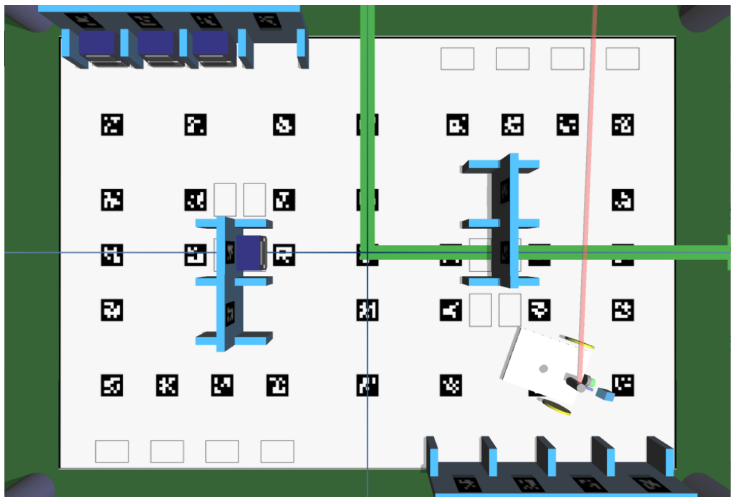


Figure 4.9: Position i=4

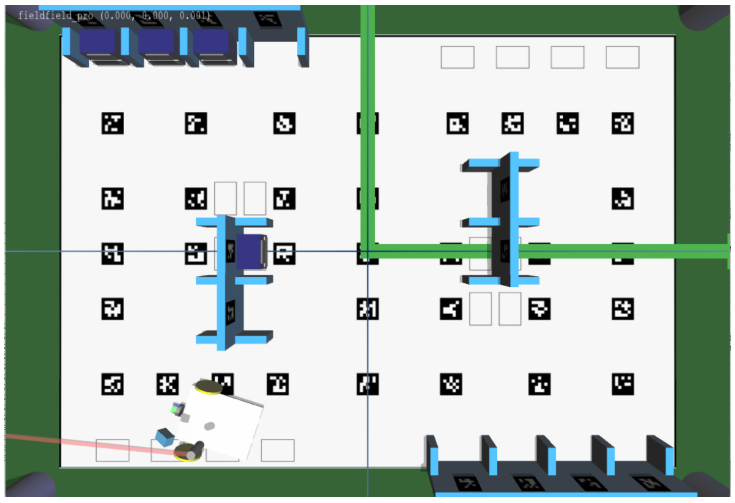


Figure 4.10: Position i=5

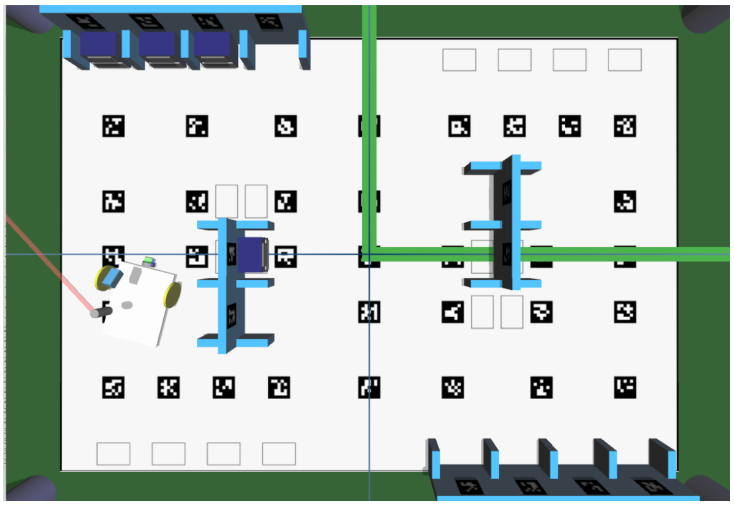


Figure 4.11: Position i=6

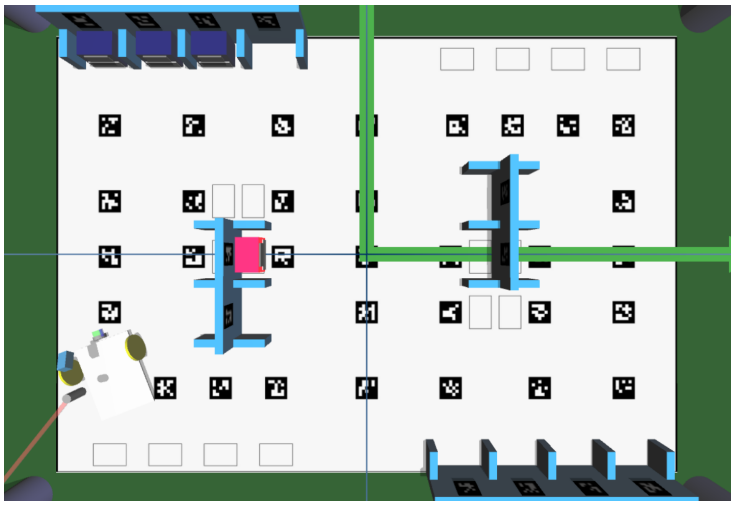


Figure 4.12: Position i=7

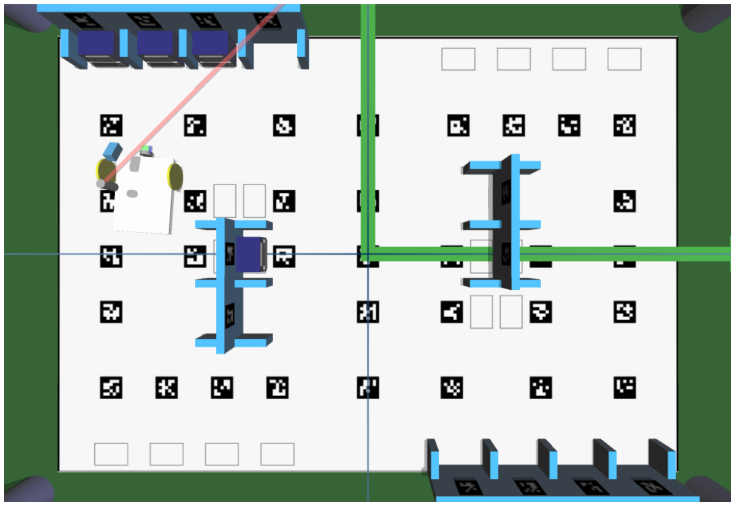


Figure 4.13: Position i=8

Chapter 5

Conclusions

Considering the objectives set on Chapter 1, the results of this work were plausible. Even though the trilateration technique was not properly used, but a modified bilateration, the errors found were acceptable. The results obtained from the Beacon Identification algorithm and the Robot Estimated Position algorithm provide valuable insights into the effectiveness and limitations of the developed project.

In the case of the Beacon Identification algorithm, it was observed that the algorithm successfully identified beacons and calculated their coordinates for various robot orientations. However, some limitations were identified, such as the algorithm's inability to function if the robot is moved outside the factory area, and an increase in error when the robot approaches one of the beacons. Additionally, the algorithm's applicability is limited to this specific beacon layout due to its mirrored approach.

Regarding the Robot Estimated Position algorithm, the results showed that the algorithm accurately estimated the robot's global position in terms of x and y coordinates, with errors within a reasonable range given the dimensions of the model. However, there was an issue in calculating the θ value, as the error remained consistently at zero for all tested points. This suggests that further refinement is needed in determining the orientation of the robot.

The study also highlighted the impact of using bilateration instead of trilateration for sequential beacons and the need to consider the angle of the LiDAR in relation to the

beacon's center for more accurate results.

In retrospect, there are several aspects that could have been approached differently, such as improving the accuracy of θ calculation and addressing the limitations of the algorithm when the robot is outside the factory area. Nevertheless, the project succeeded in going beyond the initial objectives by developing functional algorithms for beacon identification and robot position estimation.

Overall, the results obtained in this study contribute to the understanding of the project's strengths and weaknesses and provide a foundation for future improvements and enhancements in the field of robotics and localization systems.

