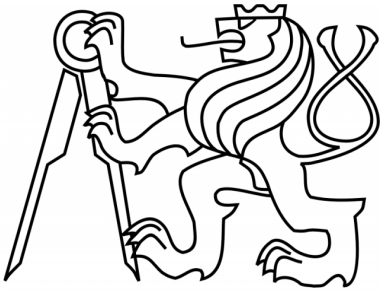


Czech Technical University In Prague

Faculty of Electrical Engineering
Dept. of Control Engineering

Master's Thesis

Rapid Prototyping of Mobile Robot Control Algorithms

Andrés España Cabrera

Supervisor: Michal Sojka, PhD.

June 2014

Declaration of Authorship

I, Andrés España Cabrera, declare that this thesis titled, 'Rapid Prototyping of Mobile Robot Control Algorithm' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Espana Cabrera Andres**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **Rapid prototyping of mobile robot control algorithms**

Guidelines:

1. Get familiar with the a mobile demo robot available at DCE and an internal communication middleware ORTE.
2. Add support for ORTE to Simulink, i.e. create the corresponding Simulink blocks and S-functions.
3. Implement the robot control algorithm in Matlab Simulink/Stateflow and use Embedded code generate C code of the algorithm. The task is to find a cylinder with a bottle and pick up the bottle.
4. Implement an algorithm for robot localization in known environment (e.g. ICP) and integrate it with the control algorithm.
5. Test and document everything.

Bibliography/Sources:

[1] Mathworks: Simulink 2013a - Developing S-functions
[2] Smolik P. et al.: ORTE user manual

Diploma Thesis Supervisor: Ing. Michal Sojka, Ph.D.

Valid until the summer semester 2014/2015

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 15, 2014

"Let the future tell the truth and evaluate each one according to his work and accomplishments. The present is theirs; the future, for which I really worked, is mine."

Nicolas Tesla

Abstract

The design of control algorithms for mobile robots has been always a big challenge, due to the complexity of control techniques and the difficulty that this represents when trying to implement them in the robots software. This thesis work presents a method to establish communication and control of a mobile robot from Matlab/Simulink platform. The robot used is an undergoing project from the Department of Control Engineering of the Czech Technical University in Prague. Furthermore, this work presents performance tests and analysis of control algorithms implemented to test communication link and controllability of the system, from which a set of conclusions and recommendations are given.

Acknowledgements

First, I would like to express my sincere gratitude towards my supervisor PhD. Michal Sojka, who gave the opportunity to undertake this project. His guidance, patience and support during this time were fundamental to complete this work.

As a scholarship holder student, I would like to thank the European Commision for their finantial support to carry out my studies as part of the Erasmus Mundus Joint European Master in Space Science and Technology.

Finally, but importantly, I would like to thank my parents, girlfriend and friends for their unending support during these years abroad. This would not have been possible without you.

Contents

List of Figures

List of Tables

Abbreviations

DCE	**D**epartment **C**ontrol **E**ngineering
ORTE	**O**pen **R**eal-**T**ime **E**thernet
RTPS	**R**eal **T**ime-**P**ublisher- **S**ubscribe
UDP	**U**ser **D**atagram-**P**rotocol

To my family and friends...

Chapter 1

Introduction

Eurobot Association is an international organization created in May 2004 and registered in France. It was born 6 years after the Eurobot Contest to structure the organization of this growing contest, and to favor the spirit of exchange and co-operation between the different organizers [4].

The robots participating in the competition are autonomous robots. Every year a different topic is set for the competition, which determines the type of tasks robots are built and programmed for. Overall, the main goal of Eurobot is to encourage interest in robotics in young people on an international scale.

CTU Dragons, representing the Department of Control Engineering of the Czech Technical University in Prague, participated in this competition for the first time in 2007 with robot *DragonBot*.

The topic of 2007 competition was *Robot Recycling Rally*. The robots design and programming were meant to sort waste. The task was to find bottles, cans and batteries, pick them up and once having identified the type of waste, place it in the proper bin [5].

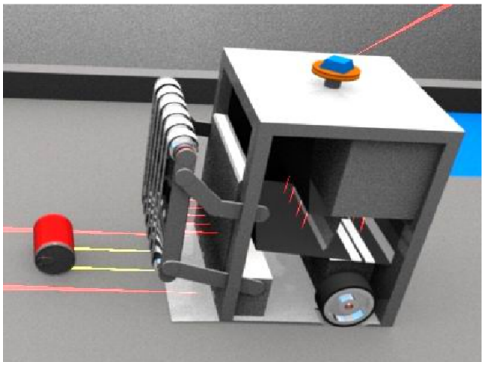


FIGURE 1.1: 3D model of DragonBot

For the construction of DragonBot knowledge and technologies from industry and others developed in the Control Department, were used. Most of the programming back then was done in C/C++ with the aid of free software tools and libraries. Components were designed in a versatile and efficient way. The main control program of the robot was built on finite state machine architecture.

One core component of the robot, which is worth mentioning due to the ultimate goal of this thesis work, is the middleware used for interaction between different components of the robot, ORTE.

ORTE is an open source implementation of Real-Time Publish-Subscribe (RTPS) communication protocol. The basics and advantages of using such a protocol, as well as the importance of it for the accomplishment of the goal of this work, will be explained ahead in this document.

1.1 Main Objective

This thesis work has as main objective, enabling rapid prototyping of new control algorithms and strategies for mobile robot, through the use of ORTE from Matlab Simulink/Stateflow platform.

Among the many advantages of using a middleware such as ORTE, that later in this document will be listed, is the fact of having a *Plug and play* connectivity. This meaning that new applications and/or services are easily detected and any of them can join or leave the network at any time without any type of reconfiguration. These applications can be built using different programming languages or platforms. At the end, ORTE will be in charge of managing the inputs of these different origin applications and translate them into data that other application, sensor or actuator will be able to interpret accordingly.

After seven years of application developing for DragonBot, there is still no method designed for applications running in Matlab/Simulink that can interact with the robot. Due to the great potential of such a platform, specially for Control Systems, designing such a method results convenient for current and future projects.

This work intends to develop this communication method and implement a controller as a way of testing the capabilities of it. The success of this project will help future projects aiming at having better and easier to design controllers.

1.2 Overview

Thesis work comprehends the following aspects:

1. Familiarization with mobile demo robot available at DCE and internal communication middleware ORTE.
2. ORTE communication through Matlab S-Functions and Simulink blocks.
3. Implementation of robot control algorithms in Matlab Simulink/Stateflow and performance testing of controllers.

1.3 Structure of Thesis

The thesis work is structured such that reader understands the development of the work and the achievement of the goals set at the beginning.

- Chapter 2 Introduces the reader with the ORTE middleware. The understanding behind it, was a key factor in future development of this work. Objectives of RTPS protocol are presented, as well as a general overview of the Publish-Subscribe architecture. Basic concepts for Real-Time applications are explained on this chapter too.

- Chapter 3 Describes how the integration of ORTE middleware was done through the use of Matlab S-Functions and blocks in Simulink. A very general overview of S-Functions is presented to the reader. The methodology followed for the design of these functions is contained in this chapter, along with descriptions of the final implemented functions.

- Chapter 4 Describes the implementation for the two algorithms to achieve control of the mobile robot. Tests and performance of both algorithms is presented to the reader, highlighting advantages, disadvantages and opportunity areas for future implementations.

- Chapter 5 Presents a summary and an analysis of the results obtained in this work. Suggested lines of research and development for future projects are also included in this chapter.

Chapter 2

Open Real-Time Ethernet Basics

2.1 ORTE Overview

The Open Real-Time Ethernet (ORTE) is an open source implementation of Real-Time Publish-Subscribe (RTPS) communication protocol [1], developed at CTU.

RTPS is an application layer protocol targeted to real-time communication applications. This protocol is built on the top of standard *User Datagram Protocol* (UDP) stack. Since there are many TCP/IP stack implementations under many operating systems and RTPS protocol does not have any other special HW/SW requirements, it should be easily ported to many HW/SW target platforms. Because it uses only UDP protocol, it retains control of timing and reliability [1].

2.1.1 RTPS Protocol objectives

Among the main objectives to use RTPS protocol [2], that in this case are relevant for this project we find:

- **Plug and play connectivity.** New applications and services are automatically discovered and applications can join and leave the network at any time without the need for reconfiguration.
- **Performance and Quality.** Performance and quality-of-service properties to enable best-effort and reliable publish-subscribe communications for real-time applications over standard IP networks.

- **Modularity.** In order to allow simple devices to implement a subset and still participate in the network.
- **Scalability.** This will enable systems to potentially scale to very large networks.

2.2 Publish-Subscribe Architecture

The publish-subscribe architecture has as ultimate goal simplifying data distribution from one source to many recipients. A **publisher** does not have to have any knowledge of the number or location of subscribers [5].

Subscribers on the other side, receive the data in an anonymous way, without knowing anything about the publisher. As well is important to mention that an application can be a publisher and a subscriber at the same time. Having these features available, the developer simply writes a given application to send or receive the data.

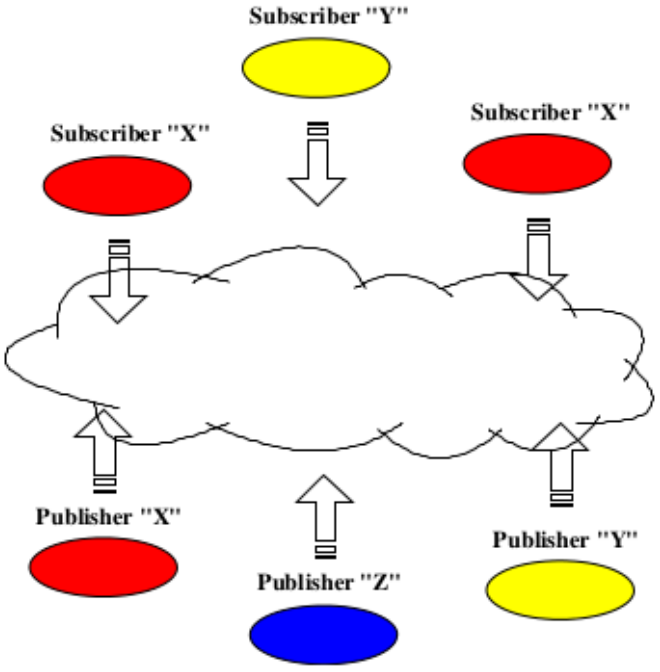


FIGURE 2.1: Publish-Subscribe Architecture [1]

The publish-subscribe services are typically made available to applications through middleware, that sits on top of the operating system network interface and presents an application programming interface [5]. In the case of *DragonBot*, the middleware used as explained before is ORTE.

The communication process occurs in three simple steps [1]:

- Publisher declares intent to publish a publication.
- Subscriber declares interest in a publication.
- Publisher sends a publication issue.

2.2.1 Publish-Subscribe in Real Time Applications

The publish-subscribe explained in previous section has several useful features for real-time applications [1]:

- This type of architecture provides an efficient and fast way of distributing data, due to its efficiency in bandwidth and latency for periodic data exchange.
- Since it provides a high connectivity, this type of architecture is ideal in applications where publishing and subscribing applications are added and removed dynamically .

Real-time applications often require [6]:

- **Delivery timing control:** This type of applications must know when data is delivered and how long it remains valid.
- **Reliability control:** Since reliable delivery conflicts with deterministic timing, each time-constrained application needs to specify its particular reliability characteristics (for example, how long it is willing to wait).
- **Fault-tolerance:** The communications layer should not introduce any single-node points of failure. Moreover, support for "hot standby" or backup data production is often a requirement.
- **Selective degradation:** Each real-time logical data-channel must be protected from the others. The performance of a channel should not be affected by other channels slowing due to dropouts, network congestion, receiver CPU overload and so on.

2.3 Real Time Publish-Subscribe Model

The Real-Time Publish-Subscribe (RTPS) communications model includes protocols to handle the administrative chores underlying plug-and play and fault-tolerant distributed system configurations [6]. Among the features included in the model is the use of timing parameters for both publishers and subscribers, to have control over different types of data flows and in this way achieve the performance, robustness and reliability objectives. The following subsections describe the parameters/structure for publishing or subscribing to data within the network.

2.3.1 Publication Parameters

Each publication is characterized by four parameters [1]:

- *Topic* → Label that identifies each flow of data
- *Type* → Data format
- *Strength* → Publishers weight or priority relative to to other publishers sharing the same topic.
- *Persistence* → Amount of time each publication will be valid.

These parameters are of importance for all subscribers waiting for the data they require for their own processes. Figure 2.2 depicts the way a subscriber assesses among all available publications, using the properties above explained.

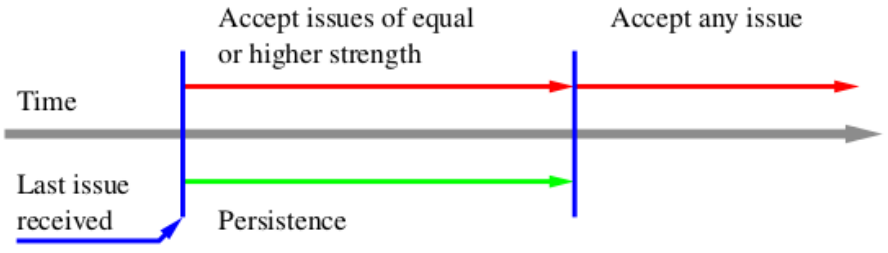


FIGURE 2.2: Publication Arbitration [1]

In the case where there are multiple publishers sending the same publication, the subscriber accepts the issue if its strength is greater than the last-received issue or if the last issues persistence has expired. Typically, a publisher that sends issues with a period of length T will set its persistence to some time Tp where Tp > T. Thus, as long as the strongest publisher is functional, its issue will take precedence over a publication issue of lesser strength [1].

