



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Campus de Terrassa

---

# FINAL DEGREE PROJECT: TRAJECTORY CONTROL DESIGN OF A MOBILE ROBOT WITH COMPUTER VISION

---

*Degree in Electrical Engineering*

*Escola Superior d'Enginyeries Industrial, Aeroespacial i  
Audiovisual de Terrassa (ESEIAAT)*



**Tutors:** Ramón Pérez Magrané, Bernardo Morcego Seix

**Author:** Xavier Gutiérrez Ramón

**Defence:** October 2016

## ***Agraïments***

En primer lloc, vull agrair al departament de Enginyeria de sistemes, automàtica i informàtica industrial de la ESEIAT y en especial al meu tutor Ramon Pérez, al professor Bernardo Seix i al professor Albert Masip. La seva ajuda en aquest projecte ha estat inestimable i els hi vull agrair la oportunitat d'haver-me introduït al món de la robòtica.

També vull agrair el suport a tots els meus companys que han estat amb mi tots aquests anys. Cada any ha sigut millor que l'anterior gracies a ells. En especial a "Los Felinos".

Per últim a la meua família, la meua mare i els meus germans que sempre han estat allà per aixecar-me quan defallia i encoratjar-me a continuar fins al final.

## ***Abstract***

Before readings this memory, the reader has to know that it was made over the premise that the reader doesn't have any knowledge of tracking paths by robots. The personal objective is the use of this memory in the future in the practical classes for their study or expansion.

The field of mobile robot control has been the focus of active research in the past decades. On this present project, we address the concept of path tracking based on the position and orientation of the robot, basically the mobile robot has to be able to move along lines on the floor and crossings.

At the end of this present memory, I did a statistical study in order to compare two methods to work and determine which one is better.

## INDEX

1. INTRODUCTION .....	11
1.1 Objectives .....	11
1.2 State of the art.....	12
1.2.1 State of the art of the robots .....	12
1.2.2 State of the art of the path tracking .....	12
1.2.3 Synthesis of the state of the art.....	15
2. HARDWARE OF ROBOTINO .....	16
2.1 About Robotino.....	16
2.1.1 Chassis:.....	17
2.1.2 Battery System .....	17
2.1.3 Battery Charger. ....	18
2.1.4 Control unit. ....	18
2.2 Sensors of the mobile robots.....	24
2.2.1 Classification of the sensors:.....	24
2.3 About the wheels of the mobile robots.....	32
2.3.1 Wheels: .....	32
2.3.2 Tracks: .....	33
2.3.3 Legs: .....	33
2.4 Type of locomotion:.....	34
2.4.1 Differential .....	34
2.4.2 Synchronous.....	34
2.4.3 Tricycle .....	35
2.4.4 Ackerman .....	36
2.4.5. Omnidirectional (Wheels with rollers – Swedish Wheels).....	36
2.4.6 Locomotion by sliding tapes.....	37
2.4.7 Legs.....	38
2.5 Sensors that use Robotino .....	39
2.5.1 Collision sensor or the sensor of the bumper. ....	39
2.5.2 Infrared sensors. ....	39
2.5.3 Camera (Webcam) .....	39
2.6 Wheels that use Robotino .....	48

3. IDENTIFICATION AND CONTROL OF THE PROPULSION WHEELS .....	49
3.1 Introduction .....	49
3.2 Design the model for the wheels .....	49
3.3 Procedure for the calculation of the model and the controller of a wheel.....	49
3.4 First-order model .....	51
3.5 Parameters of the controller .....	52
3.6 Methodology and the plant of Simulink to validate the controller .....	57
3.7 Study of the Error.....	60
4. KINEMATIC MODEL .....	62
4.1 Introduction .....	62
4.2.1 Differential Drive .....	64
4.2.2 Synchronous drive.....	65
4.2.3 Tricycle .....	66
4.2.4 Omnidirectional wheels .....	68
4.3 Simulation of the kinematic model on world reference .....	70
4.3.1 Breakdown of each part of the general scheme: .....	71
4.4 Practical Cases.....	74
4.4.1 Case 1 .....	74
4.4.2 Case 2 .....	76
4.4.3 Case 3 .....	78
5. SIMULATION OF THE TRAJECTORY CONTROL .....	81
5.1 Introduction .....	81
5.2 Theory of the trajectory algorithm .....	81
5.2.1 Development.....	82
5.3 Implementation the trajectory control algorithmf .....	83
5.3.1 Simulation of the trajectory control.....	85
5.4 Simulation for any path with trajectory control .....	86
5.4.1 Rotation Model .....	86
5.4.2 Kinematic Model .....	88
5.4.3 Wheels Model .....	89
5.4.4 Trajectory Control Algorithm .....	90
5.4.5 Change axes of coordinate .....	92
5.4.6 Implementation of the trajectory of the Robot in Simulink.....	97
5.5. Scheme of simulations .....	105

5.5.1 Method 1:.....	105
5.5.2 Method 2:.....	106
5.76 Simulation of two methods .....	107
5.6.1 Workspace for both methods: .....	107
5.6.2 Results of the simulation method 1: .....	108
5.6.3 Results of the simulation method 2: .....	110
5.6.4 Conclusion of the simulation.....	112
6. IMPLEMENTATION OF TRAJECTORY CONTROL, VISION AND TRAJECTORY PLANNER.....	113
6.1 Implementation the interface for the communication between the robot and the trajectories planner.....	113
6.2 Programming code for implementing the trajectory control, vision by camera, controller PI of the wheels and trajectory planner.....	115
6.2.1 Input Variables .....	115
6.2.2 Matlab instructions for Robotino.....	115
6.3 Test methodology .....	115
6.3.1 Introduction .....	116
6.3.2 Tests performed.....	117
6.4 Laboratory experiments.....	119
6.4.1 Comparison between both methods .....	119
6.4.2 Speed of each wheel for both methods:.....	126
6.4.3 Final comparison .....	127
7. CONCLUSIONS OF THE PROJECT .....	128
8. BIBLIOGRAPHY.....	131
9. ANNEX .....	133
9.1 Programs designed .....	133
9.1.1 Programing code of XavmesuraDistanciaARuta.m .....	133
9.1.2 Programming code for the simulation of the method 1 .....	137
9.1.3 Programming code for the simulation of the method 2 .....	138
9.1.4 Programming code for both methods on the real robot .....	141

## INDEX OF ILLUSTRATIONS

Illustration 1. Control scheme for the mobile robot from [1].....	12
Illustration 2. General scheme model reference adaptive control.....	13
Illustration 3. Scheme used for the fuzzy controller.....	13
Illustration 4. Chassis of the Robotino .....	17
Illustration 5. Localization of the battery of the Robotino .....	17
Illustration 6. Localization of the control unit parts of Robotino .....	18
Illustration 7. Scheme of the communication of the board EA09 .....	19
Illustration 8. Scheme of the communication using Wireless .....	22
Illustration 9. Interfaces to connect robotino.....	22
Illustration 10. Touch control command of Robotino .....	23
Illustration 11. Vision range in degrees .....	25
Illustration 12. Representation of the reflections.....	25
Illustration 13. Representation of the Cross-Talk .....	25
Illustration 14. Representation of the triangulation.....	26
Illustration 15. Representation of the encoder disc .....	31
Illustration 16. Disposition of the differential wheels .....	34
Illustration 17. Disposition of the Synchronous wheels .....	34
Illustration 18. Disposition of the tricycle wheels.....	35
Illustration 19. Disposition of the Ackerman wheels.....	36
Illustration 20. Disposition of the omnidirectional wheels.....	36
Illustration 21. Disposition of the locomotion by sliding tapes .....	37
Illustration 22. Disposition of the locomotion by legs .....	38
Illustration 23. Localization of the webcam.....	39
Illustration 24. Work area for calibration .....	42
Illustration 25. Interactive image once marked the rectangular region.....	44
Illustration 26. Localization of the different parts of the omnidirectional wheels.....	48
Illustration 27. Experimental results (realized by professors) .....	50
Illustration 28. Experimental results (realized by professors) .....	50
Illustration 29. Data variations between samples time .....	51
Illustration 30. Model representation for the wheel 0 .....	52
Illustration 31. Simulation of the system with the controlled signal and the PI controller that we calculated.....	54
Illustration 32. Graph of the controlled output and the input speed for the wheel 0 .....	54

Illustration 33. Plant used in Simulink to simulate the response .....	55
Illustration 34. Graph of the controlled output and the input speed .....	56
Illustration 35. Simulink scheme to find the error .....	57
Illustration 36. Above figure: Compare actual speed with the theoretical model Below figure: The error of the comparison .....	58
Illustration 37. Plant used in Simulink to simulate the response with the error .....	58
Illustration 38. Simulink plant response with the error .....	59
Illustration 39. Error for the permanent region .....	60
Illustration 40. Representation of the Fourier transform .....	61
Illustration 41. Theoretical representation of the axis .....	62
Illustration 42. Scheme representation of the axis .....	63
Illustration 43. Theoretical representation of the axis for differential drive .....	64
Illustration 44. Theoretical representation of the axis for steered wheel .....	65
Illustration 45. Theoretical representation of the axis tricycle .....	66
Illustration 46. Theoretical representation of the omnidirectional wheels .....	68
Illustration 47. General scheme of simulation .....	70
Illustration 48. Scheme of the input steps .....	71
Illustration 49. Scheme of the model reference .....	71
Illustration 50. Scheme of the matrix kinematic model in the robot reference .....	72
Illustration 51. Outputs of the simulation: .....	72
Illustration 52. General scheme structured in subsystems .....	73
Illustration 53. Response of the simulation for the case 1 (units are in meters) .....	75
Illustration 54. Response of the simulation for the case 2 (units are in meters) .....	77
Illustration 55. Response of the simulation for the case 3.1 (units are in meters) .....	79
Illustration 56. Response of the simulation for the case 3.2 (units are in meters) .....	80
Illustration 57. Theoretical scheme of the robot position when is out of line .....	81
Illustration 58. Scheme of the robot position when is out of line .....	83
Illustration 59. Scheme of Simulink model to simulate the behaviour when the robot is out of line.... .....	84
Illustration 60. Robot behaviour simulation for redirection, when placed 0.4 m from point of origin of coordinates .....	85
Illustration 61. Rotation model in Simulink .....	87
Illustration 62. Kinematic model in Simulink .....	88
Illustration 63. Wheels model in Simulink .....	89
Illustration 64. Scheme of Simulink model to simulate the behaviour when the robot is out of line	91



Illustration 65. Rotation on the axis Z.....	92
Illustration 66. Sign criteria.....	93
Illustration 67. Implementation of change of coordinates in Simulink .....	96
Illustration 68. Block MATLAB Function.....	97
Illustration 69. Matlab function method 1 .....	98
Illustration 70. Kinematic Matrix .....	100
Illustration 71. Matlab function method 2 .....	101
Illustration 72. Scheme: Method 1 .....	105
Illustration 73. Scheme: Method 2 .....	106
Illustration 74. Method 1, Position .....	108
Illustration 75. Method 1, Speed in axes XY .....	108
Illustration 76. Method 1, Path Tracking .....	109
Illustration 77. Method 2, Position .....	110
Illustration 78. Method 2, Speed in axes XY .....	110
Illustration 79. Method 2, Path Tracking .....	111
Illustration 80. Representation of a route in a 3x3 grid.....	113
Illustration 81. Representation of the path of the experiment in a 3x3 grid.....	117
Illustration 82. Speed of the wheels during the experiment, Method 1 .....	123
Illustration 83. Speed of the wheels during the experiment, Method 2 .....	123
Illustration 84. Representation of the desired path of the experiment .....	124
Illustration 85. Path of the experiment .....	125

## INDEX OF TABLES

Table 1. Advantages and Disadvantages using wheels .....	32
Table 2. Advantages and Disadvantages using tracks.....	33
Table 3. Advantages and Disadvantages that using legs .....	33
Table 4. Advantages and Disadvantages using differential wheels .....	34
Table 5. Advantages and Disadvantages using Synchronous wheels .....	35
Table 6. Advantages and Disadvantages using Tricycle wheels.....	35
Table 7. Advantages and Disadvantages using Ackerman wheels.....	36
Table 8. Advantages and Disadvantages using omnidirectional wheels .....	37
Table 9. Advantages and Disadvantages using locomotion by sliding tapes .....	37
Table 10. Advantages and Disadvantages using locomotion by legs.....	38
Table 11. Parameters of the first-order model for each wheel .....	56
Table 12. Parameters of the PI controller after the conversion .....	56
Table 13. Parameters for the simulation (Input step for each wheel).....	57
Table 14. Parameters of the first-order model for each wheel .....	57
Table 15. Parameters of the first-order model for each wheel .....	89
Table 16. Parameters of the PI controller after the conversion .....	89
Table 17. Increments of theta.....	94
Table 18. Criteria of directions.....	97
Table 19. Possible directions of the robot .....	103
Table 20. Variables of the Workspace .....	107
Table 21. Example of the route.....	113
Table 22. Transformations of the route.....	114
Table 23. Path of the experiment .....	117
Table 24. Summary of all input values.....	118
Table 25. Working time for each method.....	119
Table 26. Results of the distance of the robot to the vertical line, Method 1.....	119
Table 27. Results of the distance of the robot to the vertical line, Method 2.....	119
Table 28. Results of the distance of the robot to the horizontal line, Method 1 .....	120
Table 29. Results of the distance of the robot to the horizontal line, Method 2 .....	120
Table 30. Results of the angle of the robot to the vertical line, Method 1 .....	121
Table 31. Results of the angle of the robot to the vertical line, Method 2 .....	121
Table 32. Results of the angle of the robot to the horizontal line, Method 1 .....	122
Table 33. Results of the angle of the robot to the horizontal line, Method 2 .....	122

Table 34. Results of the speed, Method 1 .....126

Table 35. Results of the speed, Method 2 .....126

## **1. INTRODUCTION**

### ***1.1 Objectives***

**Main objective:**

Design and implement a control algorithm for robot path tracking using computer vision as the position sensors.

**Partial objectives:**

1. Development of the motor model and identification of its parameters. Design and implementation of the wheel controller to operate the omnidirectional wheels.
2. Development of a kinematic and dynamic model to control the robot.
3. Design and implementation of the algorithm of trajectory tracking, in particular, straight lines and crosses.
4. Determination of the robot position using computer vision.
5. Control of the Robotino using the kinematic model, path tracking algorithm and the computer vision system.
6. Design and implementation of the interface between the robot and the trajectory planner.
7. Writing a memory to be used later in this university as an academic material.

## 1.2 State of the art

### 1.2.1 State of the art of the path tracking

In [1] three different controllers are presented for position control and path tracking: adaptive PID controller, adaptive control with reference model and a fuzzy controller, in order to compare them and to determine which one has better results on the control of the path tracking.

As we see on the illustration 3. Control scheme for the mobile robot. If we want to achieve control and path tracking, we must obtain the position control of the mobile robot, starting from the desired references and considering their values in global reference. In this way, it establishes a control loop that provides as outputs the angular speed for each wheel, in order to get the desired position.

The angular speeds are references for the second control loop (internal loop). Depending on the manipulation of the input voltage of the actuators (motors) that allows to change the speeds required for each wheel.

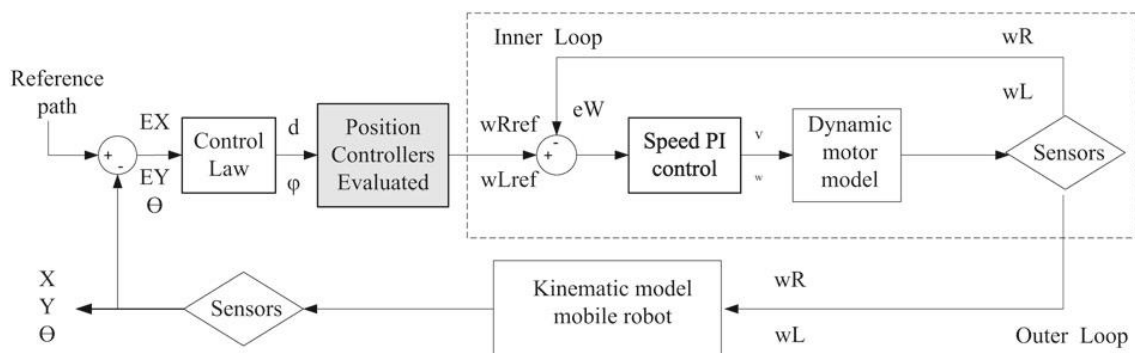


Illustration 1. Control scheme for the mobile robot from [1]

- Path tracking of mobile robot with Model Reference Adaptive Control (MRAC)

The idea behind of the Model Reference Adaptive Control (MRAC) (Illustration 2. General scheme model reference adaptive control) is to create a closed loop controller with parameters that can be updated to change the response of the system. The output of the system is compared to a desired response from a reference model. The control parameters are updated based on this error. The goal of changing the parameters is to find a plant response that matches the response of the reference model.

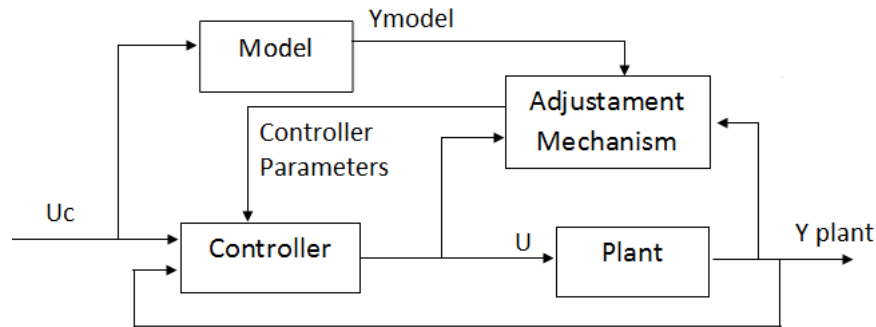


Illustration 2. General scheme model reference adaptive control

For the implementation of the MRAC, we have to transform the kinematic model into a kinematic model with the adaptive parameters of the controller.

- Path tracking of mobile robot with PID Control with Parameter Adaptation (PID ADA).

Like in the previous case, the kinematic model has to be transformed. Here, into a dynamic model which includes the PID with Parameter Adaptation, and the kinematic model for the mobile robot.

- Path tracking of mobile robot with Fuzzy Controller (Illustration 3).

For the implementation of a fuzzy controller for path tracking, they considered two inputs, corresponding to the distance error and the deviation angle between the mobile robot and the target point. This two outputs correspond to the linear speed and the angular speed in case of mobile robot.

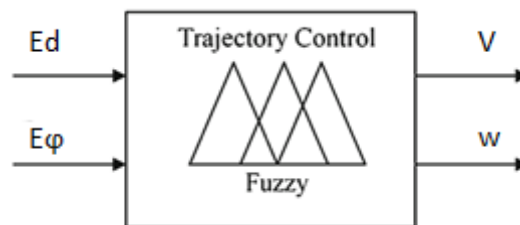


Illustration 3. Scheme used for the fuzzy controller

In case of mobile robot the fuzzy controller works in the following way: when the distance between the mobile robot and the target point is large, it is necessary to increase the linear speed. On the other hand, the speed must be reduced if the distance

is short. There is a high dependence between the linear speed and the distance to be covered, but the adjustment of deviations of the mobile robot is mainly related to variations in the angular speed. The modification of the speed allows to give at internal control loop the adequate references for path tracking.

The article [2] is a study of the trajectory tracking using a camera and controlled by a fuzzy controller. This paper presents an algorithm for autonomous path tracking of a mobile robot to track straight and curved paths traced in the environment. The algorithm uses a fuzzy logic based approach for path tracking so that human driving behavior can be emulated in the mobile robot. The method combines a fuzzy steering controller, which controls the steering angle of the mobile robot for path tracking, and fuzzy velocity controller which controls the forward linear velocity of the mobile robot for safe path tracking. A camera is used to capture images of the path ahead of the mobile robot and the vision system determines the lateral offset, heading error and the curvature of the path and performs experiments using a mobile robot platform.

The article [3] is focused on the application of model-based predictive control (MPC) to the trajectory tracking problem of non-holonomic wheeled mobile robots (WMR). The trajectory tracking problem is solved using two approaches: nonlinear MPC and linear MPC. Simulation results are provided in order to show the effectiveness of both schemes. Considerations regarding the computational effort of the MPC are developed with the purpose of analysing the real-time implementation viability of the proposed techniques.

The article [4] presents a solution for the path tracking problem of a mobile robot using a PID controller. The proposed method uses a linearized model of the mobile robot composed of an integrator and a delay. The PID controller has been tuned considering the nominal performance and the robustness as control specifications. Experimental results demonstrate the good performance and robustness of the proposed controller

The article [6] develops methodologies and techniques for control architecture design, path tracking laws and posture estimation of a vision-based wheeled mobile robot (WMR). To solve the problem of position/orientation tracking control of the WMR, two kinematical predictive control laws are developed to manipulate the vehicle. Simulation and experimental results are included to illustrate the feasibility and effectiveness of the proposed control laws.

The article [7] develops a trajectory controller of 3-wheels omnidirectional mobile robot using fuzzy azimuth estimator. A trajectory controller for an omnidirectional mobile robot, which each motor is controlled by an individual PID law to follow the speed command from inverse kinematics, needs a precise sensing data of its azimuth and exact estimation of reference azimuth value. In this paper, they are solved by using fuzzy logic inference which can be used straight forward to perform the control of the mobile robot by means of the fuzzy behaviour based scheme already existent in literature.

The article [8] develops a kinematic path tracking algorithm for a non-holonomic mobile robot using an improved iterative learning control (I-ILC) technique. The proposed algorithm produces a velocity command to the wheeled robot, in addition, the state disturbances and measurement noises are taken into consideration. The MATLAB software is used for simulation to verify the feasibility and validity of the proposed learning algorithm.

### *1.2.2 Synthesis of the state of the art*

We have seen several articles with different methods of trajectory control, and different types of robots including the robots with omnidirectional wheels. There are many ways trajectory tracking, but let's focus on the articles [4] [2] [7]. In these articles, it used a camera for tracking trajectory, a robust PID to control the robot and the application of the fuzzy controller in a robot with omnidirectional wheels.

According to the study [1], after testing the three models, the results indicate that the MRAC controller apparently is the best solution for the path tracking of the mobile robot. But, it also says the difference respect to the other controllers is not so significant as to discard them. MRAC is the best option because it is based on the results of the torque requirements that shows smooth curve which extends the life of the motors.

**That means it can be control lay for one of the three controllers and any one that we use doesn't represent a big difference between them.**

In this project we are going to focus on two concepts that we saw in two articles, the use of a camera to track path will be used and control the wheels using a PID.



## **2. HARDWARE OF ROBOTINO**

### **2.1 About Robotino**

Robotino is a robot of the German company Festo whose main line of development is the automation and control production processes.

Robotino is a product of the Didactic division from FESTO. It is an omnidirectional mobile robot usually used for the learning and improvement for control systems, it is also used for research and development at technical schools. The omnidirectional system allows to make movements forward, backward and sideways and turn on a particular point. All of this added to their analog/digital sensors, and a webcam for interaction through computer vision.

***All of this devices built into their control unit***, which features computer performance industrial PC-104, compatible with MOPS1cdVE, 300 MHz, Linux operating system and a flash memory disk (1 GB) with C ++ API to control Robotino. Their three motors and her wheels are separated by 120° each other which allows the movement in different locations in a coordinate systems of two dimensions.

Robotino can be controlled by several softwares. In this project we use Matlab Software to control the robot. One of the main advantages to use Matlab is use a wireless LAN which allows permanent and continuous connection with the robot computer, sensors and their webcam.

Physical Features of Robotino:

- Diameter: 370 mm
- Height, including chassis: 210mm
- Total weight: approx. 11 kg

Based on the different components that make up the Robotino, the next paragraph gives a brief description of each part:

### 2.1.1 Chassis:

The chassis consists of a stainless steel platform with laser welding. The chassis has two handles which are made specifically for times when we should raise or transport the Robotino because subjecting from the other components can cause significant damage to the robot (Illustration 4). Also, inside of the chassis there's rechargeable batteries, drive units and the webcam. Besides, the chassis provides an additional mounting options to add others sensors and / or motors. The highly sensitive system components, such as the controller, the module I / O and interfaces are located on the bridge of the robot.

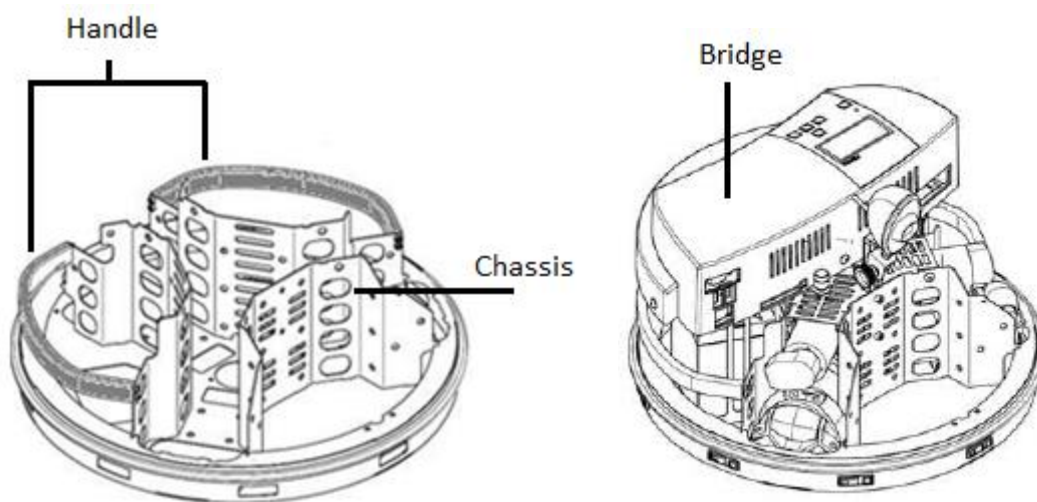


Illustration 4. Chassis of the Robotino

### 2.1.2 Battery System

Robotino works with two Effekta batteries connected in series, each of 12 V (Volts), 5 Ah (Ampere hour). (Illustration 5)

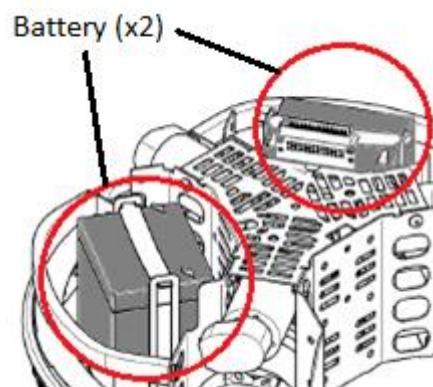


Illustration 5. Localization of the battery of the Robotino

### 2.1.3 Battery Charger.

For charging the battery uses an electric charger ANSMANN ALCT 24-2 with a conversion voltage 220 (V) to 24 (V)/2 (A) with a connector connecting adapter 2.4 [mm] of thickness. Robotino works while it is charging but this limits the mobility of the robot.

### 2.1.4 Control unit.

The control unit is which manages and distributes the instructions of the program to the different parts of the Robotino. Disposition in illustration 6. The information of the sensors and the webcam are sent it by the Unit Control.

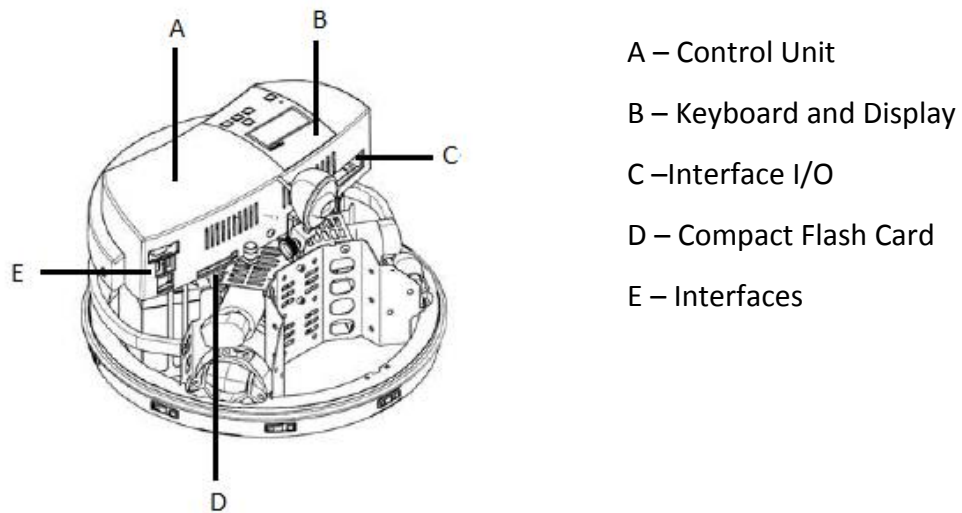


Illustration 6. Localization of the control unit parts of Robotino

This control unit consists of the following elements:

- PC/104 processor
- EA09
- Connectivity
- Compact Flash card
- Interfaces.
- Keyboard and Display
- Interface I / O

### PC/104 processor

The computer is PC/104 (MOPSlcdLX) with a processor AMDLX800 at 400 MHz. The computer has 1 GB of SDRAM memory and a Compact Flash 1 MB where the operating system is installed. In this case we can work on Linux and QNX:

- Linux with Kernel in real time (RTAI) and all the necessities libraries to run applications.
- A QNX Neutrino real time operating system (RTOS: Real-Time Operating Systems) developed by QNX Software Systems Ltd which consists of a micro-kernel surrounded by a set of optional modules (resource managers).

### EA09

The mother board controls the inputs and outputs of the board EA09 which provides access to motors, sensors, analog and digital inputs, digital outputs and relays of the Robotino®.

The illustration 7 is a representation of how the Robotino communicates with the computer:

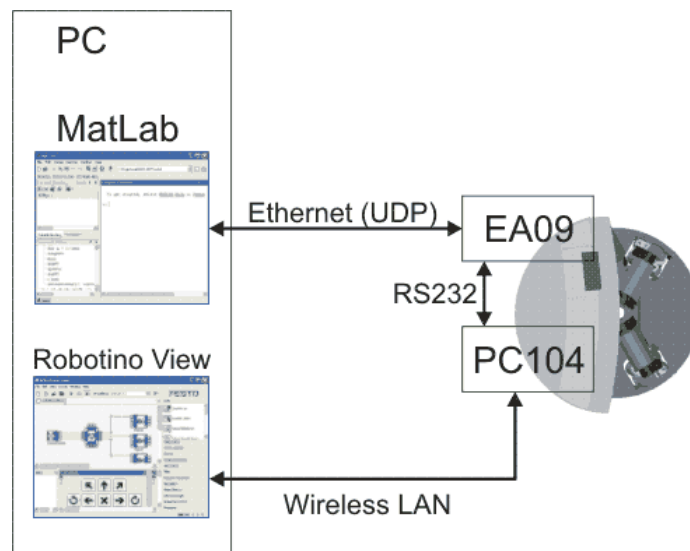


Illustration 7. Scheme of the communication of the board EA09

In the illustration 7 we saw how Matlab communicates with the Robotino. As we see the communication between Matlab and the Robotino has to be through the intercommunication with the EA09 board. This part of the unit control adapts the orders in Matlab to the kind of orders that understands the Robotino.

On the computer we use Matlab for data monitoring and hardware in the loop control. The board EA09 works as an interface between the computer and the drive motors, the nine distance sensors, I/O connector, battery management. In normal operation, the set point values are received through RS232 from the PC104.

Ethernet of the board EA09 can be connected to Robotino by using their access point. Robotino communicates with the board EA09 through Wireless LAN. But that depends of the quality of the Wireless connection. This depends if there is hardware in the loop or high speed monitoring, among other things.

The I/O board and PC / 104 have connected through one of two serial interfaces (RS-232) of the interfaces from PC/104. A feature of this board allows a loop speed control on each of the actuators with a frequency of 1 kHz. It also incorporates a timer and even a counter of position for each encoder.

The EA09 board firmware implements four PID controllers for the three motors. The PID controller equation (1) is:

$$u(t) = K_p \left( e(t) + \frac{1}{T_N} \int_0^t e(t') dt' \right) + K_d \dot{e}(t) \quad (1)$$

### Connectivity

The connectivity of robotino features two modes for data transmission:

- LAN Ethernet, Cable RJ-45.
- WLAN, WIFI 802.11 b/g.

### **Lan Ethernet**

Using their Acces Point to adapt the TCP IP protocol, such as a LAN using a cable RJ - 45 for connection. PC Ethernet access is configured using their AP incorporated in the robot.

### ***Wireless LAN access point***

The characteristics of the connection of Wireless LAN access point is WLAN Standard IEEE 802.11b (11 Mbit to 2.4 GHz) and 802.11g (54 Mbps at 2.4 Ghz), with a limit range up to 100 meters, with a WEP encryption or WPA-PSK secure network.

Robotino has own access point: Wireless Access Point. This device can work as a mode of AP (Access Point) or as a client. In our case, Roboino is configured to work in Client mode.

In this mode (Client), the Access Point works exactly as a wireless modem wifi, the Access Point tries to find an external modem with certain specifications. The PC can connect to the Access Point through external WLAN.

In our case, the computers of the laboratories are connected through wireless modem, which is connected via WLAN with our Robotino (Illustration 8). The IP for the Robotino in our laboratory is 192.168.1.221.

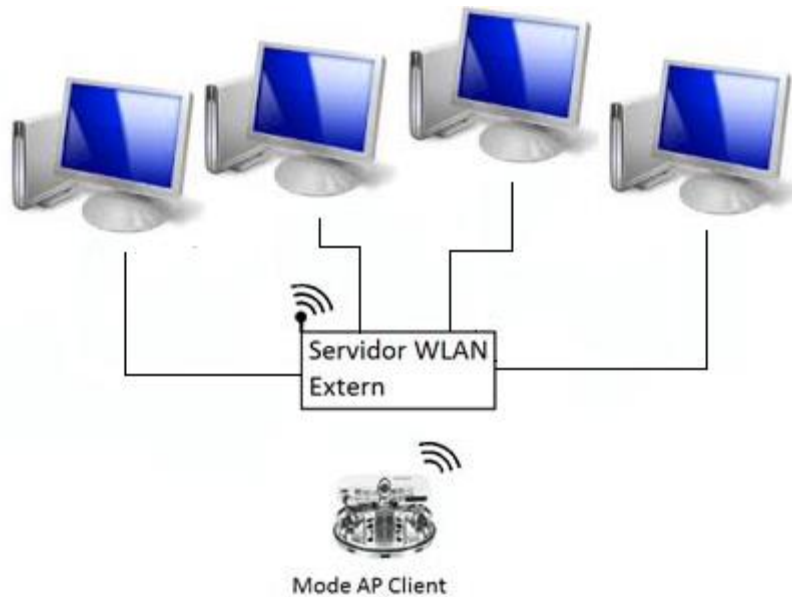


Illustration 8. Scheme of the communication using Wireless

### Compact Flash card.

The compact flash card contains the operating system of the computer, libraries of the functions it also can load a program with the instructions for the Robotino. The card used in this project has a capacity of 1 GB of memory.

### Interfaces.

For the connectivity, Robotino brings to us these interfaces as it shows in the illustration 9:

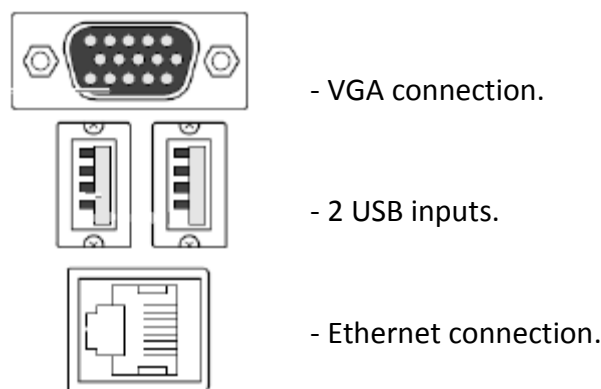


Illustration 9. Interfaces to connect Robotino

These interfaces are created for connecting a mouse, a keyboard and a screen to access the operating system, there is also an Ethernet input in case of failures in the WLAN communication, but this limits the mobility of the robot.