5.1 Future work

In future work, it would be prudent to explore alternative avenues for the simulation framework utilized in this project. Although SimTwo was employed for the entirety of the code development, it became apparent that the simulation imposed a substantial computational load. Therefore, one promising direction for improvement could involve transitioning to a more computationally efficient programming language, such as Python or C++. Given that SimTwo offers UDP communication capabilities, this migration could facilitate a smoother and more streamlined simulation process.

Furthermore, enhancing the beacon identification algorithm represents another promising avenue for future research. While the current approach relies on a quadrant-based methodology, which may not be suitable for all layouts, a more sophisticated and adaptable beacon identification system could be developed. This would not only make the system more versatile but also align it with a broader range of real-world scenarios, providing enhanced accuracy and applicability in diverse contexts.

About the calculation of the robot's orientation angle, θ, in future studies should be a key focal point. As observed in the current work, the error associated with θ remained consistently at zero for all tested points, indicating a potential issue in the calculation process. Investigating and refining the method for determining the robot's orientation

angle is paramount to achieving more accurate and reliable results in future applications.

By resolving these limitations and improving the accuracy of θ calculation, the project's overall performance can be significantly enhanced, making it more robust and adaptable to a wider range of scenarios and layouts. This step would be critical for achieving comprehensive and precise localization of the robot in real-world environments, thereby broadening the practical applications of the developed algorithms.

Bibliography

[1] YDLIDAR X4_ydlidar|Focus on lidar sensor solutions.

[2] J. Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880, Dec 1996.

[3] Paulo Costa. SimTwo - A realistic simulator for robotics, June 2023. original-date: 2018-09-04T18:11:04Z.

[4] John J. Craig. *Introduction to Robotics*. Pearson Educación, 2006. Google-Books-ID: hRzOp_qdxG8C.

[5] Amanda Davis. Straight Out of Sci-Fi, Shakey Was the First Mobile Robot Built With AI, 2017.

[6] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, July 2010. Google-Books-ID: oIVptQEACAAJ.

[7] Paulo Costa Emily Maggioli, Motaz Ayiad. Simtwo tutorial - Tutorial 1 - Getting started, 2018.

[8] L. Jetto, S. Longhi, and G. Venturini. Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots. *IEEE Transactions on Robotics and Automation*, 15(2):219–229, April 1999.

[9] Eric Krotkov, Reid Simmons, Fabio Cozman, and Sven Koenig. Safeguarded teleoperation for lunar rovers: From human factors to field trials. In *In Proc. IEEE Planetary Rover Technology and Systems Workshop*, 1996.

[10] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[11] NASA/JPL-Caltech. Mars Exploration Rover-a semi-autonomous exploration robot.

[12] Rudy Negenborn. *UTRECHT UNIVERSITY Robot Localization and Kalman Filters.* 2003.

[13] S. I. Roumeliotis and G. A. Bekey. Bayesian estimation and Kalman filtering: a unified framework for mobile robot localization. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 3, pages 2985–2992 vol.3, April 2000.

[14] Vitor Luiz Martinez Sanches. Um sistema de localização robótica para ambientes internos baseado em redes neurais., 2009.

[15] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots.* MIT Press, Cambridge, Mass, February 2011.

[16] Chanop Silpa-Anan, Samer Abdallah, and David Wettergreen. Development of autonomous underwater vehicle towards visual servo control, 2000.

[17] F. Thomas and L. Ros. Revisiting trilateration for robot localization. *IEEE Transactions on Robotics*, 21(1):93–101, February 2005.

[18] Chih-Ming Jack Wang. Location estimation and uncertainty analysis for mobile robots. pages 1231 – 1235, 05 1988.

Appendix A

Proposta Original do Projeto

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia Industrial - Eletrotécnica

Desenvolvimento e Avaliação de um Sistema de Localização por Trilateração Utilizando Quatro Beacons na Plataforma Simtwo

Orientador: José Luís Sousa de Magalhães Lima

Coorientador: Marcos Banheti Rabello Vallim

1 Objetivo

Implementar um sistema de localização por trilateração através da plataforma Simtwo utilizando quatro beacons. Fazer análise de erros, compararando aos dados reais fornecidos pela plataforma.