In the case where the strongest publisher stops sending issues (willingly or due to a failure), other publisher(s) sending issues for the same publication will take over after Tp elapses. Through this mechanism an inherently robust, quasi-stateless communications channel is established between the then-strongest publisher of a publication and all of its subscribers [1].

2.3.2 Subscription Parameters

Each subscription is described through four parameters [1]:

- *Topic & Type* → Label that identifies each flow of data and data format respectively, as in parameters for publications.
- *Minimum Separation* → Period of time during which no new issues are accepted for that specific subscription.
- *Deadline* → This parameter specifies how long the subscriber is willing to wait for the next issue.

As seen in Figure 2.3, in one hand the *Minimum Separation* protects for instance a slow subscriber against a fast publications. On the other the *Deadline* provides a fixed amount of time, that in the case of communication delays could be used to take proper corrective actions.

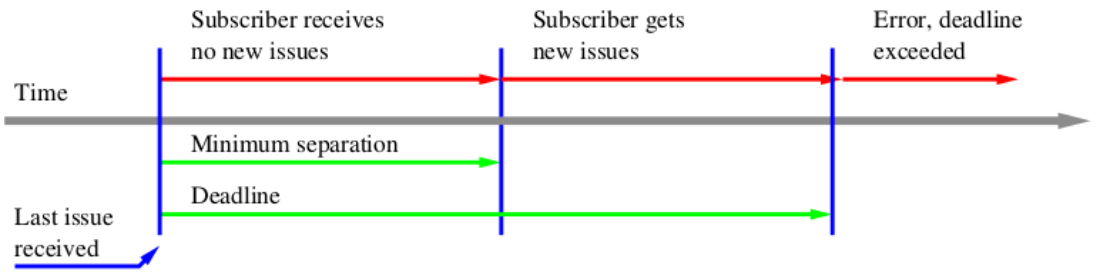


FIGURE 2.3: Subscription Issue Structure [1]

Chapter 3

Matlab S-Functions & ORTE

The communication with robot *DragonBot* was done through the implementation of Level-2 C S-Functions in Matlab. These functions were used later in the corresponding blocks in Simulink.

The first section in this chapter, intends to give a very general overview about these functions, focusing on the specific aspects that were important for the accomplishment of the objectives in this project. For further technical details please refer to [7].

In the same way, section 3.2 will introduce the reader with the implemented functions, important parameters and general strategy followed.

3.1 S-Function Callback Methods

An S-function is a computer language description of a Simulink block written in MATLAB, C, C++, or Fortran. C, C++, and Fortran S-functions are compiled as MEX files using the *mex* utility [7]. The S-functions for this project, were developed in C.

Like a Level-2 MATLAB S-function, a MEX S-function consists of a set of callback methods that the Simulink engine invokes to perform various block-related tasks during a simulation. The engine directly invokes MEX S-function routines instead of using function handles as with MATLAB S-functions. Because the engine invokes the functions directly, MEX S-functions **must follow standard naming conventions specified by the S-function API** [7].

C MEX S-functions must implement the following callback methods [7]:

- *mdlInitializeSizes* - Specifies the sizes of various parameters in the SimStruct, such as the number of output or input ports for the block. This method was used to define the sizes of the output ports in case of Subscribers and input ports for Publishers.

- *mdlInitializeSampleTimes* - Specifies the sample time(s) of the block. For the case of publishers the INHERITED_SAMPLE_TIME option was used. For subscribers it was defined according to the type of data subscribing to.

- *mdlOutputs* - Calculates the output of the block. For publishers, in this method the corresponding assignation and issuing of data was done (ORTEPublication-Send). In the case of subscribers only the assignation of data from subscription was assigned to the defined outputs.

- *mdlTerminate* - Performs any actions required at the termination of the simulation. If no actions are required, this function can be implemented as a stub. In this case for all publishers or subscribers no action was required, and this method was left empty.

Besides these callback methods, there are others defined as *optional*, that could be used by the developer in case of needing to fulfill specific requirements for more complex applications. Since this project required of a method that could run only at the beginning of the model execution, due to ORTE initialization conditions, the *optional* method *mdlStart* was used. Please refer to literature for further knowledge on all available optional methods.

3.2 S-Functions Implementation

Implementation of these functions, required a good understanding on both ORTE protocol and S-Function's basic concepts. Implementation was divided into Publishers and Subscribers, due to differences in the rules and methods to issue or receive data.

3.2.1 Publishers

Publisher S-Functions implemented, will be in charge of publishing the robot data coming from Simulink. This data, once calculated in a Simulink block diagram, is sent through ORTE to the robot, to make the Robot perform a certain task.

Figure 3.1 depicts the general process taken in every S-Function developed. The main differences between each data publisher, lie on the number of input ports and the width of them. As well as in the *Topic, Type, Strength* and *Persistence* parameters, that will have to be modified depending on the data wishing to issue.

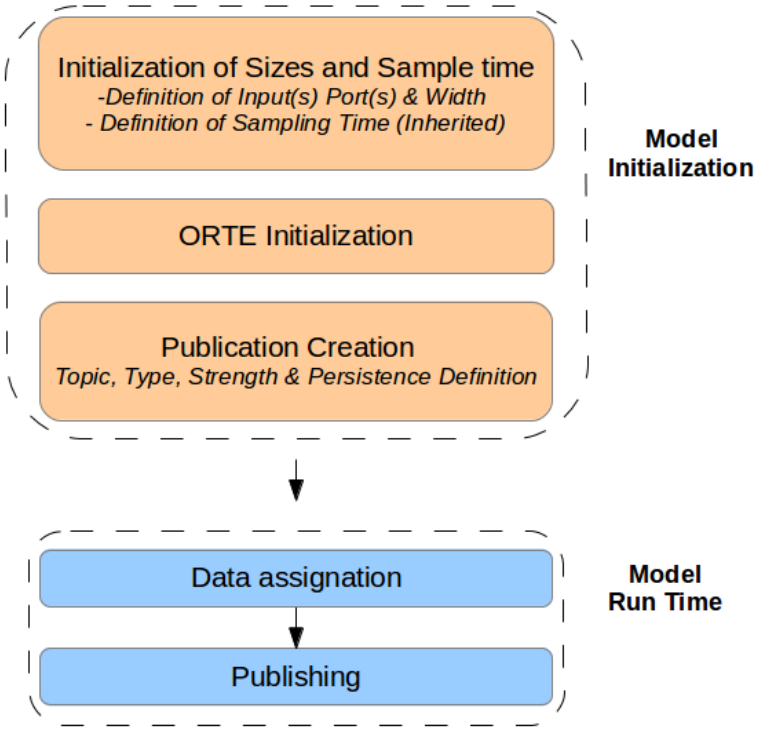


FIGURE 3.1: Publisher S-Function Diagram

Tables 3.1, 3.2 and 3.3 enlist a summary of the S-Function parameters for the functions developed, aligned with the objectives of this project. The *Strength* and the *Persistence* parameters are omitted in these tables; all functions were designed and tested with the same values of 1 and 3 seconds respectively.

PublisherRef	
Action	Publishes Reference Robot Position
Topic	*ref_pos*
Type	*robot_pos*
Input Port Width	3
Inputs	x, y and *phi* reference position of robot

TABLE 3.1: Reference Robot Position Publisher – S-Function parameters

Publisher	
Action	Publishes Robot Best Estimated Position
Topic	*est_pos_best*
Type	*robot_pos*
Input Port Width	3
Inputs	x, y and *phi* values

TABLE 3.2: Robot Estimated Best Position Publisher – S-Function parameters

PublishSpeed	
Action	Publishes Reference Motion
Topic	*motion_speed*
Type	*motion_speed*
Input Port Width	2
Inputs	*Left* and *right* robot wheel speeds

TABLE 3.3: Robot Motion Speed Publisher - S-Function parameters

**These S-Functions were tested individually with satisfactory results. For controllability tests (further explained in Chapter 4), only PublisherRef and Publisher functions were used. PublishSpeed function was not used due to limitations of simulator.*

3.2.2 Subscribers

Subscriber S-Functions implemented, will provide a given Simulink design with data coming from the robot software through ORTE. This data after being processed by the function itself, will be ready to be used as part of a design within Simulink platform.

Figure 3.2 depicts the general process taken in every subscriber S-Function developed. Unlike the publisher functions, an auxiliary function is required to process the data being published by other applications within the network. This function is needed as an intermediate step between data being published and data being subscribed to. Through array manipulation, the function stores published data, to later on assign it to the outputs of the Subscriber.

Similarly as for publisher functions, the main differences between each data subscriber function, lie in the the number of output ports and the width of them. As well as in the *Topic, Type, Minimum Separation* and *Deadline* parameters, that will have to be modified depending on the data wishing to be subscribed to.

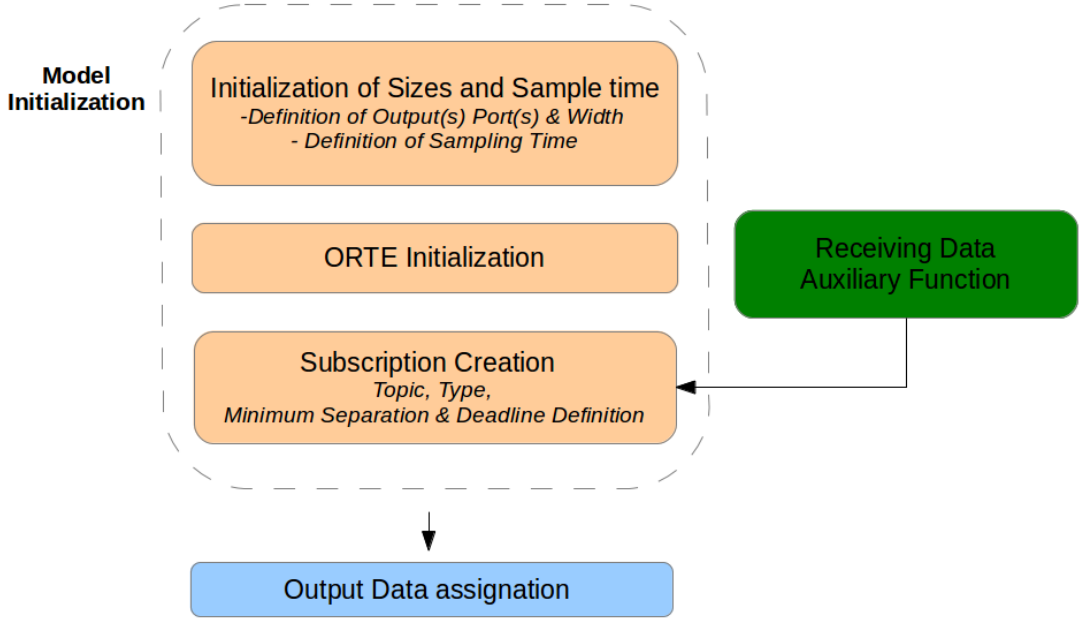


FIGURE 3.2: Subscriber S-Function Diagram

Tables 3.4, 3.5 and 3.6 enlist a summary of the subscription S-Function parameters for the functions developed. Its worth mentioning that the *Minimum Separation* and *Deadline* parameters are omitted in these tables; all functions were designed and tested with the same values of 300 and 0 seconds respectively.

Subscriber	
Action	Subscribes to Motion data
Topic	*motion_speed*
Type	*motion_speed*
Output Port Width	2
Outputs	*Left* and *right* robot wheel speeds

TABLE 3.4: Robot Motion Data subscriber - S-Function parameters

SubscriberPos	
Action	Subscribes to Robot Estimated Position data
Topic	*est_pos_best*
Type	*robot_pos*
Output Port Width	3
Outputs	x, y and *phi* robot position data

TABLE 3.5: Robot Estimated Best Position Subscriber - S-Function parameters

SubscriberHokuyo	
Action	Subscribes to Laser Sensor (Hokuyo) data
Topic	*hokuyo_scan*
Type	*hokuyo_scan*
Output Port Width	681
Outputs	Vector of values containing distances in mm to possible obstacles in trajectory

TABLE 3.6: Laser Sensor "Hokuyo" Subscriber - S-Function parameters

**These S-Functions were tested individually with satisfactory results. For controllability tests (further explained in Chapter 4), only SubscriberPos function was used to close the control loop. Subscriber function couldn't be used due to limitations of simulator. SubscriberHokuyo performs well with simulator, but further work is required to process data coming from laser sensor.*

Chapter 4

S-Functions Test -*Mobile Robot Control*

4.1 Control Setup

The design of control loops for a mobile robot, to perform specific tasks is always a big challenge. An easy and friendly way to design and test different types of control techniques is of high importance to develop more efficient and complex algorithms.

Now that a communication channel is set between a powerful platform like Matlab/ Simulink and the mobile robot through the use of S-Functions described in previous chapter, its necessary to test the behavior of these functions. Running them on a close-loop scheme, will provide us with an analysis of the performance of such functions and evaluate the viability of their use in future projects.