### Keyboard and Display.

We can see on top of the Robotino a keyboard and a display (Illustration 10), from this keyboard can observe and select options that we need to run a program manually. When it turns on the Robotino, we can see the computer name, IP address, version of software and the charge level of the battery in the display.



Illustration 10. Touch control command of Robotino

### Interface I/O.

It is an interface for connecting sensors and additional actuators, using:

- 8 analog inputs (0-10 V) (AIN0 to AIN7).
- 8 digital inputs (DI0 to DI7).
- 8 digital outputs (DO0 to DO7).
- 2 relays for additional actuators (REL0 and REL1). The relay contacts can be used as NO, NC or switched.



## **2.2 Sensors of the mobile robots**

The next part discusses the various sensors used in mobile robots. We discuss and explain some of their features.

### **2.2.1 Classification of the sensors:**

The classification of the sensors of the mobile robots is based on the characteristics of the sensors: reading speed, error rate and tolerance, but the sensors are mainly classified in internal receptors and external receptors.

#### ***External receptors:***

##### **Sensors for measuring distances based on ultrasound:**

Ultrasonic sensors emit an ultrasonic signal, subsequently the reflection is received. Ultrasonic sensors are formed by a capsule emitting and a receiving station or by a transducer that acts as transmitter and receiver.

This type of sensor has some limitations when it applies on a mobile robot:

- The speed of sound is variable: it can be affected by the density of air, humidity and dust concentration in the air.
- The blank time (blanking time): it is necessary to save a little time from being emitted until the receiver prepares to receive, aiming that not be influenced by the wave leaving the issuer.
- Attenuation: the ultrasonic wave leaves the sender, it will disperse, and the signal will be reduced depending on the environment.
- The measuring angle (Illustration 11):  
The signal has a profile complex amplitudes. Therefore, the echo not calculates accurately the location of the object.

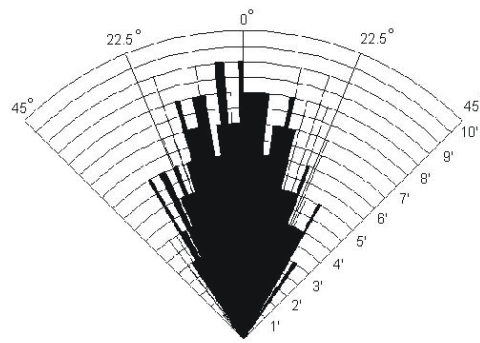


Illustration 11. Vision range in degrees

- Reflections: It is a serious impediment to detect the position of an object as it shows in the illustration 12. It can't ensure that the echo received is a result of a series of complex reflections by the environment.



Illustration 12. Representation of the reflections

- Cross-Talk: This effect occurs if the echo generated is received by another receptor as it shows in the illustration 13. This problem can be avoided, if it leaves a time between the measurements of two ultrasonic sensors.



Illustration 13. Representation of the Cross-Talk

About all the restrictions mentioned, the most important are: variation of the speed of sound, reflections and the angle measurement.

### Sensors for measuring distances operating in the infrared spectrum

Through these sensors we can estimate the distances between the sensor and the objects in the environment. There are different methods to measure the distance from an object:

- Triangulation: It uses geometric relationships between the output beam and the sensor position. As it shows in the illustration 14:

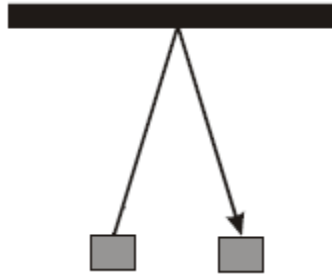


Illustration 14. Representation of the triangulation

- Flight time: Measures the time from the exit until is received the beam, after bounce off an object.

The advantages of this type of sensor can be summarized as follows:

- The laser can generate a million measurements in one second.
- Sensors are ideal for depth measurements.

On the other hand, there are a set of very important drawbacks:

- In the case of laser the price is very high (3000-10000 euros).
- The consumption of the laser.
- It does not detect obstacles or above/below the horizontal plane of measurement therefore it is not good to avoid obstacles.

### Artificial vision:

It can be distinguished at least two types of cameras, working in the visible spectrum and working in the infrared spectrum. On this field, there are algorithms to filter noise, offset lighting problems, finding lines, matching lines with models, draw shapes and build three-dimensional representations.

Video camera has the following advantages:

- It offers much information on each image.
- It is ideal for detecting colors, textures and objects.

On the other hand, there are disadvantages such as:

- High computational cost. Thus, the robots usually have several processors, some of them exclusively devoted to processing images.
- The price is high. A machine vision system, usually consists of one or two video cameras.
- The images depends on the lighting.
- It is not the right sensor for measuring depth.

### Proximity sensors:

Proximity sensors usually have a binary output indicating the presence of an object within a specified range of distance. Under normal conditions, the proximity sensors used in robotics for work in near field to catch or avoid an object. Any sensor to measure distance can be used as a proximity sensor.

There are three types of typical proximity sensors in mobile robots.

#### - *Inductive sensors*

Sensors based on a change in inductance due to the presence of a metallic object are proximity sensors most commonly used on the industrial environment. An inductive sensor is essentially a coil next to a permanent magnet packaged in a simple and robust receptacle. The effect of bringing the sensor to the proximity of a ferromagnetic material produces a change in the position of the flux lines of the permanent magnet. In static conditions there is no movement in the flow lines and therefore it does not induce any current in the coil.

- *Hall Effect sensors:*

The Hall Effect relates the tension between two points of a conductor or semiconductor material with a magnetic field through the material. The Hall Effect sensors only detects magnetized objects.

- *Capacitive sensors:*

Like with inductive and Hall Effect sensors, this sensor only detects ferromagnetic materials. Capacitive sensors are potentially capable (with varying degrees of sensitivity) to detect all solid and liquid materials.

### **Lighting sensors: photo resistance, photodiodes and phototransistors**

- The light sensors allows to hide in the dark the robot and when a light shows up it moves toward that. The photoresistance depends of the resistance of the sensor.
- The phototransistors have greater sensitivity to light than the photoresistance. The phototransistor is basically a transistor which has been generated by the base collector.
- The photodiodes have high sensitivity, produces a linear output over a wide range of light levels, and responds quickly to changes in lighting. This makes them useful in communication systems in order to detect modulated lights; the remote control for almost all TV, stereos and CD players employ them.

### ***Other external receptors***

Other external receptors are of contact, touch, radar-based microwave, thermocouples, etc.

### ***Internal receptors:***

This type of sensors give information of the position and orientation of the robot.

The positioning sensors can be classified into two groups:

- Absolute measurement sensors: They give a measure of the pose respect to a fixed reference environment. Absolute sensors make mistakes when it measures but these errors do not accumulate over time.
- Incremental sensors: These sensors measures an incremental movement respect to a point, but it has the disadvantage that the measurement errors are cumulative.

### **GPS:**

The GPS is a system that provides a measure of the position of the robot in open air, and therefore it is considered an absolute measurement sensor.

The GPS has the following advantage:

- It is the only sensor of absolute position which works on any external environment.

On the other hand, it has several drawbacks:

- It can't be used in buildings, because that blocks the satellite signal.
- It is expensive

### **Sensors for measuring the orientation of the robot**

#### ***Inclinometer:***

An inclinometer is a device that measures orientation of the gravity vector. The most common inclinometer uses mercury as a liquid for equilibrate. To measure the inclination needs to be on a platform that is not subject to acceleration, otherwise the measure is flawed. It is very sensitive to vibration.

### *Gyroscope:*

The compass uses the earth's magnetic field to determine the orientation of the robot.

The main advantage is:

- It is the only absolute measurement sensor which measures the orientation of the robot anywhere in the world.

It has important limitations:

- It is sensitive to external magnetic fields and it is sensitive to metallic elements that are very close to the robot. If magnetic field distortion is caused by the robot, the problem can be corrected. This effect is called Soft Iron Distortion.

### **Incremental encoder**

Optical encoders or incremental encoders consists of a transparent disk which includes a series of opaque markings placed radially, equidistant from a lighting system and a light receiving element as it shows in the illustration 15. The axis is coupled to the transparent disk. With this arrangement, the receiver pulses counts each time the light passes through each brand and taking account of these pulses is possible to know the position of the axis and the rotation speed.

Main limitations are:

- The information about the position is lost when the system is unpredictably switched off or when there are strong disturbances in the environment.
- Devices are particularly sensitive to shock and vibration.
- There is lack of information if the encoder changes the direction.

One solution to the latter problem, it has another strip of marks offset from the previous, this extra pulse train (B) is generated displaced 90° electrical respect to generated by the first strip A.

It is possible to obtain an additional signal in order to indicate the direction of rotation which acts on the corresponding counter indicating the increase or decrease of the direction. It is also necessary have a reference mark on the disk Z which indicates a full rotation, therefore it has to start counting again. This brand also serves to start counting after recovering from a power failure.

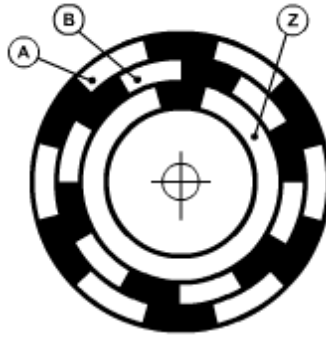


Illustration 15. Representation of the encoder disc

The resolution of this type of sensor depends directly on the number of brands that are on the disc. A relatively simple method to increase resolution is counting the rising edges of the pulse trains and the pulse account down, with this solution it increases the resolution of the sensor.

The unit of measurement to define the accuracy of an encoder is called electrical degree.

To know how many mechanical degrees correspond to 360° electric, apply the following equation:

$$360^\circ \text{ electric} = \frac{360^\circ \text{ mechanical}}{\frac{n^\circ \text{ pulse}}{\text{turn}}} \quad (2)$$

### Speed sensors

The speed of movement of each actuator is normally fed to an analog control loop implemented in the motor driver element. Normally, the speed control loop is analog, the sensor uses a generatrix which provides the rotational speed of the axis (10 mV per rpm).



## 2.3 About the wheels of the mobile robots

### 2.3.1 Wheels:

The wheels are the most popular method to provide mobility on a robot and they are used to propel many different sized robots and platforms. There's multiple sizes of wheels, different types of tire and different numbers of wheels. The little robots tend to have the smallest wheels, usually less than 2 centimetres in diameter, 3 and 4 are the most common number of wheels.

Usually, a robot with three wheels uses two of them as normal wheels for axis, and the last one is a caster wheel.

The robot which uses a gyroscopic stabilization are a complex robot. A robot with four or six wheels have the advantage of using multiple drive motors (one connected to each wheel) which reduces the slip.

But the wheels that use our Robotino are omnidirectional wheels. This omnidirectional wheels give the robot a significant mobility over the other types. Summary of the advantages and disadvantages in the table 1.

Advantage:	Disadvantages:
– Usually low-cost	– May lose traction (slip)
– Simple design and construction	– Small contact area
– Six wheels replaces a track system	
– Diameter, width, material, weight, tread etc. can all be custom to your need	
– Excellent choice for beginners	

Table 1. Advantages and Disadvantages using wheels

### 2.3.2 Tracks:

Track drives are similar to the wheels that tanks. Use track drives are the best option for robots used outdoors. Tracks do not provide an extra grip strength but can reduce slip and distribute the weight of the robot. Summary of the advantages and disadvantages in the table 2. Making them useful for surfaces such as sand and gravel.

Advantage:	Disadvantages:
– Constant contact with the ground prevents slipping	– There is a sideways force that acts on the ground; <b>this causes damage to the surface the robot</b>
– Evenly distributed weight helps robot to tackle a variety of surfaces	– Not many tracks are available.
	– Increases the mechanical complexity and connections

Table 2. Advantages and Disadvantages using tracks

### 2.3.3 Legs:

An increasing number of robots use legs for mobility. Legs are often preferred for robots that must run on uneven terrain. Several of the amateur robots are designed with six legs which allows the balance of robot. Robots with fewer legs are harder to balance. Researchers made experiments with bipeds, quadrupeds and hexapods. Summary of the advantages and disadvantages in the table 3.

Advantage:	Disadvantages:
– Closer to organic/natural motion	– Increased mechanical, electronic and coding complexity
– Can potentially overcome large obstacles and navigate through very rough terrain	– Higher cost to build

Table 3. Advantages and Disadvantages that using legs

## 2.4 Type of locomotion:

### 2.4.1 Differential

From the point view of programming and construction, the differential design is one of the least complicated systems of locomotion as it shows in the illustration 16. The robot can go straight, turn on itself or take a curve. Summary of the advantages and disadvantages in the table 4.

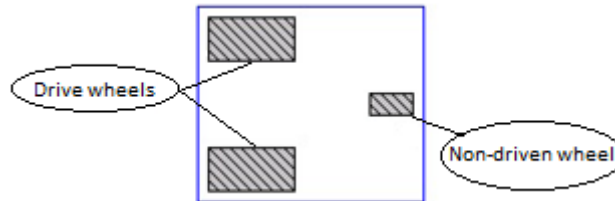


Illustration 16. Disposition of the differential wheels

Advantages:	Drawbacks:
- Inexpensive system	- Difficult to control
- Easy to implement	- Requires control precision and trajectories straight
- Simple design	

Table 4. Advantages and Disadvantages using differential wheels

### 2.4.2 Synchronous

In this design, all of the wheels (usually three, Illustration 17) are steering and driving, the wheels are locked so that always point in the same direction. To change the direction of the robot, it has to turn simultaneously all wheels around a vertical axis, but the chassis is still pointing in the same direction. Summary of the advantages and disadvantages in the table 5.

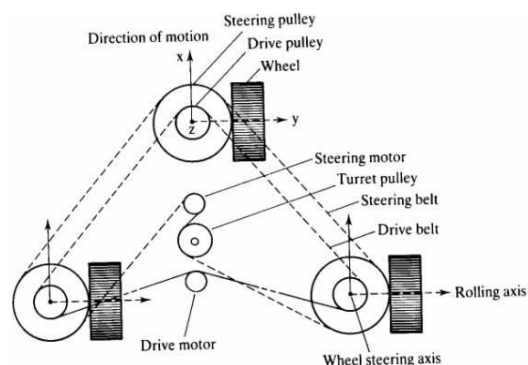


Illustration 17. Disposition of the Synchronous wheels

Advantages:	Drawbacks:
- Each motor works independent to each other and the rotation is simplify to control	- Design and implementation are complex
- The straight-line control is guaranteed mechanically	

Table 5. Advantages and Disadvantages using Synchronous wheels

### 2.4.3 Tricycle

The tricycle has two wheels connected by a shaft and one steerable wheel. The steerable wheel, only serves to determine the direction of the robot. As it shows in the illustration 18.

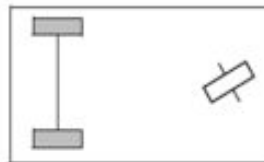


Illustration 18. Disposition of the tricycle wheels

Advantages and disadvantages of Synchronous locomotion (Table 6):

Advantages:	Drawbacks:
- Easy construction	- Requires control precision trajectories
- Simple design	- Restriction of certain movements
- Reduced slip	

Table 6. Advantages and Disadvantages using Tricycle wheels

#### 2.4.4 Ackerman

Ackerman wheels are based on the Ackermann steering geometry. Disposition wheels in the illustration 19. It is a geometric solution to solve the problem of the wheels when it takes a curve from the inside or the outside which traces out circles of different radii in the curves. Summary of the advantages and disadvantages in the table 7.

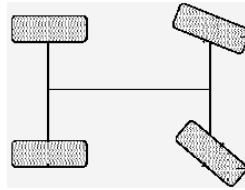


Illustration 19. Disposition of the Ackerman wheels

Advantages:	Drawbacks:
- Easy construction	- The robot can move immediately forward or backward but not laterally by the wheel spin.
- A system of 4 wheels controls the direction.	

Table 7. Advantages and Disadvantages using Ackerman wheels

#### 2.4.5. Omnidirectional (Wheels with rollers – Swedish Wheels)

It is a wheel with small discs around the circumference which are perpendicular to the turning direction (Illustration 20). The wheels moves with full force, but it also can slide laterally with great easiness. Advantages and Disadvantages in the Table 8.

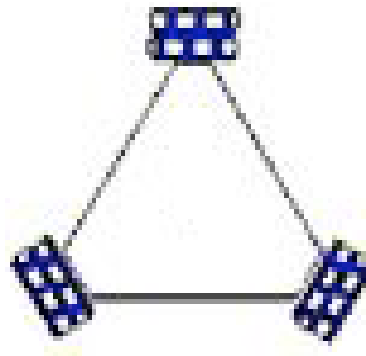


Illustration 20. Disposition of the omnidirectional wheels

Advantages:	Drawbacks:
- Allows to do complicated movements	- The straight-line motion is not guaranteed by mechanical constraints: it is necessary implement a control
	- Complex implementation

Table 8. Advantages and Disadvantages using omnidirectional wheels

#### 2.4.6 Locomotion by sliding tapes

Many robots use this method of travel, but most of them are oriented to dangerous or where the displacement is complex, as military purposes or space exploration environments. Robots with slip tapes are the least common for simple activities. Since they have higher costs, greater scuffing and be slower.

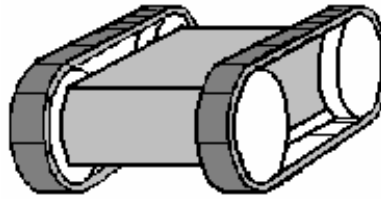


Illustration 21. Disposition of the locomotion by sliding tapes

Advantages and Disadvantages in the Table 9.

Advantages:	Disadvantages:
- <b>System easy to control</b>	- It doesn't have an accurate model of rotation
	- Consumes much power and time to rotate.

Table 9. Advantages and Disadvantages using locomotion by sliding tapes

### 2.4.7 Legs

The human being is the model of bipedal locomotion, so the biped robots are mainly designed trying to emulate our legs, as it shows the illustration 22. But control of bipedal locomotion has two major problems: stability control, necessary to maintain an upright posture and motion control that enables a shift forward at various speeds. Despite this, bipedalism has several advantages, such as access to very difficult terrain.



Illustration 22. Disposition of the locomotion by legs

Advantages and Disadvantages in the Table 10.

Advantages:	Drawbacks:
- It can moves through any terrain.	- Many degrees of freedom, it is very difficult to control.
	- Maintaining stability is complicated.
	- Consumes a lot of energy and time for movements

Table 10. Advantages and Disadvantages using locomotion by legs

## 2.5 Sensors of Robotino

### 2.5.1 Collision sensor or the sensor of the bumper.

It is formed by a band surrounding the detection chassis (black band) and contains a chamber with two conductive surfaces, these surfaces maintain a distance from each other, and with them it can detect the minimum change of the pressure on the band. This allows the robot to be able to stop when it collides with an obstacle.

### 2.5.2 Infrared sensors.

Robotino is equipped with nine infrared sensors, SHARP brand, model GP2D12, which are mounted on the chassis all around the robot with a separation of 40 cm to each other which allowing to monitor obstacles or elements in their surrounding area.

It is capable to measure distances between the sensor and the obstacle with a range between 4 and 30 cm.

### 2.5.3 Camera (Webcam)

The Robotino works in a programming environment that allows to work with images and video in real time, it is possible with a VGA CMOS color camera with an USB 1.1 connection. Disposition of the camera in illustration 23.

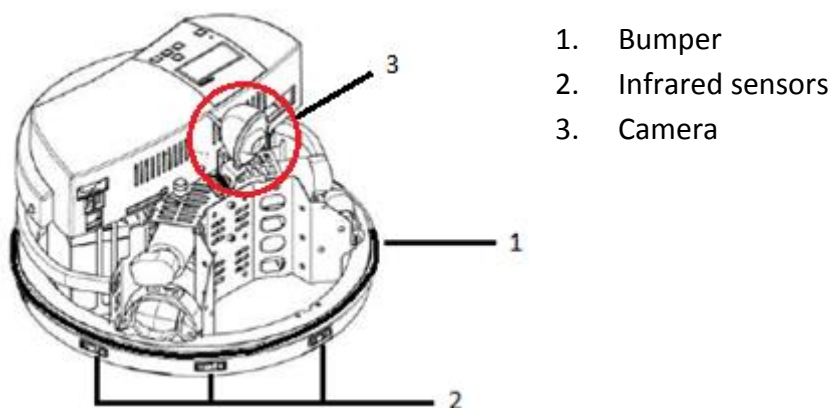


Illustration 23. Localization of the webcam



### ***Determination of the robot position through camera***

In order to determine the position of the robot, we use a web camera to take the photos and obtain information from the line. We use two functions of Matlab already programmed to obtain this information.

First one, "pintaimatge.m" draws an image on a 3x2 grid, and the second one, "mesuraDistanciaARuta.m" gives us the distance and the angle between the robot and the line on the x axis. One of the objectives of this project is to find a way that the robot follows a line using vision camera.

The "mesuraDistanciaARuta.m" has been modified slightly to be able to obtain the distance and the angle respect to a vertical line, but it also obtains the distance and the angle respect to a horizontal line, renamed to "XavmesuraDistanciaARuta.m".

The complete program of the XavmesuraDistanciaARuta.m is on the Annex 9.1.1 Programing code of the XavmesuraDistanciaARuta.m

### ***Camera***

In order to obtain the information from photo as accurate as possible and using this functions it is necessary to calibrate the parameters of the camera. There are two types of parameters: intrinsic and extrinsic.

The intrinsic parameters are invariant since they depend on the camera, but the extrinsic parameters depend on the camera calibration. We have to make a calibration to obtain the rotation and translation matrices to represent a pixel of the captured image to a point in coordinates of robot every time that we start a test in the laboratory and the camera is on a new position.

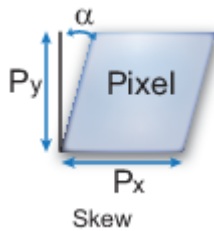
### *Intrinsic parameters of the camera*

All intrinsic parameters can be collected in one matrix named camera matrix. This matrix allows to relate by a linear equation, the pixel on coordinate vector with the vector on normalized coordinates.

The camera matrix is defined as:

$$KK = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The pixel is defined as:



$[c_x, c_y]$  — Optical center (the principal point), in pixels.

$(f_x, f_y)$  — Focal length in pixels.

$f_x = F/p_x$

$f_y = F/p_y$

$F$  — Focal length in world units, typically expressed in millimetres.

$(p_x, p_y)$  — Size of the pixel in world units.

$s$  — Skew coefficient which is non-zero if the image axes are not perpendicular.

$s = f_y \cdot \tan(\alpha)$

The calibration of these intrinsic parameters have already been made previously, I am going to use their results because they use the same camera and the intrinsic parameters are invariant.

The values of the camera matrix are as follows:

$$KK = \begin{bmatrix} 404.8412 & 0 & 143.9544 \\ 0 & 408.8882 & 108.0107 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

### Extrinsic camera parameters

#### Camera Calibration ( $T_{c\_ext}$ , $R_{c\_ext}$ )

I proceed to calibrate the extrinsic parameters and determine the rotation and translation matrices to represent one point in world coordinates to a point in camera coordinates.

Before performing the calibration, at least we should take one photo of the grid of squares in black and white. Note that we should place the Robotino in the drawing where indicates in the illustration 24.

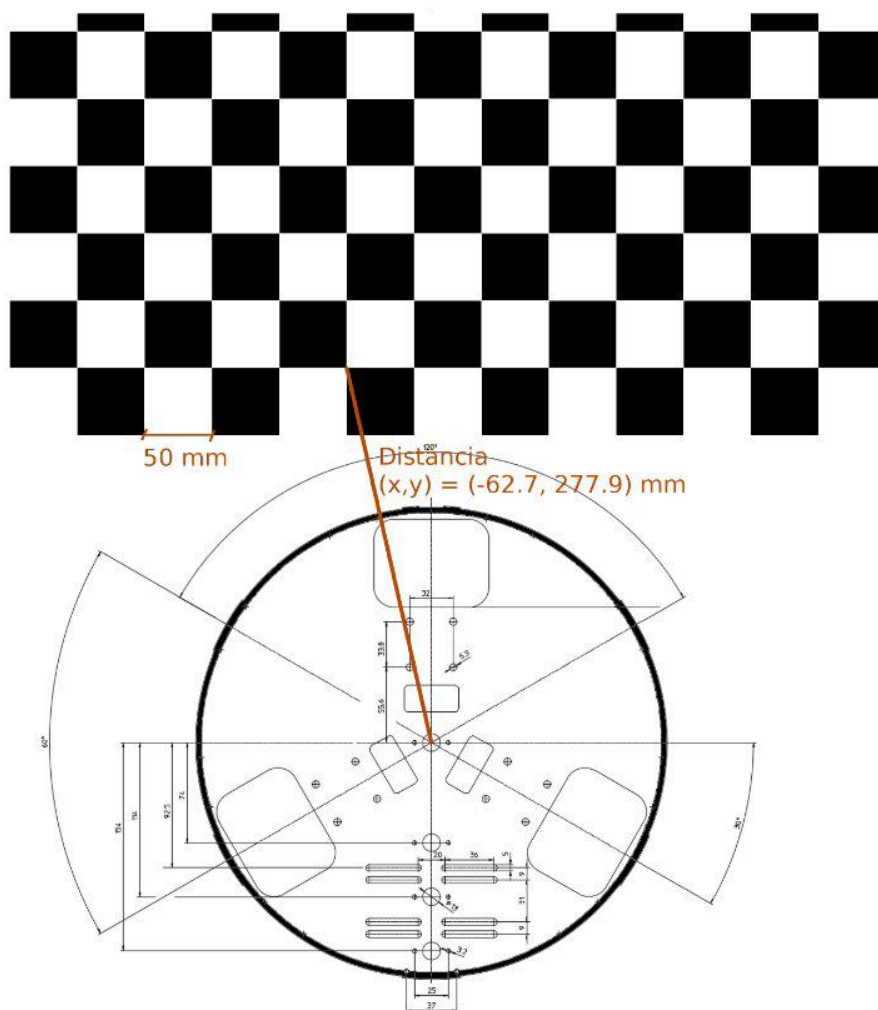


Illustration 24. Work area for calibration

There is a toolbox in matlab previously installed that allows to obtain these matrices. Once the photo has been taken we will save it in the directory that we work with the name 'Quadricula' in "tif" format. Using the command:

```
imwrite (img, 'Quadricula.tif')
```

The KK matrix along with others parameters such as their deviation of the error are in a file called Calib\_Results.mat. We need to load this file using the command of Matlab:

```
load Calib_Results.mat
```

Once this has been done, we have to run on the main page of Matlab the command:

```
extrinsic_computation
```

Follow the next steps that indicates Matlab:

The first indication from Matlab says:

```
Computation of the extrinsic parameters from an image of a pattern.  
The intrinsic camera parameters are assumed to be known (previously  
computed)
```

```
Image name (full name without extension):
```

We write the name of the photo that we already take without the extension (in our case 'Quadricula'):

```
Image name (full name without extension): Quadricula
```

The next step asks the image format:

In our case, we have to write the letter "t" as indicates the legend.

```
Image format: (['r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg',  
'm'='ppm']) t
```

**Warning: The image format can only be in a format that indicates the legend.**

The program want's to indicate the length (in pixels) of the window for extracting the vertices of the square, by default, 5 pixels for x and y.

```
Extraction of the grid corners on the image  
Window size for corner finder (wintx and winty):  
wintx ([ ] = 5) = 5  
winty ([ ] = 5) = 5  
Window size = 11x11
```

It now appears an interactive image where we have to indicate the vertices of a rectangular region (Illustration 25). The order of vertices doesn't matter, but the first one must be the upper left vertex.

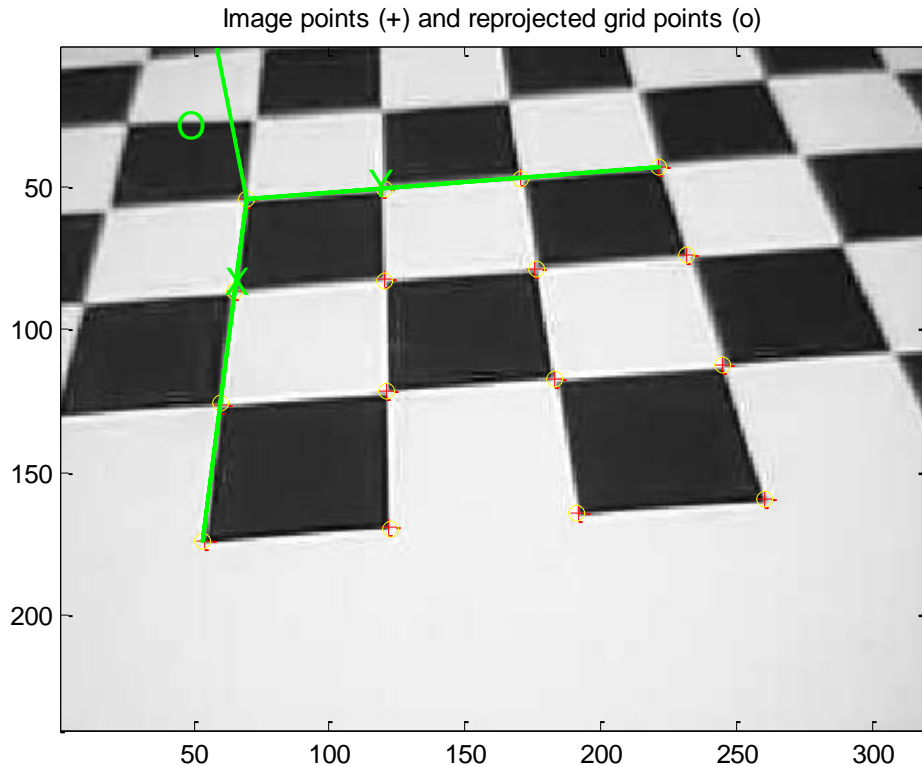


Illustration 25. Interactive image once marked the rectangular region

The next step is necessary and sometimes not, it depends on the size of the chosen region. Sometimes the program is not able to identify how many squares are on the x axis and the y axis, if it were the case we would have to indicate it manually:

```
Could not count the number of squares in the grid. Enter manually.
Number of squares along the X direction ([] = 10) = 3
Number of squares along the Y direction ([] = 10) = 3
```

Finally, the program requests the length of the squares on the x axis and y in mm.

In the case of our grill in the lab, the length of the square is equal to 50mm

```
DX size of each square along the X direction ([] = 30 mm) = 50
DY size of each square along the Y direction ([] = 30 mm) = 50
```

When we finished to introduce the orders, the program extracts the matrices of rotation, translation and the pixel error. This error is important to make sure that it is small to ensure accuracy, otherwise the calibration should be repeated until the error is acceptable. (Less than 0.5 is acceptable)

```
Corner extraction...
Extrinsic parameters:
Translation vector: Tc_ext = [-74.543326  -53.266620  401.815736]
Rotation vector:   omc_ext = [ 1.798734   1.667851  -0.787365]
Rotation matrix:   Rc_ext = [ 0.054699   0.997407  -0.046763
                             0.669964   -0.071387  -0.738953
                             -0.740375   0.009090  -0.672132]

Pixel error:      err = [ 0.33049  0.31560 ]
```

### Step to convert coordinates of robot into coordinates calibration

The illustration 26 shows the distance from a known point on the grid and the center of the robot. From this image of calibration we can determine the translation and rotation that have to done to convert the coordinate system of the robot into a calibration coordinates with a point that we know. The upper left point of the rectangular region in illustration 27 is located two square above the point showed in the illustration 26, (-62.7,277.9). Thus the translation matrix is the reverse of [-62.7, 377.9, 0] and the rotation is -90°. The axes is what we want to change that is why we must take the opposite sign of the angle, 90°.

I transform the matrices homogeneously to operate with them:

Rotation:

$$R_{rc} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 5.1.a$$

Translation:

$$t_{rc} = \begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(377.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 5.1.b$$

### Step to convert coordinates of robot into coordinates camera

We proceed with two steps in order to convert the coordinates: first the translation and then the rotation. Secondly: the calibration coordinates into a camera coordinates. Following this equation:

$$M = t_{cal \rightarrow cam} * R_{cal \rightarrow cam} * R_{rob \rightarrow cal} * t_{rob \rightarrow cal} \quad (5.2)$$

This gives to us an array called "M" which includes the matrix of rotation and translation (R, t):

$$M = \begin{bmatrix} R_{3 \times 3} & t_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{bmatrix} \quad (5.3)$$

Command Execution:

```
Rrc=[0,-1,0,0;1,0,0,0;0,0,1,0;0,0,0,1];
Trc=[1,0,0,-(-62.7);0,1,0,-(377.9);0,0,1,0;0,0,0,1];

Tc_ext_homo=[1,0,0,Tc_ext(1);0,1,0,Tc_ext(2);0,0,1,Tc_ext(3);0,0,0,1];
Rc_ext_homo=[Rc_ext(1,1),Rc_ext(1,2),Rc_ext(1,3),0;Rc_ext(2,1),Rc_ext(2,2),Rc_ext(2,3),0;Rc_ext(3,1),Rc_ext(3,2),Rc_ext(3,3),0;0,0,0,1];

M=Tc_ext_homo*Rc_ext_homo*Rrc*Trc;

t=[M(1,4);M(2,4);M(3,4)];
R=[M(1,1),M(1,2),M(1,3);M(2,1),M(2,2),M(2,3);M(3,1),M(3,2),M(3,3)];
```

$$t = \begin{bmatrix} 8.6647 \\ 195.7370 \\ 122.5978 \end{bmatrix} \quad (5.4. a)$$

$$R = \begin{bmatrix} 0.9974 & -0.0547 & -0.0468 \\ -0.0714 & -0.6700 & -0.7390 \\ 0.0091 & 0.7404 & -0.6721 \end{bmatrix} \quad (5.4. b)$$

Remember that we must perform calibration every time that the camera is moved.

### ***Method to binarization of image***

The two functions that provide us the information regarding the position of the robot line were already programmed.

The first function is "pintaimatge.m ". This function is used within the XavmesuraDistanciaARuta.m function to draw the images binarized in the 3x2 grid.

The second function "XavmesuraDistanciaARuta.m" is the function that really provides the distance and angle that we want. We below discuss more deeply about this function.

#### ***pintaimatge.m***

This function draws an image on a grid of 3x2 which binarizes the image in three matrices of 320x240. Each matrix is defined on the colour plane R G B. From that matrix, we obtain an image in black and white (the background in black and the line in white).

#### ***XavmesuraDistanciaARuta.m***

[D1, theta] = XavmesuraDistanciaARuta (img, R, t, KK, []) is the Matlab command that calls the script with the same name "XavmesuraDistanciaARuta". This function returns the distance and angle from the axis y.

This script has 4 outputs:

d1\_ver, theta\_ver: The distance and angle of the robot respect of the vertical line.

d1\_hor, theta\_hor: The distance and angle of the robot respect of the horizontal line.

**Criteria of signs:**

*Respect to a vertical line:*

If the orientation is to the left, the distance and angle are positive.

If the orientation is to the right, the distance and the angle are negative.

*Respect to a horizontal line:*

If the orientation is to the right, the distance and angle are positive.