2 Detalhes

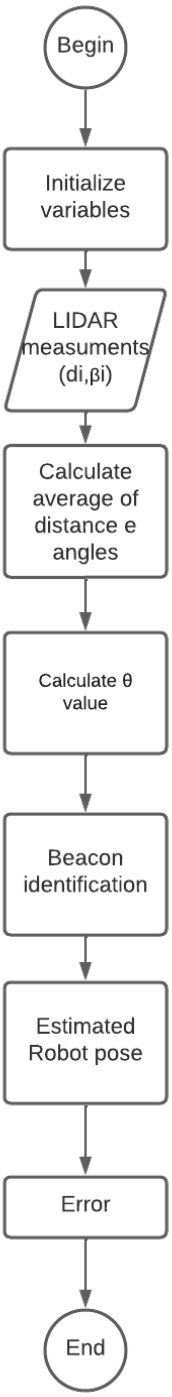
A localização precisa é uma necessidade crítica em várias aplicações, como rastreamento de ativos, navegação indoor, e automação industrial. A trilateração é uma técnica comum usada para determinar a localização de um objeto com base nas medições de distância de múltiplos beacons. Este projeto de tese visa desenvolver e avaliar um sistema de localização por trilateração utilizando a plataforma Simtwo, empregando quatro beacons para melhorar a precisão da localização. O objetivo é analisar a eficácia do sistema, comparando os resultados com dados reais fornecidos pela plataforma.

3 Metodologia de trabalho

- Configuração da Plataforma Simtwo: Configurar um ambiente simulado na plataforma Simtwo, com quatro beacons posicionados estrategicamente e um objeto a ser localizado.
- Desenvolvimento do Sistema de Trilateração: Implementar o sistema de localização por trilateração, que coleta medições de distância dos beacons para calcular a posição estimada do objeto.
- Coleta de Dados: Realizar uma série de experimentos nos quais o objeto se move dentro do ambiente simulado, coletando dados de medições de distância.
- Análise de Erros: Analisar os erros de localização, considerando fatores como ruído nas medições, interferência e propagação do sinal, e outros possíveis desafios.

Appendix B

Fluxogram


Begin
Initialize variables
LIDAR measuments (di,βi)
Calculate average of distance e angles
Calculate θ value
Beacon identification
Estimated Robot pose
Error
End