Figure 4.1 depicts the high-level design of the controllers implemented in this project to test the S-Functions. From the figure it can be seen that the system will *Subscribe* to the current status of the robot (1). This status could contain wheel velocities data or a given position defined by x, y and ϕ. (In the present work, due to limitations of the simulator used for the project, as it will be explained in section 4.2, only position publishers and subscribers were used).
This data provided by the S-Function, will be then compared to the reference(2) given, in order to calculate errors in position or speed and provide reference velocities to the controller(3).

The controller (4) will process these inputs and will try to minimize the errors in the system by two control signals: v and w, linear and angular velocity respectively.

These control signals need to be processed taking into account the kinematics of the robot. Block (5) will transform this v and w into a position or a left/right wheel velocity.

Finally, the loop is closed by simply *Publishing* (6) this data through ORTE. This data will be interpreted by a simulator or the real robot software and transformed into a control action.

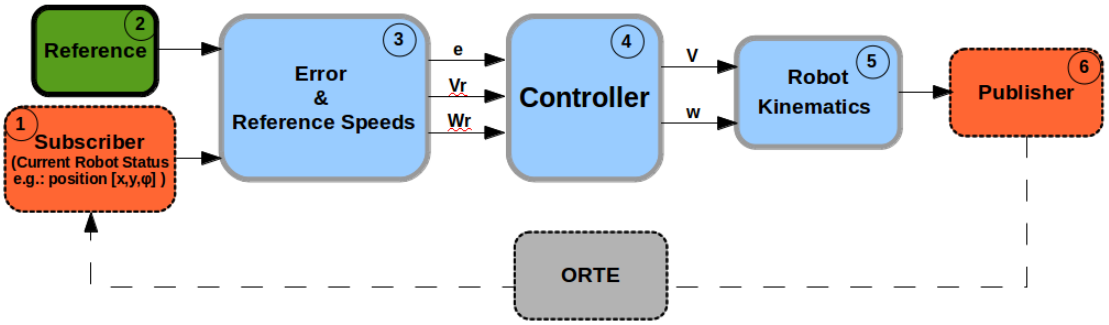


FIGURE 4.1: High Level Control Diagram

4.2 Simulator - *Robomon*

Since the mobile robot has been a project ongoing for several years now, several applications have been developed to support the design and test tasks the robot has faced. It's clear that due to the nature of a project like this, a simulator that could help visualize the behavior and/or parameters related to robot's performance is essential for design and testing activities.

Robomon is this simulator, developed by CTU Control Department, that has evolved during the last several years, trying to meet new and more complex requirements. Aligned with the main goal of this project, it was used to test the functionality of S-Functions and Controllers developed.

Figure 4.2 shows the main graphical interface of *Robomon*. The mobile robot is simulated by the boxes with arrows, where the uncolored box represents the reference position and the gray box its current position.

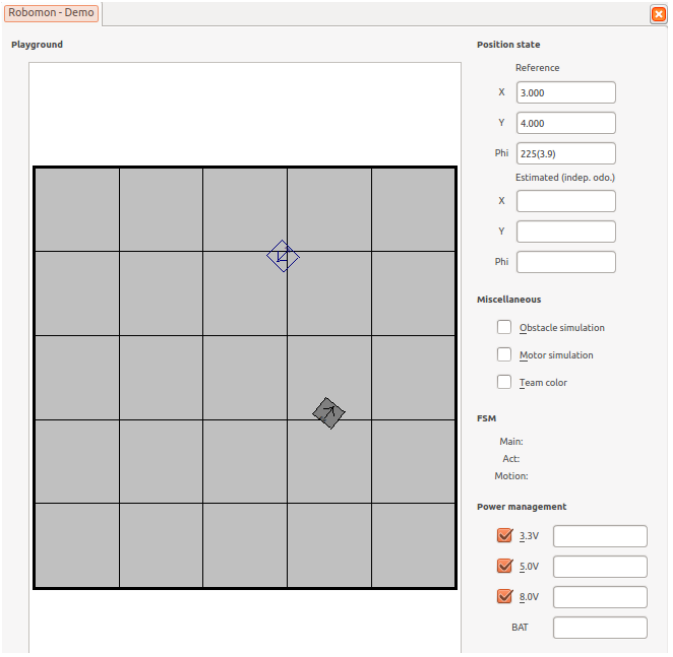


FIGURE 4.2: *Robomon* - Mobile Robot Simulator

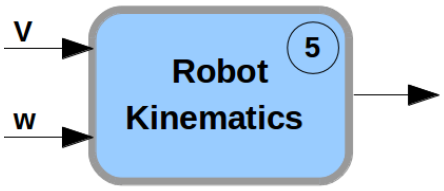
This simulator has the capability to respond to published data through ORTE, simulating and displaying graphically a given trajectory. Besides, it should be able to publish robot position, wheel motion, among other data, that could be used by any given subscriber within the network.

For the purpose of this work, *Robomon* is receiving data, coming from S-Functions *PublisherRef* and *Publisher* (Tables 3.1 and 3.2). This data will be used to simulate the reference and the current positions of the robot respectively.

During the development of this project it was found that *Robomon* has not yet implemented the functionality of publishing position of the robot, nor wheel velocities. For this reason, *SubscriberPos* (Table 3.5) S-Function is used, in order to subscribe to the data being published by S-Function *Publisher*.

Once *Robomon* has the capability to publish position or motion data, *Subscriber* function (Table 3.4) could be used or *SubscriberPos* could be modified to interact directly with data being published.

4.3 Robot Kinematics



The mobile robot position can be represented in global Cartesian coordinate system as seen in Figure 4.3.

The figure shows how its position is described by x, y and ϕ values, where the sub-index c is an abbreviation for *current*, to represent the robot's current position [9].

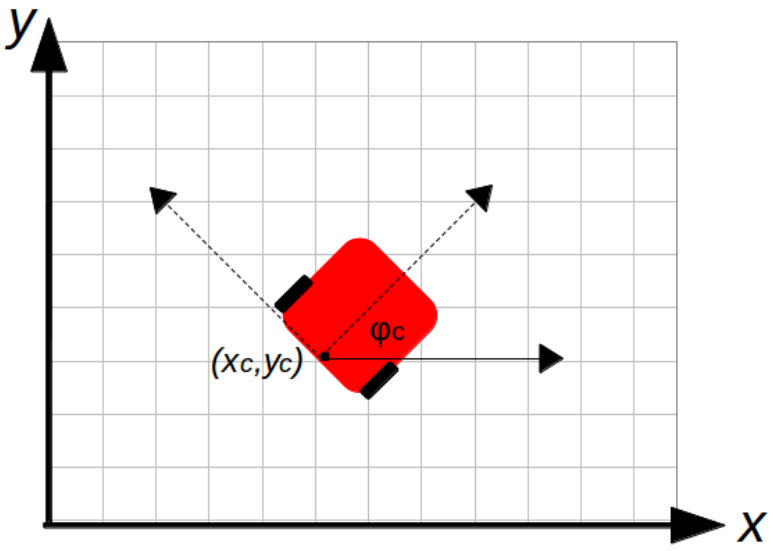


FIGURE 4.3: Mobile Robot

The motion of the robot can be controlled by its linear v and angular ω velocities. These velocities are described by [10]:

$$v = \frac{v_{left} + v_{right}}{2} \tag{4.1}$$

$$\omega = \frac{v}{r} \tag{4.2}$$

Where r stands for the radius of the robot wheels. Since the robot has two, the velocity for each wheel can be calculated by the following equations:

$$v_{left} = v - \frac{r}{2w} \tag{4.3}$$

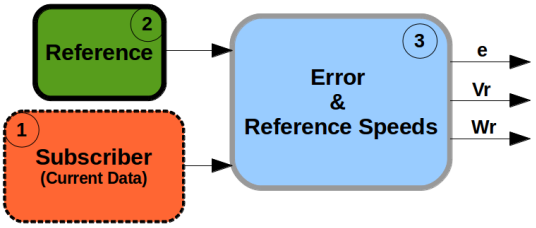
$$v_{right} = v + \frac{r}{2w} \tag{4.4}$$

From this, the robot's kinematics (motion) model are described by the following states [10]:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} cos(\phi) & 0 \\ sin(\phi) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.5}$$

This model will be used to calculate the errors in position, that will after serve as inputs for controllers.

4.4 Error Definition & Reference Velocities



After having defined the kinematics of the robot, the errors in position can be defined considering robot's current position and any reference given as it is detailed in Figure 4.4.

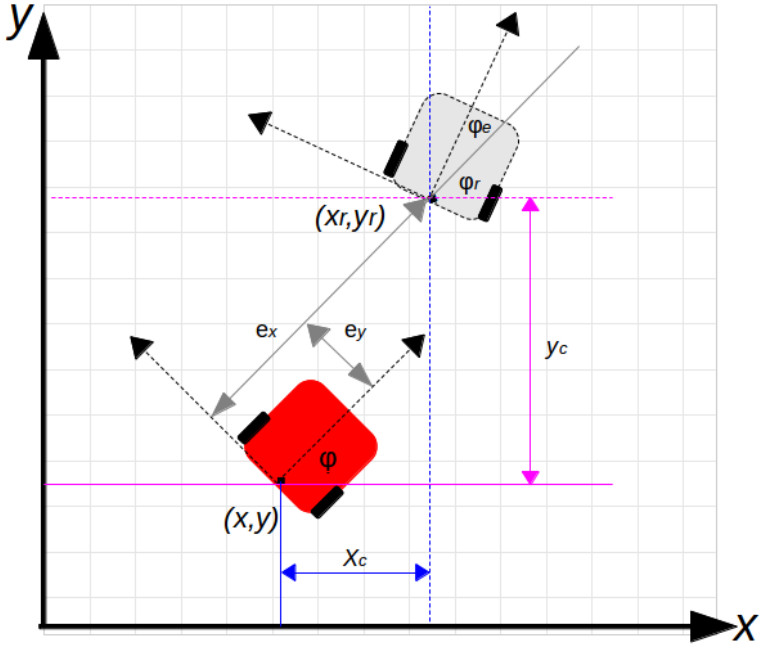


FIGURE 4.4: Error Geometry from Current to Reference Position

Using previous figure as an aid, three different type of errors in position can be defined and described by the following equations [9]:

- *Driving Error*

$$e_D = cos(\phi_c) \cdot (x_r - x_c) + sin(\phi_c) \cdot (y_r - y_c) \tag{4.6}$$

- *Lateral Error*

$$e_L = -sin(\phi_c) \cdot (x_r - x_c) + cos(\phi_c) \cdot (y_r - y_c) \tag{4.7}$$

- *Orientation Error*

$$e_\phi = \phi_r - \phi_c \tag{4.8}$$

Where sub-indexes r and c correspond to $reference$ and *current* positions, respectively. These errors along with reference linear and angular velocities, will be used as the input for the controllers that will be presented on the following sections.

4.4.1 Reference Velocities v_r & ω_r

The reference linear (v_r) and angular (ω_r) velocities are necessary for the controller to calculate the corresponding control action. These velocities will be calculated by comparing the reference position (subindex r) and the current position (subindex c) of the robot as calculated in equations 4.9 and 4.10, where the term T_s stands for the sample time in the system. In the case of a tracker robot, v_r and w_r would depend on the reference trajectory (at samples k and $k-1$) only and not on the robot's position [9].

$$v_r = \frac{\sqrt{(x_r - x_c)^2 - (y_r - y_c)^2}}{T_s} \tag{4.9}$$

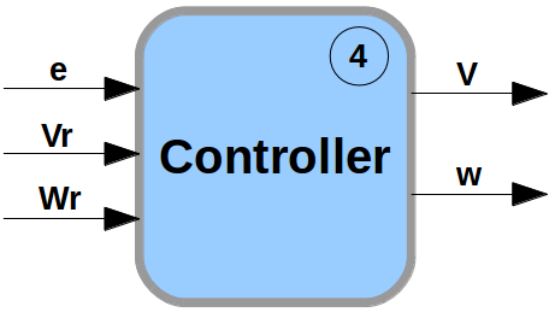
$$\omega_r = \frac{\phi_r - \phi_c}{T_s} \tag{4.10}$$

In the potential scenario where the robot needs to rotate for more than a circle (ϕ from $2\pi \rightarrow 0$), the control scheme is protected, assuming $\omega_r = \omega_c$, if $|\omega_c| > 2\pi$ as suggested by Karer, et al [10].

4.5 Controller Proposed: *Kanayama*

4.5.1 Design

The first controller implemented to test the S-Functions developed, follows the research done by Kanayama [8], in which a stable control rule is found using a Lyapunov function [8]. The analysis for the final derivation of such a controller can be found on the literature referenced in [8].



Its worth mentioning that the analysis from which the outcome is the previous controller, assumes a perfect velocity tracking.

The control law proposed is as follows:

$$v = v_r \cdot cos(e_\phi) + k_1 \cdot e_D \tag{4.11}$$

$$w = w_r + v_r(k_2 \cdot e_L + k_3 \cdot sin(e_\phi)) \tag{4.12}$$

Equation 4.11, shows how the controller is fixing for orientation and driving errors. The first part of equation $v_r \cdot cos(e_\phi)$, will provide the controller with an extra amount of energy, to fix for e_ϕ in combination with equation 4.12. The proportional controller k_1 will fix for any given driving error, in an effort to assure an optimal minimization of error.