If the orientation is to the left, the distance and the angle are negative.

**Requirements:**

The program needs that the far side of the line corresponds to the positive direction of the axis of the world.

**Required parameters:**

**Img:** This variable is the photo that takes while the robot is following the line. Therefore, it takes photos at sampling time that indicates the camera (30 fps). Although this value can vary and be lower between 15-20 fps for unknown reasons.

**R:** rotation matrix.

**KK:** Matrix of the camera.

**Warning:**

D1: The units of the distance is in millimeter.

Theta: The margin of the angle are [-180.180] and the units is in degrees.



## 2.6 Robotino wheels

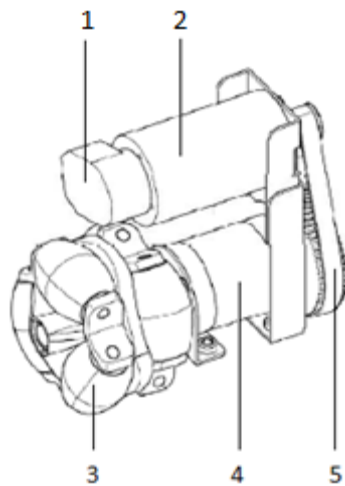
Robotino uses omnidirectional wheels, there are three motors and three wheels separated by  $120^\circ$  each one, each drive unit contains the following elements:

- DC Motor.
- Incremental Encoder.
- Omnidirectional rollers.
- Reductor with a reduction ratio of 16: 1.
- Timing Belt.

**In the case of omnidirectional rollers are directed in one direction by their own axis.**

But it can move in any direction, this is because the work of two rolls together can create a different direction to the established. For example, if it moves two of the three wheels in the opposite direction, the robot will move forward. But if the direction is the same, the robot will move backward.

Disposition of the elements in the Illustration 26:



- 1 - Incremental Encoder.
- 2 - DC Motor.
- 3 - Omnidirectional wheels.
- 4 - Reductor with a reduction ratio of 16: 1.
- 5 – Timing Belt.

Illustration 26. Localization of the different parts of the omnidirectional wheels

### **3. IDENTIFICATION AND CONTROL OF THE PROPULSION WHEELS**

#### ***3.1 Introduction***

In this present chapter I find how to control the Robotino's omnidirectional wheels. First, we have to find a model for each wheel, also identify their parameters, and then design a PI controller (with corresponding parameters) for each wheel of the Robotino.

The second part of the chapter includes the disturbances from the captured data. We are going to check using Matlab tool "Simulink" if these disturbances affect the performance of the PI controller.

To obtain the model for each wheel we used experimental data which have been collected by professors of ESEIAAT.

#### ***3.2 Design the model for the wheels***

In the board EA09 of the Robotino there are controllers ( $K_p$  and  $K_i$ ) to control the wheels of the robot. From the data experiments performed by others, we to tune these parameters. **This tuning relates the tension send to the motor and the measured speed of the wheels when the Robotino is on the floor.**

#### ***3.3 Identification experiments***

I explain below the procedure for the wheel 0 and the same procedure repeats for the other two wheels. At the end of this chapter, we are going to show the results for each wheel. The next illustrations (27), (28) are the response for the three wheels when wheel 0 is the only one which receives input signal. (Sample time equal to 0.001).

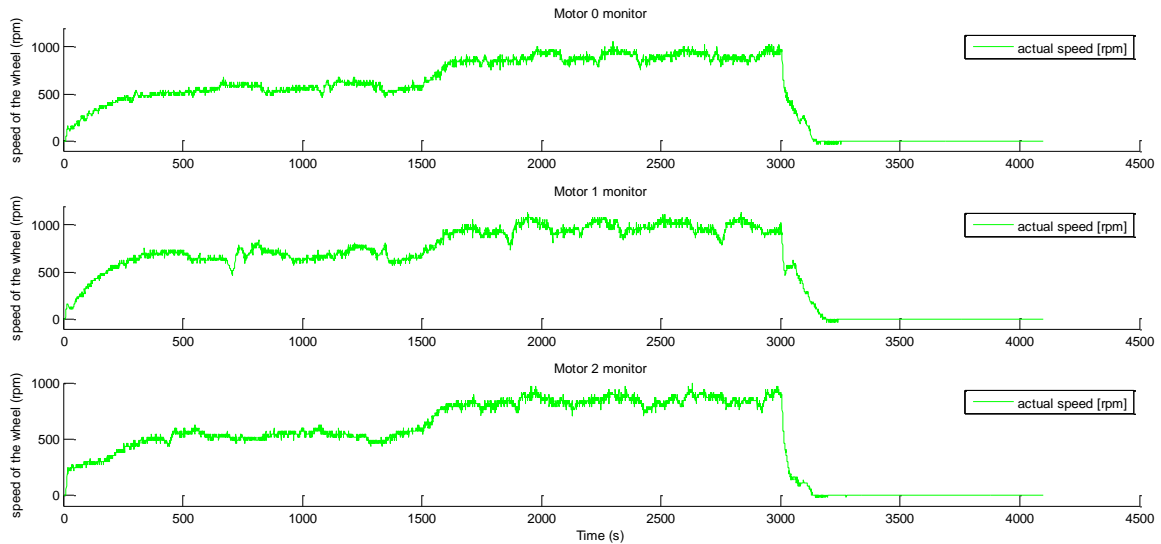


Illustration 27. Experimental results (realized by professors)

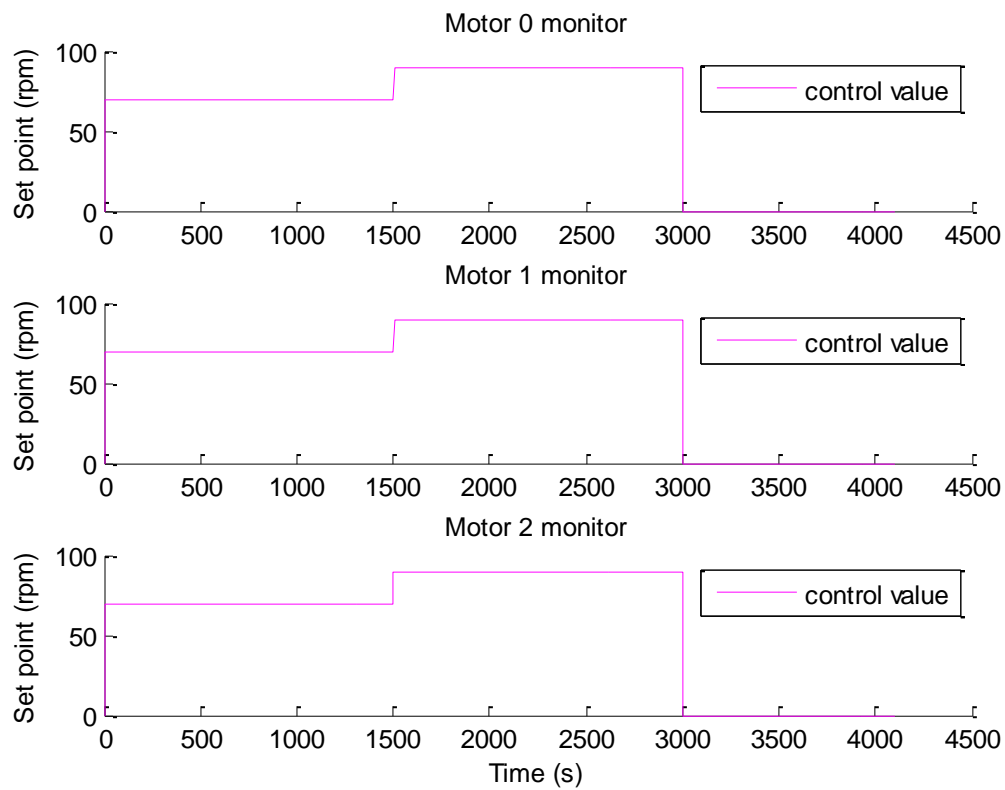


Illustration 28. Experimental results (realized by professors)

I focus on the response of the wheel 0. I show a better image of the response of the wheel 0.

### 3.4 First-order model

I have two transient regimes and two permanent regimes. In order to work better with this data, I find a first-order model high frequency noise observed in next illustration 29.

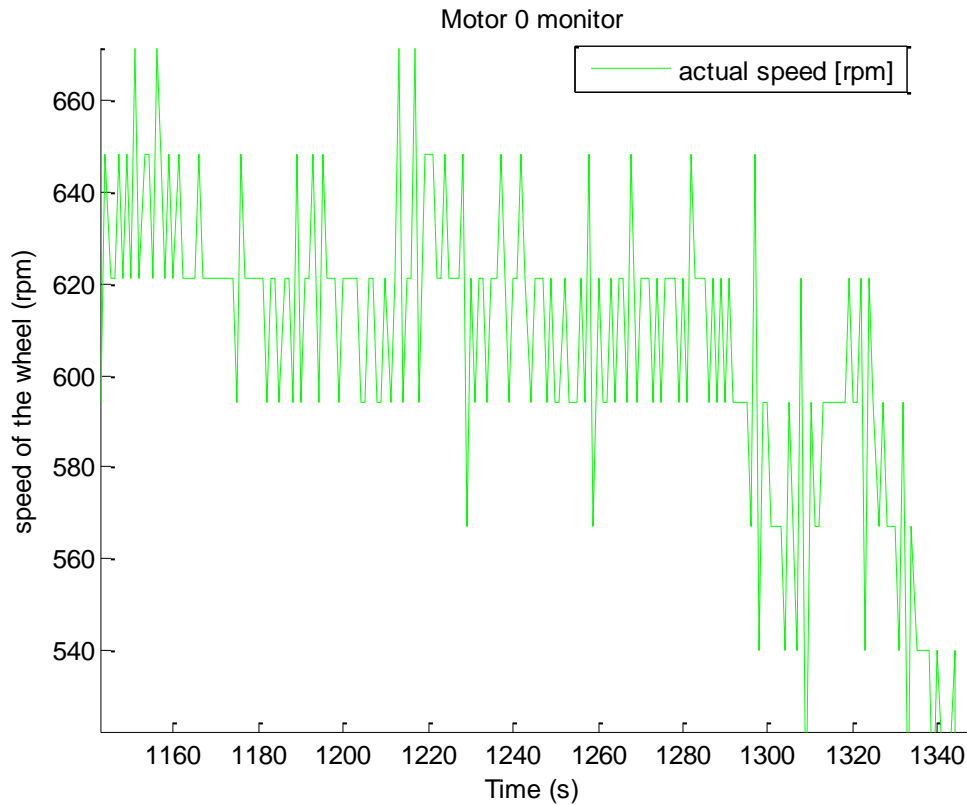


Illustration 29. Data variations between samples time

The values after the 1000 samples of the sampling period have been taken as zero reference, the delay time are consider it equal to 1 ( $N = 1$ ).

In order to obtain this reference, I make the average of the values for the first steady state regime and subtract it the values. I repeat the process with the control signal, but the average was not necessary because it was a fix value. I use the Matlab function “oe” to find a model.

The transfer function obtained is:

$$G(z) = \frac{0.1553 z^{-1}}{1 - 0.9907 \cdot z^{-1}} \quad (6.1)$$

The comparison between the experimental data and the model in the illustration 30:

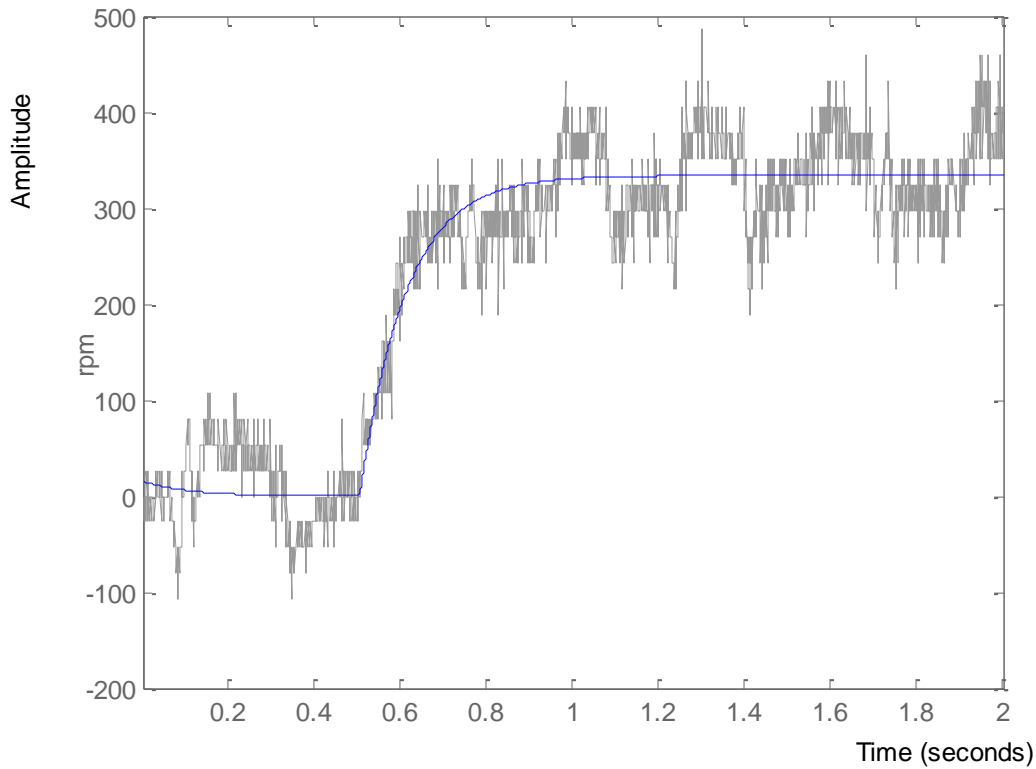


Illustration 30. Model representation for the wheel 0

### 3.5 Parameters of the controller

Desired plant:

$$G(z) = \frac{(1 - a_{ideal}) \cdot z^{-1}}{1 - a_{ideal}z^{-1}} \quad (6.2. a)$$

Next step is to calculate Kp and Ki controller parameters that is why I calculate a first order desired model in discrete time introducing into a controller. This is the desired model if we want to reduce the error:

$$G_{PI}(z) = \frac{1 - a \cdot z^{-1}}{b \cdot z^{-1}} \cdot \frac{\frac{(1 - a_{ideal}) \cdot z^{-1}}{1 - a_{ideal}z^{-1}}}{1 - \frac{(1 - a_{ideal}) \cdot z^{-1}}{1 - a_{ideal}z^{-1}}} = \frac{1 - az^{-1}}{b \cdot z^{-1}} \cdot \frac{\frac{(1 - a_{ideal}) \cdot z^{-1}}{1 - a_{ideal} \cdot z^{-1}}}{\frac{1 - z^{-1}}{1 - a_{ideal} \cdot z^{-1}}} =$$

$$\begin{aligned}
&= \frac{1 - az^{-1}}{b \cdot z^{-1}} \cdot \frac{(1 - a_{ideal}) \cdot z^{-1}}{1 - z^{-1}} = \frac{(1 - az^{-1}) \cdot (1 - a_{ideal}) \cdot z^{-1}}{b \cdot z^{-1} \cdot (1 - z^{-1})} = \\
&= \frac{(1 - a_{ideal}) \cdot z^{-1} - a(1 - a_{ideal}) \cdot z^{-2}}{b \cdot z^{-1} \cdot (1 - z^{-1})} = \\
&= \frac{1}{b \cdot z^{-1}} \cdot \frac{(1 - a_{ideal}) \cdot z^{-1} - a(1 - a_{ideal}) \cdot z^{-2}}{(1 - z^{-1})} = \\
&= \frac{\frac{(1 - a_{ideal})}{b} - \frac{a(1 - a_{ideal})}{b} \cdot z^{-1}}{(1 - z^{-1})} \quad (6.2. b)
\end{aligned}$$

PI control discretization:

$$H_{PI}(z) = kp + ki \frac{1}{1 - z^{-1}} \quad (6.2. c)$$

I have to compare the model with the function for a PI controller in discrete-time, and then determine the equation for Kp and Ki, this is the solution:

$$kp = \frac{a(1 - a_{ideal})}{b} \quad ki = \frac{(1 - a_{ideal})}{b} - kp \quad (6.3)$$

I want a controller that makes the error zero in steady state fast, I adjust the pole not too close to 1, in a first case I choose the pole equal to 0.998.

For  $a = 0.9907$ ,  $b = 0.1553$ ,  $a_{ideal} = 0.998$

$$kp = \frac{a(1 - a_{ideal})}{b} = \frac{0.9907 \cdot (1 - 0.998)}{0.1553} = 31.89633 \cdot 10^{-3} \quad (6.4. a)$$

$$ki = \frac{(1 - a_{ideal})}{b} - kp = \frac{(1 - 0.998)}{0.1553} - 31.89633 \cdot 10^{-3} = 2.9942 \cdot 10^{-4} \quad (6.4. b)$$

The following illustrations (31, 32) are the simulation result:

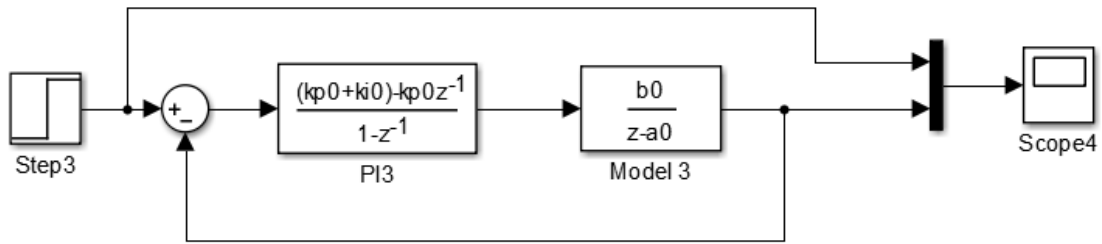


Illustration 31. Simulation of the system with the controlled signal and the PI controller that we calculated

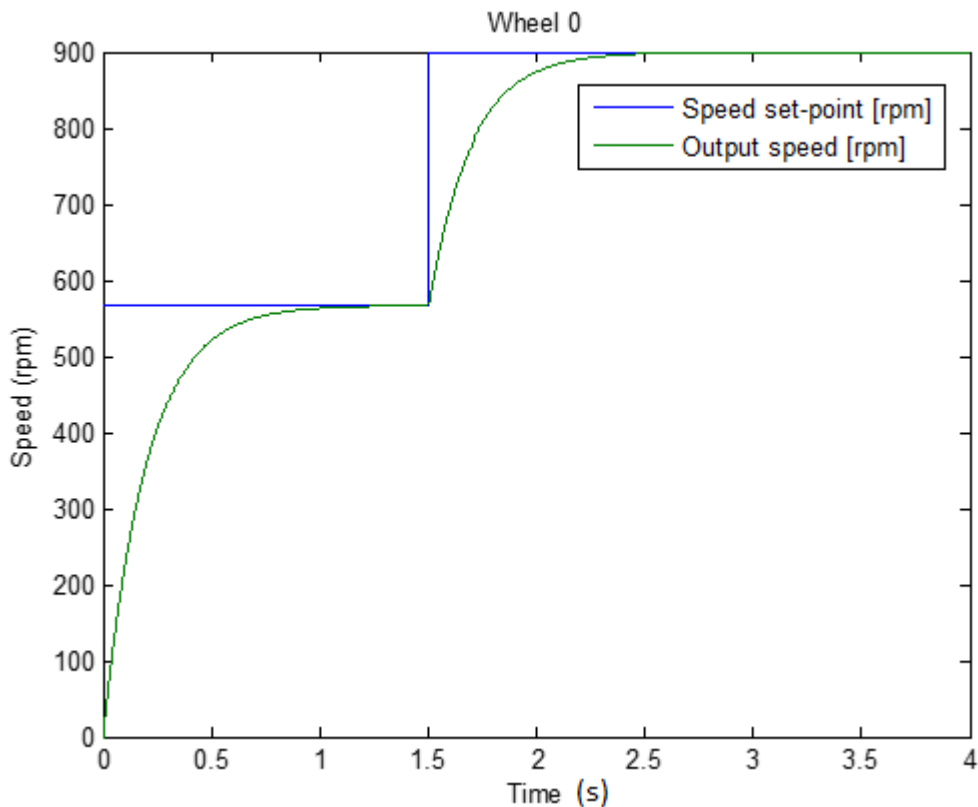


Illustration 32. Graph of the controlled output and the input speed for the wheel 0

The step comes with a signal of 567 rpm and it ends in a permanent regime of 900 rpm for the wheel 0.

The simulation model provides a response without overshoot which means that our model is correct. But according to the specifications of the Robotino there are certain conversions to be performed on the parameters of Kp and Ki controller.

First, the units of the control signal are revolutions per minute (rpm) while the variable that reads the EA09 board has their own units, this relationship is as follows:

$$27 \text{ rpm} = 1 \text{ controller unit}$$

Second, the instruction of Matlab introduces the values of the PI controller of Robotino but it needs the following conversion:

$$kp = kp' \cdot 0.01$$

$$ki = ki' \cdot 0.001$$

$kp$  and  $ki$  are the values of the controller without the conversion.

$kp'$  and  $ki'$  are the values that we put into the controller of the Robotino.

These conversions are extracted from the website of the bibliography [7].

I proceeded then to make these changes on the simulation in the driver but adding a gain of  $1/27$  in the model of Simulink to simulate the conversion that will do the EA09 board in the reality.

#### PI Wheel 0:

We apply the conversion in order to obtain the new PI parameters to identify each controller for each wheel:

$$kp'0 = kp0 \cdot \frac{27}{0.01} = 31.89633 \cdot 10^{-3} \cdot \frac{27}{0.01} = 86.12 \quad (6.5. a)$$

$$ki'0 = ki0 \cdot \frac{27}{0.001} = 2.9942 \cdot 10^{-4} \cdot \frac{27}{0.001} = 8.08435 \quad (6.5. b)$$

The new plant for simulation in Simulink in the illustration 33:

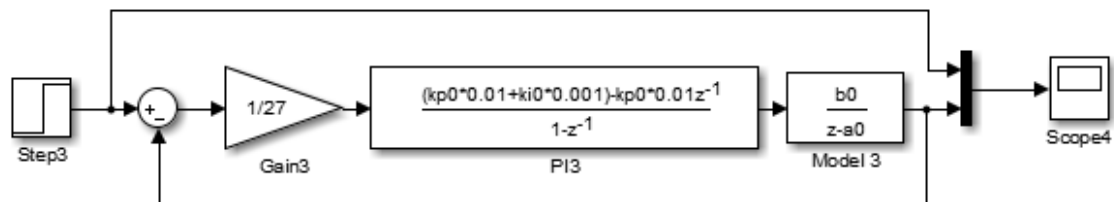


Illustration 33. Plant used in Simulink to simulate the response



The response of the output in the illustration 34:

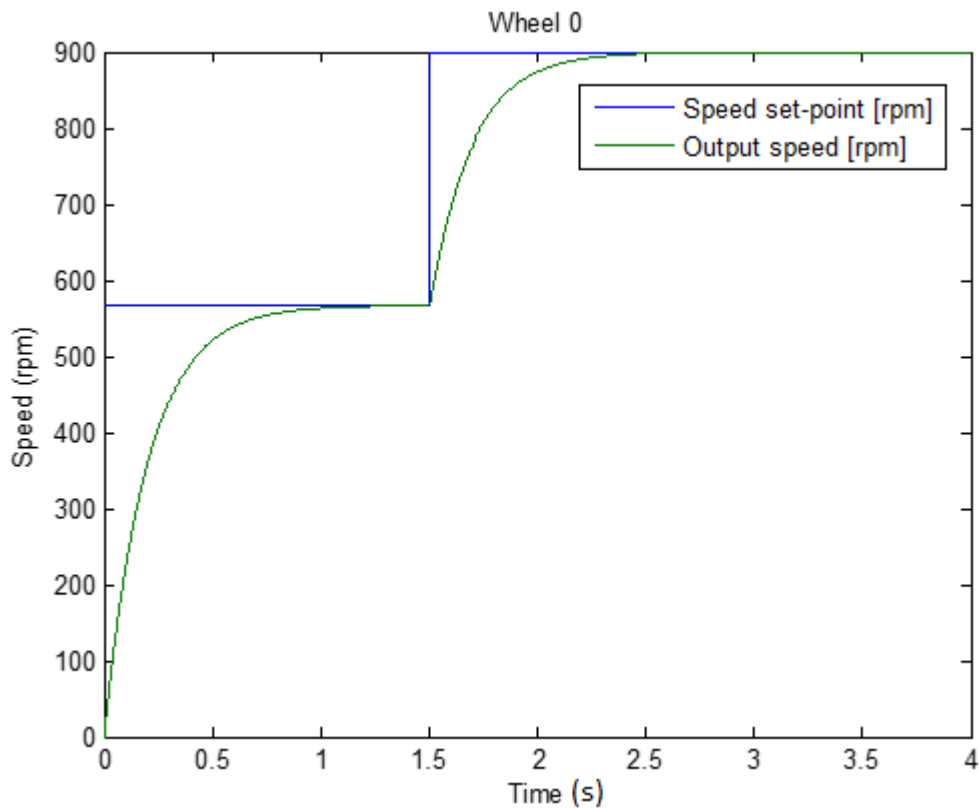


Illustration 34. Graph of the controlled output and the input speed

The simulation response is exactly the same after the conversion. This changes that I do in the plant Simulink are correct to implement these parameters in to the real robot. Perhaps this seems a not necessary step. But an error was detected in the calculation of PI controller with this testing.

Results for each wheel in the table 11 and table 12:

Model for the wheel 0	Model for the wheel 1	Model for the wheel 2
$a_0 = 0.9907$	$a_1 = 0.9893$	$a_2 = 0.9879$
$b_0 = 0.1553$	$b_1 = 0.1596$	$b_2 = 0.1702$

Table 11. Parameters of the first-order model for each wheel

Controller parameters for the wheel 0 (PI)	Controller parameters for the wheel 1 (PI)	Controller parameters for the wheel 2 (PI)
$kp'_0 = 86.12$	$kp'_1 = 83.681391$	$kp'_2 = 78.3587$
$ki'_0 = 8.0843528$	$ki'_1 = 9.05075$	$ki'_2 = 9.5975$

Table 12. Parameters of the PI controller after the conversion

### 3.6 Methodology and of Simulink model to validate the controller

I used the Simulink tool to find the error between the model and the experimental data of the speed. The same scheme serves to repeat with the wheel 1 and 2, but we have to change the data of 'from workspace' for the corresponding data for each wheel (actual\_speed, control\_value). The scheme proceeds as follows in the illustration 35:

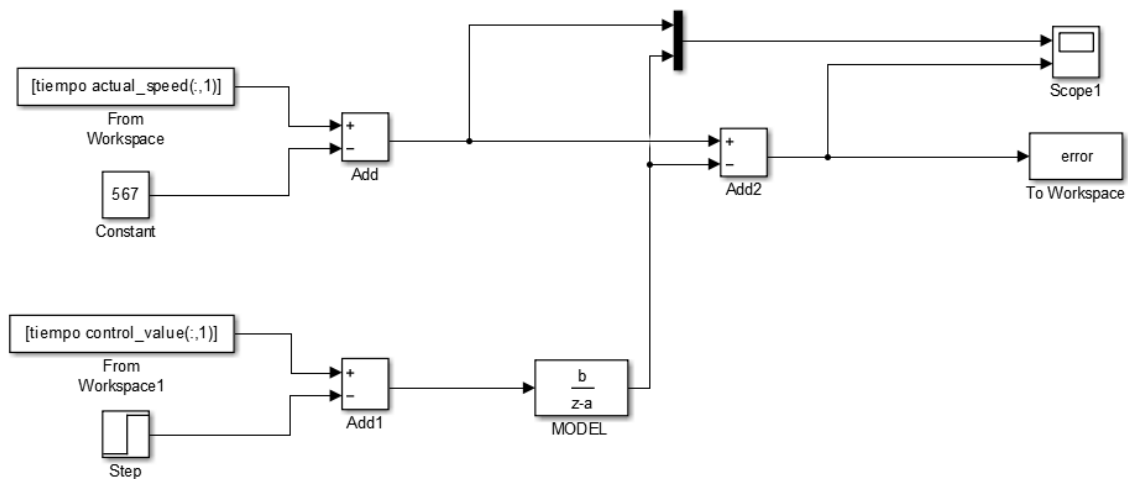


Illustration 35. Simulink scheme to find the error

The value of the block **"Constant"** of the Simulink model is the average value of the speed that we have used to change the zero reference (Table 13).

Constant for the wheel 0	Constant for the wheel 1	Constant for the wheel 2
Constant = 567	Constant = 675	Constant = 560

Table 13. Parameters for the simulation (Input step for each wheel)

The value of the block **"Step"** of the plant of Simulink is the average value of the sign of control that in case of all wheels was **70**.

The values of 'a' and 'b' for this plant was the values that we found with the approximation to 0.998. So that the values for each wheel are the following table (Table 14):

Model for the wheel 0	Model for the wheel 1	Model for the wheel 2
a0 = 0.9907	a1 = 0.9893	a2 = 0.9879
b0 = 0.1553	b1 = 0.1596	b2 = 0.1702

Table 14. Parameters of the first-order model for each wheel

Results graphics for the wheel 0 (Illustration 36):

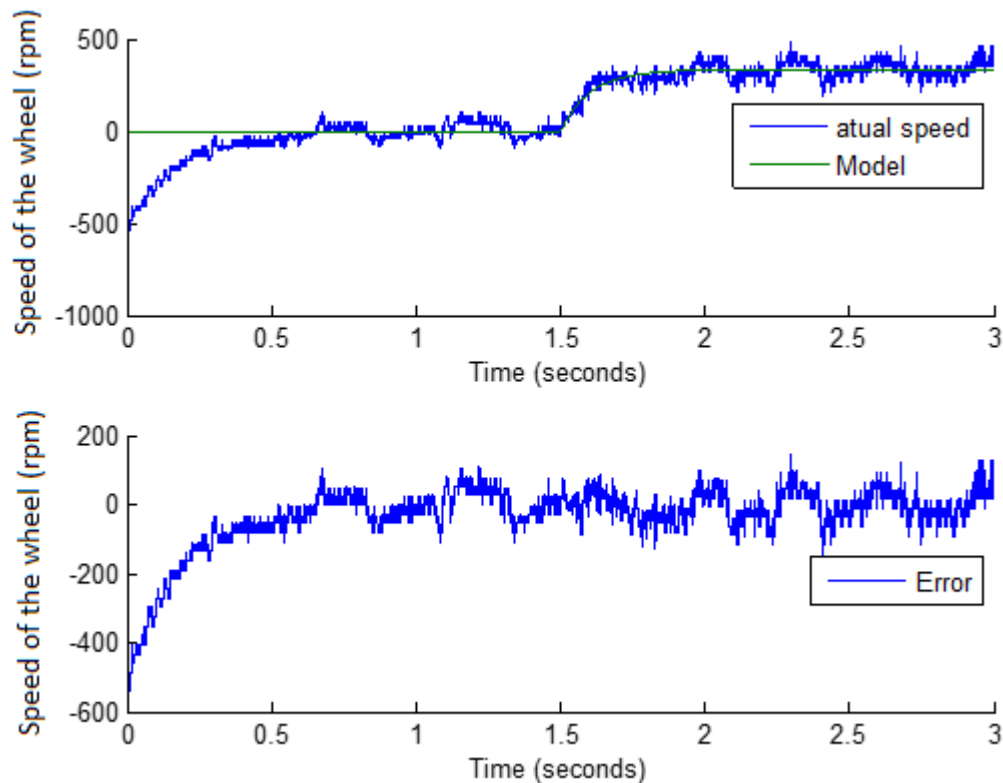


Illustration 36. Above figure: Compare actual speed with the theoretical model  
Below figure: The error of the comparison

I get the error that gets in the permanent regime of captured data to implement the error in our plant of simulation (Last 1000 points).

The Simulink model for the test in the illustration 37:

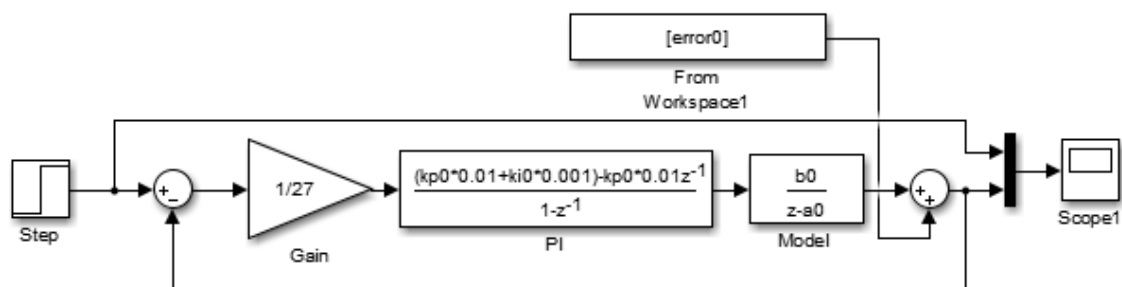


Illustration 37. Plant used in Simulink to simulate the response with the error

This is how response the Simulink model of the illustration 37:

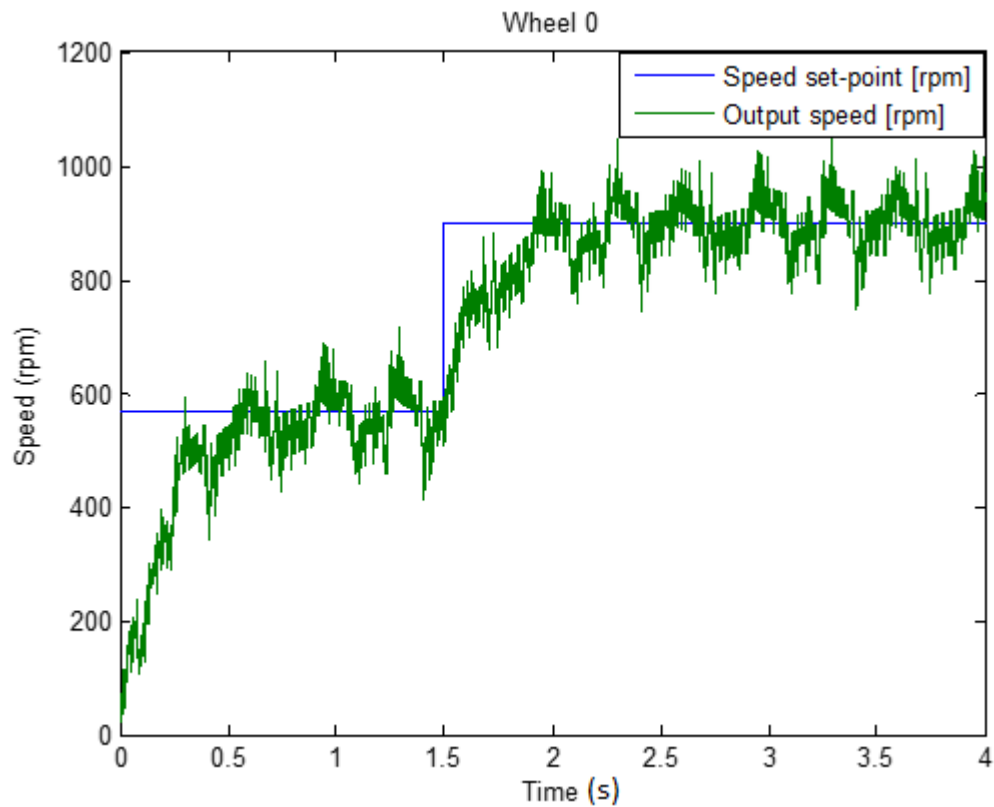


Illustration 38. Simulink plant response with the error

### 3.7 Study of the Error

This is the procedure:

We have to make the pole nearest to 1 fails to mitigate the error. Therefore, we studied the error to corroborate if the noise can be filtered or not.