Appendix C

Code

```
function PrepareLaserSweepPackage(LaserScanRays: Matrix) :
   string;


var
    i, j, l, m, alfa, bf, k11, k12, k13, nrows, gama, n, fi
       : integer;
    xcal1, ycal1, xcal2, ycal2, xcal3, ycal3, xcal4, ycal4,
        xdiag, ydiag, xavg, yavg, a1, b1, c1, a2, b2, c2,
       a3, b3, c3, a4, b4, c4, m31, n31, m24, n24, m34, n34
       , m41, n41, rb: double;
    previous_phi : array of double;
    actual_theta: double;
    vx, vy, xanterior, yanterior, omega, yatual, xatual:
       double;
    previous_theta: double;
    actual_phi : array of double;
    temp : string;
```

```
13        averageangle: array of double;
14        distance : array of double;
15        angle : array of double;
16        numberofpoints: array of double;
17        sumdistance: array of double;
18        sumangle: array of double;
19        average: array of double;
20        bf1,bf2,bf3,bf4: array of double;
21        xb: array of double;
22        yb: array of double;
23        bi1: array of double;
24        bi2: array of double;
25        bi3: array of double;
26        bi4: array of double;
27    begin
28      temp := '';
29      nrows := MNumRows(LaserScanRays); // numero de linhas
30
31    SetLength(distance, nrows - 1);
32    SetLength(angle, nrows - 1);
33    SetLength(sumdistance, 5);
34    SetLength(sumangle, 5);
35    SetLength(numberofpoints, 5);
36    SetLength(average, 5);
37    SetLength(averageangle, 5);
38    SetLength(previous_phi,5);
39    SetLength(actual_phi,5);
40    SetLength(xb, 4);
41    SetLength(yb, 4);
```

```
42  SetLength(bi1, 3);
43  SetLength(bi2, 3);
44  SetLength(bi3, 3);
45  SetLength(bi4, 3);
46
47    vx:=0;
48    vy:=0;
49    omega:=0;
50    fi:=1;
51    actual_theta:= 0;
52    a1:=0;
53    a2:=0;
54    a3:=0;
55    a4:=0;
56    b1:=0;
57    b2:=0;
58    b3:=0;
59    b4:=0;
60    c1:=0;
61    c2:=0;
62    c3:=0;
63    c4:=0;
64    m31:=0;
65    n31:=0;
66    m24:=0;
67    n24:=0;
68    xcal1:=0;
69    ycal1:=0;
70    xcal2:=0;
```

```
71      ycal2:=0;
72      xcal3:=0;
73      ycal3:=0;
74      m34:=0;
75      n34:=0;
76      m41:=0;
77      n41:=0;
78      rb:=0.05;
79      gama:=1;
80      n:=0;
81      l:=0;
82
83
84      j:= 1; // coluna da matriz
85      k11:=1;
86      k12:=1;
87      k13:=1;
88
89    //  beacon:=[4, 2, 1,3];
90
91      bf1:=[0.9,0.65,0.150];  // 4
92      bf2:=[-0.9,0.650,0.150]; // 2
93      bf3:=[-0.900,-0.65,0.150]; // 1
94      bf4:=[0.9,-0.650,0.150];    // 3
95
96
97      bf:=0; // beacon found
98      alfa:=0;
99
```

```
100      begin
101      bf:=0; // nenhum beacon encontrado
102           for i := 0 to nrows - 1 do
103             begin
104               temp := temp + FloatToStr( Mgetv(LaserScanRays,
                      i, 0) ) + ','; // armazena a leitura do
                     LIDAR em um array temp
105              end;
106
107    // ATRIBUTES VALUES TO DISTANCE AND ANGLE ARRAY OK
108           for i:= 1 to (length(distance)-1) do
109             begin
110             distance[i-1]:= Mgetv(LaserScanRays, i, 0); //
                   atribui o valor da dist ncia no array
111             angle[i-1]:=i; // atribui os valores dos angulos
                   no array
112             j:=j+1;
113             end;
114
115    // SUM DISTANCES FROM EACH UNKNOWN BEACON AND NUMBER OF
          POINTS MEASURED PER BEACON
116    for i:=1 to (length(distance)) do
117      begin
118
119           if ((((distance[i-1])<0) and ((distance[i])>0)) or
                ((distance[0])> 0) and (i =1)) then
120           begin
121
122               bf:=bf+1;
```

```