Equation 4.12 in combination with equation 4.11 will eliminate both, e_L and e_ϕ with the aid of gains k_2 and k_3. The w_r term will assure the robot angular velocity not to be affected by potential noise [10].

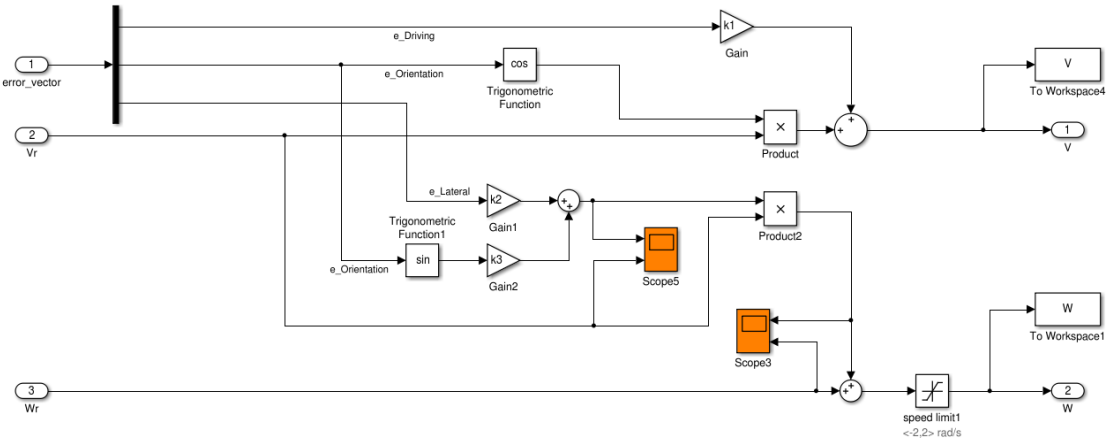


FIGURE 4.5: Controller *Kanayama* Implemented in Simulink

4.5.2 Gain Optimization

Suitable values for the gains in our controller, will depend on the reference point or a trajectory the robot is designed to follow [10]. In this case the mobile robot is configured to minimize position errors and not follow a trajectory.

Controller initial constants were found with the aid of an unconstrained nonlinear optimization, from point $(1, 1, 0°)$ to $(2, 3, 235°)$, as it can be seen in Figure 4.16.

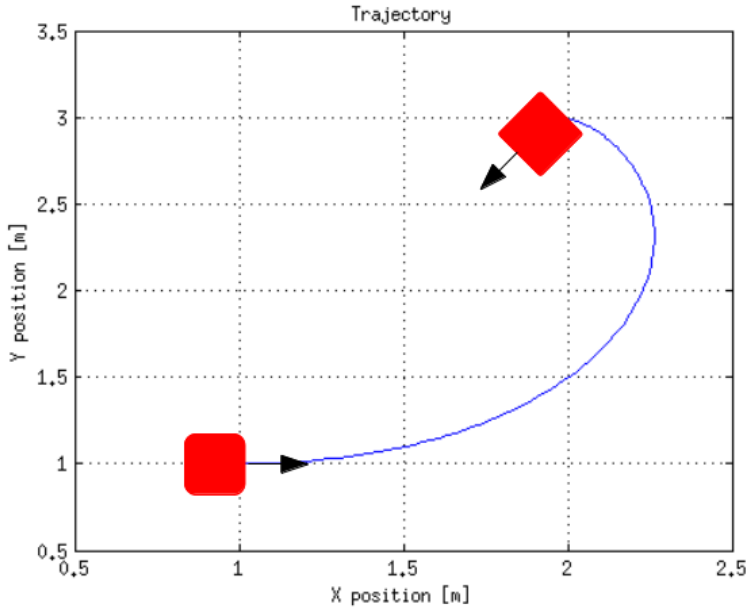


FIGURE 4.6: Initial and Final Position of Mobile Robot

Figure 4.7 shows the code used for this optimization. The function *criterial* represents the criterion chosen to minimize the error and therefore provide the best possible path between the two points. This criterion is described by equation 4.13 [10]:

$$c = \sum e^2(k) = \sum (x_c - x_r)^2 + (y_c - y_r)^2 + 0.2 \cdot (\phi_c - \phi_r)^2 \tag{4.13}$$

```
%% Controller #1
k1_opt_0 = 0.5;   %starting constants
k2_opt_0 = 2;
k3_opt_0 = 0.5;

options = optimset('fminsearch');
options = optimset(options,'Display','iter','TolFun',1e-3,'TolX',1e-1,'MaxIter',500);
[K_s,fval,exitflag] = fminsearch('criterial',[k1_opt_0 k2_opt_0 k3_opt_0],options);

k1=K_s(1,1);
k2=K_s(1,2);
k3=K_s(1,3);
```

FIGURE 4.7: Unconstrained Nonlinear Optimization

Having optimized for this designed trajectory these are the final gains used to test the performance of the controller:

$$k_1 = 2.8691 \qquad k_2 = -0.3958 \qquad k_3 = -0.8516$$

The final implemented Simulink diagram is depicted in Figure 4.8. As it can be seen, this implementation follows the High-Level description done in section 4.1 (Figure 4.1).

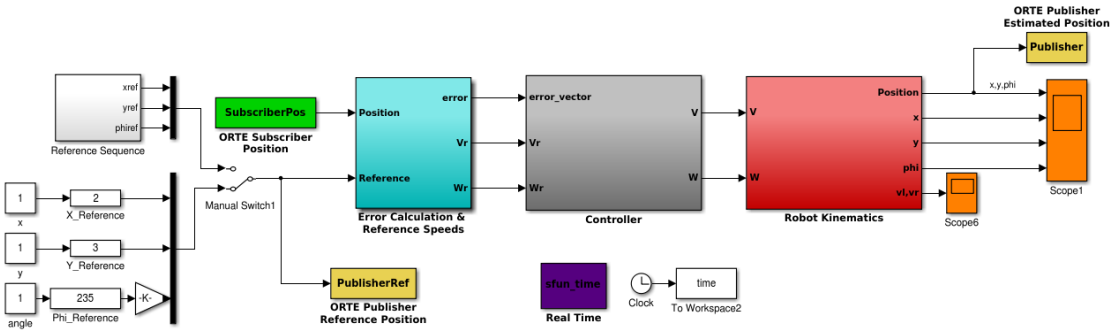


FIGURE 4.8: Simulink Final Implementation

4.5.3 Simulation Results

The plots in the figures below are meant in one hand, to present the performance of the controller implemented in the design, which will be discussed ahead.

On the other hand, these results also confirm that the S-Functions designed, perform accordingly. Thus, the communication channel established between ORTE and Matlab/Simulink is functional and reliable.

The first set of results (Figures 4.9 to 4.11), show the behavior of the mobile robot when trying to fix its position in the (x, y) plane, along with its orientation. A second set of results is presented, containing the errors in the system.

Differences between reference and position can be seen for x and y, meaning that the control signal v is a not enough to correct properly for these mismatches. The system shows a stabilization time of about 6 *seconds*.

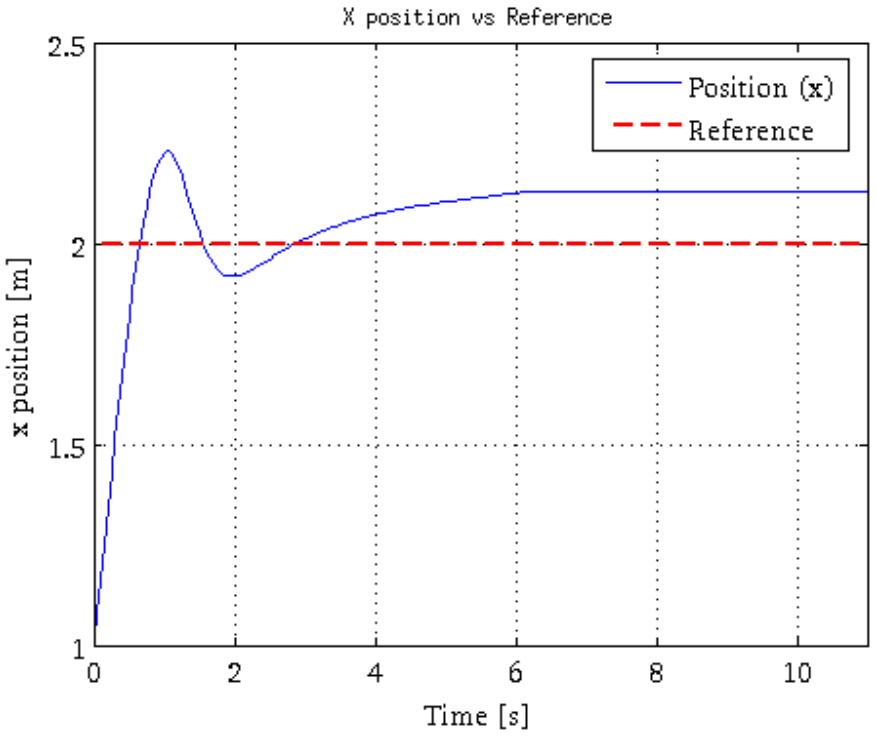


FIGURE 4.9: X Position vs Reference Results – *Kanayama*

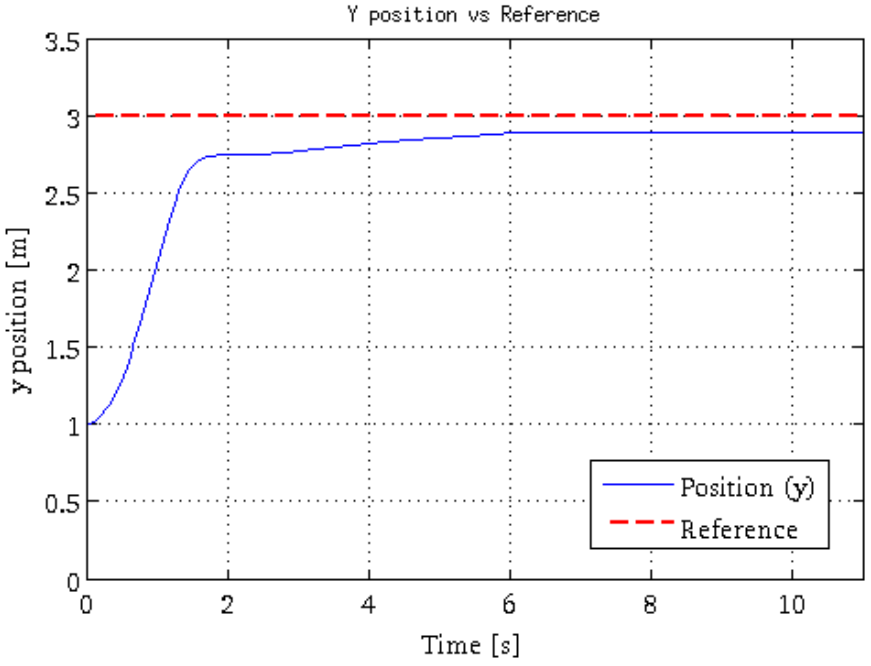


FIGURE 4.10: Y Position vs Reference Results – *Kanayama*

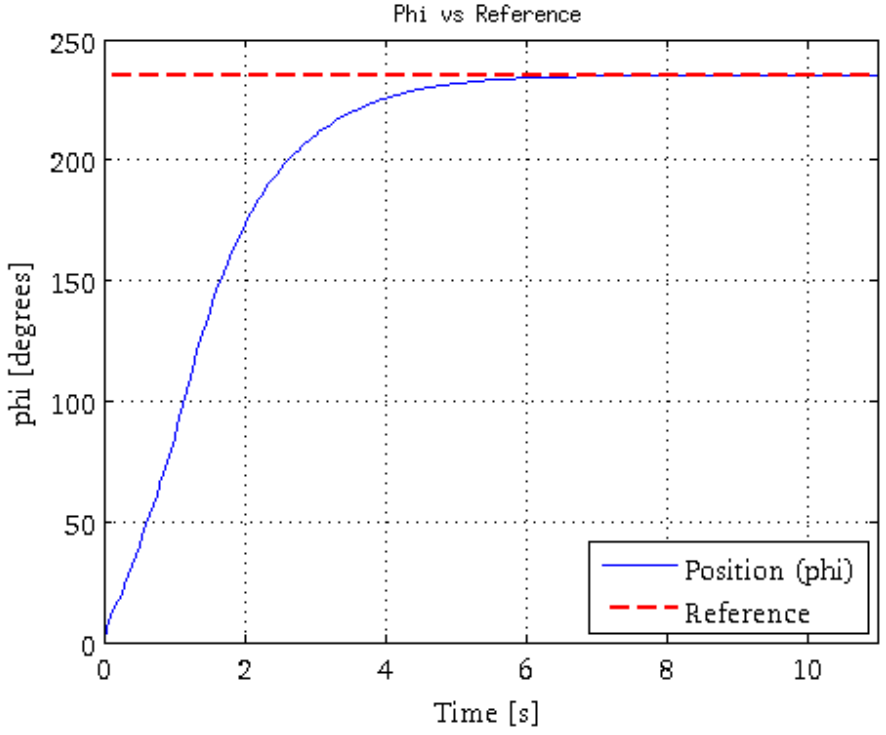


FIGURE 4.11: ϕ Position vs Reference Results – *Kanayama*

For ϕ, as seen in Figure 4.11, the controller performs satisfactorily, counteracting completely the error in orientation.