The error of the wheel 0:

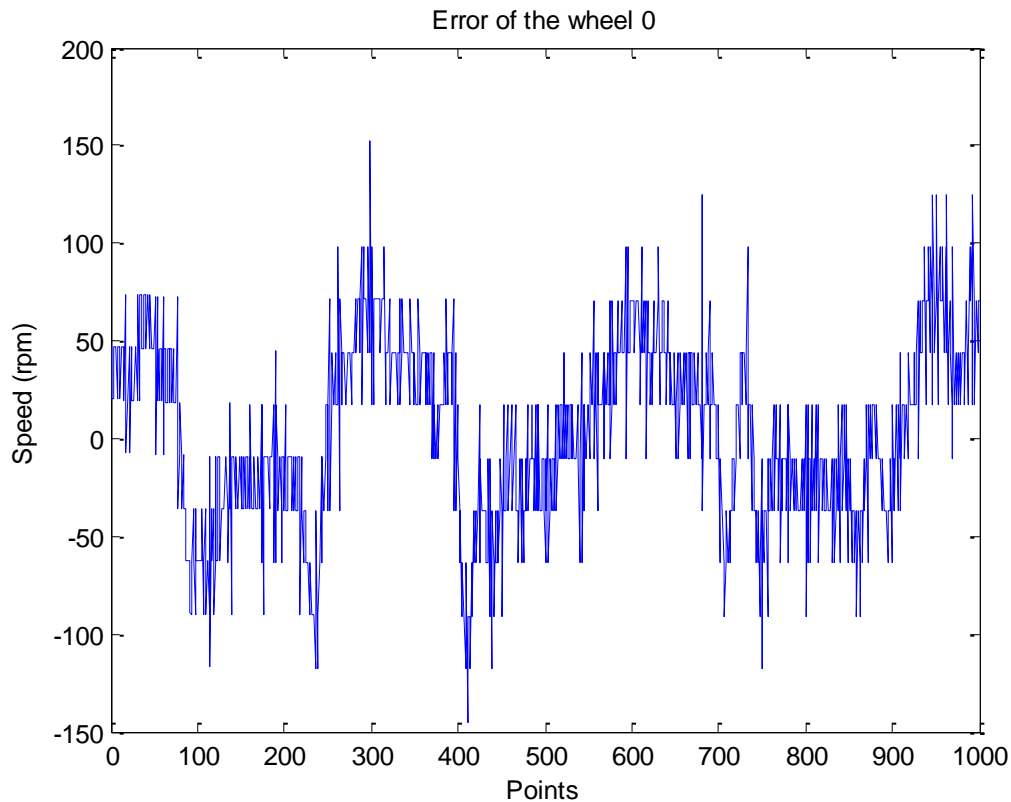


Illustration 39. Error for the permanent region

The error signal has a sinusoidal shape that it repeated approximately every 300 points. We use the Fourier transform to express the temporal frequency domain and the sinusoidal harmonics which compose it.

The problem happens when the signal is decomposed in harmonics, we were not sure if that signal includes integer multiples of each harmonic that decompose it. In order to solve this problem we proceed to work with a window.

A window is used to limit the length of a real signal captured. The basic idea of the process of window, it is extracts the average value of the signal and delimitates the window as broad as the number of points of the signal, then it multiplies each point of the signal without the average for the corresponding coefficient of the window and it finally performs Fourier transform.

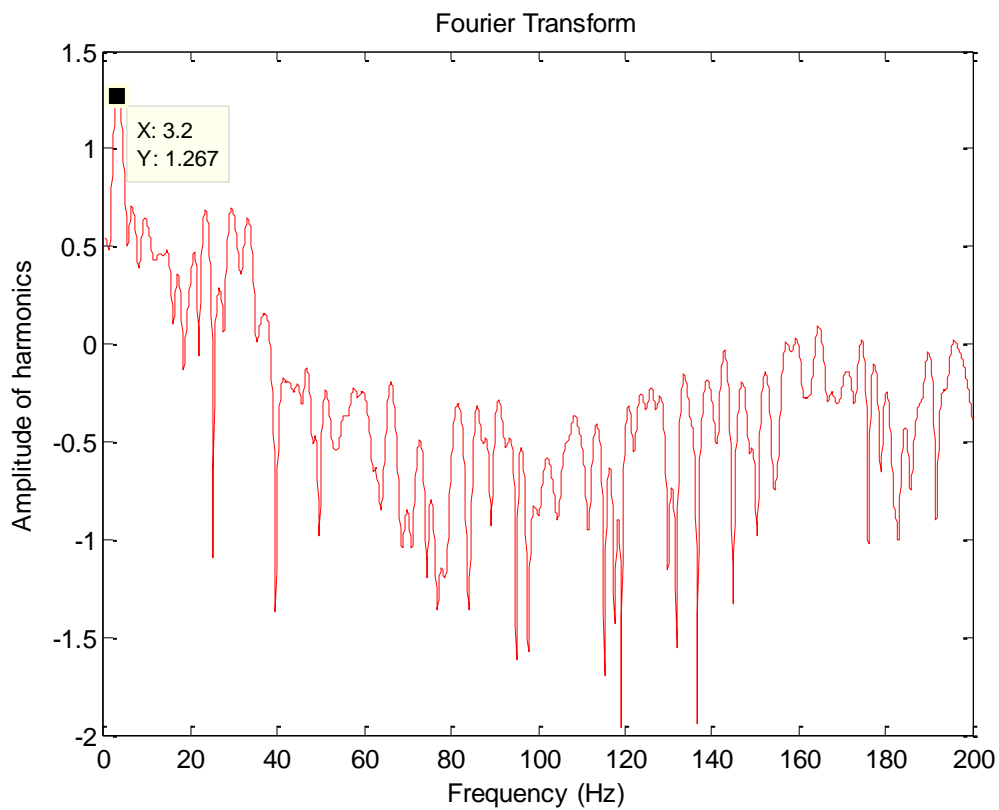


Illustration 40. Representation of the Fourier transform

We saw that in the illustration 40 there is a fundamental harmonic of 3.2 Hz. The noise is constant across the spectrum, so we can say that it was a white noise. The noise will not be filtered in order to be attenuated because the signal will be so low that robot will remain immobile.

## 4. KINEMATIC MODEL

### *4.1 Introduction*

In this chapter, different kinematic models for different types of robots are exposed. For more information see [9] [10] of the bibliography.

Dynamic Model: The study of motion which models the forces.

- Includes the energies and speeds associated with these motions (Gravity, etc..).

Kinematic Model: The study of the mathematics of motion without considering the forces that affect the motion.

- It deals with the geometric relationships that govern the system
- It deals with the relationship between control parameters and the behaviour of a system in the space.

*Notation:*

Idealized model of wheel:

- If the wheels are free to rotate about its axis (x axis), the robot exhibits preferential rolling motion in one direction (y axis) and a certain amount of lateral slip as it shows the illustration 41.

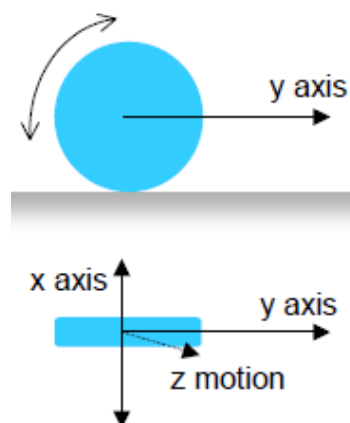


Illustration 41. Theoretical representation of the axis

Robot and World reference in illustration 42:

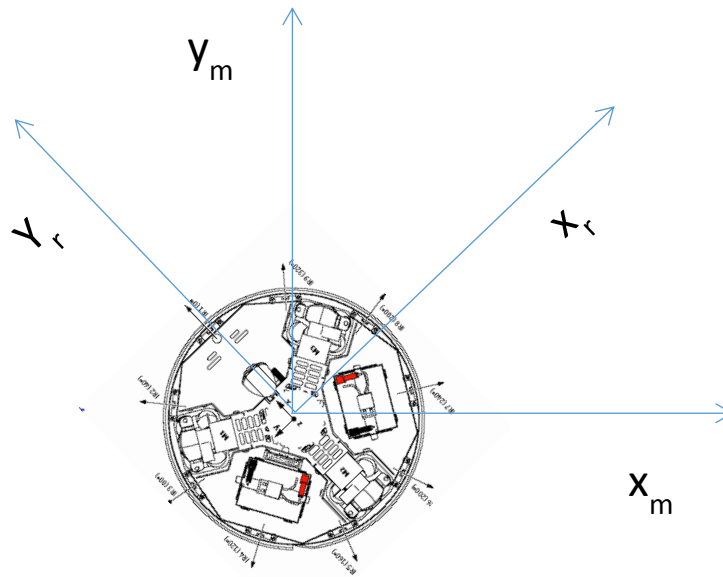


Illustration 42. Scheme representation of the axis

- $\{X_r, Y_r\}$  – robot reference
- $\{X_m, Y_m\}$  – world reference

Kinematic Model in robot reference

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix}$$

Rotation matrix expressing the orientation of the world reference with respect to the robot reference

$$\begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) & 0 \\ \sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



## 4.2 Kinematic Model for each locomotion type:

### 4.2.1 Differential Drive

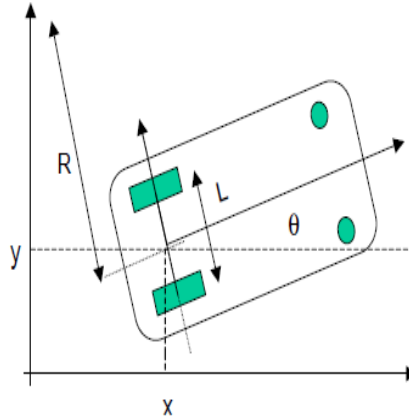


Illustration 43. Theoretical representation of the axis for differential drive

$V_r(t)$ : Linear speed of right wheel.

$V_l(t)$ : Linear speed of left wheel.

$r$ : Nominal radius of each wheel.

$R$ : Instantaneous curvature radius of the robot trajectory, relative to the mid-point axis.

$R - \frac{L}{2}$ : Curvature radius of trajectory described by left wheel.

$R + \frac{L}{2}$ : Curvature radius of trajectory described by right wheel.

$$\begin{aligned} w(t) &= \frac{V_r}{R + \frac{L}{2}} \\ w(t) &= \frac{V_l}{R - \frac{L}{2}} \end{aligned} \rightarrow \begin{aligned} w(t) &= \frac{V_r(t) - V_l(t)}{L} \\ R &= \frac{L}{2} \cdot \frac{(V_l(t) + V_r(t))}{(V_l(t) - V_r(t))} \end{aligned} \rightarrow v(t) = w(t) \cdot R = \frac{1}{2}(V_r(t) + V_l(t))$$

(6.6)

Kinematic Model in robot reference:

$$\begin{bmatrix} V_x(t) \\ V_y(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r-L \end{bmatrix} \cdot \begin{bmatrix} w_l(t) \\ w_r(t) \end{bmatrix}$$

(6.7)

Kinematic Model in world reference:

$$v(t) = w(t) \cdot R = \frac{1}{2} (V_r(t) + V_l(t)) \quad (6.8. a)$$

$$w(t) = \frac{V_r(t) - V_l(t)}{L} \quad (6.8. b)$$

↓

$$\dot{x}(t) = v(t) \cdot \cos(\theta) (t) \quad (6.8. c)$$

$$\dot{y}(t) = v(t) \cdot \sin(\theta) (t) \quad (6.8. d)$$

$$\dot{\theta}(t) = w(t) \quad (6.8. e)$$

↓

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta) (t) & 0 \\ \sin(\theta) (t) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v(t) \\ w(t) \end{bmatrix} \quad (6.8. f)$$

#### 4.2.2 Synchronous drive

- In a synchronous drive robot (synchro drive), each wheel is capable of being driven and steered.
- Typical configurations:
  - Three steered wheels arranged as vertices of an equilateral triangle by a cylindrical platform.
  - All the wheels turn and drive it in unison.

Steered wheel dispose in the illustration 44:

- The orientation of the rotation axis can be controlled

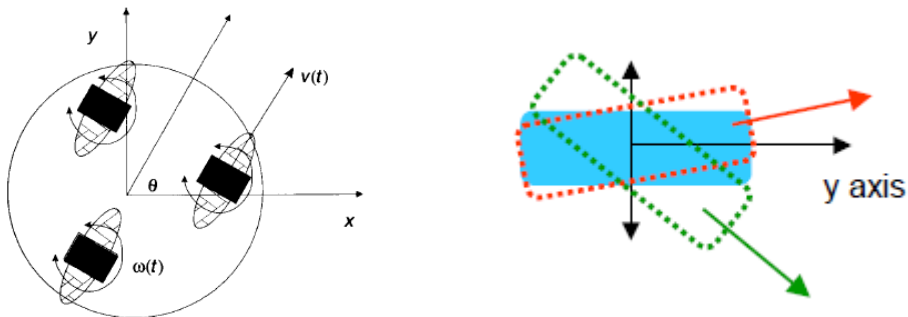


Illustration 44. Theoretical representation of the axis for steered wheel

- The three wheels point in the same direction and turns at the same rate.
- The vehicle controls the direction of the wheels and their speed.
- Because all wheels remain parallel, the synchro drive rotates always about the center of the robot.
- The synchro drive robot has the ability to control the orientation  $\theta$ .

$$x(t) = \int_0^t v(\sigma) \cdot \cos(\theta(\sigma)) d\sigma \quad (6.9.a)$$

$$y(t) = \int_0^t v(\sigma) \cdot \sin(\theta(\sigma)) d\sigma \quad (6.9.b)$$

$$\theta(t) = \int_0^t w(\sigma) d\sigma \quad (6.9.c)$$

### 4.2.3 Tricycle

- It has three wheels and two odometers on the two rear wheels (Illustration 45).
- Steering and power are provided through the front wheel.

Control variables:

- Steering direction  $\alpha(t)$
- Angular velocity of steering wheel  $w_s(t)$
- $r$  = steering wheel radius

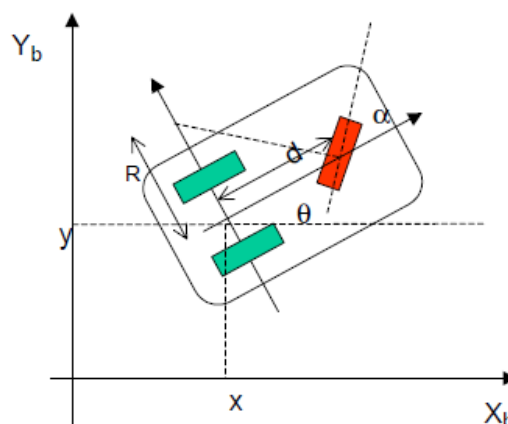


Illustration 45. Theoretical representation of the axis tricycle

$$V_s(t) = w_s(t) \cdot r \quad 6.10. a$$

$$R(t) = d \cdot tg\left(\frac{\pi}{2} - \alpha(t)\right) \quad 6.10. b$$

$$w(t) = \frac{w_s(t) \cdot r}{\sqrt{d^2 + R(t)^2}} \quad 6.10. c$$

$$w(t) = \frac{v_s(t)}{d} \cdot \sin(\alpha) (t) \quad 6.10. d$$

Kinematic model in robot reference:

$$v_x(t) = v_s(t) \cdot \cos(\alpha) (t) \quad 6.11a$$

$$v_y(t) = 0 \quad 6.11. b$$

$$\dot{\theta}(t) = \frac{v_s(t)}{d} \cdot \sin(\alpha) (t) \quad 6.11. c$$

Kinematic model in world reference:

$$\dot{x}(t) = v_s(t) \cdot \cos(\alpha) (t) \cdot \cos(\theta) (t) \quad 6.12. a$$

$$\dot{y}(t) = v_s(t) \cdot \cos(\alpha) (t) \cdot \sin(\theta) (t) \quad 6.12. b$$

$$\dot{\theta}(t) = \frac{v_s(t)}{d} \cdot \sin(\alpha) (t) \quad 6.12. c$$

↓

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta) (t) & 0 \\ \sin(\theta) (t) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v(t) \\ w(t) \end{bmatrix} \quad \begin{array}{l} v(t) = v_s(t) \cdot \cos(\alpha) (t) \\ w(t) = \frac{v_s(t)}{d} \cdot \sin(\alpha) (t) \end{array}$$

6.12. d

#### 4.2.4 Omnidirectional wheels

Robotino has three omnidirectional wheels with these constraints as it shows the illustration 46:

- Omnidirectional wheels: No kinematic restrictions on the wheel axis.
- In the perpendicular axis, there is no sliding: there is a univocal relationship between the angular speed and the speed of the chassis.
- Robot reference system has an independent relationship with the time.

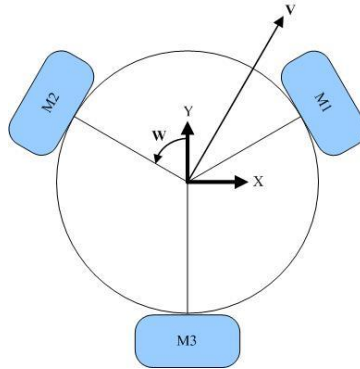


Illustration 46. Theoretical representation of the omnidirectional wheels

Kinematic equation of the omnidirectional wheels:

$$r \cdot w(t) = -v_x(t) \cdot \sin(\delta) + v_y(t) \cdot \cos(\delta) + R \cdot \dot{\theta} \quad (6.13.a)$$

Kinematic matrix for each wheel from the Kinematic equation in robot reference:

$$\begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{r} \cdot \begin{bmatrix} -\sin(\delta_0) & \cos(\delta_0) & R \\ -\sin(\delta_1) & \cos(\delta_1) & R \\ -\sin(\delta_2) & \cos(\delta_2) & R \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} \quad (6.13.b)$$

$$\delta_0 = 150^\circ \quad \delta_1 = 270^\circ \quad \delta_2 = 30^\circ \quad R = 0.15 \quad r = 0.05$$

$$\begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{0.05} \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} \quad (6.13.c)$$

This equation is used to find the speed of the robot based on the world speed reference, but what we want to find is the speed and the position of the robot based of the setpoint speed which is the inverse of the kinematic matrix that we calculated.

The equation after the inverse:

$$\begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} = 0.05 \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} \quad (6.13.d)$$

Kinematic model in world reference:

$$\begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\theta}_m \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) & 0 \\ \sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot 0.05 \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} \quad (6.13.e)$$

The implementation of this equation using Simulink tool allows the simulation of the behaviour in function of the input step that we introduce on wheels.

### 4.3 Simulation of the kinematic model on world reference

The illustration 47 is the application of the equation 25. Matrix of the Kinematic Model for Omnidirectional wheels in world reference. The dynamic model designed in chapter (the first order models for each wheel and the PI controller) is included in the model.

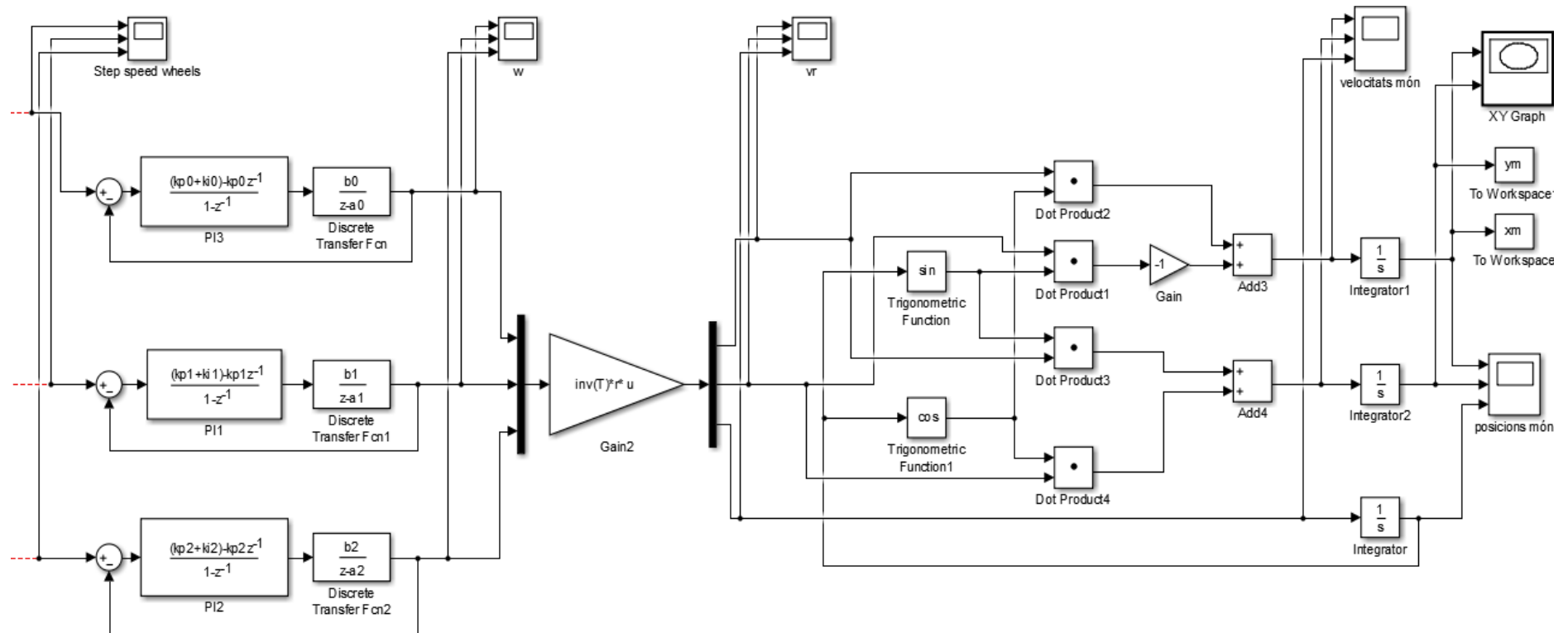


Illustration 47. General scheme of simulation

#### 4.3.1 Breakdown of each part of the general scheme:

The first part of the general scheme is setpoint input. We use the Simulink block "Step" to enter setpoint speed at every time. We use the block "add" to add all steps that we need for each wheel in one signal. The illustration 50 is just a representation of how it looks the Simulink once we introduces several "Steps". Note that the red arrow from this setpoint input (illustration 48) is connected to the red arrow to the left of the illustration 47.

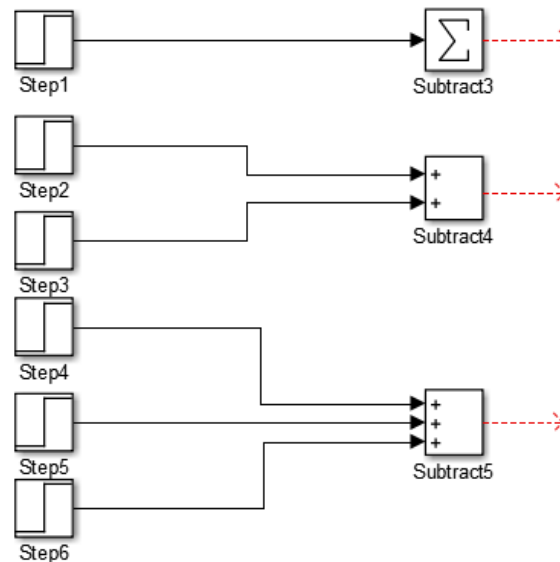


Illustration 48. Scheme of the input steps

The second part is a discrete model of the wheels from chapter 4 which also includes their controller PI as it shows in the illustration 49:

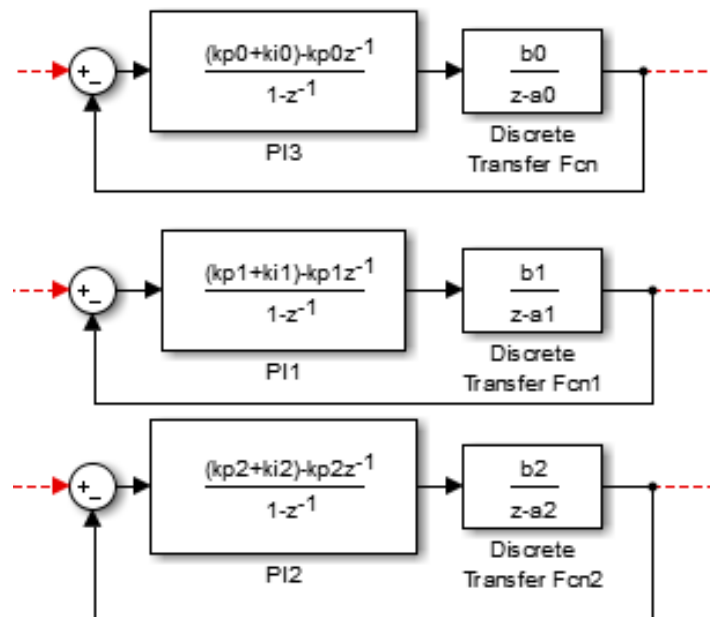
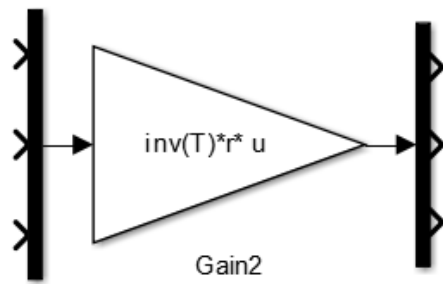


Illustration 49. Scheme of the model reference

We have the angular speed of each wheel once overcomes this part.



In the third part (Illustration 50), there is a matrix of the direct kinematic model that calculates the speed of the robot from the wheel speeds in the robot reference .



T is kinematic model robot reference:

$$T = \begin{bmatrix} -\sin(\delta_0) & \cos(\delta_0) & R \\ -\sin(\delta_1) & \cos(\delta_1) & R \\ -\sin(\delta_2) & \cos(\delta_2) & R \end{bmatrix}$$

Illustration 50. Scheme of the matrix kinematic model in the robot reference

Finally, the rotation matrix (Illustration 51)to move from robot reference into world reference:

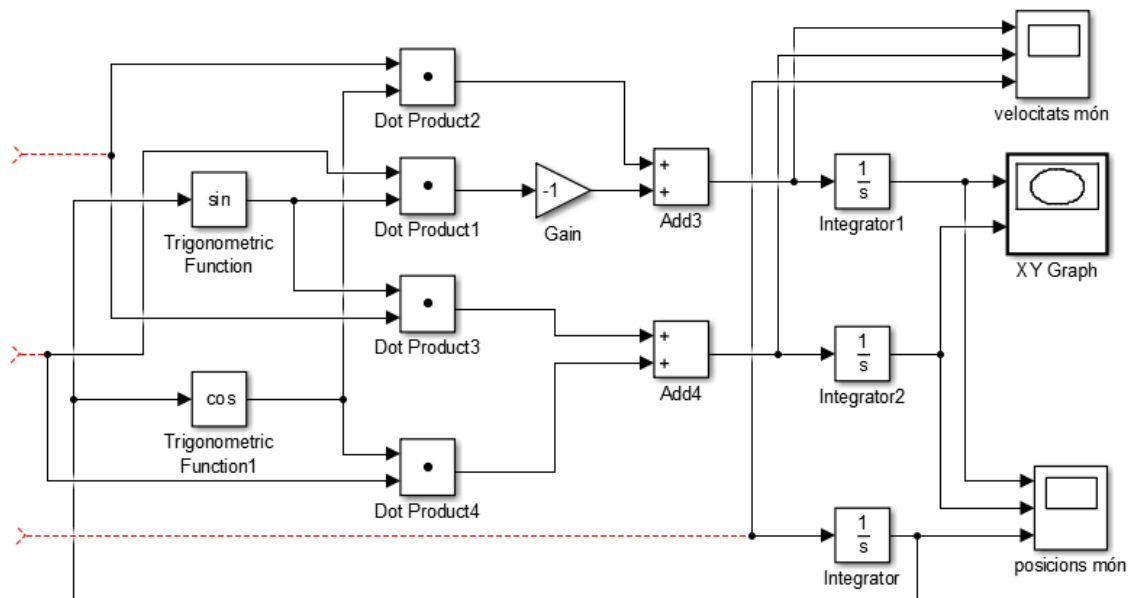


Illustration 51. Outputs of the simulation:

The outputs of this system are the speed on XY components. If we integrate these results we have the position of robot in XY components which allows to visualize the robot path throughout the simulation. The three red arrows that we see on the illustration 51 are connected to the outputs of the block "demux" (Illustration 50). It is interesting the highlight that no integration has sense in the robot reference as it moves with the robot.

It is recommended put every part inside a subsystem block to make a better file of Simulink which is easy to work with and easy to understand Like the illustration 52.

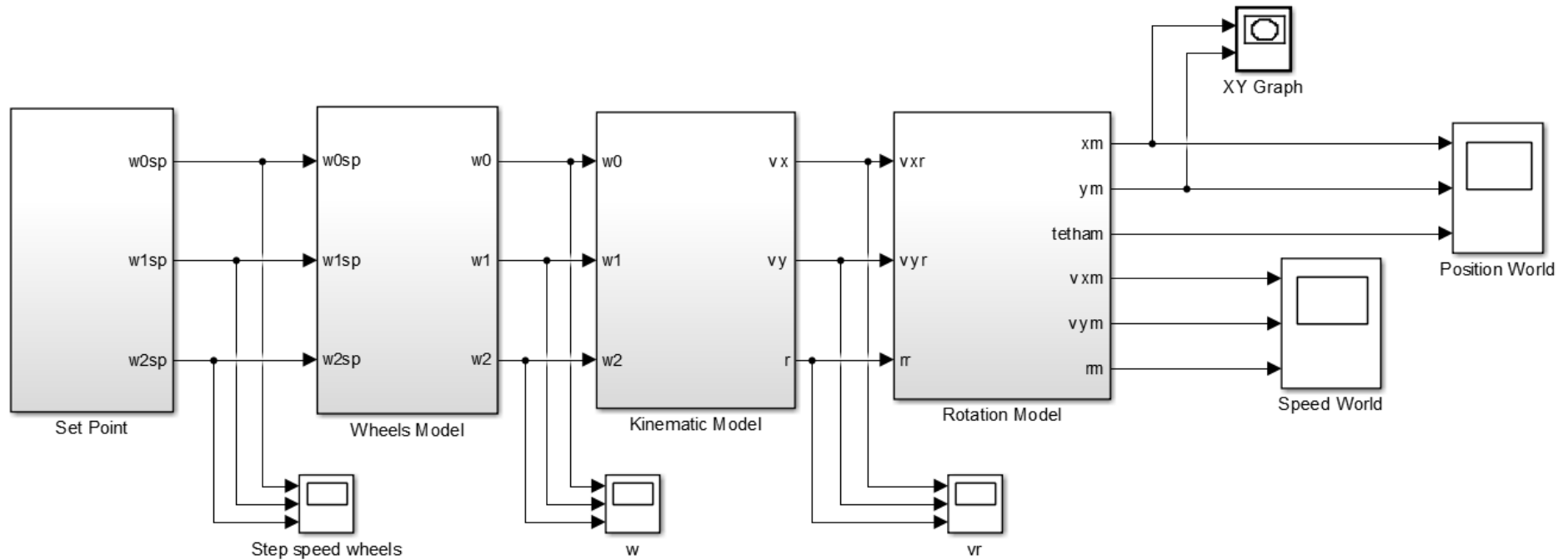


Illustration 52. General scheme structured in subsystems

## 4.4 Practical Cases

### 4.4.1 Case 1

The kinetic model that relates the wheel speeds and considering that the radius of the wheel is  $r = 0.05\text{m}$  and the chassis  $R = 0.15$ , is as follows:

$$w = \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{0.05} \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix}$$

The Robotino is following a straight line with a speed of  $V = 0.2\text{m} / \text{s}$  and detects a crossing to 40cm (calculated over their center). Then enters in open loop control mode to turn on itself just on the crossing.

Calculate the setpoints speed of the wheel to reach this point and turn right (East).

*Resolution of the problem 1:*

Introduce a signal (-3.46) for the wheel 0, (3.46) for the wheel 2 and (0) for the wheel 1. After 2 seconds reaches 40cm and send a contrary signal to the wheel 0 and 2, (3.46) and (-3.46), respectively.

The simulation stops right at the point and rotates, apply the same signal (-4.71) to the three wheels for 1 second. Then the robot returns to the same signal as the first time, (-3.46) for the wheel 0, (3.46) and for the wheel 2 (0) for the wheel 1.

If we want to move the robot to up:

$$w_1 = \frac{1}{0.05} \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0.2 \\ 0 \end{bmatrix} = \begin{bmatrix} -3.46 \\ 0 \\ 3.46 \end{bmatrix}$$

If we want to rotate:

$$w_1 = \frac{1}{0.05} \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ -1.57 \end{bmatrix} = \begin{bmatrix} -4.71 \\ -4.71 \\ -4.71 \end{bmatrix}$$

Steps to introduce in the setpoint input with the Simulink block (Step):

Wheel 0:

Step Time	0	2	2.5	3.5	4
Set point	- 3.46	3.46	- 4.71	4.71	- 3.46

Wheel 1:

Step Time	2.5	3.5
Set point	- 4.71	4.71

Wheel 2:

Step Time	0	2	2.5	3.5	4
Set point	3.46	- 3.46	- 4.71	4.71	3.46

Response of the simulation:

This is the representation of the behaviour of a real robot with wheels model that have found in world reference:

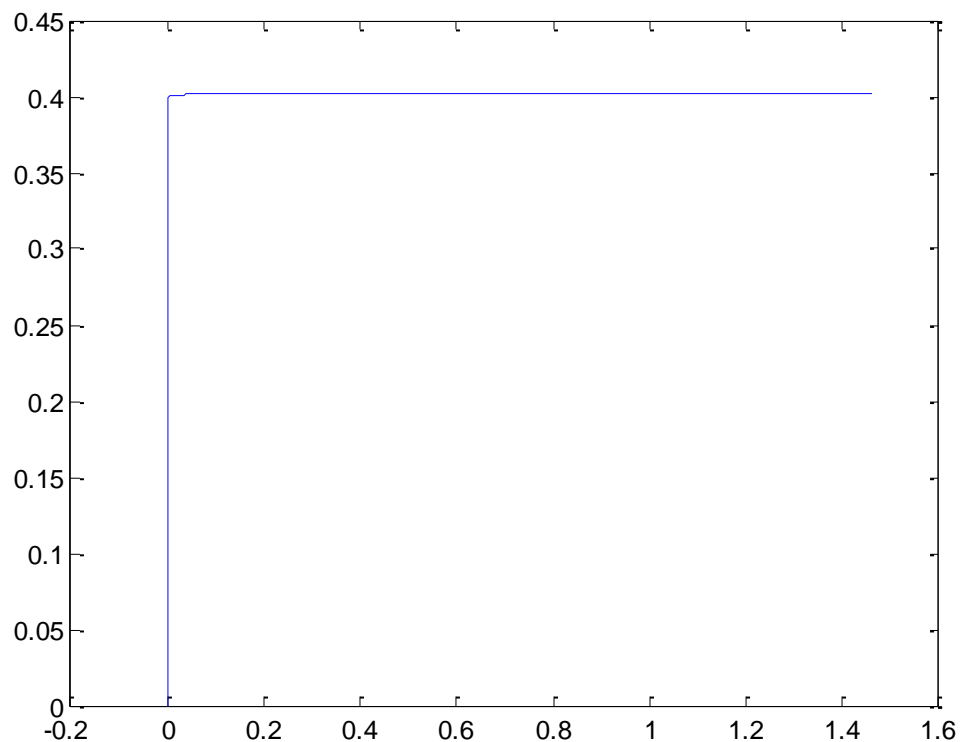


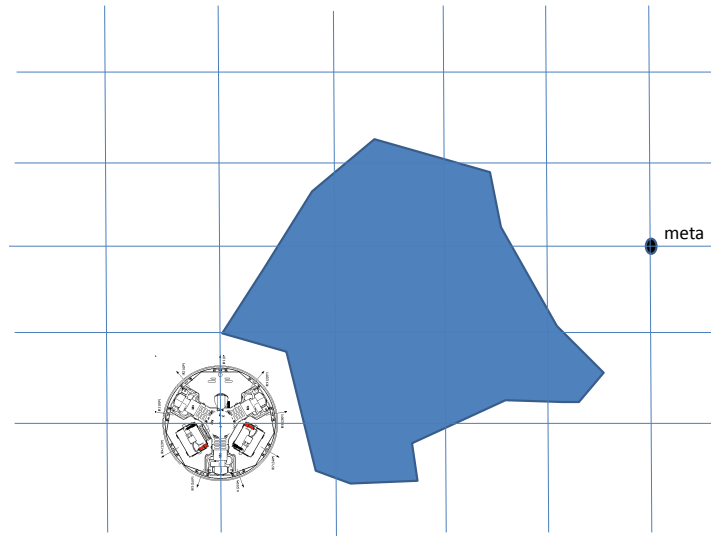
Illustration 53. Response of the simulation for the case 1 (*units are in meters*)

#### 4.4.2 Case 2

Given the kinematic model in the robot system reference:

$$w = \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{0.05} \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix}$$

We want the Robotino to go to the goal point of the scheme Figure 1 where the squares are 30cm side.



