Dissertation Template for Princeton University

First Middle Last

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Electrical Engineering
Adviser: Professor Smith

June 2010

Abstract

Recent works on the control of wheeled mobile robots have shifted from the only use of the kinematic model to the use of the dynamic model as well. First papers treated the inputs to the dynamic model as torques, and since most commercially available robots take velocities as inputs, several works shifted to use velocity-based dynamic model, in case of designing the robot platform the previous two approaches are not suitable.

This work describe how to design and interface a robot platform electrical hardware, propose a LQG servo velocity controller based on a formulated and estimated voltage-based dynamic model, and use a kinematic controller for trajectory tracking purpose.

Résumé

Des travaux récents sur le contrôle des robots mobiles à roues sont passés de la seule utilisation du modèle cinématique à l'utilisation du modèle dynamique également. Les premiers papiers traitent les entrées du modèle dynamique en tant que couples, et puisque la plupart des robots commercialement disponibles prennent des vitesses en tant qu'entrées, plusieurs travaux sont migrés pour utiliser le modèle dynamique basé sur la vitesse, dans le cas de la conception du plate-forme du robot, les deux approches précédentes ne conviennent pas.

Ce travail décrit la façon de concevoir et d'interfaçer le matériel électrique du robot, proposer un servo contrôleur de vitesse LQG basé sur un modèle dynamique formulé et estimé qui prennent des tensions en tant qu'entrées, et utiliser un contrôleur cinématique pour le suivi de la trajectoire.

Acknowledgements

After an intensive period of three months, today is the day: writing this note of thanks is the finishing touch on my dissertation. It has been a period of intense learning for me, not only in the scientific arena, but also on a personal level. Writing this dissertation has had a big impact on me. I would like to reflect on the people who have supported and helped me so much throughout this period. I would like to thank my tutor, Mr.A.KACIMI, for his valuable guidance. You definitely provided me with the tools that I needed to choose the right direction and successfully complete my dissertation. In addition, I would also like to acknowledge Mr.A.FERADJI and Mr.K.AIZI as the jurors and readers of this thesis, and I am gratefully indebted to them for their very valuable comments on this thesis. Finally, I must express my very profound gratitude to my parents, my sisters, my little brother, to all my family without forgetting all my friends and also to some specific person that I met in my life: LAID Sara, my partner MAMECHE Hamza, CHERIF Bilel, BENZIERA Sedik, BESBES Abdelfettah, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

AZZI Oussama

I would like to thank all the people who contributed in some way to the work described in this thesis. First and foremost, I thank my academic advisor, Mr.KACIMI Abderrahmane. During my tenure, he contributed to a rewarding graduate school experience by giving me intellectual freedom in my work and engaging me in new ideas. Additionally, I would like to thank my supervisory committee members Mr.FERADJI Ahmed, and Mr.AIZI Kamel for their interest in my work.

Every result described in this thesis was accomplished with the help and support of my friend and colleague AZZI Oussama.

Finally, I would like to acknowledge friends and family who supported me during my time here. First and foremost I would like to thank Mom, Dad, and my sister Assia for their unconditional love and support. Special mention to my friend LARBI Soheib for being my projects mate, and for his debates, dinners and life advices. I'm lucky to have met BAICHE Fadoua here, and I thank her for her friendship, love, and unyielding support. I owe a debt of gratitude to all my friends and colleagues at University of Blida of which I was a student for 2 years.

Author

MAMECHE Hamza

To our parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most mobile robots are wheel-based structures because of their efficiency and simple mechanical implementation [13]. A very common configuration for mobile robots is the differential drive, which has two independently driven wheels in the rear (or front) and one or more unpowered wheels to balance the structure. Due to their good mobility and simple configuration, Wheeled mobile robots are becoming more popular for performing tasks that are too dangerous or tedious for humans, such as exploration, rescue, industrial load transportation, autonomous wheelchairs, floor cleaning, and others. And with the increase of technology availability, it is becoming more cost-effective to replace a worker with a mobile robot. Typical environments for wheeled mobile robots include warehouses, factories, military applications, and planetary exploration. In situations where the robot must maneuver with precision, a feedback controller for robot motion is necessary.

Considering differential drive mobile robots, an important issue is that the design of most of its motion controllers is based only on the robot's kinematic model. The main reasons for that are: (a) the dynamic model is more complicated than the kinematic one and its precise determination depends on the knowledge of several parameters associated with the vehicle and its actuators (like mass, moment of inertia etc.), and

(b) mobile robots frequently have low-level velocity control loops for their motors, which take a desired angular velocity as input and stabilize the motor angular velocity at this value [12]. However, because the robot's low-level velocity control loops do not guarantee perfect velocity tracking, when high-speed movements and/or heavy load transportation are required, it becomes essential to consider the robot dynamics as well. Thus, some motion controllers that compensate for the robot dynamics have been proposed in the literature. As an example, in [6] a combined kinematic/torque control law with a robust-adaptive controller based on neural networks is proposed to deal with disturbances and non-modeled dynamics. Notice that the control commands they used were torques, but commercial robots usually receive velocity commands, like the Pioneer robots from Adept Mobile Robots, the Khepera robots from K-Team Corporation, and the robuLAB-10 from Robosoft Inc. Therefore several works use velocity based dynamic model such as [8], and [11] which proposed an adaptive dynamic compensation controller. In our case we re-adapted an obsolete mobile robot platform for the purpose of control implementation, Therefore having direct access to motors and encoders, so we used a dynamic model that include motor dynamics and support voltages as inputs.