123             if (i = 1) and (distance[0]> 0) then begin
124             sumdistance[bf-1]:=sumdistance[bf-1]+distance
                   [0];   //@Stefany: distance[i], not distance
                   [i-1] (Also need to include the first
                   iteration execption
125             sumangle[bf-1]:=sumangle[bf-1]+angle[0];
126             numberofpoints[bf-1]:=numberofpoints[bf-1]+1;
127             end;
128
129             sumangle[bf-1]:=sumangle[bf-1]+angle[i];
130             sumdistance[bf-1]:=sumdistance[bf-1]+distance[i
                   ];   //@Stefany: distance[i], not distance[i
                   -1] (Also need to include the first
                   iteration execption
131             numberofpoints[bf-1]:=numberofpoints[bf-1]+1;
132
133         end
134
135         else if  (((distance[i-1]) > 0) and ((distance[i])
               > 0)) then      // @Stefany: Use elseif here to
               optimize
136         begin
137
138             sumdistance[bf-1]:=sumdistance[bf-1]+distance[i
                   ];
139             sumangle[bf-1]:=sumangle[bf-1]+angle[i];
140             numberofpoints[bf-1]:=numberofpoints[bf-1]+1;
141
142         end
```

```
143
144         end;
145
146
147         if (bf=5) then
148         begin
149
150            sumdistance[0]:=sumdistance[0]+ sumdistance[4];
151            sumangle[0]:=sumangle[0]+ sumangle[4] - 360*
                  numberofpoints[4];
152            numberofpoints[0]:=numberofpoints[0]+ numberofpoints
                  [4];
153            bf := 4;
154
155         end;
156
157             // SHOWS THE AVERAGE OF THE ANGLE OF EACH BEACON
                  AND THE DISTANCE
158              begin
159   for m := 0 to 3 do
160   begin
161     average[m] := ((sumdistance[m]) / numberofpoints[m]) + rb
         ;
162     averageangle[m] := (sumangle[m]) / numberofpoints[m];
163   //   if ((averageangle[m] >= 0) and (averageangle[m] <= 90))
        then
164        actual_phi[m] := averageangle[m]
165   //   else if ((averageangle[m] > 90) and (averageangle[m] <=
        180)) then
```

```
166   //    actual_phi[m] := 180 - averageangle[m]
167   //  else if ((averageangle[m] > 180) and (averageangle[m]
        <= 270)) then
168   //    actual_phi[m] := averageangle[m] - 180
169   //  else if ((averageangle[m] > 270) and (averageangle[m]
        <= 360)) then
170   //    actual_phi[m] := 360 - averageangle[m];
171
172     if (gama = 1) then
173     begin
174       previous_phi[m] := actual_phi[m];
175       previous_theta := (180 * GetRobotTheta(irobot)) / pi;
176       gama := 0;
177     end;
178   end;
179
180   for n := 0 to 3 do
181    begin
182      if actual_phi[n] <> previous_phi[n] then
183      begin
184       actual_theta := previous_theta + previous_phi[0] -
           actual_phi[0];   //corrigir
185       previous_phi[n] := actual_phi[n];
186       previous_theta := actual_theta;
187      end;
188   end;
189
190            SetRCValue(33,1,FloatToStr(actual_phi[0]));
191            SetRCValue(33,2,FloatToStr(actual_phi[1]));
```

```
192             SetRCValue(33,3,FloatToStr(actual_phi[2]));
193             SetRCValue(33,4,FloatToStr(actual_phi[3]));
194
195
196   // BEACON IDENTIFICATION
197  if fi=1 then begin
198   xatual:=0;
199   yatual:=0;
200   fi:=0;
201 end;
202      for j:=0 to 3 do
203
204         begin
205             xb[j]:=average[j]*(cos(rad(actual_theta))*cos(rad
                   (averageangle[j]))-sin(rad(actual_theta))*sin(
                   rad(averageangle[j])))+xatual;
206             yb[j]:=average[j]*(sin(rad(actual_theta))*cos(rad