**Fig. 1: Scheme of the environment**

- Find the wheel speed of the Robotino to do this path and stops. Give these signal in function of time (taking as  $u(t)$  a unitary step)

*Resolution of the problem 2:*

As case 2 does not establish a predetermined speed, I use the same values of case 1. The only difference is that when we want to rotate left the sign is positive.

Steps to introduce in the setpoint input:

Wheel 0:

Step Time	0	1	1.5	3	3.5	4.5	5	11	11.5	12.5
Set point	4.71	- 4.71	- 3.46	3.46	- 4.71	4.71	- 3.46	3.46	- 4.71	4.71

Step Time	13	20.5	21	22	22.5	25.5
Set point	- 3.46	3.46	- 4.71	4.71	- 3.46	3.46

Wheel 1:

Step Time	0	1	3.5	4.5	11.5	12.5	21	22
Set point	4.71	- 4.71	- 4.71	4.71	- 4.71	4.71	- 4.71	4.71

Wheel 2:

Step Time	0	1	1.5	3	3.5	4.5	5	11	11.5	12.5
Set point	4.71	- 4.71	3.46	- 3.46	- 4.71	4.71	3.46	- 3.46	- 4.71	4.71

Step Time	13	20.5	21	22	22.5	25.5
Set point	3.46	- 3.46	- 4.71	4.71	3.46	- 3.46

Simulation of the trajectory of the robot:

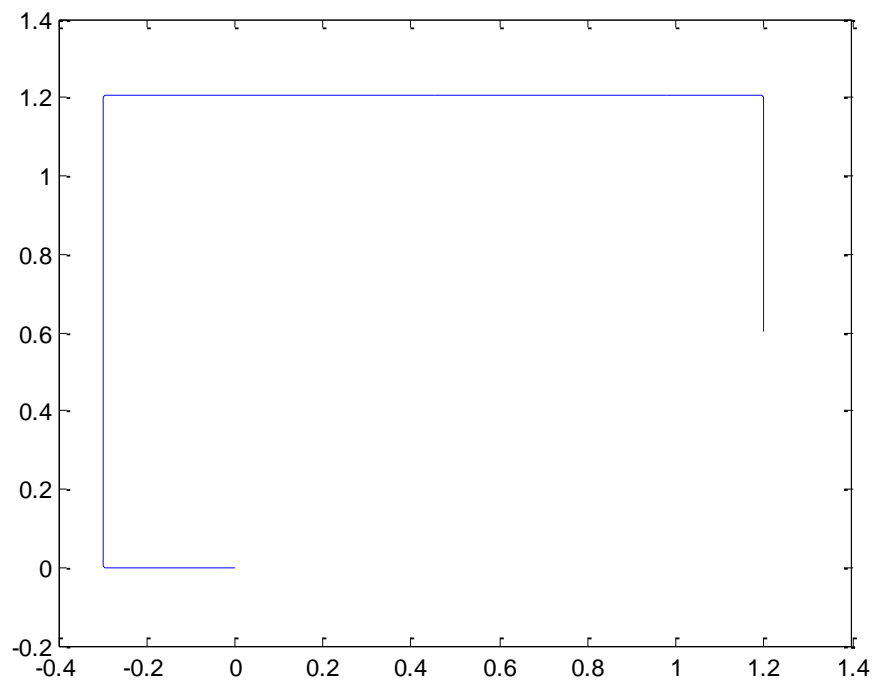
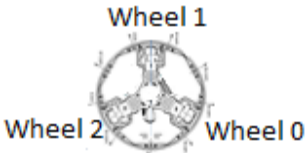
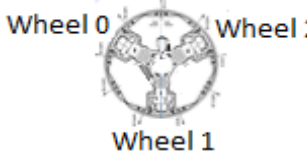


Illustration 54. Response of the simulation for the case 2 (*units are in meters*)

- b. Which orientation will have the Robotino to stop?

Looking to the south

Initial position of the robot	Final position of the robot
	

#### 4.4.3 Case 3

The Robotino has in front of it an object 10 meters away. We want to take photos of the object from all angles. Given the kinematic model of the robot in the robot system reference:

$$w = \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{0.05} \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix}$$

- 3.1 Find the speeds of the wheels which will move Robotino to 1 meter from the object and stops it. Give these signal in function of time (taking as  $u(t)$  a unitary step)

As case 3 does not establish a predetermined speed, I use the same speed calculated values for case 1 and 2.

Steps to introduce in the set point input:

Wheel 0:

Step Time	0	45
Set point	- 3.46	3.46

Wheel 1:

Step Time	0
Set point	0

Wheel 2:

Step Time	0	45
Set point	3.46	- 3.46

Representation of how will response the robot:

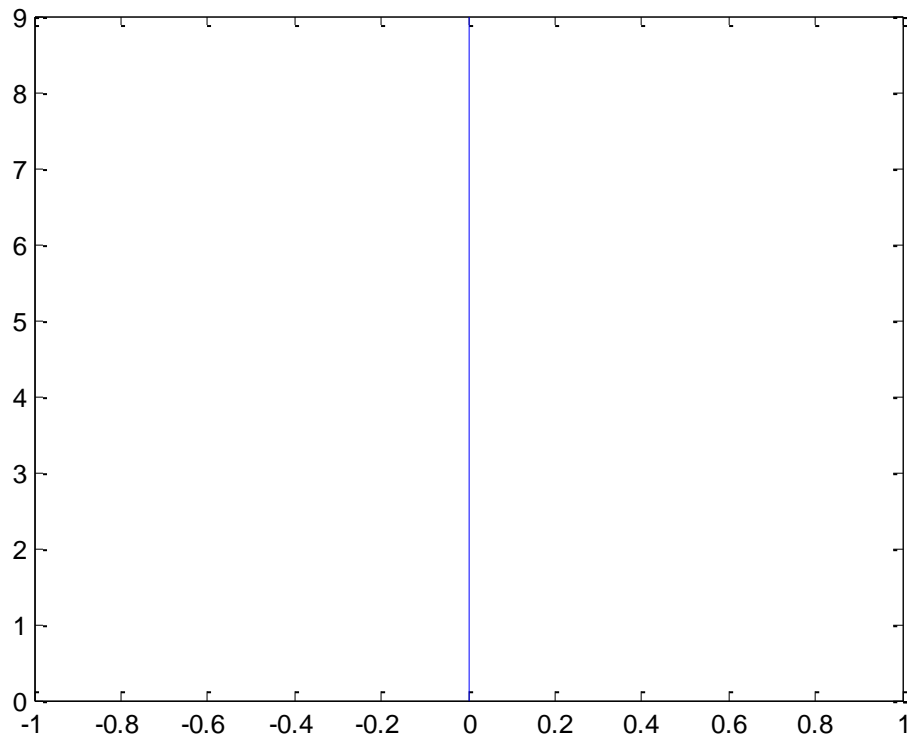


Illustration 55. Response of the simulation for the case 3.1 (*units are in meters*)

3.2 Find the wheel speed that turns the robot 360° around the object at 1 meter away. Remember that the angular velocity of circular motion ( $w$ ), the tangential velocity ( $V_y$ ) and radius ( $r$ ) are related by  $w \cdot r = V_y$

$$w = \frac{V_y}{r} = \frac{0.2}{1} = 0.2$$

We assume that the robot starts at a distance of 1 meter from the object because it has already travelled 9 meters in the last paragraph “3.1”.

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \frac{1}{0.05} \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \cdot \begin{bmatrix} 0.2 \\ 0 \\ 0.2 \end{bmatrix} = \begin{bmatrix} -1.4 \\ 4.6 \\ -1.4 \end{bmatrix}$$



Steps to introduce in the setpoint input:

Wheel 0:

Step Time	0
Set point	- 1.4

Wheel 1:

Step Time	0
Set point	4.6

Wheel 2:

Step Time	0
Set point	- 1.4

Representation of how the robot behaves:

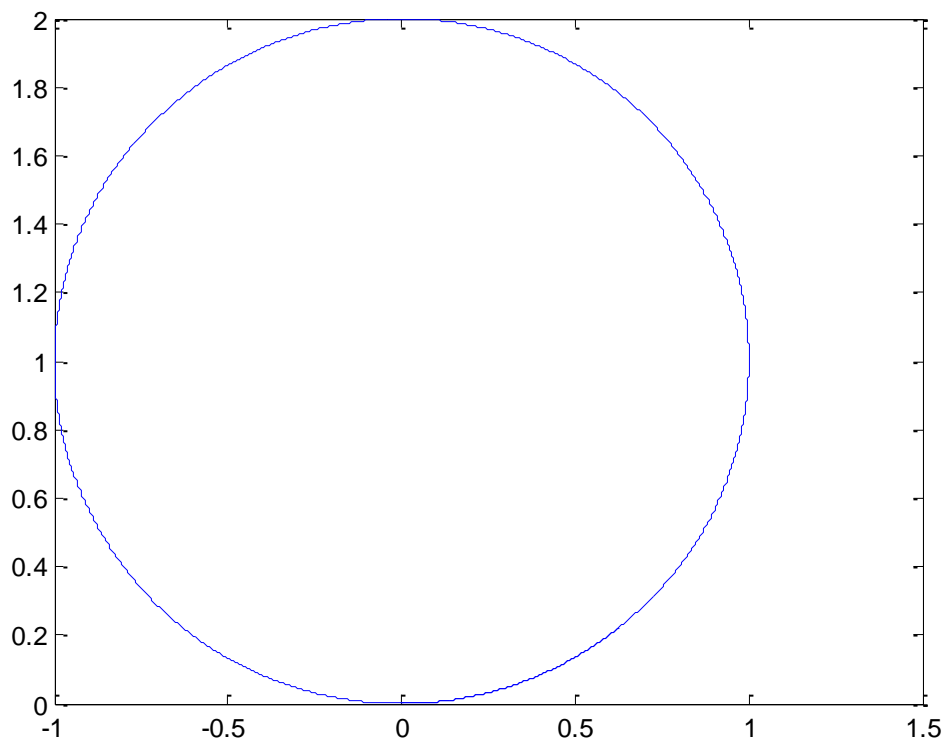


Illustration 56. Response of the simulation for the case 3.2 (*units are in meters*)

## 5. TRAJECTORY CONTROL

### 5.1 Introduction

This chapter presents the trajectory control algorithm and some simulations to show its effectiveness. But now we are going to introduce the trajectory control which allows the robot to follow a linear path.

Path tracking is a tracking algorithm which works by calculating the curvature that will move a vehicle, from its current position to some goal position. The point of the algorithm is to choose a position some distance ahead of the vehicle.

### 5.2 Theory of the trajectory algorithm

Pure pursuit is a geometrical method which determines the curvature that drives the robot to a chosen point on the path (P). The robot and this point form an arc as shown in illustration 57. The chord length of this arc acts as the third constraint in determining a unique arc that joins these two points.

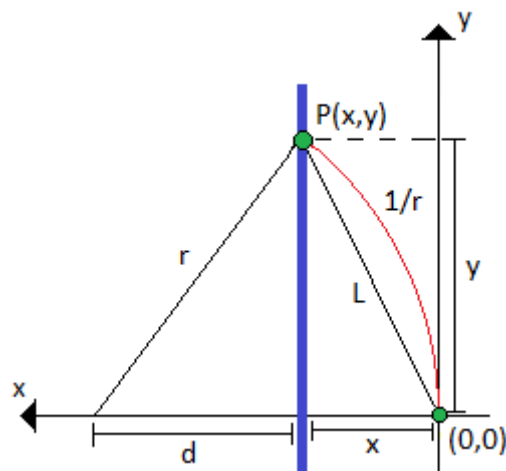


Illustration 57. Theoretical scheme of the robot position when is out of line

The point (P) has to be on the path and the robot is located on the origin point (0,0).

### 5.2.1 Development

Applying the Pythagorean Theorem on illustration 57 and adding up the distances on the x axe. We get the following equations:

$$x^2 + y^2 = L^2 \quad d^2 + y^2 = r^2 \quad x + d = r \quad (7)$$

In order to develop the path control, it is necessary the length of the red arc ( $1/r$ ) which is the path the robot follows to reach the point of the path (P). I calculate the arc radius ( $r$ ) with the above equations and apply the inverse of the result.

Development:

$$d^2 + y^2 = r^2 \quad (7.1.a)$$

$$(r - x)^2 + y^2 = r^2 \quad (7.1.b)$$

$$r^2 - 2 \cdot r \cdot x + x^2 + y^2 = r^2 \quad (7.1.c)$$

$$x^2 + y^2 = 2 \cdot r \cdot x \quad (7.1.d)$$

$$L^2 = 2 \cdot r \cdot x \quad (7.1.e)$$

$$r = \frac{L^2}{2x} \quad (7.1.f)$$

$$\frac{1}{r} = \frac{2}{L^2} \cdot X_L \quad (7.1.g)$$

- Calculation of the distance between the point of the line which the robotino should follow and the imaginary line formed in the direction of Robotino facing forward:

In order to find the distance between the point of the line which should follow the Robotino and the imaginary line formed in the direction of Robotino facing forward ( $X_L$ ), we need the distance “d” that we see on the next illustration (Illustration 58, (7.3.c)). But it is also necessary to find the distance “b” (Illustration 58,(7.3.d)).

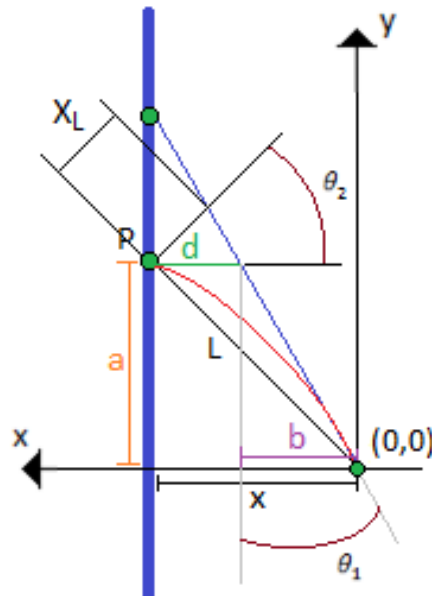


Illustration 58. Scheme of the robot position when is out of line

From the illustration 58:

Applying the Pythagorean Theorem to find "a". The tangent of the angle " $\theta_1$ " to obtain "b". Subtracting " $x - b$ " to obtain "d". Finally, it's applying the cosine of " $\theta_2$ " to obtain  $X_L$

Development:

$$a = \sqrt{L^2 - x^2} \quad (7.2.a)$$

$$b = \sqrt{L^2 - x^2} \cdot \tan(\theta_1) \quad (7.2.b)$$

$$d = x - \sqrt{L^2 - x^2} \cdot \tan(\theta_1) \quad (7.2.c)$$

$$X_L = \left( x - \sqrt{L^2 - x^2} \cdot \tan(\theta_1) \right) \cdot \cos(\theta_2) \quad (7.2.d)$$

$$X_L = \left( x \cdot \cos(\theta_2) - \sqrt{L^2 - x^2} \cdot \sin(\theta_1) \right) \quad (7.2.e)$$

This is the final equation for the trajectory control:

$$\dot{\theta}_r = V \cdot \frac{2}{L^2} \cdot \left( x \cdot \cos(\theta_2) - \sqrt{L^2 - x^2} \cdot \sin(\theta_1) \right) \quad (7.3)$$

Where:

- $V$  is the Speed of the robot in m/s.

### 5.3 Implementation of the trajectory control algorithm

We have to implement equation at the beginning in our model of Simulink if we want to obtain the set point speed as we saw in chapter 4.

$$T = \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = \frac{1}{0.05} \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} \quad (7.4)$$

In the block "Gain" is implemented equation 7.4 of the illustration 59.

We get the setpoint speed of wheel doing th.is conversion which is the reverse process that we did before.

From that model of Simulink we can get  $\dot{x}_r$  and  $\dot{y}_r$  but to get  $\dot{\theta}_r$ , we need to implement the equation of the trajectory control.

As  $\dot{\theta}_r$  is equal to  $\gamma$  from the equation, we show the illustration of the Simulink blocks needed to calculate  $\dot{\theta}_r$ :

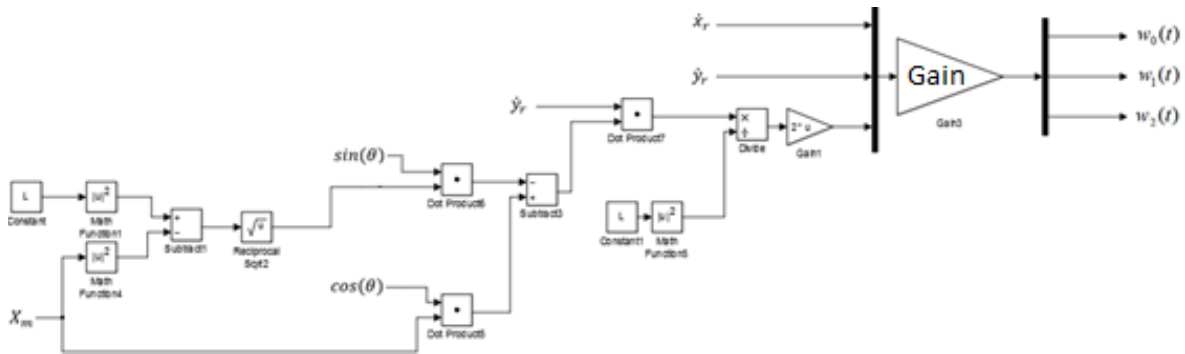


Illustration 59. Scheme of Simulink model to simulate the behaviour when the robot is out of line

This is just one example of how to implement the control path in the following example.

### 5.3.1 Simulation of the trajectory control

If we locate the robot outside of the origin point (0,0), it will be redirected towards the vertical axis.

The initial conditions of the scheme of Illustration 59:

$$\begin{aligned}V_x &= 0 \\V_y &= 0.4 \text{ m/s} \\L &= 0.7 \text{ m}\end{aligned}$$

The blue line represents the position of the robot on world reference:

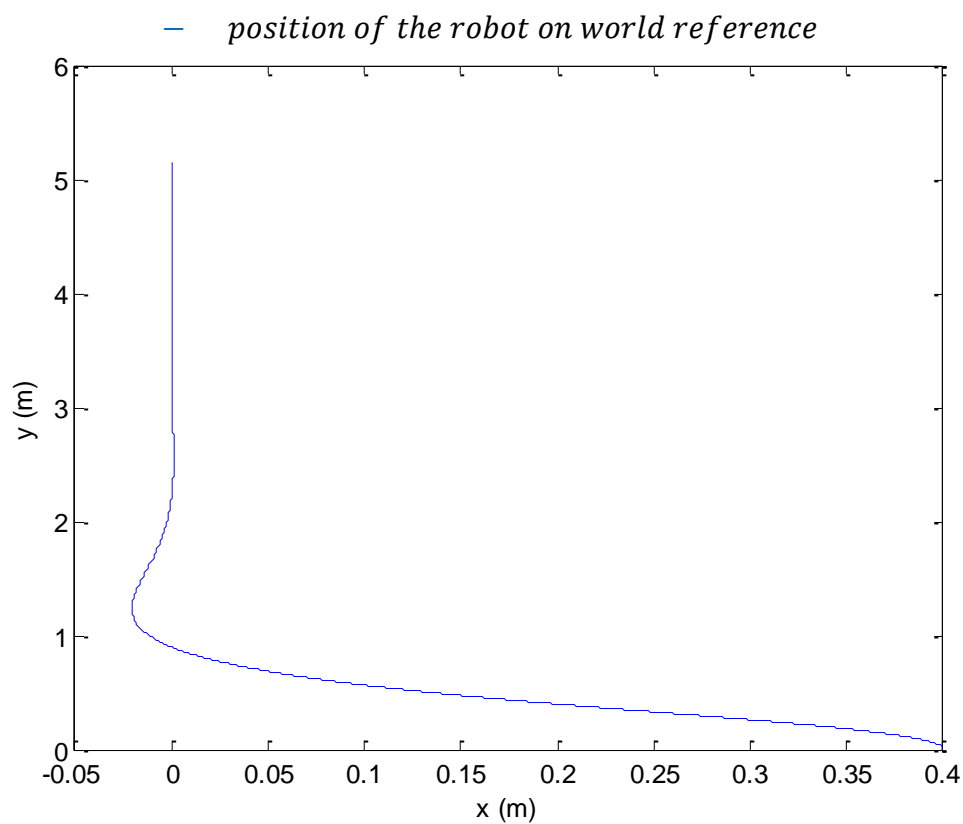


Illustration 60. Robot behaviour simulation for redirection, when placed 0.4 m from point of origin of coordinates

This procedure gives the order to move forward following the line with the trajectory control.

### 5.4 Simulation for any path with trajectory control

So far we studied how to follow a straight trajectory. Now we are interested in following this kind of trajectory in an imaginary grid. This section deals with the problem of switching from a horizontal to a vertical line and viceversa.

We will to study two methods of path tracking:

- Closed loop switching, the robot is continuously moving without stopping.
- Open loop switching, the robot stops when arrives on the rotation point, turn on itself and it follows the new line.

We have to apply a translation and a rotation to coordinates  $x$  and  $y$  in both methods. These new coordinates are in world references on a 2D plane. The robot have to know its position and orientation throughout the simulation and how it changes during that time.

The simulation scheme consists of several parts that have to be explained separately, at the end of this chapter, we will show the scheme and the response for both methods that were mentioned following the same path. Each part of the scheme are separated by subsystem blocks:

#### 5.4.1 Rotation Model

We obtain the speed of the robot in robot coordinates as we saw in the chapter 4 and must apply a transformation to the speeds in order to become in world reference.

$$\begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\theta}_m \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) & 0 \\ \sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{bmatrix} \quad (7.5)$$





### 5.4.2 Kinematic Model

The equation for the robot speeds is given by the following formula:

$$\begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} = r \cdot \begin{bmatrix} -\sin(\delta_0) & \cos(\delta_0) & R \\ -\sin(\delta_1) & \cos(\delta_1) & R \\ -\sin(\delta_2) & \cos(\delta_2) & R \end{bmatrix}^{-1} \cdot \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} \quad (7.6)$$

Where:

- $r$  is the radius of the omnidirectional wheels.
- $R$  is the radius of the robot.
- $\delta$  are the angle of the wheels.

Then, replacing the values of equation in the chapter 4:

$$\begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} = 0.05 \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} \quad (7.7)$$

#### Implementation of the Kinematic model (Illustration 62)

The inputs of this block are the speed of the robot. The outputs are speeds XY ( $V_x$ ,  $V_y$ ) and gamma ( $r$ ) in robot reference.

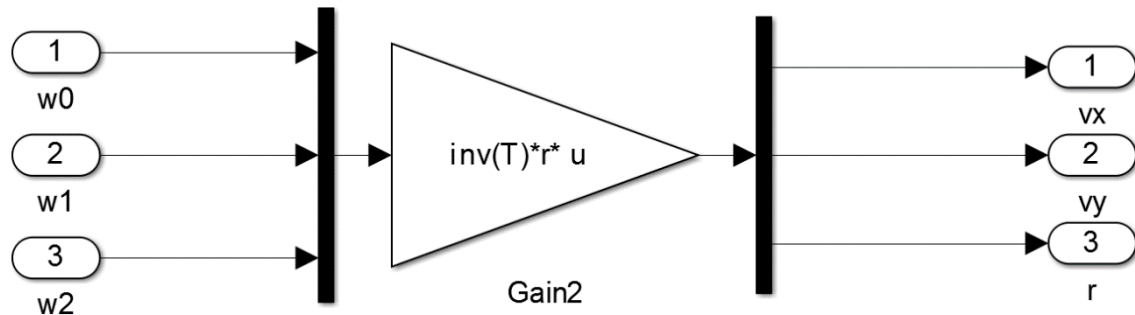


Illustration 62. Kinematic model in Simulink

- $T$  is the array of the Kinematic formula
- $r$  is the radius of the omnidirectional wheels.

### 5.4.3 Wheels Model

In Chapter 3 we obtained the discrete transfer function for each wheel and their own PI controller. Summary of the results in the table 15 and table 16:

Model for the wheel 0	Model for the wheel 1	Model for the wheel 2
$a_0 = 0.9907$ $b_0 = 0.1553$	$a_1 = 0.9893$ $b_1 = 0.1596$	$a_2 = 0.9879$ $b_2 = 0.1702$

Table 15. Parameters of the first-order model for each wheel

Controller parameters for the wheel 0 (PI)	Controller parameters for the wheel 1 (PI)	Controller parameters for the wheel 2 (PI)
$kp'_0 = 86.12$ $ki'_0 = 8.0843528$	$kp'_1 = 83.681391$ $ki'_1 = 9.05075$	$kp'_2 = 78.3587$ $ki'_2 = 9.5975$

Table 16. Parameters of the PI controller after the conversion

### Implementation of the Wheels Model (Illustration 63)

The inputs of this block are:

- Setpoint speed for each wheel

The outputs of this block are:

- Speed for each wheel.

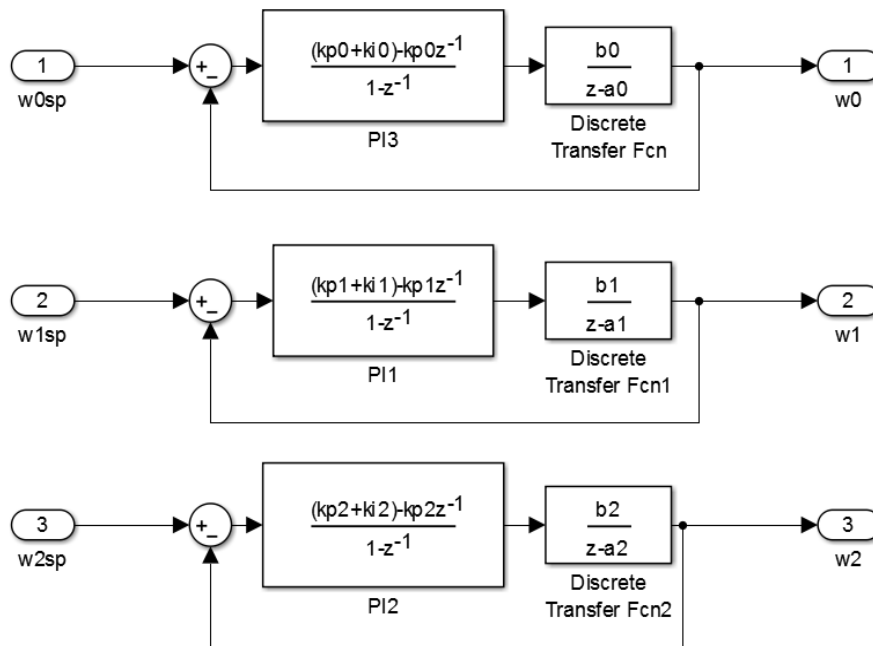


Illustration 63. Wheels model in Simulink

#### 5.4.4 Trajectory Control Algorithm

We need to apply two calculations in this Subsystem block. The first one is an array to get the setpoint speed for the robot, these outputs feed subsystem block "Wheels Model". Using the XY robot speeds and gamma (r).

$$T = \begin{bmatrix} w_{sp0} \\ w_{sp1} \\ w_{sp2} \end{bmatrix} = \frac{1}{0.05} \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_r(t) \\ \dot{y}_r(t) \\ \dot{\theta}_r(t) \end{bmatrix} \quad (7.8)$$

The second one is  $\dot{\theta}_r$  which depends on the control algorithm. The result was the following formula (7.9):

$$\dot{\theta}_r = V \cdot \frac{2}{L^2} \cdot \left( x \cdot \cos(\theta) - \sqrt{L^2 - x^2} \cdot \sin(\theta) \right) \quad (7.9)$$

- V is the Speed of the robot in m/s.
- L is the distance between the centre of the robot and one point of the path line.
- x is the position of the robot in the axes X.
- $\theta$  is the orientation of the robot in degrees.

### Implementation of the Trajectory Control

The inputs of this subsystem block are the x coordinates of the robot, orientation (theta), XY speeds and the variable called “ControlTray” which can activate or deactivate the trajectory control.  $x_m$  and theta are the result of the change of coordinates applied in another subsystem block.

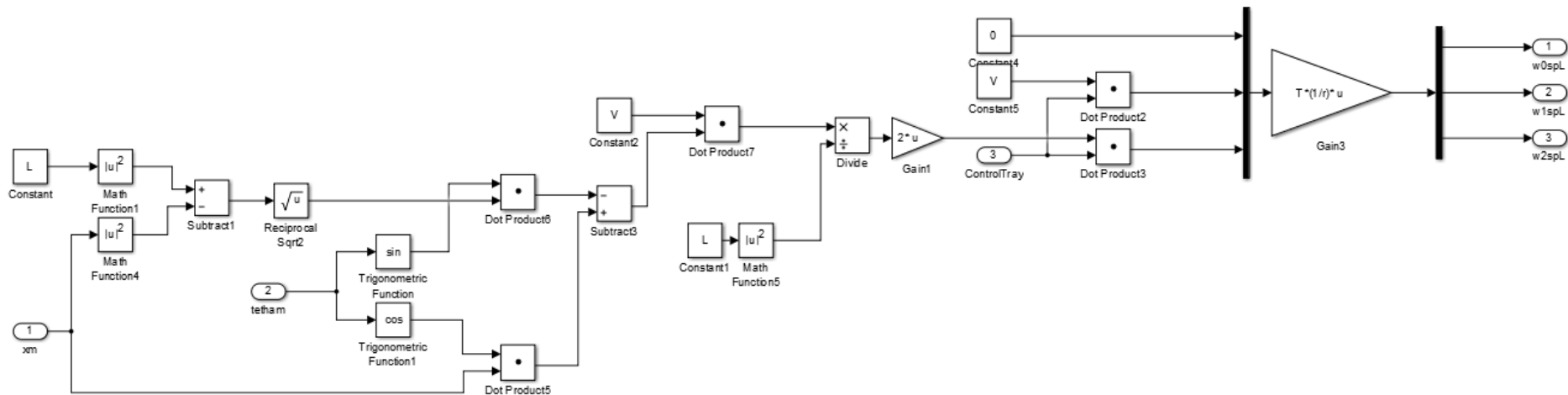


Illustration 64. Scheme of Simulink model to simulate the behaviour when the robot is out of line

The  $V_x$  speed is 0 because what we want is the robot to move forward and the  $V_y$  speed is equal to the speed in m/s.

The variables  $V_y$  and theta have to be multiplied by the variable “ControlTray” which can only have two values 0 and 1 (to activate or deactivate block “Trajectory Control Algorithm”).

### 5.4.5 Change axes of coordinate

The previous case was a simple case where the robot has to be redirected and continue straight. But if we want turn right or left, we must apply the inverse of rotation matrix to the coordinate system of the robot. We show below calculations of the rotation matrix and translation matrix.

#### Rotation matrix

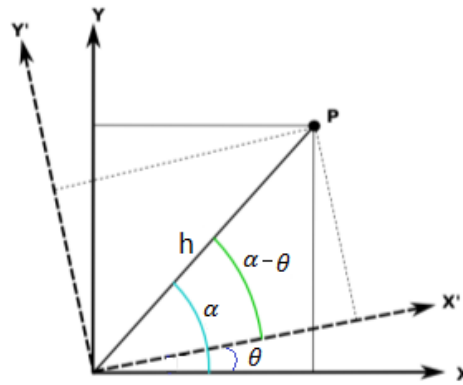


Illustration 65. Rotation on the axis Z

$$x = h \cdot \cos(\alpha) \quad x' = h \cdot \cos(\alpha - \theta) \quad (7.10.a)$$

$$y = h \cdot \sin(\alpha) \quad y' = h \cdot \sin(\alpha - \theta) \quad (7.10.b)$$

$$x' = h \cdot \cos(\alpha - \theta) = h \cdot (\cos(\alpha) \cdot \cos(\theta) + \sin(\alpha) \cdot \sin(\theta)) = \quad (7.10.c)$$

$$= h \cdot \cos(\alpha) \cdot \cos(\theta) + h \cdot \sin(\alpha) \cdot \sin(\theta) = \quad (7.10.d)$$

$$x' = x \cdot \cos(\theta) + y \cdot \sin(\theta) \quad (7.10.e)$$

$$y' = h \cdot \sin(\alpha - \theta) = h \cdot (\sin(\alpha) \cdot \cos(\theta) - \cos(\alpha) \cdot \sin(\theta)) = \quad (7.10.f)$$

$$= h \cdot \sin(\alpha) \cdot \cos(\theta) - h \cdot \cos(\alpha) \cdot \sin(\theta) = \quad (7.10.g)$$

$$y' = -x \cdot \sin(\theta) + y \cdot \cos(\theta) \quad (7.10.h)$$

$$x' = x \cdot \cos(\theta) + y \cdot \sin(\theta) \quad (7.10.i)$$

$$y' = -x \cdot \sin(\theta) + y \cdot \cos(\theta) \quad (7.10.j)$$

The equations can be written in matrix form:

$$R = \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) \\ \sin(\theta(t)) & \cos(\theta(t)) \end{bmatrix} \quad (7.11)$$

The sign criteria is positive if the angle rotates counter clockwise and it is negative if rotates clockwise. Where the reference 0 is on the axis Y.