This thesis is organized as follows; In chapter 2, we describe the process of re-adapting the platform hardware to make in suitable for implementing the developed control scheme.Than in chapter 3, first the kinematics of a differential drive mobile robot are reviewed. The dynamic model of a mobile robot which takes voltages as inputs, and includes actuator dynamics is then derived. In chapter 4, we present the dynamic model parameters identification, and the the motors' drivers effect on the loop, than in chapter 5 we propose a centralized LQG servo controller for velocity control, than we use the kinematic controller presented in [10] for trajectory tracking, and finally we present the simulation results. Chapter 6 presents experimental results and discussion for velocity control, point to point stabilization and path tracking for circle and figure-

eight trajectories, and in chapter 6 we summarize the results, form a conclusion and propose the possibility of future works that hasn't been addressed in this document.

Chapter 2

Designing and interfacing The robot Hardware

2.1 Designing The robot Hardware

In this section, we will discuss the design and working of our robot hardware and selection of its hardware components. To achieve the main goal of the robot in hardware, we need to select all hardware components and find how to interconnect all these components. We know that the main functionality of this robot is tracking a generated path; this robot will have the ability to follow the path from the start position to the end position so we will discuss the different hardware components required to achieve this goal. We will see a block diagram representation and its explanation, and also discuss the main working of the robot. For our robot we had to build a new hardware platform that includes robot chassis, motors as an actuator ,quadrature encoder as a sensor,an Arduino Due as a controller board and a battery.

2.1.1 the updated platform for our robot

The robot we had in the beginning was a different platform called Intelligent robot kit ED-7273 (figure 2.1) made by ED corporation [1].

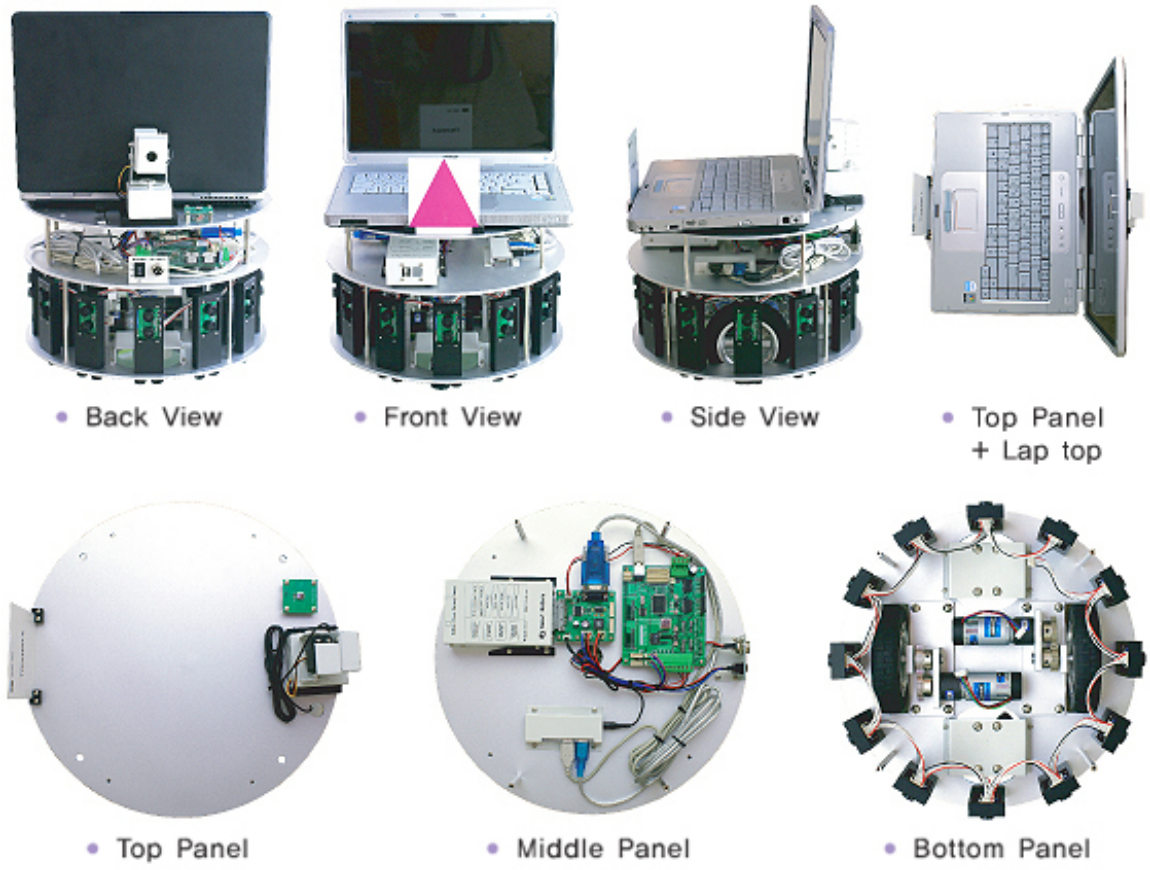


Figure 2.1: Intelligent robot kit ED-7273 [1]

The ED-7273 platform based on flow chart programming with C++ on a windows XP as software. the hardware system is a closed system compatible just with this software.