The data seen in previous plots, is translated into errors in the system as defined in equations 4.6 − 4.8, graphically shown in Figures 4.12 − 4.14.

From Figures 4.12 and 4.13, it can be seen that the controller can eliminate both, the driving error and the orientation error. Nevertheless, the lateral error is not properly eliminated as it can be seen in Figure 4.14.

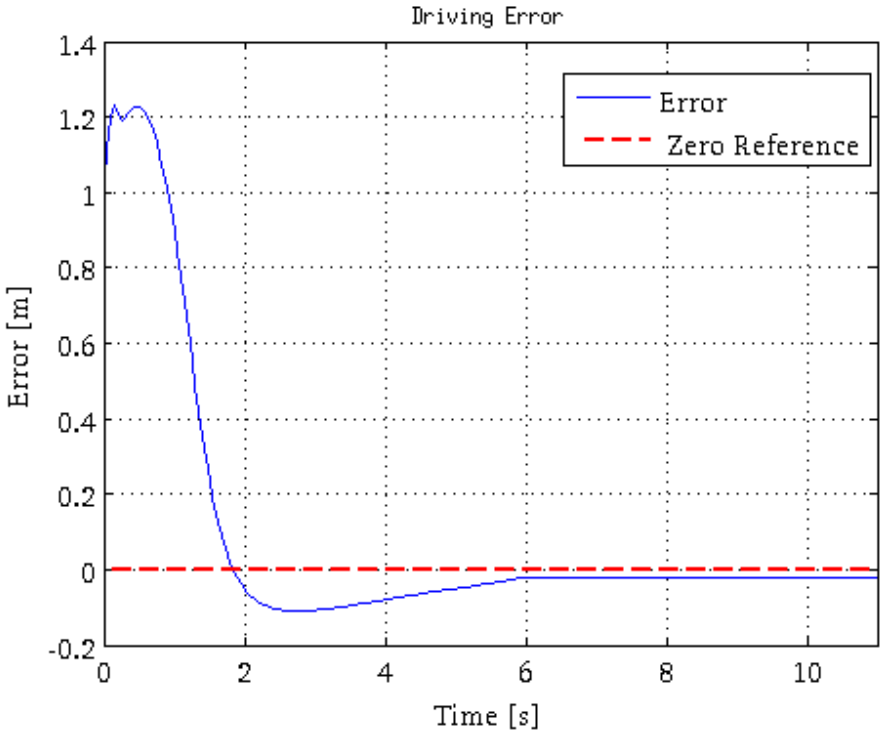


FIGURE 4.12: Driving Error – *Kanayama*

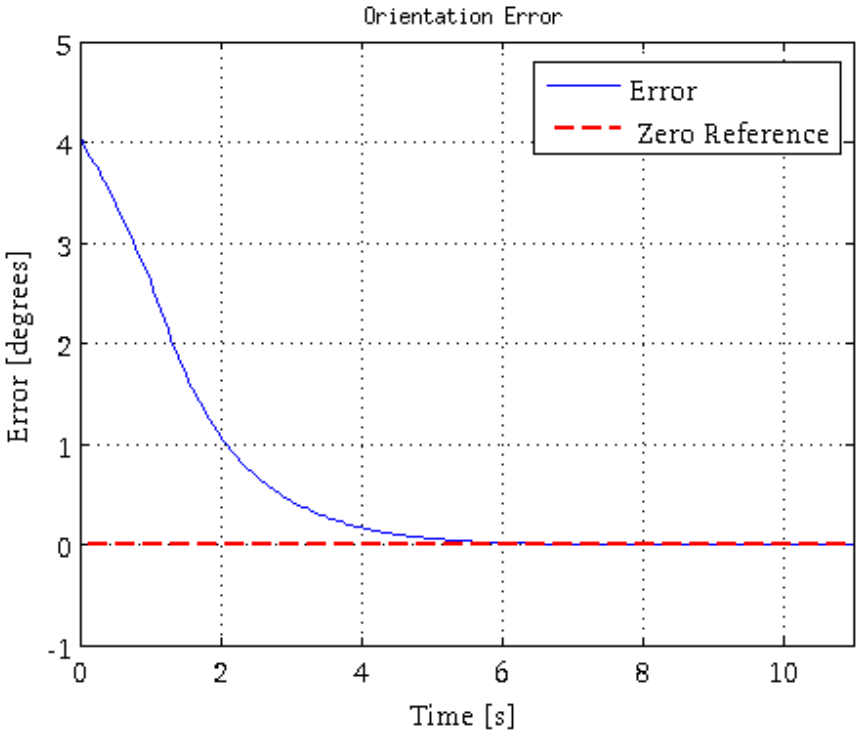


FIGURE 4.13: Orientation Error – *Kanayama*

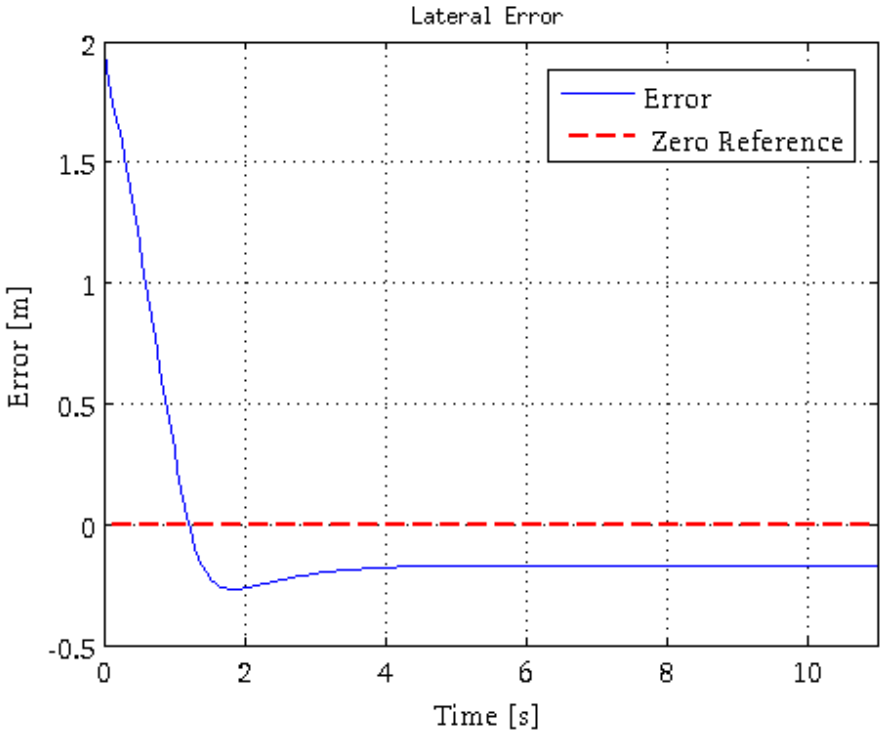


FIGURE 4.14: Lateral Error – *Kanayama*

Finally, Figure 4.15 shows the final position of the robot vs the reference along with the trajectory followed by the robot. As it can be seen, the final position, as well as the trajectory followed could be certainly improved through a better tuning or a new control algorithm.

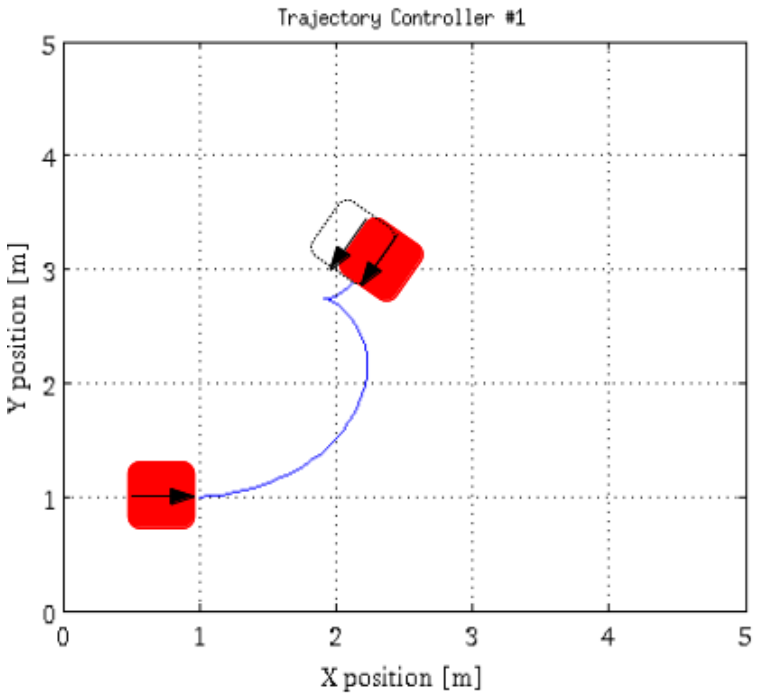


FIGURE 4.15: Trajectory followed and final position for Controller *Kanayama*

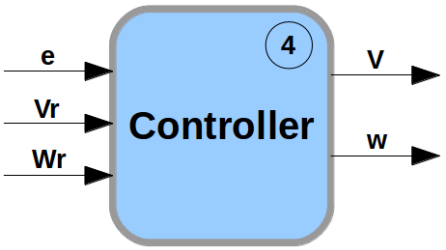
Analyzing equations 4.11 and 4.12 it can be seen that the response of the controller could be slow when the error in the driving direction is small and the error of the lateral direction is large.

Consider the case where the error in the driving direction $e_D = 0$ and the orientation error $e_\phi = 2 \cdot \pi$, where, the linear velocity $v = 0$. In this case, the lateral error cannot be eliminated to zero especially when e_L is relatively large [9].

These limitations of this controller led to research and implement a better controller that could counteract these scenarios, controller which will be presented in the following section.

4.6 Controller Proposed: Sanhoury

4.6.1 Design



The second controller implemented, follows the research done by Sanhoury I, et al [9].

A new linear velocity tracking controller is proposed under this research, which is based on a direct Lyapunov method, where the lateral error is taken into account when designing the linear velocity tracking controller [9]. The angular velocity calculation remains the same as in eq 4.12. The analysis for such a design, as for *Kanayama* controller, can be found on the literature referenced in this work.

The control law proposed by the authors is as follows:

$$v = v_r \cdot cos(e_\phi) + k_1 \cdot e_D + k_4 \cdot sign(e_D) \cdot e_L^2 \tag{4.14}$$

$$w = w_r + v_r(k_2 \cdot e_L + k_3 \cdot sin(e_\phi)) \tag{4.15}$$

where,

$$sign(e_D) = \begin{cases} -1 & e_D < 0 \\ 1 & e_D \geq 0 \end{cases}$$

As seen in equation 4.14 the controller provides the system with an extra factor in velocity to correct for lateral errors, tuned through k_4.

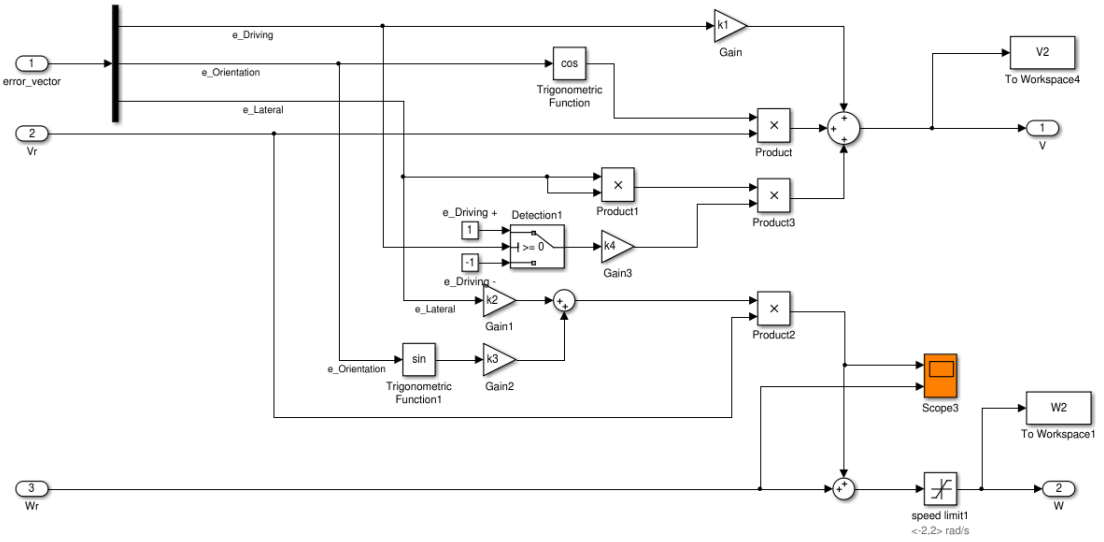


FIGURE 4.16: Controller Implemented in Simulink

To compare the effectiveness of this new factor in the controller, the same values for k_1, k_2 and k_3 as in controller *Kanayama* were used, while k_4 was manually tuned to evaluate the difference in response. The final value used in this case was of $k_4 = 0.350$

4.6.2 Simulation Results

As done before for Controller *Kanayama*, a first set of results is presented in this section (Figures 4.17 to 4.19), to show the response of the controller when counteracting for differences in x, y and ϕ. A second set of results is presented (Figures 4.20 to 4.23), containing the errors in the system.