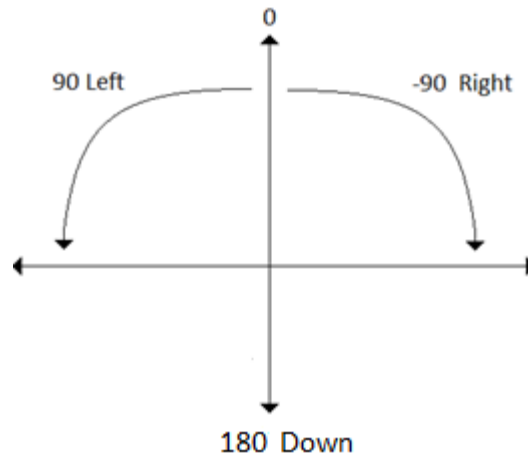


Illustration 66. Sign criteria

We want to express a point in another reference system when the robot rotated, therefore, we have to rotate the angle and change the sign. For example, in case it rotates clockwise an angle = - 90°, we use 90°.

$$R_{forward} = \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) \\ \sin(\theta(t)) & \cos(\theta(t)) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (7.12.a)$$

$$R_{Right} = \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad (7.12.b)$$

$$R_{down} = \begin{bmatrix} \cos(180) & -\sin(180) \\ \sin(180) & \cos(180) \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \quad (7.12.c)$$

$$R_{Left} = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad (7.12.d)$$

$\theta$  has to be increased depending on its direction when the robot turns. We follow the same sign criteria but the units are in radians as we see in the table 17:

Increments of theta	Direction
0	Up
$-\pi/2$	Right
$-\pi$	Down
$\pi/2$	Left

Table 17. Increments of theta

### Translation matrix

The translation matrix is a 2x1 matrix with the position in XY axes.

$$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (7.13)$$

The input variables are  $x_m$  and  $y_m$  in XY coordinates. This matrix changes the position of the robot. It takes the last point and it becomes as origin point for the next position in the simulation. This action will take place each sampling time (Sample time = 0.001).

$$p_{xyz} = p_{uvw} \cdot t \quad (7.14.a)$$

$$T p_{uvw} = p_{xyz} \cdot (-T) \quad (7.14.b)$$

We use the reverse to obtain new coordinates because the inputs are in XY coordinates.

Properties of the translation in a vector space:

- The composition of two translations switches in another translation.
- Identity is a translation.
- Every translation is bijective and the inverse of other translation.

$$\forall u, v \in V \rightarrow t_v \circ t_u = t_{u+v}, I = t_0, (t_u)^{-1} = t_{-u} \quad (7.15.a)$$

$$t^{-1} = -t = \begin{bmatrix} -tx \\ -ty \end{bmatrix} \quad (7.15.b)$$

### *Translation and Rotation of the coordinates*

The next matrix represents the transformation of a vector in homogeneous coordinates about the transformations.

$$\begin{bmatrix} \text{Rotation} & \text{Translation} \\ \text{Perspective} & \text{Scale} \end{bmatrix} \quad (7.16)$$

In robotics, the perspective transformation is void and the scale is equal to the unit (1).

$$\begin{bmatrix} \text{Rotation} & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & \text{Translation} \\ 0 & 1 \end{bmatrix} \quad (7.17)$$

We applied a translation before a rotation to the coordinates. Multiply the matrices to obtain this equation that have to be implemented in the simulation:

$$\begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} R & -R \cdot t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = R' \cdot \begin{bmatrix} x \\ y \end{bmatrix} - R \cdot t \quad (7.18)$$



*Implementation of the change of coordinates in Simulink (Illustration 67)*

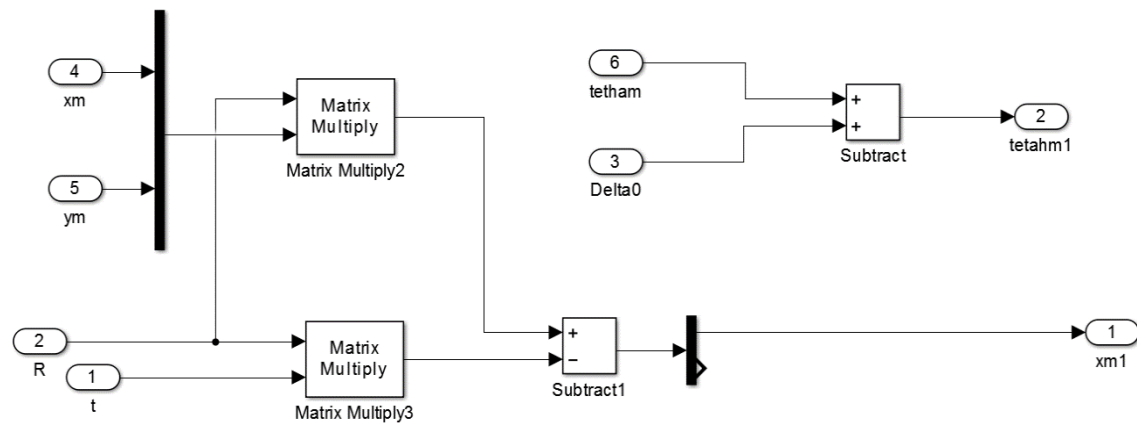


Illustration 67. *Implementation of change of coordinates in Simulink*

- “R” is the rotation matrix that change according if we want to turn left or right.
- “t” is the transfer matrix that also change depending on the orders that we introduce.
- “Delta0” is the increment of theta ( $\theta$ ).
- Theta ( $\theta$ ) the orientation of the robot
- xm the position of the robot in coordinates XY
- ym the position of the robot in coordinates XY

### 5.4.6 Implementation of the trajectory of the Robot in Simulink

I build do a subsystem block with multiple inputs and multiple outputs. The number of inputs and outputs depends on the MATLAB Function (Illustration 68).

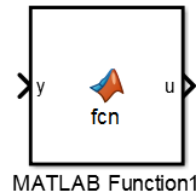


Illustration 68. Block MATLAB Function

#### How to implement the trajectory in the simulation

I use a different function Simulink model for each method. But in both methods, I use the same sign criteria to introduce the path tracking, at least in the simulation.

The Matlab function has an input variable called "ListWaypoints" consisting of a matrix of 3 columns and as many rows as rotation points it has.

- The first column is the position on the x axis of the turning point.
- The second column is the position on the y axis of the rotation point.
- The third column is the direction to be followed by the robot to reach that point.

Note that the last row of this matrix is 0 by requisites of the program.

How interpret the orders of the third column:

Direction		
↑	0	Up
→	1	Right
↓	2	Down
←	3	Left

Table 18. Criteria of directions

Example:

x	y	Direction
0	2	0
-2	2	3
-2	3	0
3	3	1
3	0	2

**Method 1.**

The inputs of the matlab function are the position of the robot in world reference and the variable called "ListWaypoints". The outputs are the rotation matrix, the matrix of translation, the increment of theta and the variable called "ControlTray", which is equal to 1, because the robot is in constant motion.

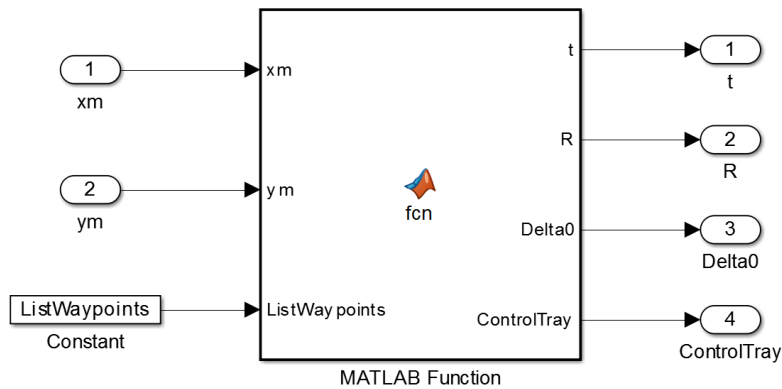


Illustration 69. Matlab function method 1

**Description of the programming code:**

The complete program is in the Annex 9.1.2 Programming code for the simulation of the method 1 and its main parts are explained here.

**1st Variable statement:**

```
persistent waypoint
persistent Vector

if (isempty(waypoint))
    waypoint=1;
    Vector=[0 0];
end;
```

- waypoint:

This variable is the row of the array "ListWaypoints". For the first case: waypoint = 1.

- Vector:

This variable indicates the direction to be followed by the robot. The value [0 0] is not used.

Both variables are persistent variables which keeps the last value that have been given what is essential for the simulation throughout of time.

## 2nd. Path tracking

```
if (norm([ListWaypoints(waypoint,1)-xm,ListWaypoints(waypoint,2)-ym])<0.7)
    waypoint=waypoint+1;
    if waypoint==size(ListWaypoints,1)
        waypoint=waypoint-1;
    end
end
```

These lines of code adds 1 to the variable called “waypoint” and goes to the second line of the array called "ListWaypoints" when the robot is at distance less than 0.7m from the rotation. Then goes to the next rotation point.

But when the variable "waypoint" reaches the maximum value (the row where all values are 0) the code subtract 1 to the variable, and the robot moves permanently in the last direction.

## 3rd. Conversion of criteria address

```
if ListWaypoints(waypoint,3)==0
    Vector=[0 1];
elseif ListWaypoints(waypoint,3)==1
    Vector=[1 0];
elseif ListWaypoints(waypoint,3)==2
    Vector=[0 -1];
elseif ListWaypoints(waypoint,3)==3
    Vector=[-1 0];
end;
```

These lines of code convert the values 0, 1, 2, 3 of the third column of the variable called "ListWaypoints" in a binary value that applies to the equation of the rotation matrix and the increased  $\theta$ .

## 4th. ControlTray and Change of coordinates

```
ControlTray=1;
R=Vector(1,1)*[0 -1;1 0]+Vector(1,2)*[1 0;0 1];
t=[ListWaypoints(waypoint,1);ListWaypoints(waypoint,2)];
Delta0=Vector(1,1)*pi/2+Vector(1,2)*(1-Vector(1,2))*pi/2;
```

Remember that variable called "ControlTray" is always 1 because it is in constant motion.

The second line (“R” equation) depends on the values of the variable called "Vector":

The result will be [1 0; 0 1] if the values are [0 1] which correspond to the robot going forward.

If the values are [1 0], the resulting matrix is [0 -1; 1 0] which correspond to the matrix of rotation to the right, and successively for the values to the down and left ([-1 0; 0 -1] and [0 1; -1 0] respectively).

The third line ("t" equation) is the translation matrix, it takes the values of the first and second column of the variable called "ListWaypoints" which corresponds to the location of the rotation point in XY coordinates.

The last line ("Delta0" equation) depends on the values of the variable called "Vector":

If the values are [0 1], the result is zero.

If the values are [1 0], the result is ( $\pi/2$ ) which allows turn to the left and successively for the values to the right and down ( $-\pi/2$  and  $-\pi$  respectively).

### Method 2.

We have to add a new subsystem block called: Open Loop and modifying the inputs and outputs of the block Micro-Controller Subsystem and the code of the MATLAB function.

#### New Block: Open Loop

This block acts when the robot stops at the rotation point and it begins to turn on left or right. (The direction depends on the next waypoint)

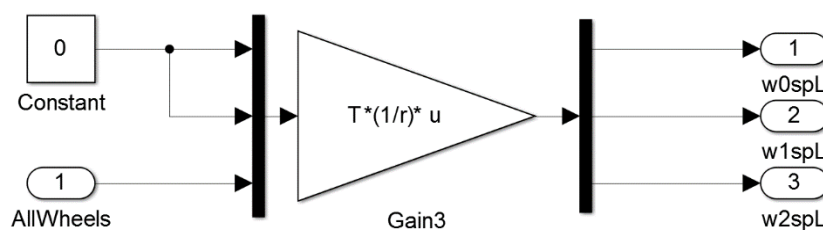


Illustration 70. Kinematic Matrix

This block has one input and three outputs. The input is the variable called "AllWheels", this value depends on the speed of robot (V). The three outputs are the setpoint speed for each wheel, this outputs are the inputs of the subsystem block called "Model Wheels".

Robot has to turn left or right with no linear speed, which is why the speed of the three wheels must be the same, so the  $V_x$  and  $V_y$  are 0. The only different value is the third input on the Simulink block called “mux”.

Once the kinematic matrix applied, the result is a 3x1 matrix with the same setpoint speed in the three rows of the matrix.

### Microcontroller Block: method 2 (Illustration 71)

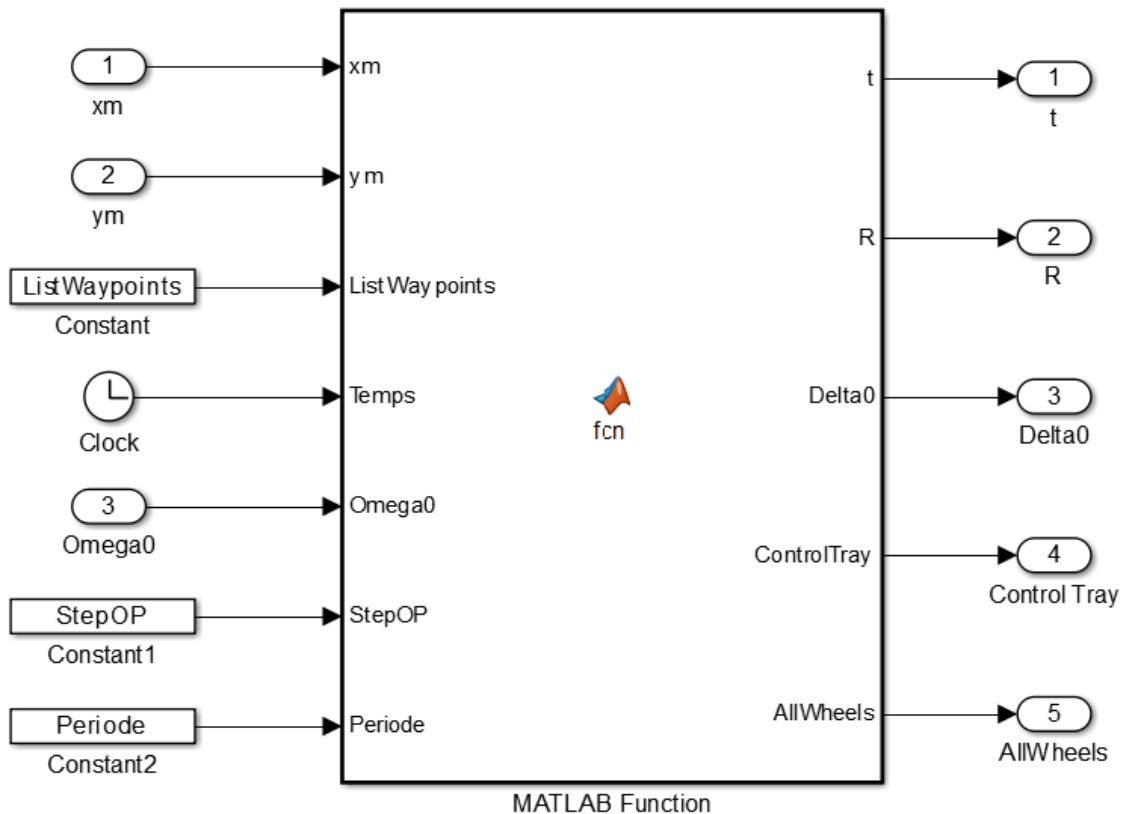


Illustration 71. Matlab function method 2

The inputs of this block are the position in coordinates XY of the robot, the variable called "ListWaypoints" same as in method 1, but now we have to add a clock to have the simulation time, the wheel speed 0 ( $\Omega_0$ ), and two new variables: “StepOP” and “Periode”.

The outputs are the same as we saw on the method 1: The rotation matrix, the translation matrix, increment of theta, the variable “ControlTray” which enables and disables the control trajectory and the new variable called "AllWheels" that we discussed before in the section block Open Loop.

First, we explain the new inputs on Matlab function before explaining the code:

- The Simulink block "Clock" is the output of current simulation time. It is a way to introduce the simulation time as a variable to use it in the code (Name: Temps).
- Omega 0 is the wheel speed 0, the first output of the subsystem block called "Wheels Model".
- StepOp is the speed of the robot when it has to turn left or right with no linear speed.
- The Periode variable is the time it takes the robot to do 1/4 turn.

### How to find the values: "StepOP" and "Periode".

Assuming that the robot is facing to the Y axis, the robot has to rotate 1/4 turn to the left if we want to move to the left, and the same thing if it have to move to the right. It needs to rotate 1/2 of turn if we want to go in the opposite direction.

Therefore, if we want to know how long needs the robot to rotate 1/4 turn with the same speed, we carry out the following calculations:

```
Pas1=((1/0.05)*T*[0;0.2;0]);  
Pas2=0.05*T^-1*[Pas1(3,1);Pas1(3,1);Pas1(3,1)];  
StepOP=Pas2(3,1)
```

Where:

T is the matrix of the Kinematic transformation

The result of "Pas1" are the setpoint speed for each wheel if we want to move forward straight.

The Pas2 is to get the speeds (Vx, Vy) and theta for the robot to turn on itself, maintaining the same speed. The setpoint speed of the wheels must be equal for each wheel (Pas1 (3.1)). The final result is Vx and Vy equal to 0 and theta equal to the variable called "StepOP".

The variable called "StepOP" must be positive, the sign of the variable called "StepOP" will change in the Matlab function. The sign depends on the rotation direction that have to rotate, the value is positive or negative depending if it have to rotate to left or

right positive for left turns and negative for right turns. If it have to go in the opposite direction that was following, the sign it is irrelevant.

If we want to know how long it takes the robot to rotate  $1/4$  turn, we use the next formula:

$$\text{Periode} = (\pi/2) / \text{StepOP}$$

### Programing code

#### - Description of the code:

This program works with three possible states for the robot:

**State 0:** The simulation enables the trajectory control ( $\text{ControlTray} = 1$ ) and it disables the subsystem block "Open loop" ( $\text{AllWheels} = 0$ ), the path simulation goes to the rotation point that marks the variable called "ListWaypoints" and "waypoint". When the simulation have reache the rotation point it goes to State 1.

**State 1:** When the simulation enters this state, the path simulation stops ( $\text{ControlTray} = 0$  and  $\text{AllWheels} = 0$ ). Once stopped ( $\Omega \sim 0$ ), the variable "ListWaypoints" changes to the next rotation point ( $\text{waypoint} = \text{waypoint} + 1$ ) and then changes to the State 2.

**State 2:** The simulation disables the trajectory control ( $\text{ControlTray} = 0$ ) and it activates the subsystem block Open loop ( $\text{AllWheels} = \text{StepOP}$ ). Then, it begins to rotate to one direction depending on the previous direction and the actual direction. There are 4 possibilities: left, right, going to the opposite direction to the previous or keep the same direction to the previous.

Summary in the table 19:

POSSIBLE DIRECTIONS OF THE ROBOT					
Direction		To turn right	To turn left	To turn opposite direction	Not turn
↑	0	1: ← 2: ↑	1: → 2: ↑	1: ↓ 2: ↑	1: ↑ 2: ↑
→	1	1: ↑ 2: →	1: ↓ 2: →	1: ← 2: →	1: → 2: →
↓	2	1: → 2: ↓	1: ← 2: ↓	1: ↑ 2: ↓	1: ← 2: ←
←	3	1: ↓ 2: ←	1: ↑ 2: ←	1: → 2: ←	1: ↓ 2: ↓

1: (Direction that the robot has been following before it stops)

2: (New direction after rotating)

Table 19. Possible directions of the robot



In case that it goes to the opposite direction, the time of rotation is double ( $\text{Periode} \cdot 2$ ). If the simulation follows the same direction, the system goes to the state 0 without performing any rotation.

If the path simulation has to rotate, when it passes the time of the variable "Periode", the system returns to state 0. We use this code for the three cases of rotation to know when it has reached the rotation time:

```
if time0==0
    time0=Temps;
end

if Temps-time0<2*Periode
    ControlTray=0;
    AllWheels=StepOP;
else
    State=0;
    time0=0;
end
```

The complete program is in the Annex 9.1.3 Programming code for the simulation of the method 2.

## 5.5. Scheme of simulations

### 5.5.1 Method 1:

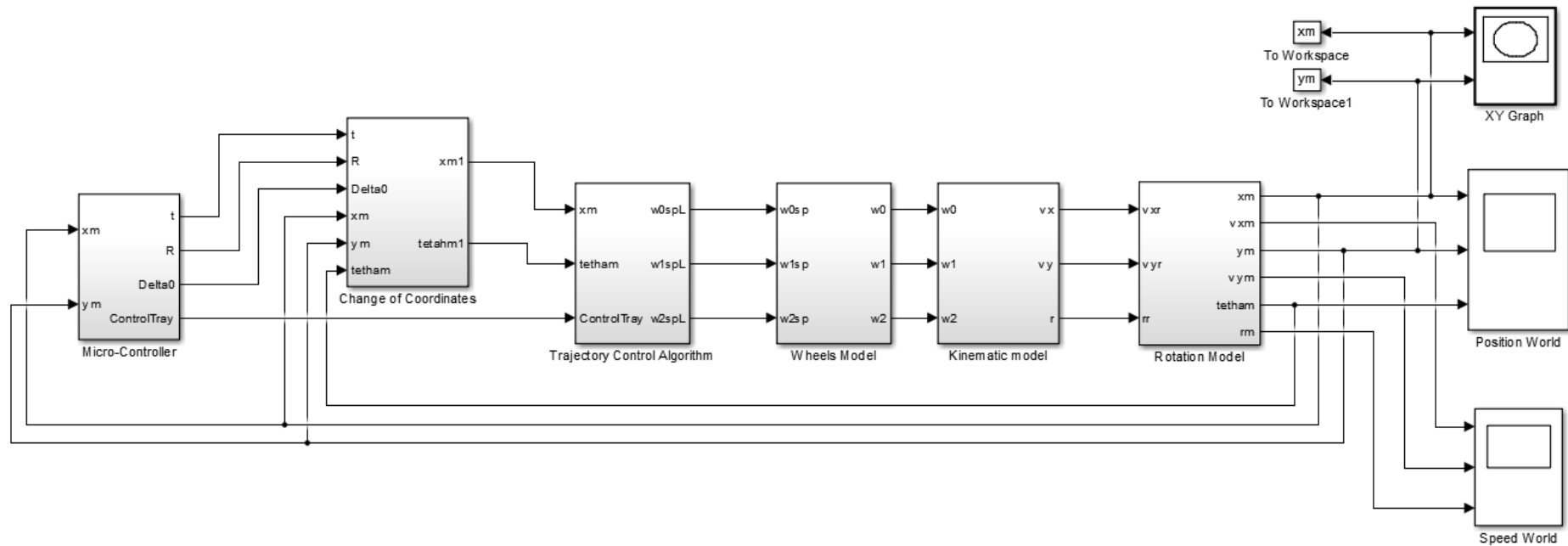


Illustration 72. Scheme: Method 1

In this scheme we can see the distribution of the subsystem blocks which were explained before and how their inputs and outputs are connected. To the right of the scheme, there are two scopes that allow us to know the position and speed of the robot during the simulation. The third block, XY Graph, gives us a graphical representation of the trajectory the robot was following.

### 5.5.2 Method 2:

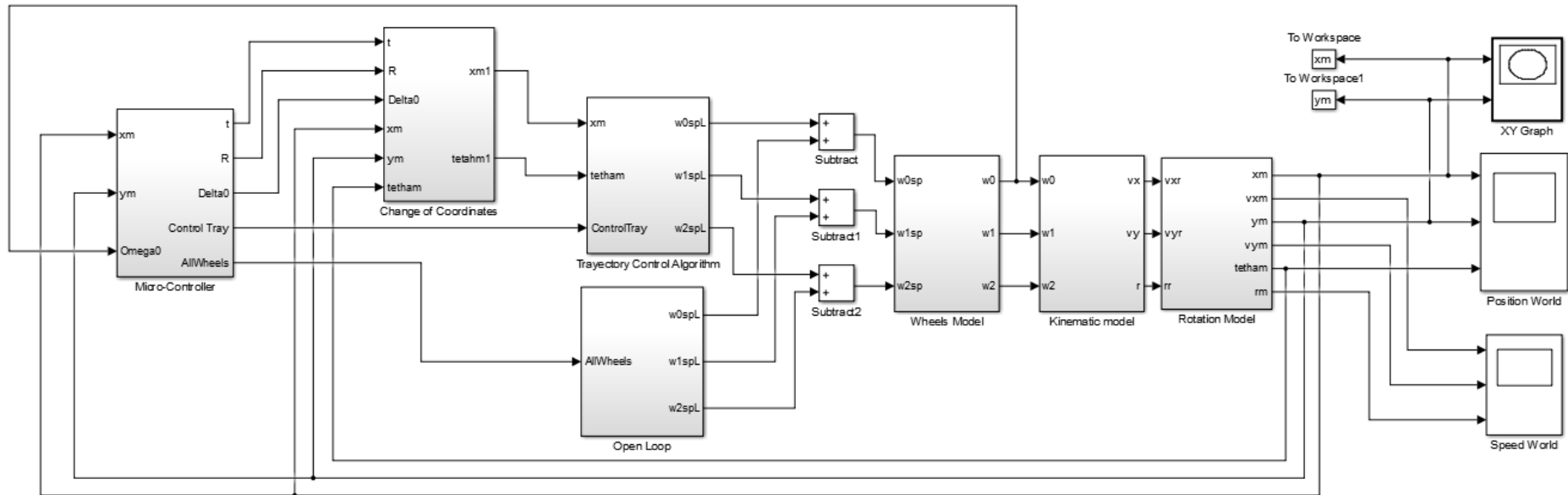


Illustration 73. Scheme: Method 2

In this scheme we can see the distribution of the subsystem blocks which were explained before and how their inputs and outputs are connected to the right of the scheme there are two scopes that allow us to know the position and speed of the robot during the simulation. The third block XY Graph gives us a graphical representation of the trajectory the robot was following.

In this second method, the outputs of the subsystem block called "Trajectory Control Algorithm" and the subsystem block called "Open loop" are summed before entering the subsystem block called "Wheels Model" because only one of them will be active.

## 5.6. Simulation of the trajectory control methods

### 5.6.1 Workspace for both methods:

Discrete Model of the wheels		PI Parameters	
a0 = 0.9907	b0 = 0.1553	kp0 = 0.0319	ki0 = 2.9942e-04
a1 = 0.9893	b1 = 0.1596	kp1 = 0.0310	ki1 = 3.3521e-04
a2 = 0.9879	b2 = 0.1702	kp2 = 0.0290	ki2 = 3.5546e-04

Variables in the simulation				
L = 0.7	Periode = 1.3603	r = 0.05	StepOP = 1.1547	V = 0.2

Robot path desired	Kinematic Matrix
ListWaypoints :	
0 3 0	T:
2 3 1	-0.5000 -0.8660 0.1500
2 5 0	1.0000 -0.0000 0.1500
0 5 3	-0.5000 0.8660 0.1500
0 3 2	
0 0 0	