                   (averageangle[j]))+cos(rad(actual_theta))*sin(
                   rad(averageangle[j])))+yatual;
207         end;
208
209             if ((xb[0]>0) and (yb[0]>0)) then   // starts at
                   B4
210
211             begin
212
213             for l:=0 to 1 do
214
215               begin
```

```
216
217             bi1[l]:=bf1[l]; //4
218             bi2[l]:=bf2[l]; // 2
219             bi3[l]:=bf3[l]; // 1
220             bi4[l]:=bf4[l]; // 3
221
222             end;
223         a1:=2*bi1[0];  // a1 se refers to beacon 4
224         b1:=2*bi1[1];
225         c1:=((average[0]*average[0])-(bi1[1]*bi1[1])-(bi1
               [0]*bi1[0]));
226
227         a2:=2*bi2[0]; // a2 se refers to beacon 2
228         b2:=2*bi2[1];
229         c2:=(average[1]*average[1]-bi2[1]*bi2[1]-bi2[0]*bi2
               [0]);
230
231         a3:=2*bi3[0];  // a3 se refers to beacon 1
232         b3:=2*bi3[1];
233         c3:=(average[2]*average[2]-bi3[1]*bi3[1]-bi3[0]*bi3
               [0]);
234
235         a4:=2*bi4[0];  // a4 se refers to beacon 3
236         b4:=2*bi4[1];
237         c4:=(average[3]*average[3]-bi4[1]*bi4[1]-bi4[0]*bi4
               [0]);
238
239           end
240
```

```
241             else if ((xb[0]>0) and (yb[0]<0)) then    //
                   starts at beacon B3
242
243             begin
244
245               bi1:=[0.9,-0.65,0.150]; //3
246               bi2:=[0.9,0.65,0.150]; // 4
247               bi3:=[-0.9,0.65,0.150]; // 2
248               bi4:=[-0.9,-0.65,0.150]; // 1
249
250
251
252         a1:=2*bi2[0];  // a1 se refers to beacon 4
253         b1:=2*bi2[1];
254         c1:=((average[1]*average[1])-(bi2[1]*bi2[1])-(bi2[0]*
               bi2[0]));
255
256         a2:=2*bi3[0]; // a2 se refers to beacon 2
257         b2:=2*bi3[1];
258         c2:=(average[2]*average[2]-bi3[1]*bi3[1]-bi3[0]*bi3[0])
               ;
259
260         a3:=2*bi4[0];  // a3 se refers to beacon 1
261         b3:=2*bi4[1];
262         c3:=(average[3]*average[3]-bi4[1]*bi4[1]-bi4[0]*bi4[0])
               ;
263
264         a4:=2*bi1[0];  // a4 se refers to beacon 3
265         b4:=2*bi1[1];
```

```
266         c4:=(average[0]*average[0]-bi1[1]*bi1[1]-bi1[0]*bi1[0])
               ;
267         end
268               else if ((xb[0]<0) and (yb[0]<0)) then begin
269               for l:=0 to 1 do
270
271                 begin
272
273                 bi1[l]:=bf3[l]; // 1
274                 bi2[l]:=bf4[l]; // 3
275                 bi3[l]:=bf1[l]; // 4
276                 bi4[l]:=bf2[l]; // 2
277
278
279
280                 end;
281       a1:=2*bi3[0];  // a1 se refers to beacon 4
282         b1:=2*bi3[1];
283         c1:=((average[2]*average[2])-(bi3[1]*bi3[1])-(bi3[0]*
               bi3[0]));
284
285         a2:=2*bi4[0]; // a2 se refers to beacon 2
286         b2:=2*bi4[1];
287         c2:=(average[3]*average[3]-bi4[1]*bi4[1]-bi4[0]*bi4[0])
               ;
288
289         a3:=2*bi1[0];  // a3 se refers to beacon 1
290         b3:=2*bi1[1];
```

```
291         c3:=(average[0]*average[0]-bi1[1]*bi1[1]-bi1[0]*bi1[0])
               ;
292
293         a4:=2*bi2[0];  // a4 se refers to beacon 3
294         b4:=2*bi2[1];
295         c4:=(average[1]*average[1]-bi2[1]*bi2[1]-bi2[0]*bi2[0])
               ;
296
297         end
298               else if ((xb[0]<0) and (yb[0]>0)) then  begin //
                     starts at B2
299
300               for l:=0 to 1 do
301
302                 begin
303
304                 bi1[l]:=bf2[l]; // 2
305                 bi2[l]:=bf3[l]; // 1
306                 bi3[l]:=bf4[l]; // 3
307                 bi4[l]:=bf1[l]; // 4