In general, figures show the improvement in performance of the system by the addition of this extra term in the controller. The response is faster, and the system shows a stabilization time of $3 - 4$ *seconds*.

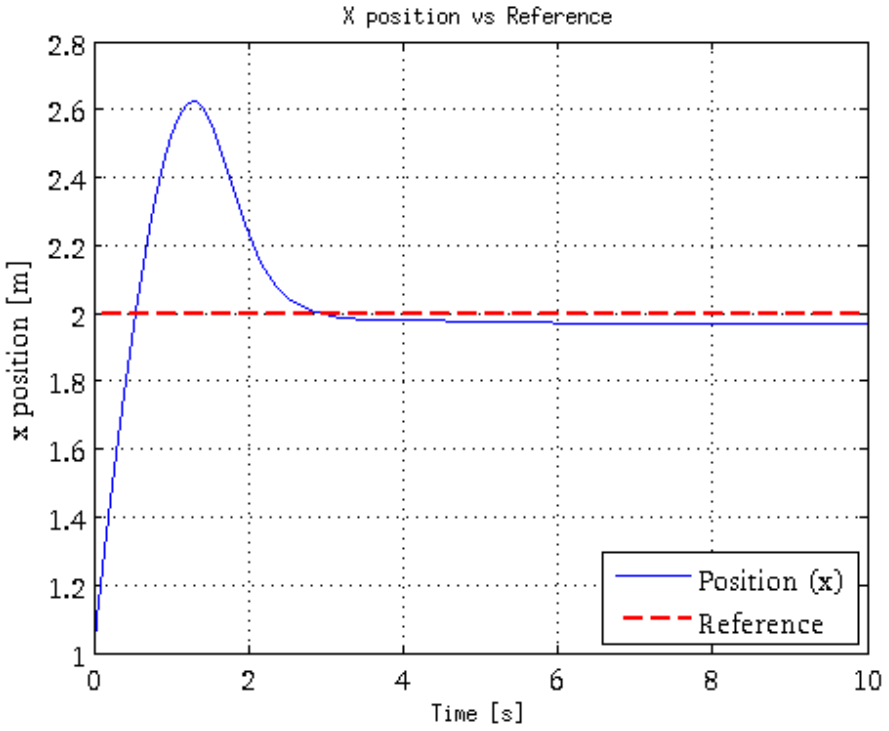


FIGURE 4.17: X Position vs Reference Results – *Sanhoury*

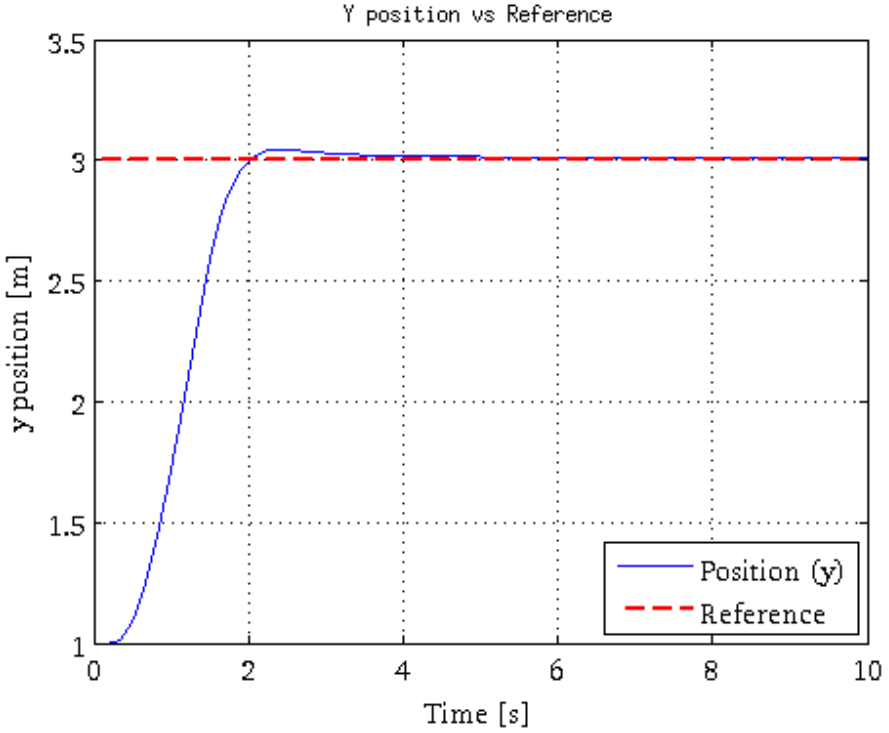


FIGURE 4.18: Y Position vs Reference Results – *Sanhoury*

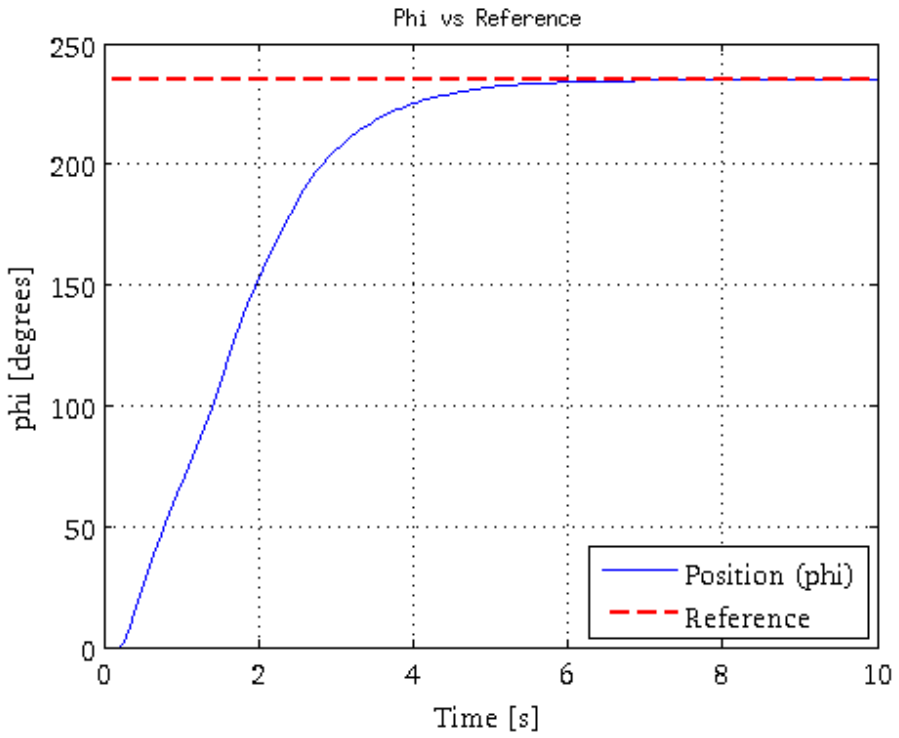


FIGURE 4.19: ϕ Position vs Reference Results – *Sanhoury*

It is evident from previous plots, that the errors will be minimized in the same way, as it can be seen in Figures 4.20 to 4.23.

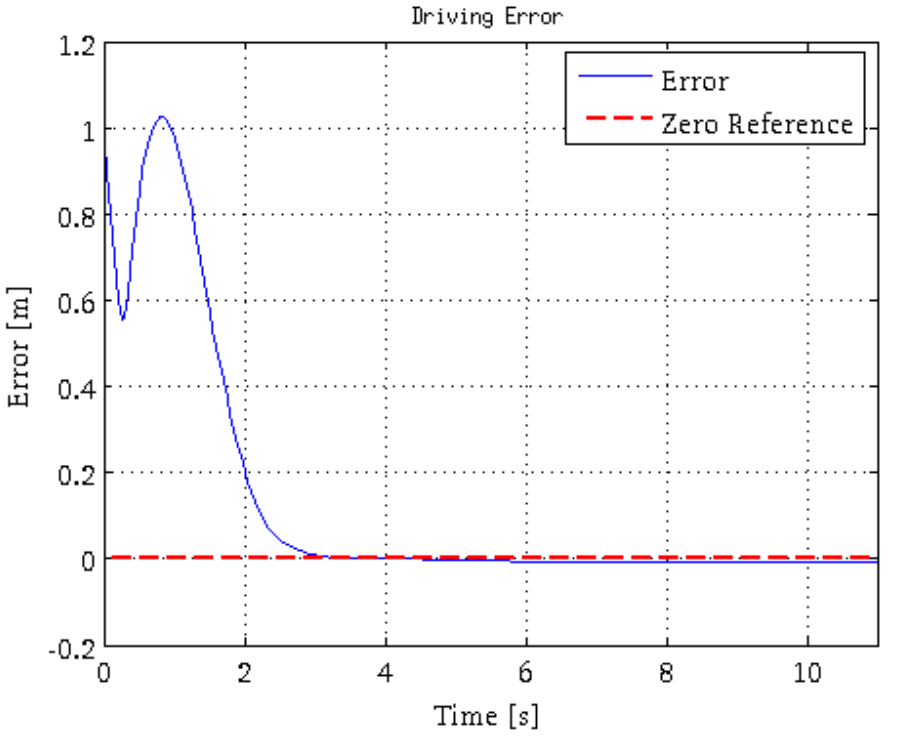


FIGURE 4.20: Driving Error – *Sanhoury*

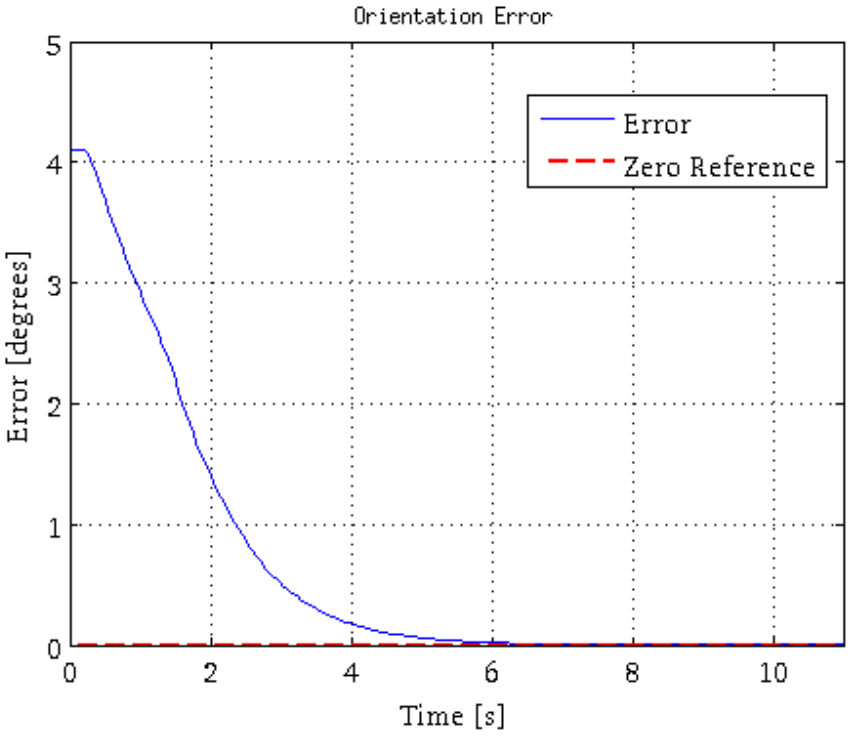


FIGURE 4.21: Orientation Error – *Sanhoury*

Important thing to notice is the improvement seen with respect to the lateral error, Figure 4.22. There is still a small error, but compared to the error seen for previous controller (Figure 4.14), it is considerably lower.

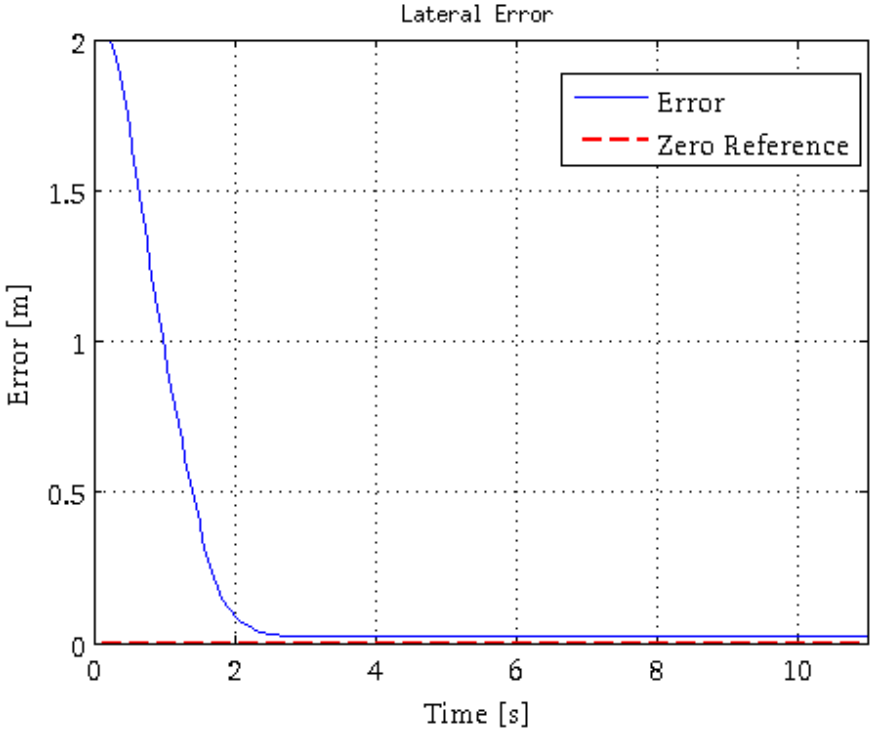


FIGURE 4.22: Lateral Error – *Sanhoury*

A higher setting in linear velocity v, lets the robot develop a better trajectory too, as it can be seen in Figure 4.23.