Table 20. Variables of the Workspace

### 5.6.2 Results of the simulation method 1:

#### Scope: Position World:

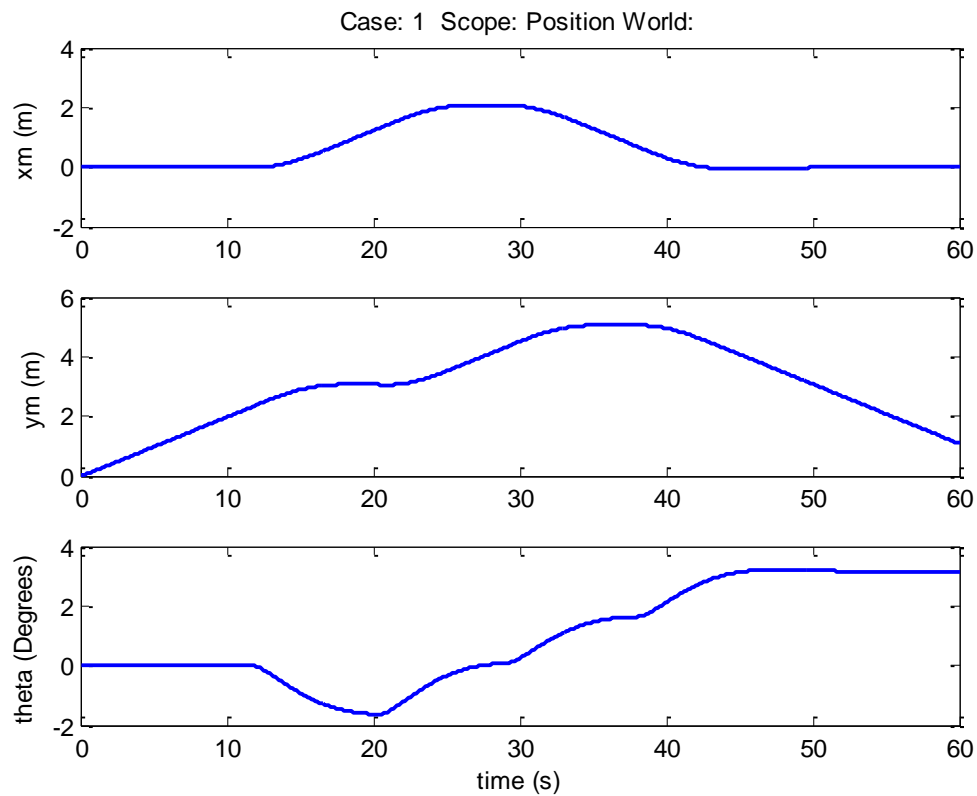


Illustration 74. Method 1, Position

#### Scope: Speed in world reference and robot reference:

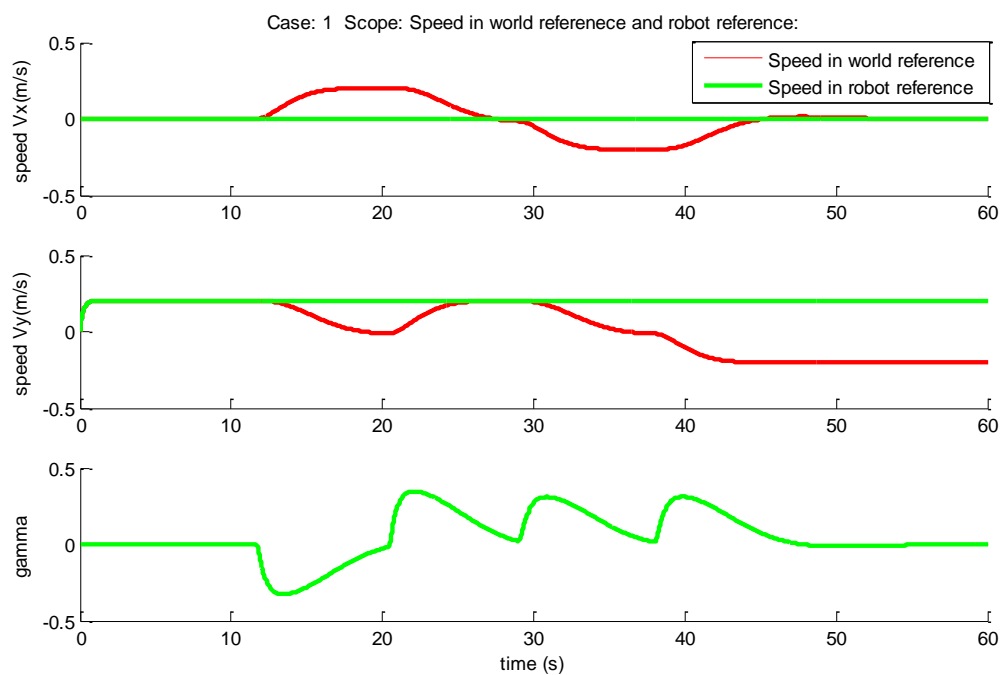


Illustration 75. Method 1, Speed in axes XY

*XY Graph: method 1*

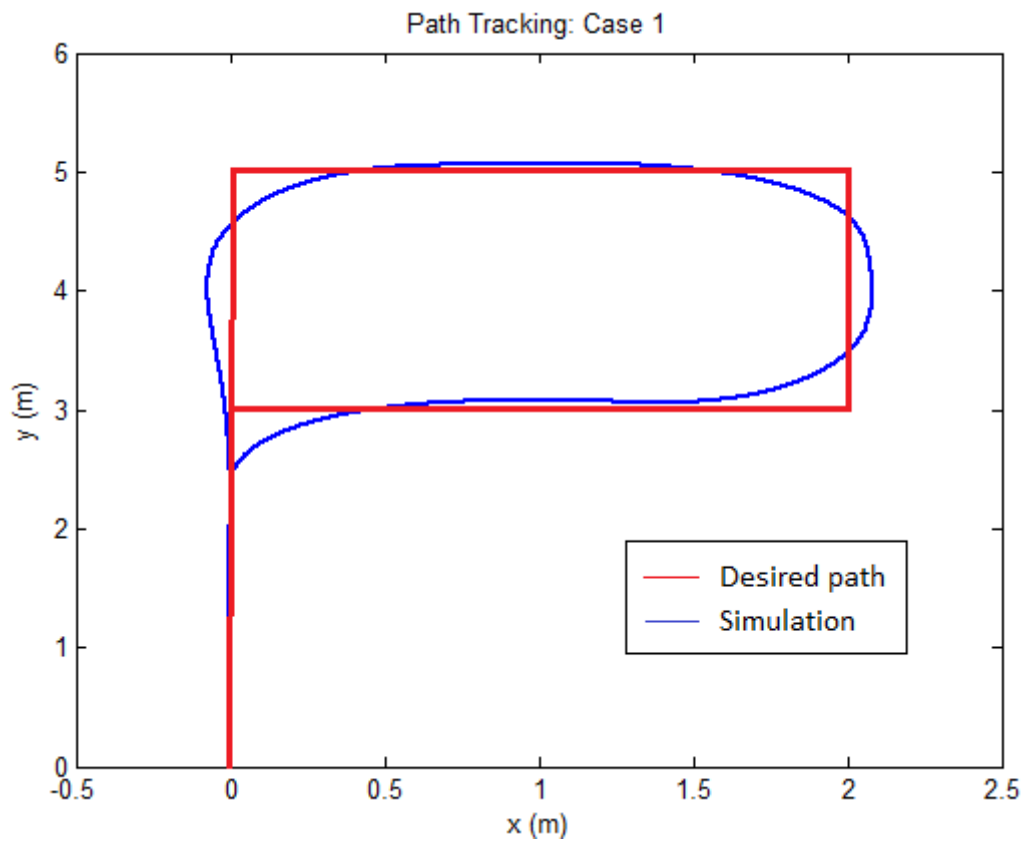


Illustration 76. Method 1, Path Tracking

### 5.6.3 Results of the simulation method 2:

#### Scope: Position World:

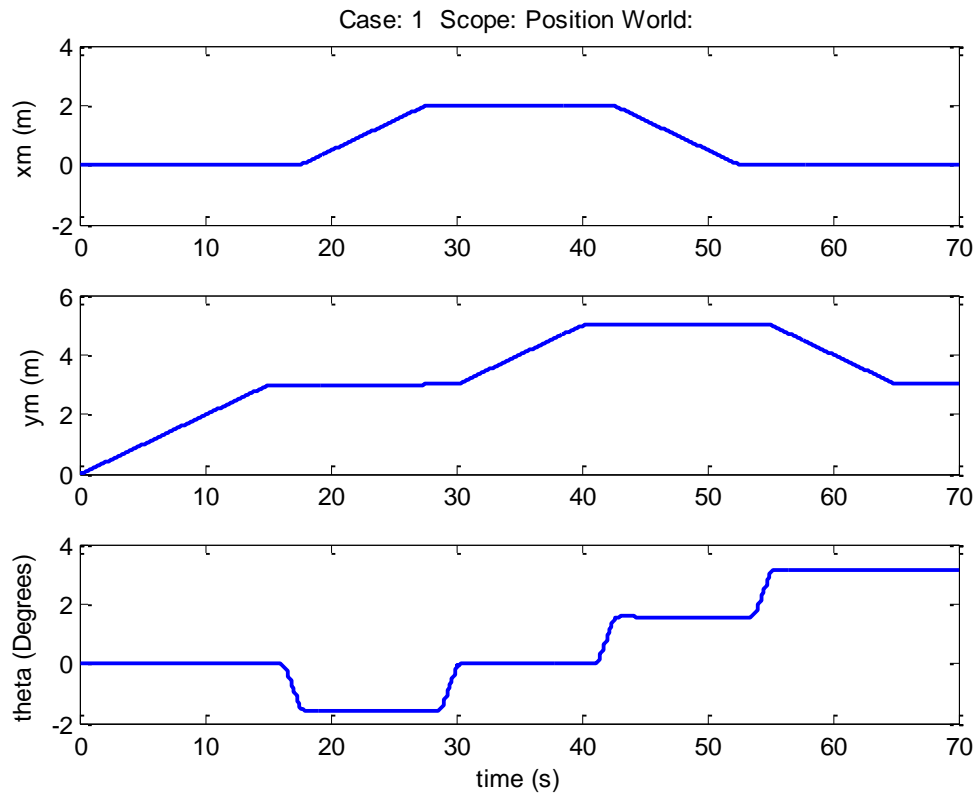


Illustration 77. Method 2, Position

#### Scope: Speed World:

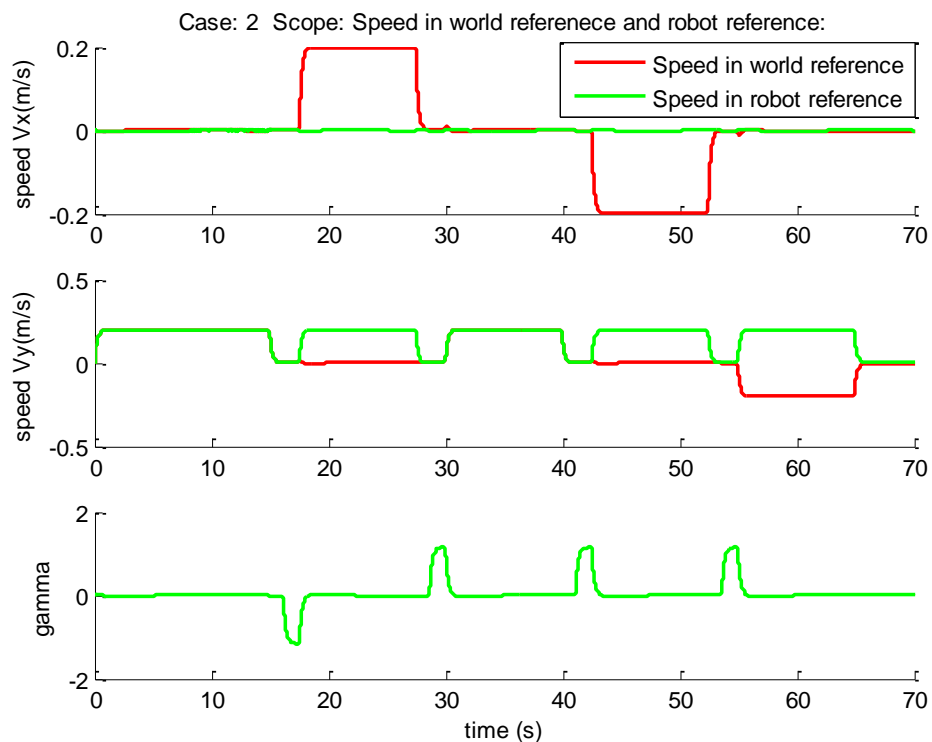


Illustration 78. Method 2, Speed in axes XY

*XY Graph: method 2*

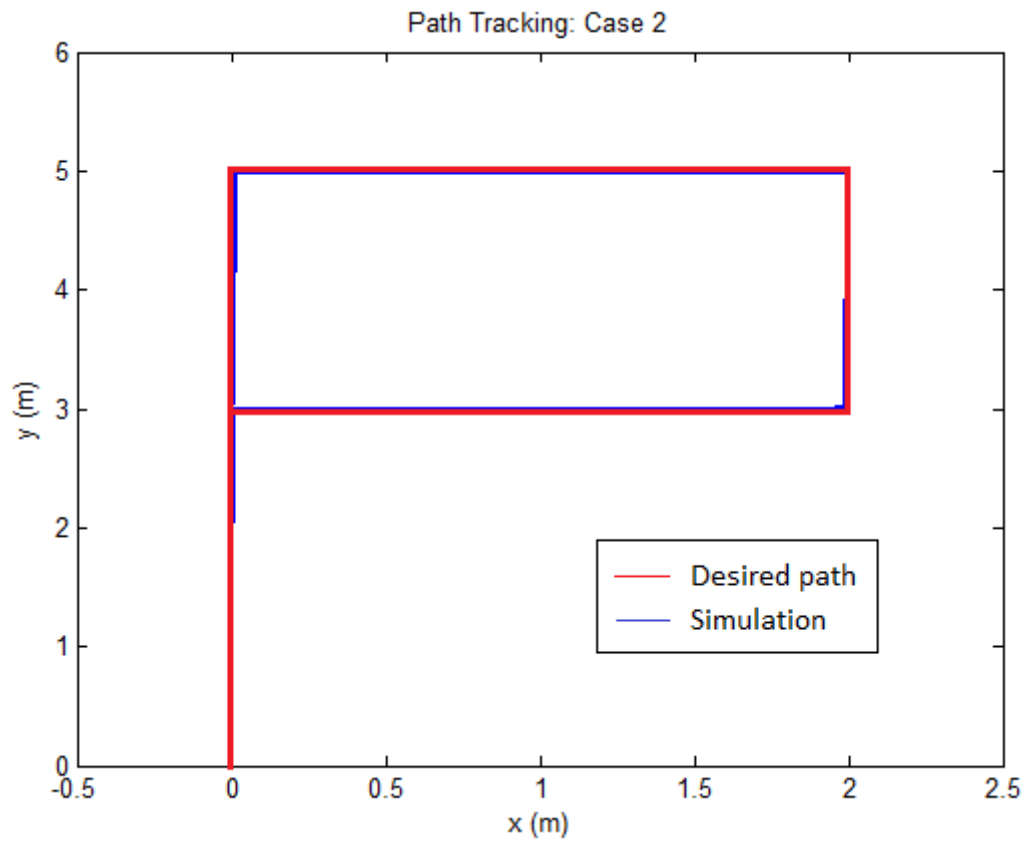


Illustration 79. Method 2, Path Tracking



#### ***5.6.4 Conclusion of the simulation***

Both simulations have followed the same route without problems, but we are going to discuss the differences between one method and another:

Method 1: The simulation reaches to the first rotation point (0,3) but exactly when it is at 0.7m of the rotation point, the robot starts to turn and it heads to the second rotation point (2,3). Once pass for all points, it follows the last direction until the end of the simulation.

Method 2: The simulation stops right at the rotation point (0,3), then the robot turns to one side and it continues on line until reach the next point (2,3). It follows the same procedure for all points of the route but when it reaches the last point ((0,3), for the second time) stops until ends the simulation .

#### ***Conclusions:***

Method 1 offers the advantage that it does not need to stop and thus it is faster than method 2. But, it have the disadvantage that needs a space free of obstacles when it has to rotate.

Method 2 is more precise, because it pursues the trajectory perfectly without leaving the line, but it is slower because the robot has to stop and rotates at every rotation point.

In order to implement one method or the other, we must take into account the space available to move. The simulation was very slow during the procedure but this can be arranged by simplifying the code or using a more powerful computer.

6. IMPLEMENTATION OF TRAJECTORY CONTROL, VISION AND TRAJECTORY PLANNER.

In this chapter, I described the experimental results of applying the methodology to the Robotino, the methodology has been presented in Chapters 3, 4 and 5.

I this section, I describe the variables used which include the variables of the trajectory control, vision camera, PI controller of the wheels and the inclusion of the trajectory planner. The variable that determines the path is different from the one used in the simulation block, therefore, it has to be explained in more details.

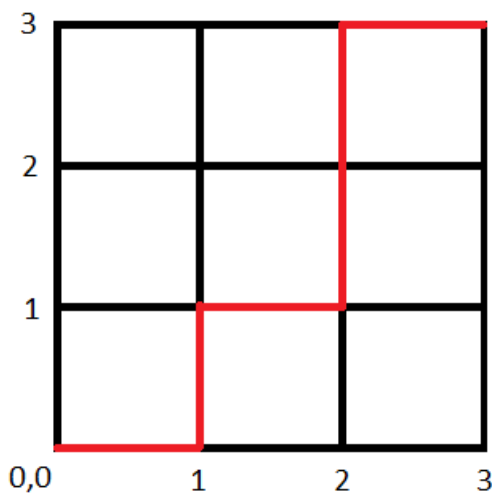
It also describes the input variables on the workspace of Matlab and the programming code which includes both methods that was used in the laboratory for the statistical study.

6.1 Implementation of the interface for the communication between the robot and the trajectory planner.

The simulation program works with an input variable that indicates the direction of the robot when it arrives at a crossroad. Now, the variable of the simulation is no longer useful for the implementation of trajectory planner, the variables of the planner is a matrix of two columns which indicates in XY coordinates of the path to be followed (The first point is always (0,0)).

The planner works with a grid that indicates the points where the robot must pass until the last point of the trajectory.

For example, the illustration 80 shows a grid and a marked route in red. Therefore, the variable of the planner is like the table 21:



X	Y
0	0
1	0
1	1
2	1
2	2
2	3
3	3

Table 21. Example of the route

Illustration 80. Representation of a route in a 3x3 grid

As I said, the variable that interprets the program and the variable that uses the planner are different. Therefore, it is necessary to apply several transformations.

First of all, subtracting the actual row to the previous row allows to transform the absolute variable called "Planificador" into an incremental variable called "ListWaypoints". (*This is not the same variable as in simulation*).

We must do a final transformation which takes the previous and the present direction of the robot based on the variable called "Ordenes". The variable was called "Traectoria". (In the last row, it adds the number 4 which is used to stop and disconnect the robot once the trajectory is completed).

The criterion of directions that has been followed this time for the both second and the third transformation are the multiple of the angles being  $0 = 0^\circ$  it turns right,  $1 = 90^\circ$  it moves backwards,  $2 = 2 \cdot 90^\circ = 180^\circ$  it turns left and  $3 = 3 \cdot 90^\circ = 270^\circ$  it moves forwards.

Input variable:	1r Transformation:
Planificador =	ListWaypoints =
0 0	0 1
0 1	1 0
1 1	0 1
1 2	0 1
1 3	
2n Transformation:	3r Transformation:
Ordenes =	Traectoria =
1	0
0	2
1	1
1	4

Table 22. Transformations of the route

## **6.2 Programming code for implementing the trajectory control, vision by camera, controller PI of the wheels and trajectory planner**

### **6.2.1 Input Parameters**

The parameters related to camera calibration:

- The rotation matrix (R), the translation matrix (t) and the intrinsic parameters of the camera (KK).

The parameters related to each motor for each wheel:

- $k_p$ ,  $k_i$  are the values of the PI controller associated for each wheel ( $k_{p0}$ ,  $k_{p1}$ ,  $k_{p2}$ ,  $k_{i0}$ ,  $k_{i1}$ ,  $k_{i2}$ ). "StepOP" are the rotational speeds applied to each wheel for the robot to turn over and the variable "Periode" is the time it takes to make a quarter turn at that speed.

Example:

```
V=0.15;  
Pas1=(1/0.05)*T*[0;V;0];  
Pas2=0.05*T^-1*[Pas1(3,1);Pas1(3,1);Pas1(3,1)];  
StepOP=Pas2(3,1)
```

```
Periode=(pi/2)/StepOP
```

The parameters related to the trajectory control:

- The transformation matrix (T), speed in cm/s of the robot (V) and the distance from the robot to a point on the line (L).

The variables related to the robot path:

- The input variable from the trajectory planner. (Planificador)

### **6.2.2 Matlab instructions for Robotino**

In the reference [11] of the bibliography, there's a summary and an explanation of each instruction of Matlab that has been used to control Robotino in the programming code.

These are the reference for instructions used:

- Com
- Motor
- Bumper
- Camera

You will find the program used in the Annex 9.1.4 Programming code for both methods on the real robot

## **6.3 Test methodology**

### 6.3.1 Introduction

Before starting the experiment we must do the camera calibration following the method explained in the section (2.5.3 Camera (Webcam)). Remember the parameters  $R$ ,  $t$  and  $KK$  are extracted from that methodology and then we introduce the rest of initial variables.

Once this is done, we can run the program in the Command Window of Matlab. After that, we must to set the method to be used.

- 1 for the closed loop (First method).
- 2 for the open loop (Second method).
- The program stops if another number is inserted.

Method 1 is in continuous movement without pause during the rotation, and method 2 is in slow movement, because it stops every time that it has to rotate.

After entering the control method number, the bumper has to be pressed. The robot takes approximately 5 seconds after pressing the bumper to start moving. The reason for this delay was not clarified.

The second method is called open loop because at the time of rotation, the robot stops on the line crossing and rotates over itself ignoring its position on the line. But the first method follows the line even when the robot rotates. Except, when there is a moment when the robot does not recognize the line as a vertical or horizontal. The robot works in an open loop in that moment for a few seconds, but just enough time to recognize the line and get the distance and the angle of the robot about the line, essential to keep moving.

6.3.2 Tests performed

Introduction

The experiment runs along the same route that contains all possible directions using the two methods developed. The same experiment was performed twice for each method at the same speed of cruise. In the end, we are going to determine which one of both methods is better by comparing the results of the statistical study with the data from angle, distance, time, and speed of the experiment.

Robot path and input variables

Robot path

The route that has been chosen for the experiment contains all possible directions that can take the robot, also it includes a path without change of direction to obtain the speed of cruise.

The following illustration shows the path of the experiment:

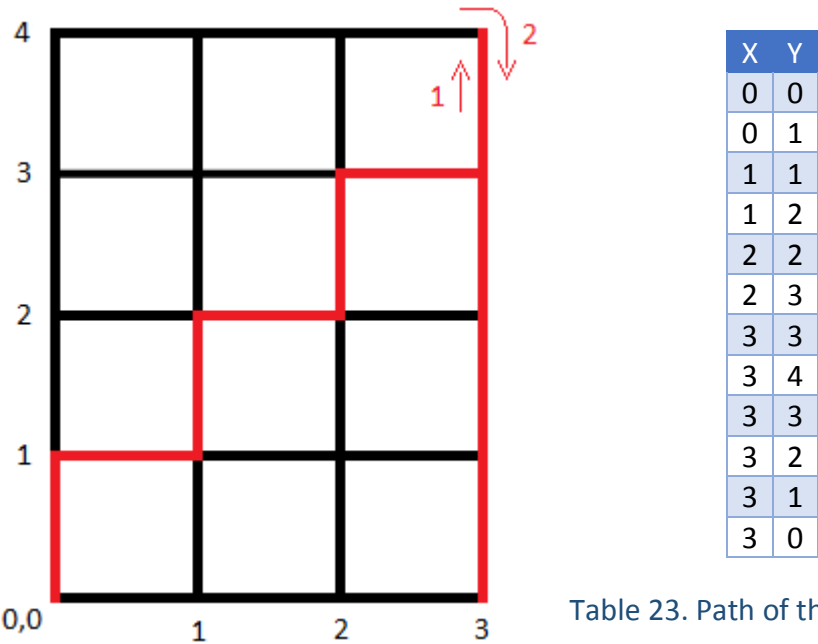


Table 23. Path of the experiment

Illustration 81. Representation of the path of the experiment in a 3x4 grid

The grid in the laboratory is formed by squares of 80 cm of side.

The robot is going straight to the point (0,1), followed by rotations of right / left until the point (3,4), half turn on itself and then continues down on the same vertical line until reaches the point (3,0).

**Input Parameters**

This are the input values for the experiments:

Kp0 = 86.1201	KK =			Planificador	
	404.8412	0	143.9544		
Kp1 = 83.6814	0	408.8882	108.0107		
	0	0	1.0000		
Kp2 = 78.3587	t =			x	y
		9.4529		0	0
Ki0 = 8.0844		179.1836		0	1
		112.7092		1	1
Ki1 = 9.0508	R =			1	2
	0.9975	-0.0574	-0.0414	2	2
Ki2 = 9.5975	-0.0695	-0.6872	-0.7231	2	3
	0.0130	0.7242	-0.6895	3	3
StepOP = 1.1547	T =			3	4
	-0.5000	-0.8660	0.1500	3	3
Periode = 1.3604	1.0000	0	0.1500	3	2
L = 0.65 m	-0.5000	0.8660	0.1500	3	1
				3	0
V = 0.15 m/s					

Table 24. Summary of all input values

### 6.4 Laboratory experiments

In this section, the result of the comparison with the data that were collected is shown.

#### 6.4.1 Comparison between both methods

##### Working time (Seconds)

Method 1:	Time Elapsed	Method 2	Time Elapsed
1r Test	61,4525 s	1r Test	69,8954 s
2n Test	62,9192 s	2n Test	70,8882 s

Table 25. Working time for each method

Method 1 is faster than the method 2 because the robot is in constant motion at method 1 and it stops at each rotation point at method 2. Below the mean, variance and standard deviation of the distance and angle of the robot to the line marked on the floor of the laboratory for both methods are presented. To check the accuracy of the trajectory control.

##### Distance of the robot to the vertical line (centimetres)

Method1:

Test 1		Test 2	
Mean	3.265 cm	Mean	3.5712 cm
Variance	2.9413 cm	Variance	1.7697 cm
Standard Deviation	1.715 cm	Standard Deviation	1.3303 cm

Table 26. Results of the distance of the robot to the vertical line, Method 1

Method 2:

Test 1		Test 2	
Mean	4.1300 cm	Mean	3.1096 cm
Variance	1.4247 cm	Variance	1.3887 cm
Standard Deviation	1.1936 cm	Standard Deviation	1.1784 cm

Table 27. Results of the distance of the robot to the vertical line, Method 2

From the data obtained of the distance of the robot to the vertical line we can see how the first method had little variation between the tests, contrary to the tests with method 2.

The mean is similar for both methods (about 3-4 cm from the line). But the standard deviation for the method 2 is smaller than for method 1. Therefore, it is more reliable to use method 2 to follow the vertical line using the distance.



***Distance of the robot to the horizontal line (centimetres)***

Method 1:

Test 1		Test 2	
Mean	15.7581 cm	Mean	15.9576 cm
Variance	1.6423 cm	Variance	3.4283 cm
Standard Deviation	1.2815 cm	Standard Deviation	1.8516 cm

Table 28. Results of the distance of the robot to the horizontal line, Method 1

Method 2:

Test 1		Test 2	
Media	12.6833 cm	Media	12.6455 cm
Variance	1.4852 cm	Variance	0.7689 cm
Standard Deviation	1.2187 cm	Standard Deviation	0.8769 cm

Table 29. Results of the distance of the robot to the horizontal line, Method 2

From the data obtained on the distance of the robot to the horizontal line we can see how the two tests for both methods have not variations appreciable between them.

This data has a different meaning for each method:

- For method 1 means that robot starts to rotate at a distance of 15 cm from the rotation point.
- For method 2 means that robot begins to stop at 12 cm from the rotation point.

But the most important thing is that the standard deviation for the method 2 is smaller than for method 1. Therefore, it is more reliable to use method 2 to redirect it to the horizontal line.

***Angle of the robot to the vertical line (Degrees):******Method 1:***

Test 1		Test 2	
Media	2.3734°	Media	2.1736°
Variance	1.7329°	Variance	2.8253°
Standard Deviation	1.3164°	Standard Deviation	1.6809°

Table 30. Results of the angle of the robot to the vertical line, Method 1

**Method 2:**

Test 1		Test 2	
Media	3.2636°	Media	2.2089°
Variance	4.0586°	Variance	2.0627°
Standard Deviation	2.0146°	Standard Deviation	1.4362°

Table 31. Results of the angle of the robot to the vertical line, Method 2

From the data obtained of the angle of the robot to the vertical line we can see how the first method had little variation between the tests, contrary to the tests with the method 2.

The mean is similar for both methods (about 2-3 degrees from the line). But the standard deviation for the method 1 is smaller than for method 2. Therefore, it is more reliable to use method 1 to follow the vertical line using the angle.

***Angle of the robot to the horizontal line (Degrees):******Method 1:***

Test 1		Test 2	
Media	87.4654°	Media	87.8504°
Variance	1.5161°	Variance	2.7767°
Standard Deviation	1.2313°	Standard Deviation	1.6663°

Table 32. Results of the angle of the robot to the horizontal line, Method 1

***Method 2:***

Test 1		Test 2	
Media	85.9520°	Media	86.7986°
Variance	2.9330°	Variance	1.7944°
Standard Deviation	1.7126°	Standard Deviation	1.3396°

Table 33. Results of the angle of the robot to the horizontal line, Method 2

From the data obtained of the angle of the robot to the horizontal line we can see how the first method had little variation between the tests, same as the tests with the method 2.

The mean is similar for both methods (about 86-87 degrees from the rotation point).

These values mean and the little standard deviation suggest that the robot had been following the straight line without problem because the horizontal line is almost perpendicular from the point of view of the robot before to reach the rotation point.

***Robot path during the experiment:***

From the speed of each wheel we can see the rotation moment at each point along the path, it is easily noticeable in the wheel 1. There are seven changes of value in wheel 0, three to turn right, three to turn left and one last change (half turn). The same for the wheel 0 and 2 but it is not much appreciable as we see on the wheel 1.

Method 1:

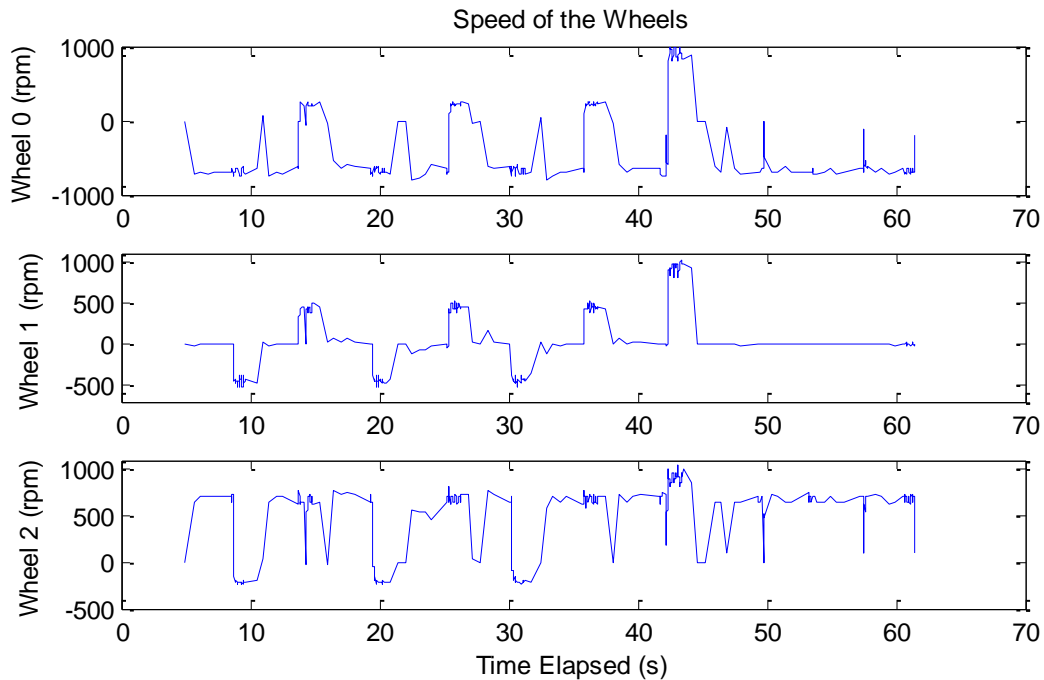


Illustration 82. Speed of the wheels during the experiment, Method 1

Method 2:

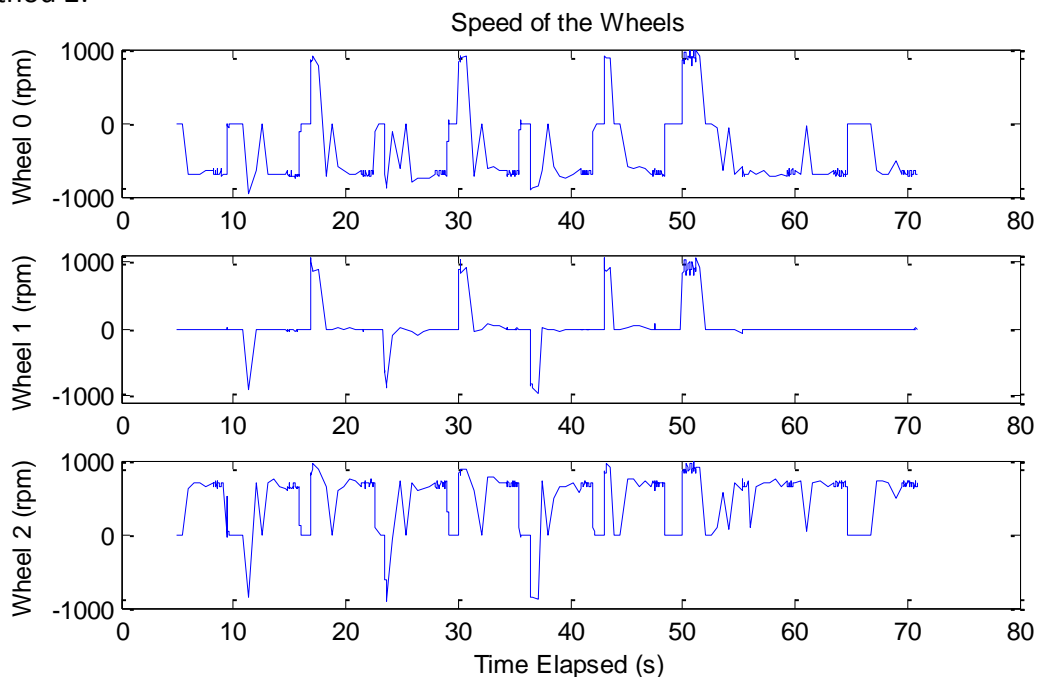


Illustration 83. Speed of the wheels during the experiment, Method 2

There is a difference between the first and the second method, the robot went straight from the point (3,4) to the point (3,0) without stopping by using method 1, but with the second method it moves from the point (3,4) to the point (3,1) and stops for a moment at this point, but then reaches to the point (3,0).

As is shown it in the Illustration 83. Method 2 can follow straight line without stopping as its proves when it goes from the point (3, 4) until the point (3, 1), but the robot stops at the point penultimate (3, 1) before to finish the path at the point (3, 0).

I apply the kinematic matrix to the speed for each wheel in order to find the  $V_x$ ,  $V_y$  and  $\dot{\theta}$ . But first we have to convert the rpm into rad/s.

$$Wheel\left(\frac{rad}{s}\right) = \frac{\pi \cdot Wheel\ (rpm)}{30} \quad (8.1)$$

$$\begin{bmatrix} V_x(t) \\ V_y(t) \\ \dot{\theta}(t) \end{bmatrix} = 0.05 \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} Wheel\ 0 \\ Wheel\ 1 \\ Wheel\ 2 \end{bmatrix} \quad (8.2)$$

By integrating of  $\dot{\theta}$  I can obtain theta ( $\theta$ ) throughout the experiment, finally with the application of the rotation matrix that we used in the simulation section in a Simulink Model.

$$R = \begin{bmatrix} \cos(\theta(t)) & -\sin(\theta(t)) & 0 \\ \sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

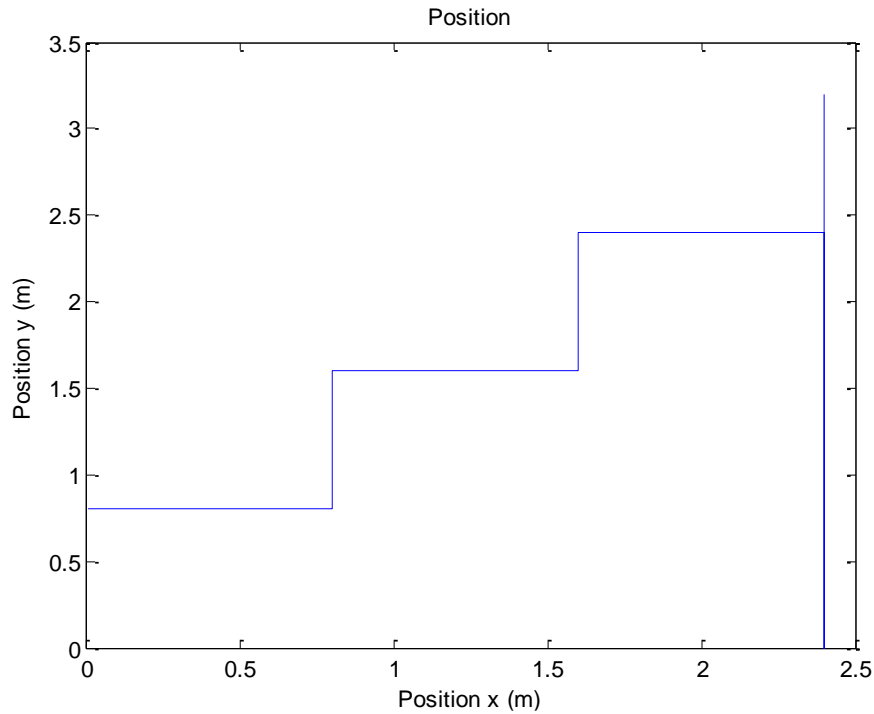


Illustration 84. Representation of the desired path of the experiment

We should have obtained a similar path to the 84 illustration with steps that we made before but the result was as follows:

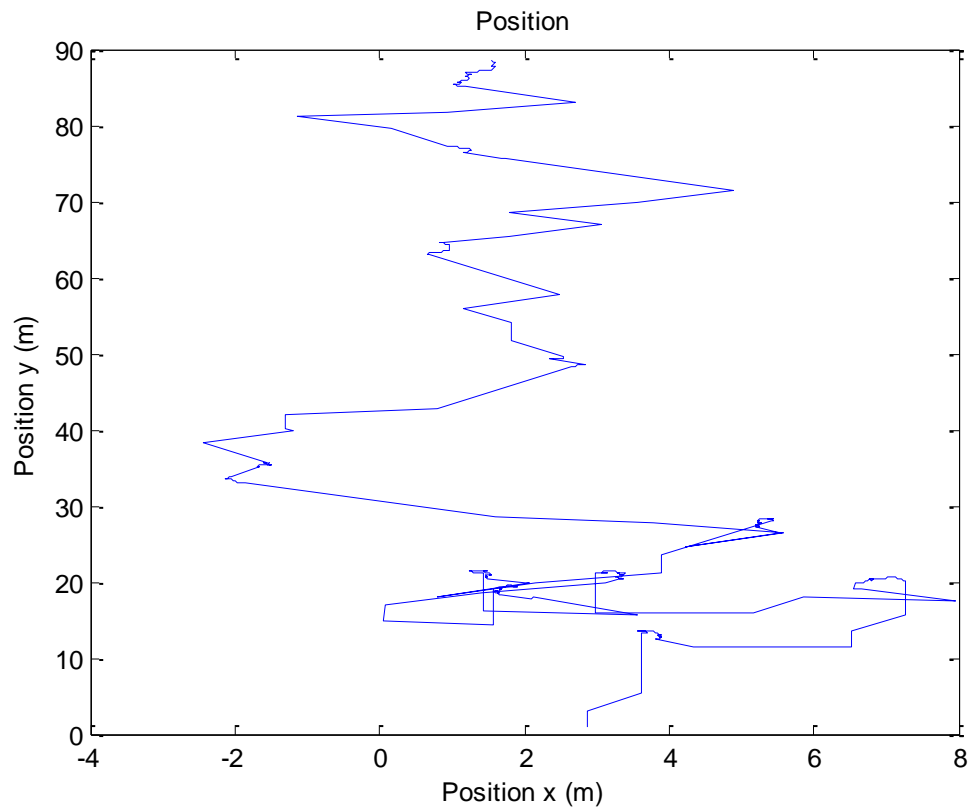


Illustration 85. Path of the experiment

The result of using the speed of each wheel to obtain the position of the robot at all times during the experiment it has not been possible. This justifies the tracking line.

#### 6.4.2 Speed of each wheel for both methods:

In this section we compare the speeds but as we have no data speed, we do from rpm data.

*Average cruise rpm for each wheel and each test, Method 1 (rpm):*

Test 1		Test 2	
Wheel 0	-632.6234 rpm	Wheel 0	-678.2932 rpm
Wheel 1	0.2730 rpm	Wheel 1	0.0211 rpm
Wheel 2	626.8231 rpm	Wheel 2	675.7811 rpm

Table 34. Results of the speed, Method 1

*Average cruise rpm for each wheel and each test, Method 2 (rpm):*

Test 1		Test 2	
Wheel 0	-620.5032 rpm	Wheel 0	-683.5093 rpm
Wheel 1	0 rpm	Wheel 1	-2.7959 rpm
Wheel 2	616.0617 rpm	Wheel 2	683.2121 rpm

Table 35. Results of the speed, Method 2

The average cruise rpm of the robot for the method 1 is 677 rpm and for method 2 is 683 rpm when the robot follows the straight line. **(The speed what I want is 0.15 m/s)**

Using the kinematic array that we saw in previous chapters, the formula is as follows:

$$v = 0.05 \cdot \begin{bmatrix} -0.5 & -0.87 & 0.15 \\ 1 & 0 & 0.15 \\ -0.5 & 0.87 & 0.15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \text{Wheel 0} \\ \text{Wheel 1} \\ \text{Wheel 2} \end{bmatrix} \cdot \left(\frac{0.1}{27}\right) \quad (10)$$

The actual speed has not reached the theoretical speed:

$$\text{Method 1:} \quad v = 0.1448 \text{ m/s} \quad (11.1)$$

$$\text{Method 2:} \quad v = 0.1461 \text{ m/s} \quad (11.2)$$

The difference between the actual speed and the desired speed was only 4% for method 1 and 3% for the method 2. Therefore, we should correct this difference at the time to introduce the value of the variable V in the workspace of Matlab, for example in this case the correction for both methods is as follows:

$$\text{Method 1:} \quad v = 0.156 \text{ m/s} \quad (12.1)$$

$$\text{Method 2:} \quad v = 0.1545 \text{ m/s} \quad (12.2)$$

### 6.4.3 Final comparison

Once done the individual analysis of the data and the comparison between both methods we can conclude that Method 1 completes the path with a 14.28% less of time than method 2. Moreover, from data obtained of the angles and the distances have been determined that obtaining distances by using the method 2 is more reliable than the method 1, but the method 1 is better to obtain the angles than the method 2.

When the experiment was performed the robot followed the path, but it has not been possible to obtain the odometry robot from the wheel speeds (applying the kinematics matrix and the matrix of rotation), this justifies the tracking line.

It must be noticed that the open loop experiment is not always successful.

It has not been changed one single line or value of the program between each repetition of the experiment. The reason of these repetitions is because it works in an open loop and there are many external and unpredictable factors in the wheels: loss communication between the robot and the PC or the state of the line drawn on the floor, among other possible factors.

Regarding desired speed and the speed obtained, we found that the difference for both methods is very small, but the method that has the smallest percentage is method 2.

As we saw, the difference between using a method or another for tracking path is not significant. Both methods are equally reliable but from the data captured by the camera we can say that the method 2 has a slight advantage over the method 1 in trajectory tracking.

We conclude from the data obtained of the tests that method 1 is a faster method, but the method 2 is more stable for trajectory tracking. Ultimately, if we have to choose between one method or another, it is a matter of priorities.

Method 1: If we want a faster system

Method 2: If we want to reliable trajectory tracking.