308
309                 end;
310
311         a1:=2*bi4[0];  // a1 se refers to beacon 4
312         b1:=2*bi4[1];
313         c1:=((average[3]*average[3])-(bi4[1]*bi4[1])-(bi4[0]*
               bi4[0]));
314
315         a2:=2*bi1[0]; // a2 se refers to beacon 2
```

```
316         b2:=2*bi1[1];
317         c2:=(average[0]*average[0]-bi1[1]*bi1[1]-bi1[0]*bi1[0])
               ;
318
319         a3:=2*bi2[0];  // a3 se refers to beacon 1
320         b3:=2*bi2[1];
321         c3:=(average[1]*average[1]-bi2[1]*bi2[1]-bi2[0]*bi2[0])
               ;
322
323         a4:=2*bi3[0];  // a4 se refers to beacon 3
324         b4:=2*bi3[1];
325         c4:=(average[2]*average[2]-bi3[1]*bi3[1]-bi3[0]*bi3[0])
               ;
326
327         end;
328
329
330             SetRCValue(29,1,'BF1');
331             SetRCValue(29,2,'BF2');
332             SetRCValue(29,3,'BF3');
333             SetRCValue(29,4,'BF4');
334             SetRCValue(30,1,FloatToStr(xb[0]));
335             SetRCValue(31,1,FloatToStr(yb[0]));
336
337             SetRCValue(30,2,FloatToStr(xb[1]));
338             SetRCValue(31,2,FloatToStr(yb[1]));
339
340             SetRCValue(30,3,FloatToStr(xb[2]));
341             SetRCValue(31,3,FloatToStr(yb[2]));
```

```
342
343             SetRCValue(30,4,FloatToStr(xb[3]));
344             SetRCValue(31,4,FloatToStr(yb[3]));
345
346
347         xcal1:=(c2-c1)/(a1-a2);    // B4 e B2
348         ycal1:=999;
349
350         xcal2:=999;      // B2 e B3
351         ycal2:=(c3-c2)/(b2-b3);
352
353         xcal3:=(c4-c3)/(a3-a4);  // B3 e B4
354         ycal3:=999;
355
356         xcal4:=999;            // B4 e B1
357         ycal4:=(c1-c4)/(b4-b1);
358
359         xdiag:= ((c4-c2)*(b1-b3)-(c3-c1)*(b2-b4))/((a3-a1)*(b2-
               b4)-(a4-a2)*(b1-b3));
360         ydiag:= ((c4-c2)*(a1-a2)-(c3-c1)*(a2-a4))/((b3-b1)*(a2-
               a4)-(b4-b2)*(a1-a3));
361
362       xavg:=(xcal1+xcal2+xcal3+xcal4+xdiag-999-999)/3;
363       yavg:=(ycal1+ycal2+ycal3+ycal4+ydiag-999-999)/3;
364
365       xatual:=xavg;
366       yatual:=yavg;
367
368       SetRCValue(23,2,'1');
```

```
369         SetRCValue(23,3,'2');
370         SetRCValue(23,4,'3');
371         SetRCValue(23,5,'4');
372         SetRCValue(23,6,'Diagonal');
373         SetRCValue(23,7,'Average');
374         SetRCValue(23,8,'Error');
375         SetRCValue(23,9,'Real Value');
376
377         SetRCValue(24,1,'x');
378         SetRCValue(25,1,'y');
379         SetRCValue(24,2,FloatToStr(xcal1));
380         SetRCValue(24,3,FloatToStr(xcal2));
381         SetRCValue(24,4,FloatToStr(xcal3));
382         SetRCValue(24,5,FloatToStr(xcal4));
383         SetRCValue(24,6,FloatToStr(xdiag));
384         SetRCValue(24,7,FloatToStr(xavg));
385         SetRCValue(24,8,FloatToStr(GetRobotX(irobot)-xavg));
386         SetRCValue(24,9,FloatToStr(GetRobotX(irobot)));
387
388         SetRCValue(25,2,FloatToStr(ycal1));
389         SetRCValue(25,3,FloatToStr(ycal2));
390         SetRCValue(25,4,FloatToStr(ycal3));
391         SetRCValue(25,5,FloatToStr(ycal4));
392         SetRCValue(25,6,FloatToStr(ydiag));
393         SetRCValue(25,7,FloatToStr(yavg));
394         SetRCValue(25,8,FloatToStr(GetRobotY(irobot)-yavg));
395         SetRCValue(25,9,FloatToStr(GetRobotY(irobot)));
396
397         SetRCValue(26,1,'Theta');
```

```
398       SetRCValue(26,2,FloatToStr(rad(actual_theta)));
399       SetRCValue(26,3,FloatToStr(GetRobotPos2D(irobot).angle))
              ;
400
401       end;
402
403     Delete(temp, Length(temp), 1);
404     Result:=temp;
405
406
407   end;
408
409
410 end;
```