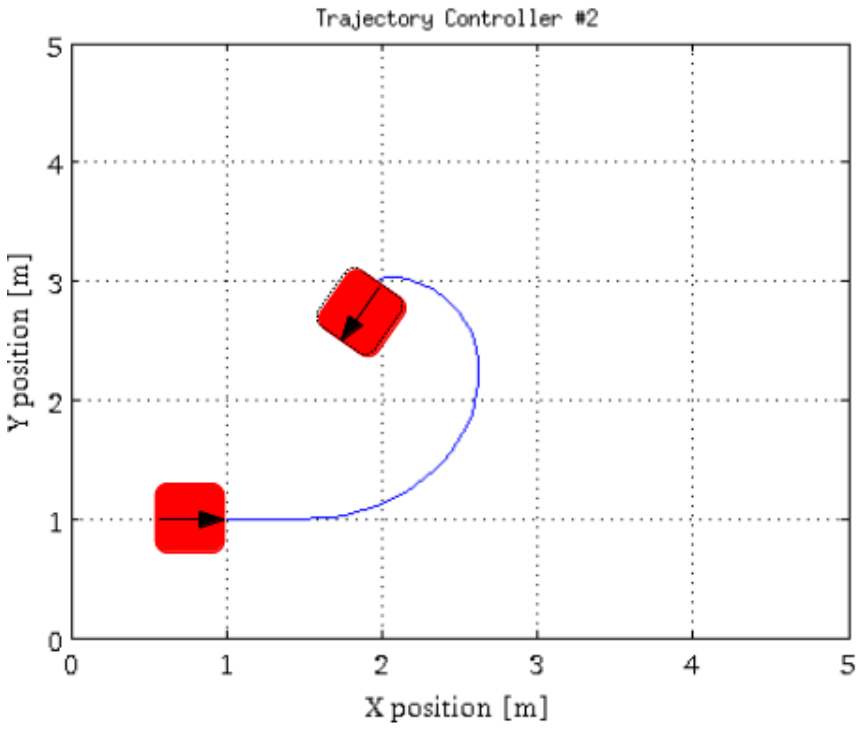


FIGURE 4.23: Trajectory followed and final position for Controller *Sanhoury*

4.7 Controllers Performance Comparison

The previous section focused on the performance of both controllers implemented, for a single reference given, since the main objective was to just to test the communication link between ORTE and Simulink. After the control scheme along with the communication showed to be functional, it was time to test it under a more complex and realistic scenario and compare the performance of both controllers.

For this reason, a sequence of different references in x, y and ϕ was designed to test for different combinations of errors in the system.

The gains for both controllers were obtained by analyzing the responses for specific movements in the plane and tuned accordingly. The final gains that are being used by both controllers are:

$$k_1 = 1.8 \qquad k_2 = 28.0 \qquad k_3 = -8.5,$$

and for Controller *Sanhoury*:

$$k_4 = 0.15$$

Since the robot has only two wheels and therefore cannot perform parallel movements from its current position, trajectories of this type were designed and tested. Table 4.1 enlists the sequence designed for this test.

Input Reference for Testing					
Ref. Point	**Time (s)**	$\boldsymbol{x}$	$\boldsymbol{y}$	$\boldsymbol{\phi}$	**Purpose**
1	Initial	1	0	0°	-
2	0	1	3	0°	*Testing for parallel constraint in y*
3	30	2	2	45°	*Small Error*
4	60	3.5	3	235°	*Large Orientation error*
5	90	1	1	0°	*Large Errors in system*
6	120	2	3	90°	-
7	150	4	3	90°	*Testing for parallel constraint in x*
8	180	1.5	3.5	20°	*Small error in y*

TABLE 4.1: Control Performance Test - Reference Sequence

Figures 4.24 to 4.26 depict the results of the reference tracking for x, y and ϕ for both controllers. Legends *Controller* #1 and *Controller* #2 stand for results of *Kanayama* and *Sanhoury* controllers, respectively.

As expected from previous analysis, Controller *Sanhoury* presents once again a faster and more aggressive response than the Controller *Kanayama*. When the system presents a large error as in seconds 60 to 120, both controllers perform similarly.

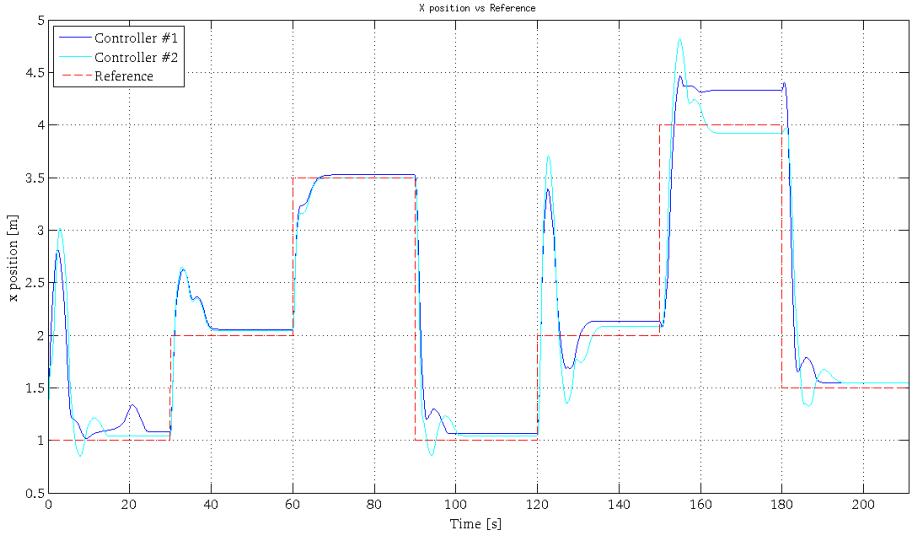


FIGURE 4.24: X Position vs Reference Results and Comparison

From Figure 4.24 it can be noticed the big difference in performance when trying to perform a maneuver to a parallel position in x, as seen from second 150 to 180.

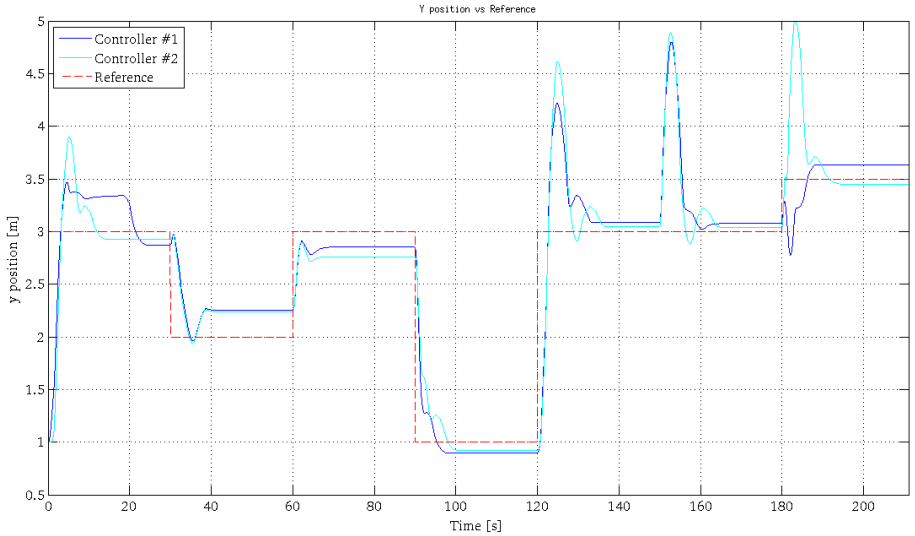


FIGURE 4.25: Y Position vs Reference Results and Comparison

Similarly as seen in Figure 4.24, 4.25 shows at the very beginning of the test, the major difference in performance during a maneuver to a parallel position in y.

Figure 4.26, other than the major error when testing for the parallel constraint in x at second 150, shows an important deficiency for both controllers. When testing the system for small errors, as from second 30 to 50, the controllers are incapable of following the given orientation angle. This can also be seen in figure 4.25, where both controllers present a poor performance.

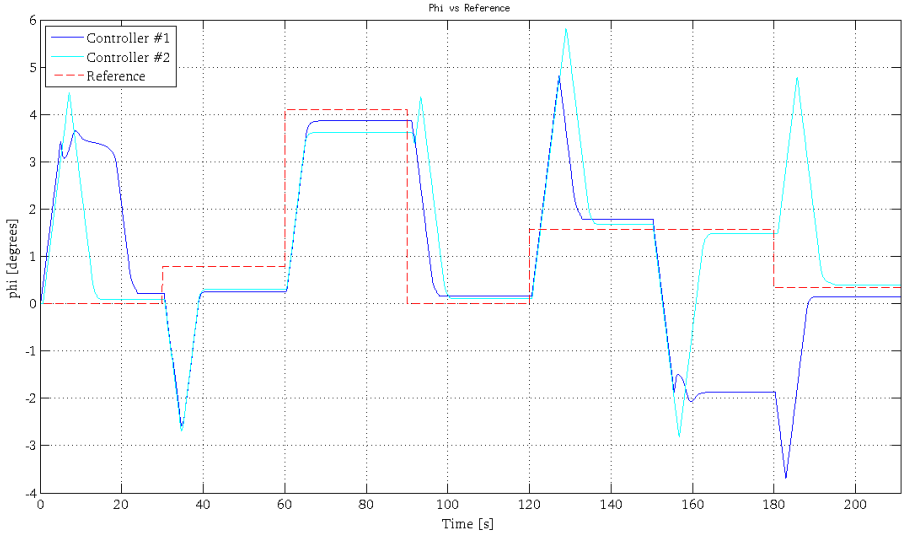


FIGURE 4.26: ϕ vs Reference Results and Comparison

As did in previous section, aiming to show the reader a different perspective of these differences and similarities, the following figures compare the *Driving*, *Lateral* and *Orientation* errors found in the system in both controllers.