But from the view as laboratory work, the reliability in trajectory tracking is more important than the speed of work time. **That is why is recommended to use Method 2 to perform the experiments with the Robotino.**



## ***7. Conclusions of the project***

It has achieved the main goal of this project: The development of a trajectory control by use a camera of Robotino. At below, it details the different points that has this project:

### **➤ Model and controller of the wheels**

It was been extracted the model and controller for each wheel from the speed data.

### **➤ Kinematic Model**

We studied the kinematic model for our robot with omnidirectional wheels and it was implemented in Simulink. (Including the PI controller and the first-order model for practical exercises of mobile robots)

### **➤ Trayectory control**

The trajectory control equation was developed following a mathematical model to be implemented later in the simulation file.

### **➤ Simulates the path tracking**

It reviews all parts of the project and it has implemented in the simulation, but it was added the development of the rotation and translation in the simulation. It has built two different methods of simulation to work with two different ways of trajectory tracking. Two methods that served as a reference at the time of working with the real robot.

Finally, we implemented a trajectory planner with a script to enter a variable that contains all directions of the path. Once both methods have been simulated under the same conditions, it has been compared the differences between using a model or another. The result has been that the method 1 is faster than the method 2, but the room for maneuver during the rotation is greater on method 2.

### **➤ Implementaton of the vision camera**

We obtained the distance and the angle of the robot thanks to the program of processing image and we also found the intrinsic and extrinsic parameters of the camera.

➤ **Build the trajectory tracking program**

I have developed a program which implements two methods of tracking lines from simulation and the vision camera. It also has been added the lines of code needed to obtain experimental data during the tests following the indications of the professor.

➤ **Conduct experiments in the laboratory**

Once done the final program, we have been able to perform four tests. Two repetitions for method in order to perform a statistic study.

➤ **Statistic study to determine the best method from laboratory data**

We concluded that Method 1 is faster than method 2 as happened in simulation, but method 2 has been more stable to path tracking and closer to the assigned speed of cruise.

From the view of laboratory, it has determined that data collection is most important and the time difference between using one method or other was not significant. So the best way to continue working in the future with the Robotino was the method 2.

- **FUTURE PROJECTS**

This project has mainly an educational purpose, it is a good starting point for initiation in control of mobile robots. This project can be expanded and improved in many ways, these are some of them:

- Obstacle detection. If an obstacle appears along the route, the vision system should detect it and issue an alarm to the planner.
- The trajectory tracking algorithm developed have both an open loop part which brings uncertainty and imprecise to the process. That could be enhanced.

***This project will be used in classes of the school ESEIAAT by teachers and students to be improved and extended, that is why this project is only the first step of more that will come.***

## 8. BIBLIOGRAPHY

### Resources in physical format.

1. Claudio Urrea, José Muñoz : Path Tracking of Mobile Robot in Crops : J Intell Robot Syst , 193–205 (2015)
2. Edwin Vans, Gancho Vachkov, Alok Sharma: Vision Based Autonomous Path Tracking of a Mobile Robot using Fuzzy Logic : University of the South Pacific, (2014)
3. Felipe Kühne, Felipe Kühne, João Manoel Gomes da Silva Jr : Mobile Robot Trajectory Tracking Using Model Predictive Control (2005)
4. Julio E.Normey-Rico a, Ismael Alcalab, Juan Gomez-Ortegab, Eduardo F.Camacho : Mobile robot path tracking using a robust PID controller : Control Engineering Practice 9, 1209–1214 (2001)
5. Ollero, A: “Robótica: manipuladores y Robots Móviles”. Marcombo Boixareu: (2001).
6. Jui-Liang Yang, Ding-Tsair Su, and Ying-Shing Shiao: Vision-Based Predictive Path Tracking Control of a Wheeled Mobile Robot : (2008)
7. Tracking control of 3-wheels omni-directional mobile robot using fuzzy azimuth estimator: Sangdae kim, Changho hyun, Youngwan cho, Seungwoo kim: (2009)
8. Trajectory Tracking Control for a Nonholonomic Mobile Robot Using an Improved ILC : Ruidong Xi, Yangmin Li, Xiao Xiao : (2014)
9. Gregory Dudek, Michael Jenkin, “Computational Principles of Mobile Robotics”, Cambridge University Press, 2000 (Chapter 1).
10. Carlos Canudas de Wit, Bruno Siciliano, Georges Bastin (eds), “Theory of Robot Control”, Springer 1996.

**Resources in electronic format.**

11. Robot manufacturer web page.

Direction: [https://www.festo.com/cms/es\\_es/index.htm](https://www.festo.com/cms/es_es/index.htm)

12. Web page with the parameters of the PI controller of the Robotino.

Direction: [http://wiki.openrobotino.org/index.php?title=Robotino3\\_PI\\_controller](http://wiki.openrobotino.org/index.php?title=Robotino3_PI_controller)

13. Website with functions to control almost every aspect of the Robotino with Matlab

Direction:

[http://doc.openrobotino.org/documentation/MATLAB/html\\_toolbox/html\\_Home.html](http://doc.openrobotino.org/documentation/MATLAB/html_toolbox/html_Home.html)

14. Website of consultations about programming between different users

Direction: <http://www.lawebdelprogramador.com/>

## 9. ANNEX

### 9.1 Programs designed

#### 9.1.1 Programing code of XavmesuraDistanciaARuta.m

```
function [dl_ver, theta_ver, dl_hor, theta_hor] = avmesuraDistanciaARuta( img, R, t, KK, metode )
switch metode(1)
case 1
    w=fspecial('disk',5);
    im=imfilter(img(:,:,1),w,'corr','symmetric');
    nivellMax = max(max(im));
    nivellMin = min(min(im));
    ibw = im2bw(im,double(nivellMax-(nivellMax-nivellMin)*0.40)/255);
case 2
    im= rgb2ycbcr(img);
    im=im(:,:,3);
    ibw = im2bw(im,150/255);
    pintaimatge(im,1,'component Cr de crominancia');
    pintaimatge(ibw,2,'binaritzat fix a nivell 150')
otherwise
    disp('Metode no implementat, pas 1')
end

switch metode(2)
case 1
    se=strel('line',50,90);
    ibww1=imopen(ibw,se);
    pintaimatge(ibww1,3,'open amb linia vertical de 50');
    se=strel('line',50,0);
    ibww2=imopen(ibw,se);
    pintaimatge(ibww2,4,'open amb linia horitzontal de 50');
case 2
    w=fspecial('sobel');
    ibww1=imfilter(ibw,w,'corr','symmetric');
    ibww2=imfilter(ibw,-w,'corr','symmetric');
    ibww=ibww1+ibww2;
```

```
se=strel('line',6,90);
ibwww1=imopen(ibww,se);
pintaimatge(ibwww1,3,'sobel doble + open amb linia vertical de 6');
w=fspecial('sobel');
ibww1=imfilter(ibw,w,'corr','symmetric');
ibww2=imfilter(ibw,-w,'corr','symmetric');
ibww=ibww1+ibww2;
se=strel('line',6,0);
ibwww2=imopen(ibww,se);
pintaimatge(ibwww2,4,'sobel doble + open amb linia horitzontal de 6');
ibww1 = ibwww1;
ibww2 = ibwww2;
otherwise
    disp('Metode no implementat, pas 2')
end

switch metode(3)
case 1
case 2
    ibwlv=bwmorph(ibww1,'thin',Inf);
    pintaimatge(img,5)
    pintaimatge(ibwlv,5,'',1)
    [i,j] = find(ibwlv(30:60,1:320));
    p1 = [mean(j);30+mean(i)];
    [i,j] = find(ibwlv(170:200,1:320));
    p2 = [mean(j);170+mean(i)];
    pintaimatge(ibwlv,5,'op. morfologica d''aprimat vertical',1)
    ibwlh=bwmorph(ibww2,'thin',Inf);
hold on
    plot(p1(1),p1(2),'bo','MarkerSize',7,'LineWidth',3)
    plot(p2(1),p2(2),'yo','MarkerSize',7,'LineWidth',3)
hold off
    pintaimatge(img,6)
    pintaimatge(ibwlh,6,'',1)
    [k,q] = find(ibwlh(1:240,170:200));
    p3 = [170+mean(q);mean(k)];
    [k,q] = find(ibwlh(1:240,20:50));
```

```

        p4 = [20+mean(q);mean(k)];
        pintaimatge(ibwlh,6,'op. morfologica d''aprimat horitzontal',1)
hold on
        plot(p3(1),p3(2),'bo','MarkerSize',7,'LineWidth',3)
        plot(p4(1),p4(2),'yo','MarkerSize',7,'LineWidth',3)
hold off
        case 3
            ccl=bwconncomp(ibww1);
            numPixels = cellfun(@numel,ccl.PixelIdxList);
            [y,i] = max(numPixels);
        otherwise
            disp('Metode no implementat, pas 3')
end

pr1 = R'*inv(KK)*[p1;1];
lambda = 0-t(3)/pr1(3);
p1r = t + lambda*pr1;

pr2 = R'*inv(KK)*[p2;1];
lambda = 0-t(3)/pr2(3);
p2r = t + lambda*pr2;

pr3 = R'*inv(KK)*[p3;1];
lambda = 0-t(3)/pr3(3);
p3r = t + lambda*pr3;

pr4 = R'*inv(KK)*[p4;1];
lambda = 0-t(3)/pr4(3);
p4r = t + lambda*pr4;

subplot(3,2,5)
hold on
text('Position',[p1(1)+20,p1(2)],'String',['(',num2str(p1r(1))',' ',num2str(p1r(2))',')'],'BackgroundColor','blue','Color','white')
text('Interpreter','latex','Position',[p2(1)+20,p2(2)],'String',['(',num2str(p2r(1))',' ',num2str(p2r(2))',')'],'BackgroundroundColor','blue','Color','white')

```



```

hold off
    subplot(3,2,6)
hold on
    text('Position', [p3(1), p3(2)-20], 'String', ['(', num2str(p3r(1)), ',', num2str(p3r(2)), ')'], 'BackgroundColor',
'blue', 'Color', 'white')
text('Interpreter', 'latex', 'Position', [p4(1), p4(2)+80], 'String', ['(', num2str(p4r(1)), ',', num2str(p4r(2)), ')'], 'Backg
roundColor', 'blue', 'Color', 'white')
hold off

p1r = [p1r(1:2);1];
p2r = [p2r(1:2);1];
p3r = [p3r(1:2);1];
p4r = [p4r(1:2);1];

if (p1r(2)<p2r(2))
    ppppv=p1r;
    p1r=p2r;
    p2r=ppppv;
    disp('p2 esta mes lluny que p1, s''intercanvien')
end

linia_ver = cross(p1r,p2r);
linia_ver = linia_ver./sqrt(linia_ver(1)^2 + linia_ver(2)^2);
theta_ver = -atan(linia_ver(2)/linia_ver(1))*180/pi;
d1_ver = linia_ver(3);

if (p3r(2)<p4r(2))
    pppph=p3r;
    p3r=p4r;
    p4r=pppph;
    disp('p3 esta mes lluny que p4, s''intercanvien')
end

linia_hor = cross(p3r,p4r);
linia_hor = linia_hor./sqrt(linia_hor(1)^2 + linia_hor(2)^2);
theta_hor = -atan(linia_hor(2)/linia_hor(1))*180/pi;
d1_hor = linia_hor(3);

```

## 9.1.2 Programming code for the simulation of the method 1

```
function [t,R,Delta0,cambio,ControlTray]= fcn(xm,ym,ListWaypoints)
persistent waypoint
persistent Vector

if (isempty(waypoint))
    waypoint=1;
    Vector=[0 0];
end;

if (norm([ListWaypoints(waypoint,1)-xm,ListWaypoints(waypoint,2)-ym])<0.7)
    waypoint=waypoint+1;
    if waypoint==size(ListWaypoints,1)
        waypoint=waypoint-1;
    end
end

if ListWaypoints(waypoint,3)==0
    Vector=[0 1];
elseif ListWaypoints(waypoint,3)==1
    Vector=[1 0];
elseif ListWaypoints(waypoint,3)==2
    Vector=[0 -1];
elseif ListWaypoints(waypoint,3)==3
    Vector=[-1 0];
end;

ControlTray=1;
R=Vector(1,1)*[0 -1;1 0]+Vector(1,2)*[1 0;0 1];
t=[ListWaypoints(waypoint,1);ListWaypoints(waypoint,2)];
Delta0=Vector(1,1)*pi/2+Vector(1,2)*(1-Vector(1,2))*pi/2;
end;
```

### 9.1.3 Programming code for the simulation of the method 2

```
function [t,R,Delta0,Canvi_del_waypoint,ControlTray,AllWheels,Status]=  
fcn(xm,ym,ListWaypoints,Temps,Omega0,StepOP,Periode)  
persistent waypoint  
persistent State  
persistent Vector  
persistent time0  
  
if (isempty(waypoint))  
    waypoint=1;  
    State=0;  
    Vector=[0 0];  
    time0=0;  
end;  
  
if ListWaypoints(waypoint,3)==0  
    Vector=[0 1];  
elseif ListWaypoints(waypoint,3)==1  
    Vector=[1 0];  
elseif ListWaypoints(waypoint,3)==2  
    Vector=[0 -1];  
elseif ListWaypoints(waypoint,3)==3  
    Vector=[-1 0];  
end;  
  
R=Vector(1,1)*[0 -1;1 0]+Vector(1,2)*[1 0;0 1];  
t=[ListWaypoints(waypoint,1);ListWaypoints(waypoint,2)];  
Delta0=Vector(1,1)*pi/2+Vector(1,2)*(1-Vector(1,2))*pi/2;  
  
if State==0  
    ControlTray=1;  
    AllWheels=0;  
    if (norm([ListWaypoints(waypoint,1)-xm, ListWaypoints(waypoint,2)-ym])<0.05)  
        State=1;  
    end  
end
```

```
elseif State==1
    ControlTray=0;
    AllWheels=0;
    if Omega0>-0.01&&waypoint~=(size(ListWaypoints,1)-1)
        waypoint=waypoint+1;
        State=2;
    end

elseif State==2
    ControlTray=0;
    AllWheels=0;
    if (ListWaypoints(waypoint-1,3)==3&&ListWaypoints(waypoint,3)==0) ||
        (ListWaypoints(waypoint-1,3)==0&&ListWaypoints(waypoint,3)==1) || (ListWaypoints(waypoint-
1,3)==1&&ListWaypoints(waypoint,3)==2) || (ListWaypoints(waypoint-1,3)==2&&ListWaypoints(waypoint,3)==3)
        if time0==0
            time0=Temps;
        end

        if Temps-time0<Periode
            ControlTray=0;
            AllWheels=-StepOP;
        else
            State=0;
            time0=0;
        end
    end

elseif (ListWaypoints(waypoint-1,3)==0&&ListWaypoints(waypoint,3)==3) || (ListWaypoints(waypoint-
1,3)==1&&ListWaypoints(waypoint,3)==0) || (ListWaypoints(waypoint-1,3)==2&&ListWaypoints(waypoint,3)==1) ||
(ListWaypoints(waypoint-1,3)==3&&ListWaypoints(waypoint,3)==2)
    if time0==0
        time0=Temps;
    end

    if Temps-time0<Periode
        ControlTray=0;
        AllWheels=StepOP;
```

```
        else
            State=0;
            time0=0;
        end

elseif (ListWaypoints(waypoint-1,3)==2&&ListWaypoints(waypoint,3)==0) || (ListWaypoints(waypoint-1,3)==0&&ListWaypoints(waypoint,3)==2) || (ListWaypoints(waypoint-1,3)==1&&ListWaypoints(waypoint,3)==3) || (ListWaypoints(waypoint-1,3)==3&&ListWaypoints(waypoint,3)==1)
    if time0==0
        time0=Temps;
    end

    if Temps-time0<2*Periode
        ControlTray=0;
        AllWheels=StepOP;
    else
        State=0;
        time0=0;
    end

elseif (ListWaypoints(waypoint-1,3)==0&&ListWaypoints(waypoint,3)==0) || (ListWaypoints(waypoint-1,3)==1&&ListWaypoints(waypoint,3)==1) || (ListWaypoints(waypoint-1,3)==2&&ListWaypoints(waypoint,3)==2) || (ListWaypoints(waypoint-1,3)==3&&ListWaypoints(waypoint,3)==3)
    State=0;

else
    ControlTray=0;
    AllWheels=0;
    disp('Error en ListWaypoints')
end
else
    ControlTray=0;
    AllWheels=0;
    disp('Error en canvio de State')
end
end
```

### 9.1.4 Programming code for both methods on the real robot

```
ComId = Com_construct;
MotorId0 = Motor_construct( 0 );
MotorId1 = Motor_construct( 1 );
MotorId2 = Motor_construct( 2 );
BumperId = Bumper_construct;
CameraId = Camera_construct;

Com_setAddress(ComId, '192.168.1.219');
Com_connect(ComId);

Motor_setComId( MotorId0, ComId );
Motor_setComId( MotorId1, ComId );
Motor_setComId( MotorId2, ComId );
Bumper_setComId(BumperId, ComId);
Camera_setComId(CameraId, ComId);

Motor_setPID( MotorId0, kp0, ki0, 25 );
Motor_setPID( MotorId1, kp1, ki1, 25 );
Motor_setPID( MotorId2, kp2, ki2, 25 );

WorkTime=[];
d_ver_exp=[];
theta_ver_exp=[];
d_hor_exp=[];
theta_hor_exp=[];
Speed_Motor0=[];
Speed_Motor1=[];
Speed_Motor2=[];
Estado=[];

clear fila i n ListWaypoints Ordenes ListWaypoints Trajectorya

for i=1:1:(length(Planificador)-1),
    ListWaypoints(i,:)=(Planificador(i+1,:)-Planificador(i,:));
end
```

```
for fila=1:1:length(ListWaypoints),
    if ListWaypoints(fila,:) == [0 1]
        Ordenes(fila,1)=1;
    elseif ListWaypoints(fila,:) == [1 0]
        Ordenes(fila,1)=0;
    elseif ListWaypoints(fila,:) == [0 -1]
        Ordenes(fila,1)=3;
    elseif ListWaypoints(fila,:) == [-1 0]
        Ordenes(fila,1)=2;
    end
end

for n=1:1:(length(Ordenes)-1),
    %dreta
    if
        (Ordenes(n,1) == 2 && Ordenes(n+1,1) == 1) || (Ordenes(n,1) == 1 && Ordenes(n+1,1) == 0) || (Ordenes(n,1) == 0 && Ordenes(n+1,1) == 3) || (O
rdenes(n,1) == 3 && Ordenes(n+1,1) == 2)
        Traectoria(n,1)=0;
        %esquerra
    elseif
        (Ordenes(n,1) == 1 && Ordenes(n+1,1) == 2) || (Ordenes(n,1) == 0 && Ordenes(n+1,1) == 1) || (Ordenes(n,1) == 3 && Ordenes(n+1,1) == 0) || (O
rdenes(n,1) == 2 && Ordenes(n+1,1) == 3)
        Traectoria(n,1)=2;
        %Reves
    elseif
        (Ordenes(n,1) == 3 && Ordenes(n+1,1) == 1) || (Ordenes(n,1) == 1 && Ordenes(n+1,1) == 3) || (Ordenes(n,1) == 0 && Ordenes(n+1,1) == 2) || (O
rdenes(n,1) == 2 && Ordenes(n+1,1) == 0)
        Traectoria(n,1)=3;
        %Seguir
    elseif
        (Ordenes(n,1) == 0 && Ordenes(n+1,1) == 0) || (Ordenes(n,1) == 1 && Ordenes(n+1,1) == 1) || (Ordenes(n,1) == 2 && Ordenes(n+1,1) == 2) || (O
rdenes(n,1) == 3 && Ordenes(n+1,1) == 3)
        Traectoria(n,1)=1;
    end
end
Traectoria(n+1,1)=4;
```

```

clear fila i n ListWaypoints Ordenes

disp('Choose the method that would you like to use:')
disp('1 -> Closed Loop    In constant movement')
disp('2 -> Open Loop     In pause movement')
Metodo=input('');

if Metodo==1
while (Bumper_value(BumperId) ~= 1)
end;

while (Bumper_value(BumperId) == 1)
    State=0;
    waypoint=1;
    tStart=tic;
    L=0.6;
end;

while (Bumper_value(BumperId) ~= 1)
    if ~(Camera_setStreaming(CameraId, 1) == 1)
        disp('Camera_setStreaming failed');
    end;

    if (Camera_grab(CameraId) == 1)
        img = Camera_getImage( CameraId );
        [d_ver,theta_ver,d_hor,theta_hor] = XavmesuraDistanciaARuta( img, R, t, KK, [2 1 2] );

        if isnan(d_hor)==1&&isnan(theta_hor)==1&&State==0
            d_ver=str2double(num2str(d_ver));
            theta_ver=str2double(num2str(theta_ver));
            gamma=V*(2/L^2)*(d_ver*0.001)*cosd(theta_ver)-sqrt(L^2-(d_ver*0.001)^2)*sind(theta_ver);
            Step=((1/0.05)*T*[0;V;gamma])*(27/0.1);
            Motor_setSetPointSpeed( MotorId0, Step(1,1));
            Motor_setSetPointSpeed( MotorId1, Step(2,1));
            Motor_setSetPointSpeed( MotorId2, Step(3,1));
            d_ver_exp=[d_ver_exp;d_ver];

```



```
d_hor_exp=[d_hor_exp;d_hor];
theta_ver_exp=[theta_ver_exp;theta_ver];
theta_hor_exp=[theta_hor_exp;theta_hor];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
Estado=[Estado;State];

elseif Traectoria(waypoint,1)==4
    time=tic;
    Temps=toc(time);
    while Temps<1
        gamma=V*(2/L^2)*(d_ver*0.001)*cosd(theta_ver)-sqrt(L^2-(d_ver*0.001)^2)*sind(theta_ver);
        Step=(1/0.05)*T*[0;V;gamma]*(27/0.1);
        Motor_setSetPointSpeed(MotorId0, Step(1,1));
        Motor_setSetPointSpeed(MotorId1, Step(2,1));
        Motor_setSetPointSpeed(MotorId2, Step(3,1));
        d_ver_exp=[d_ver_exp;d_ver];
        d_hor_exp=[d_hor_exp;d_hor];
        theta_ver_exp=[theta_ver_exp;theta_ver];
        theta_hor_exp=[theta_hor_exp;theta_hor];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        Estado=[Estado;State];
        Temps=toc(time);
    end
    break

elseif isnan(d_hor)==0&&isnan(theta_hor)==0&&State==0
    State=1;
```

```
elseif State==1
    %Dreta
    if Traectoria(waypoint,1)==0
        if d_hor>0&&theta_hor>0
            V=0.05;
            d_hor=str2double(num2str(d_hor));
            theta_hor=str2double(num2str(theta_hor));
            if isnan(d_hor)==0&&isnan(theta_hor)==0
                time=tic;
                Temps=toc(time);
                while Temps<Periode/1.2
                    gamma=V*(2/L^2)*(d_hor*0.001)*cosd(theta_hor)-sqrt(L^2-
(d_hor*0.001)^2)*sind(theta_hor);

                    Step=(1/0.05)*T*[0;V;gamma]*(27/0.1);
                    Motor_setSetPointSpeed( MotorId0, Step(1,1));
                    Motor_setSetPointSpeed( MotorId1, Step(2,1));
                    Motor_setSetPointSpeed( MotorId2, Step(3,1));
                    Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
                    Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
                    Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
                    tElapsed=toc(tStart);
                    WorkTime=[WorkTime;tElapsed];
                    d_ver_exp=[d_ver_exp;d_ver];
                    d_hor_exp=[d_hor_exp;d_hor];
                    theta_ver_exp=[theta_ver_exp;theta_ver];
                    theta_hor_exp=[theta_hor_exp;theta_hor];
                    Estado=[Estado;State];
                    Temps=toc(time);
                end
            end
            V=0.15;
            State=0;
            waypoint=waypoint+1;
        else
            V=0.05;
            d_hor=-1*str2double(num2str(d_hor));
            theta_hor=-1*str2double(num2str(theta_hor));
```

```
if isnan(d_hor)==0&&isnan(theta_hor)==0
    time=tic;
    Temps=toc(time);
    while Temps<Periode/1.2
        gamma=V*(2/L^2)*(d_hor*0.001)*cosd(theta_hor)-sqrt(L^2-
(d_hor*0.001)^2)*sind(theta_hor);

        Step=((1/0.05)*T*[0;V;gamma])*(27/0.1);
        Motor_setSetPointSpeed( MotorId0, Step(1,1));
        Motor_setSetPointSpeed( MotorId1, Step(2,1));
        Motor_setSetPointSpeed( MotorId2, Step(3,1));
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        d_ver_exp=[d_ver_exp;d_ver];
        d_hor_exp=[d_hor_exp;d_hor];
        theta_ver_exp=[theta_ver_exp;theta_ver];
        theta_hor_exp=[theta_hor_exp;theta_hor];
        Estado=[Estado;State];
        Temps=toc(time);
    end
end
V=0.15;
State=0;
waypoint=waypoint+1;
end
%Esquerra
elseif Traectoria(waypoint,1)==2
    if d_hor<0&&theta_hor<0
        V=0.05;
        d_hor=str2double(num2str(d_hor));
        theta_hor=str2double(num2str(theta_hor));
        if isnan(d_hor)==0&&isnan(theta_hor)==0
            time=tic;
            Temps=toc(time);
            while Temps<Periode/1.15
```

```
(d_hor*0.001)^2)*sind(theta_hor);

gamma=V*(2/L^2)*(d_hor*0.001)*cosd(theta_hor)-sqrt(L^2-
(d_hor*0.001)^2)*sind(theta_hor);

Step=(1/0.05)*T*[0;V;gamma]*(27/0.1);
Motor_setSetPointSpeed( MotorId0, Step(1,1));
Motor_setSetPointSpeed( MotorId1, Step(2,1));
Motor_setSetPointSpeed( MotorId2, Step(3,1));
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
d_ver_exp=[d_ver_exp;d_ver];
d_hor_exp=[d_hor_exp;d_hor];
theta_ver_exp=[theta_ver_exp;theta_ver];
theta_hor_exp=[theta_hor_exp;theta_hor];
Estado=[Estado;State];
Temps=toc(time);

end
end
V=0.15;
State=0;
waypoint=waypoint+1;
else
V=0.05;
d_hor=-1*str2double(num2str(d_hor));
theta_hor=-1*str2double(num2str(theta_hor));
if isnan(d_hor)==0&&isnan(theta_hor)==0
time=tic;
Temps=toc(time);
while Temps<Periode/1.15
gamma=V*(2/L^2)*(d_hor*0.001)*cosd(theta_hor)-sqrt(L^2-
(d_hor*0.001)^2)*sind(theta_hor);

Step=(1/0.05)*T*[0;V;gamma]*(27/0.1);
Motor_setSetPointSpeed( MotorId0, Step(1,1));
Motor_setSetPointSpeed( MotorId1, Step(2,1));
Motor_setSetPointSpeed( MotorId2, Step(3,1));
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
```

```
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
d_ver_exp=[d_ver_exp;d_ver];
d_hor_exp=[d_hor_exp;d_hor];
theta_ver_exp=[theta_ver_exp;theta_ver];
theta_hor_exp=[theta_hor_exp;theta_hor];
Estado=[Estado;State];
Temps=toc(time);

    end
end
V=0.15;
State=0;
waypoint=waypoint+1;
end

%Reves
elseif Traectoria(waypoint,1)==3
    time=tic;
    Temps=toc(time);
    while Temps<Periode/2.35
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        Temps=toc(time);
    end
    time=tic;
    Temps=toc(time);
    while Temps<Periode
        Step=(1/0.05)*T*[0;0;StepOP])*(27/0.1);
        Motor_setSetPointSpeed(MotorId0, Step(1,1));
        Motor_setSetPointSpeed(MotorId1, Step(2,1));
        Motor_setSetPointSpeed(MotorId2, Step(3,1));
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
```

```

        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        Temps=toc(time);
    end
    State=0;
    waypoint=waypoint+1;

%Seguir
elseif Traectoria(waypoint,1)==1
    time=tic;
    Temps=toc(time);
    while Temps<Periode/2.6
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        d_ver_exp=[d_ver_exp;d_ver];
        d_hor_exp=[d_hor_exp;d_hor];
        theta_ver_exp=[theta_ver_exp;theta_ver];
        theta_hor_exp=[theta_hor_exp;theta_hor];
        Temps=toc(time);
    end
    State=0;
    waypoint=waypoint+1;
end

end
end
end

elseif Metodo==2
while (Bumper_value(BumperId) ~= 1)
end;

```

```
while (Bumper_value(BumperId) == 1)
    State=0;
    waypoint=1;
    tStart=tic;
    L=0.65;
end

while (Bumper_value(BumperId) ~= 1)
    if ~(Camera_setStreaming(CameraId, 1) == 1)
        disp('Camera_setStreaming failed.');
```

```
    end;

    if (Camera_grab(CameraId) == 1)
        img = Camera_getImage( CameraId );
        [d_ver,theta_ver,d_hor,theta_hor] = XavmesuraDistanciaARuta( img, R, t, KK, [2 1 2] );

        if isnan(d_hor)==1&&isnan(theta_hor)==1&&State==0
            d_ver=str2double(num2str(d_ver));
            theta_ver=str2double(num2str(theta_ver));
            gamma=V*(2/L^2)*(d_ver*0.001)*cosd(theta_ver)-sqrt(L^2-(d_ver*0.001)^2)*sind(theta_ver);
            Step=(1/0.05)*T*[0;V;gamma]*(27/0.1);
            Motor_setSetPointSpeed( MotorId0, Step(1,1));
            Motor_setSetPointSpeed( MotorId1, Step(2,1));
            Motor_setSetPointSpeed( MotorId2, Step(3,1));
            d_ver_exp=[d_ver_exp;d_ver];
            d_hor_exp=[d_hor_exp;d_hor];
            theta_ver_exp=[theta_ver_exp;theta_ver];
            theta_hor_exp=[theta_hor_exp;theta_hor];
            tElapsed=toc(tStart);
            WorkTime=[WorkTime;tElapsed];
            Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
            Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
            Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
            Estado=[Estado;State];

        elseif Traectoria(waypoint)==4
```

```
time=tic;
Temps=toc(time);
while Temps<Periode
    gamma=V*(2/L^2)*(d_ver*0.001)*cosd(theta_ver)-sqrt(L^2-(d_ver*0.001)^2)*sind(theta_ver);
    Step=((1/0.05)*T*[0;V;gamma])*(27/0.1);
    Motor_setSetPointSpeed( MotorId0, Step(1,1));
    Motor_setSetPointSpeed( MotorId1, Step(2,1));
    Motor_setSetPointSpeed( MotorId2, Step(3,1));
    d_ver_exp=[d_ver_exp;d_ver];
    d_hor_exp=[d_hor_exp;d_hor];
    theta_ver_exp=[theta_ver_exp;theta_ver];
    theta_hor_exp=[theta_hor_exp;theta_hor];
    Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
    Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
    Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
    tElapsed=toc(tStart);
    WorkTime=[WorkTime;tElapsed];
    Estado=[Estado;State];
    Temps=toc(time);
end
break

elseif isnan(d_hor)==0&&isnan(theta_hor)==0&&State==0
    if Traectoria(waypoint+1)==Traectoria(waypoint)
        State=1;
    else
        time=tic;
        Temps=toc(time);
        while Temps<Periode/1.35
            gamma=V*(2/L^2)*(d_ver*0.001)*cosd(theta_ver)-sqrt(L^2-(d_ver*0.001)^2)*sind(theta_ver);
            Step=((1/0.05)*T*[0;V;gamma])*(27/0.1);
            Motor_setSetPointSpeed( MotorId0, Step(1,1));
            Motor_setSetPointSpeed( MotorId1, Step(2,1));
            Motor_setSetPointSpeed( MotorId2, Step(3,1));
            d_ver_exp=[d_ver_exp;d_ver];
            d_hor_exp=[d_hor_exp;d_hor];
            theta_ver_exp=[theta_ver_exp;theta_ver];
```



```
        theta_hor_exp=[theta_hor_exp;theta_hor];
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        Temps=toc(time);
    end

    time=tic;
    Temps=toc(time);
    while Temps<1
        Motor_setSetPointSpeed( MotorId0, 0);
        Motor_setSetPointSpeed( MotorId1, 0);
        Motor_setSetPointSpeed( MotorId2, 0);
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        Temps=toc(time);
    end
    State=1;
end

if State==1
    %Dreta
    if Traectoria(waypoint)==0
        time=tic;
        Temps=toc(time);
        while Temps<Periode/4.7
            Step=((1/0.05)*T*[0;0;-StepOP])*(27/0.1);
            Motor_setSetPointSpeed( MotorId0, Step(1,1));
            Motor_setSetPointSpeed( MotorId1, Step(2,1));
            Motor_setSetPointSpeed( MotorId2, Step(3,1));
```

```
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
Estado=[Estado;State];
Temps=toc(time);

end

State=0;
waypoint=waypoint+1;

%Seguir
elseif Traectoria(waypoint)==1
    time=tic;
    Temps=toc(time);
    while Temps<Periode/2
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
        Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
        Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
        tElapsed=toc(tStart);
        WorkTime=[WorkTime;tElapsed];
        Estado=[Estado;State];
        Temps=toc(time);
    end
    State=0;
    waypoint=waypoint+1;

%Esquerra
elseif Traectoria(waypoint,1)==2
    time=tic;
    Temps=toc(time);
    while Temps<Periode/4.7
        Step=(1/0.05)*T*[0;0;StepOP]*(27/0.1);
        Motor_setSetPointSpeed(MotorId0, Step(1,1));
        Motor_setSetPointSpeed(MotorId1, Step(2,1));
        Motor_setSetPointSpeed(MotorId2, Step(3,1));
        Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
```

```
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
Estado=[Estado;State];
Tems=toc(time);
end
State=0;
waypoint=waypoint+1;
%Reves
elseif Traectoria(waypoint)==3
time=tic;
Tems=toc(time);
while Tems<Periode/2.4
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
Estado=[Estado;State];
Tems=toc(time);
end
time=tic;
Tems=toc(time);
while Tems<Periode
Step=((1/0.05)*T*[0;0;StepOP])*(27/0.1);
Motor_setSetPointSpeed( MotorId0, Step(1,1));
Motor_setSetPointSpeed( MotorId1, Step(2,1));
Motor_setSetPointSpeed( MotorId2, Step(3,1));
Speed_Motor0=[Speed_Motor0;Motor_actualSpeed(MotorId0)];
Speed_Motor1=[Speed_Motor1;Motor_actualSpeed(MotorId1)];
Speed_Motor2=[Speed_Motor2;Motor_actualSpeed(MotorId2)];
tElapsed=toc(tStart);
WorkTime=[WorkTime;tElapsed];
Estado=[Estado;State];
Tems=toc(time);
end
```

```

                                State=0;
                                waypoint=waypoint+1;
                                end
                            end
                        end
                    end
                end
            end

        else
            disp('This method does not exist')
        end

        d_ver_exp=d_ver_exp*0.1;
        d_hor_exp=d_hor_exp*0.1;
        Motor_setSetPointSpeed( MotorId0, 0 );
        Motor_setSetPointSpeed( MotorId1, 0 );
        Motor_setSetPointSpeed( MotorId2, 0 );
        tElapsed=toc(tStart);

        Com_disconnect(ComId);

        Bumper_destroy(BumperId);
        Com_destroy(ComId);
        Motor_destroy(MotorId0);
        Motor_destroy(MotorId1);
        Motor_destroy(MotorId2);
        Camera_destroy(CameraId);
    end
end
```