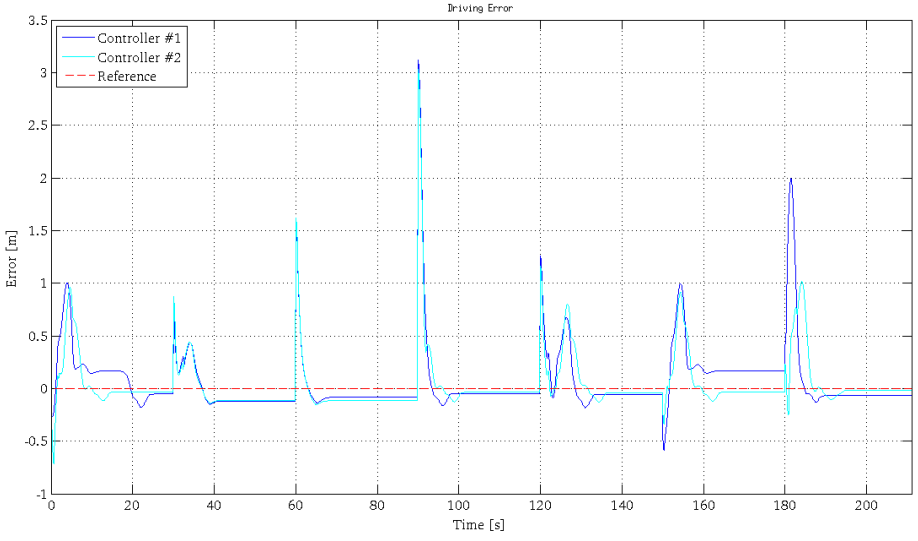


FIGURE 4.27: Driving Error Comparison

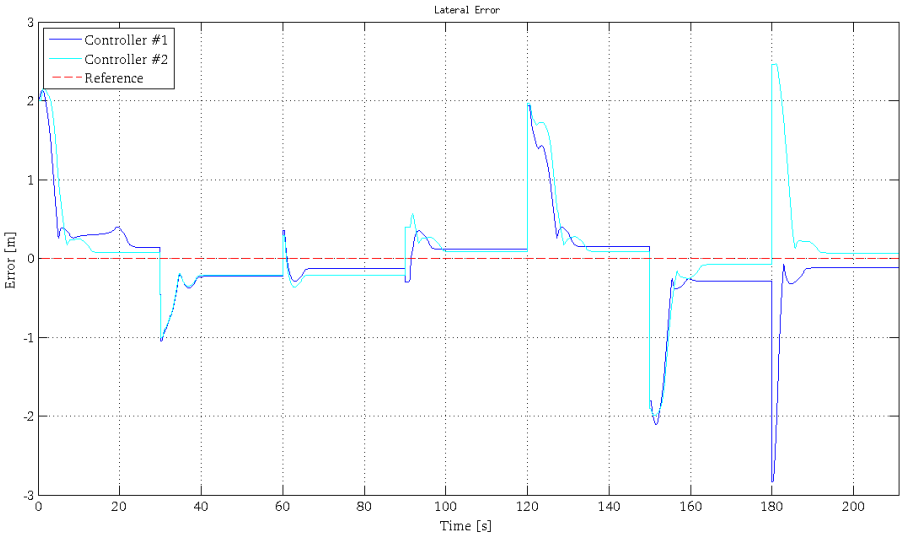


FIGURE 4.28: Lateral Error Comparison

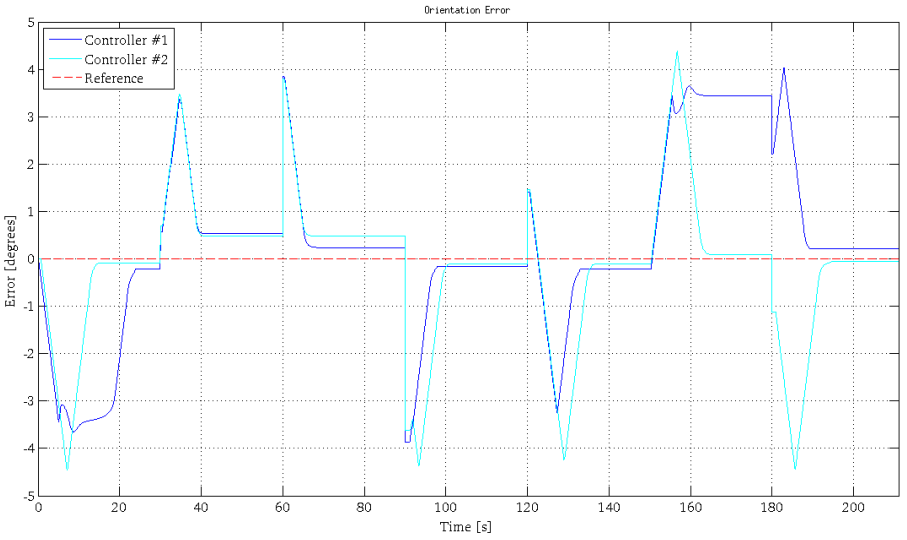


FIGURE 4.29: Orientation Error Comparison

The last set of plots compare the trajectories followed by the mobile robot using both controllers. Reference points in plots are labeled accordingly to Table 4.1.

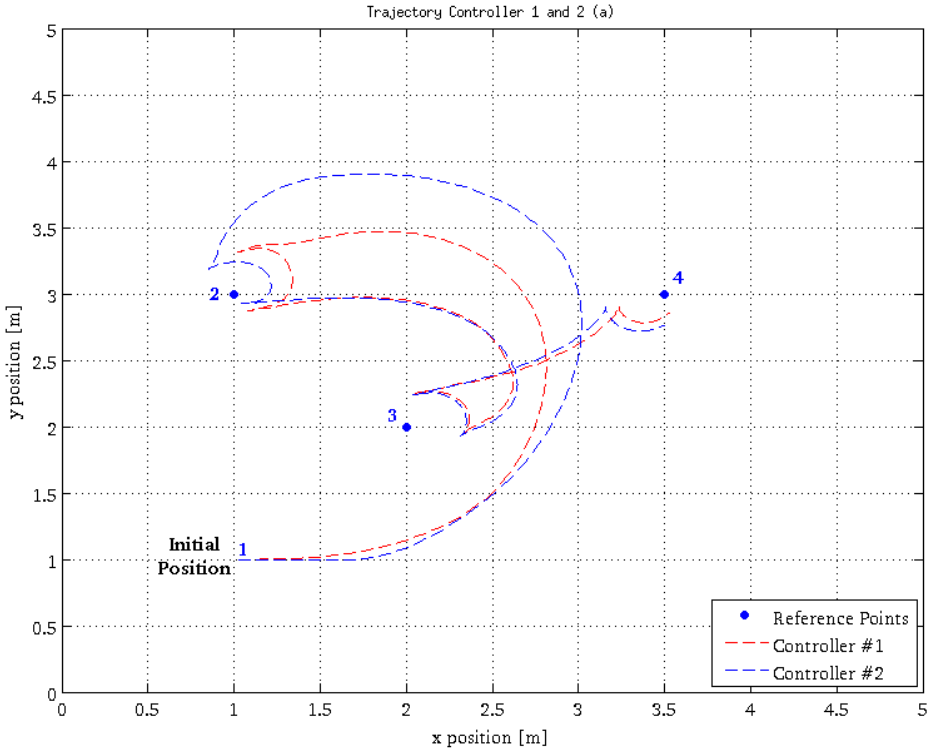


FIGURE 4.30: Trajectories comparison, from initial condition to point 4

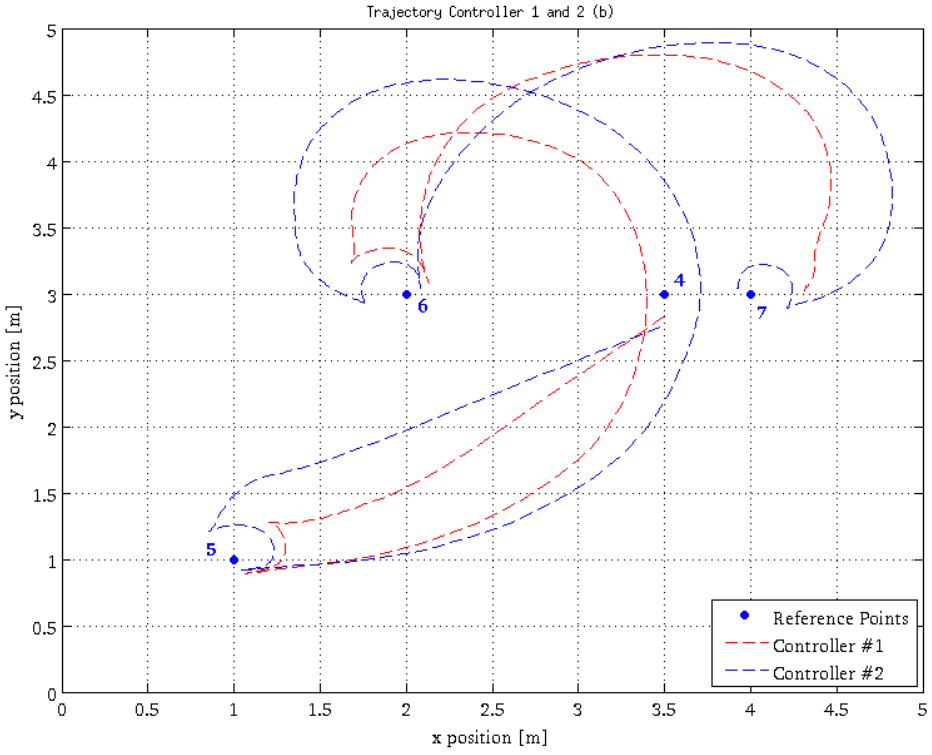


FIGURE 4.31: Trajectories comparison, from point 4 to 7

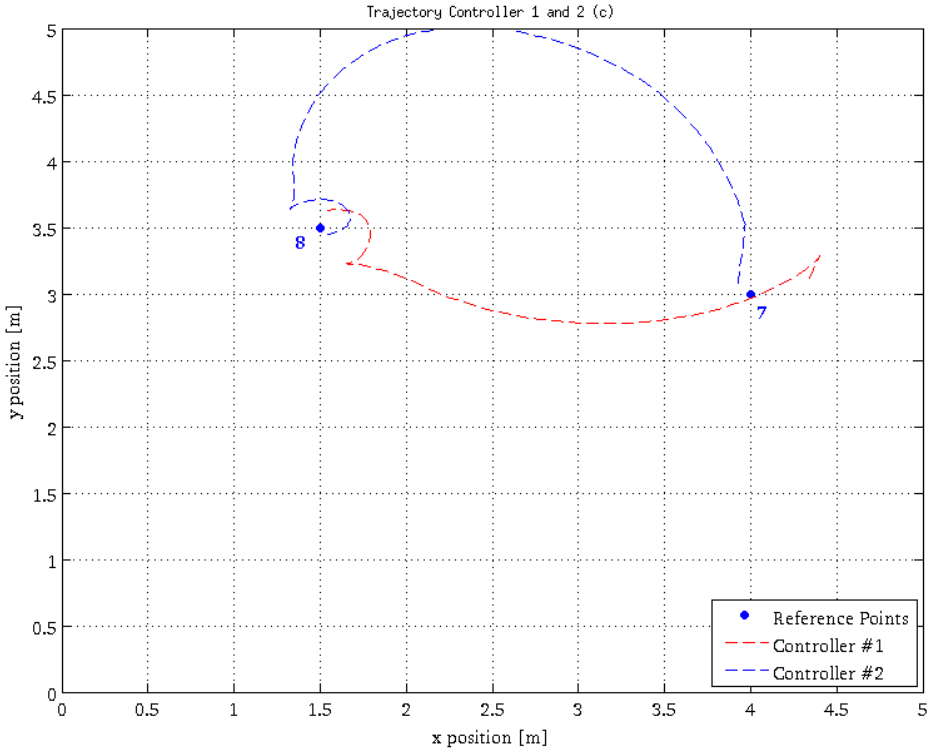


FIGURE 4.32: Trajectories comparison, from point 7 to 8

Chapter 5

Conclusions and Future Work

5.1 Summary

The main goal of this thesis work was to enable rapid prototyping of control algorithms for mobile robot, through the integration of middleware ORTE into Matlab Simulink/Stateflow platform. This work shows that the communication link established between robot and platform, is reliable and performs satisfactorily. This performance was tested under two different closed-loop control schemes, in both cases producing satisfactory results.

-The controllers implemented and tested in this project revealed the potential of a platform such as Simulink for future control related projects. The friendliness of the environment is translated into easiness to implement more complex control algorithms that could help ongoing or future projects.

Furthermore, a detailed comparison in performance for controllers implemented is given, highlighting the scenarios where the system could be prone to accuracy errors.

5.2 Discussion

The main results of this project could be divided into two. First, the communication link between ORTE and Matlab/Simulink, which showed to be efficient and reliable to use for control purposes. The process followed for the development of this link, was standard for all subscribers and all publishers separately, due to the roles they play within the

network used by the communication protocol. The differences among publishers or subscribers, rely of the type of data they are issuing/receiving to/from ORTE.

Second, the controllers implemented and tested in this project show the effectiveness of the control technique behind their design. Both controllers perform adequately, however the second controller represents a better performance, specially when it comes to counteract errors when a reference is given in a parallel position with respect to the current one.

Important thing to mention is that both controllers perform poorly when the system has small errors, specially when the degrees of difference in orientation are small, from a current to a desired position. Anyhow, the controllers here implemented were not intended to fulfill determined performance requirements (% overshoot, rise time, settling time, steady state error, etc) but just to prove that control for the mobile robot is achievable through Matlab/Simulink.

5.3 Future Work

Since this work is a basis for future projects the following areas of opportunity can be mentioned.

The results shown in this work were obtained using a simulator. S-Functions and control algorithms should be tested on hardware, to evaluate the behavior of the system and make corresponding adjustments.

As well, important work could be done regarding the data processing algorithm of the laser sensor available in the robot, from Matlab/Simulink. At this point the developed S-function only provides the system with raw data coming from the sensor. If this algorithm is developed, it will provide object detection capabilities to the robot directly from Simulink. Also, it could be used to improve control optimality for robot localization in known environments.

Future controllers should be designed trying to meet specific tasks and/or requirements. Having a powerful tool as Matlab and Simulink in hand, could led to implement in an easy way iterative, optimal or robust control techniques.

Bibliography

[**1**] Smolik P., Pisa P., Sojka M., Sebek Z., Hanzalek Z. *"ORTE – Open Real-Time Ethernet Manual"*. Czech Technical University, August 2012.

[**2**] Sonck S., et al. *"Real-Time Publish Subscribe (RTPS) Wire Protocol Specification"*. Real Innovations Inc. February 2002. Web. March 2014.

[**3**] Vokač M. *"Demonstrační robotická platforma"*. MSc thesis, Czech Technical Univer-
sity, January, 2012. http://rtime.felk.cvut.cz. Web. March 2014.

[**4**] Eurobot.*"Eurobot International Students Robotic Contest"*. Eurobot. January 2014. Web. April 2014.

[**5**] Tran Duy K et al, *Autonomous Robot Running Linux for the Eurobot 2007 Competition*. 2007. http://rtime.felk.cvut.cz. Web. November 2013.

[**6**] Castellote G., Bolton P., *"Distributed Real-Time Applications Now Have a Data Distribution Protocol"*, Real-Time Innovations Inc. Sunnyvale CA, Feb 2002. Web.November 2013.

[**7**] The MathWorks, Inc. *"Matlab/Simulink Developing S-Functions Manual"*, 2013.

[**8**] Kanayama Y, et al. *"A Stable Tracking Control Method for a Non-Holonomic Mobile Robot"* in Proc. IEEE/RSJ Int. Workshop Intelligent Robots and Systems, 1991, pp. 1236–1241

[**9**] Sanhoury I, et al. *"Tracking Control of a Nonholonomic Wheeled Mobile Robot"*. PIM Volume1, Issue 1 April 2012, pp. 7-11

[**10**] Karer G, et al. *"Robot Ballet"* in International Cultural and Academic Meeting of Engineering Students, September 2003.