As shown in Figure 2.2 all data of sensors are transmitted by a serial data via USB using SPI protocol to the PC so there is no access to transmitted data.the Pc is the calculation unit that receives data and transmits the commands to the actuators. As we see the platform is managed by a closed protocol we can't access.

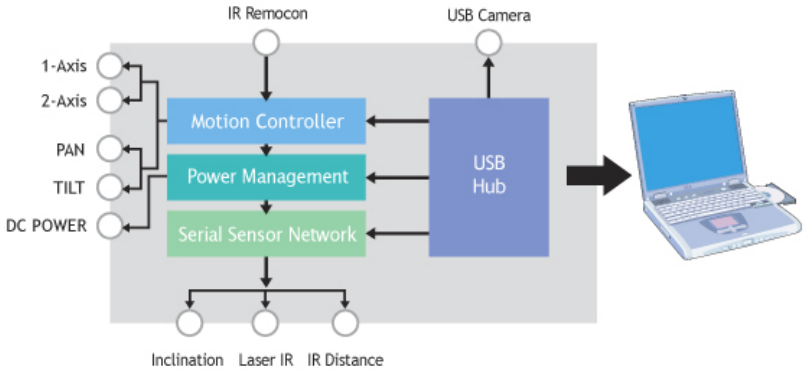


Figure 2.2: ED-7273 hardware communication diagram

The main changes we made was replacing the controller board, the drive motor, the power system and the software platform interface. the following images shows the original robot platform and our new platform (figure 2.3 & 2.4).

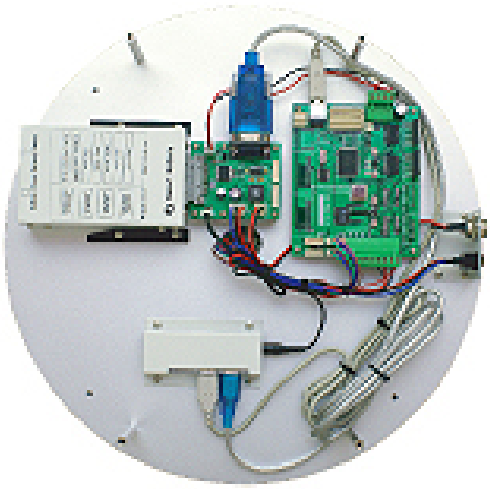


Figure 2.3: The original ED-7273 platform

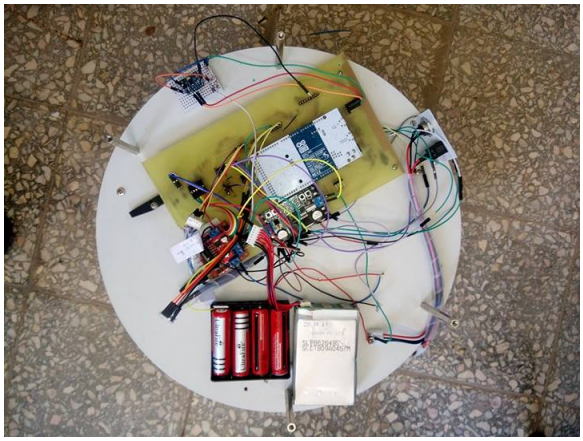


Figure 2.4: Our new platform integrated on ED-7273

2.1.2 Specifications of the Robot hardware

In this section, we will discuss some of the important specifications such as hardware Design of a Service Robot. The final robot prototype will meet the following specifications:(a) The robot chassis design should be Simple and cost effective.(b) Autonomous navigation functionality: The robot should autonomously navigate and it should contain necessary sensors for doing this.(c) Long Battery life: The robot should have a long battery life in order to work continuously. The working time should be greater than 1 hour. The robot hardware design should meet these specifications. Let's look at one of the possible ways of interconnecting the components in this robot. The next section shows the block diagram of the robot and explains it.

2.1.3 Block diagram of the robot

The robot's movement is controlled by two Direct Current (DC) gear motors with an encoder. The two motors are driven using a motor driver. The motor driver is interfaced into an embedded controller board, which will send commands to the motor driver to control the motor movements. The encoder of the motor is interfaced into the controller board for counting the number of rotations of the motor shaft. This

data is the odometer data from the robot. the following figure 2.5 shows the Block diagram of the robot:

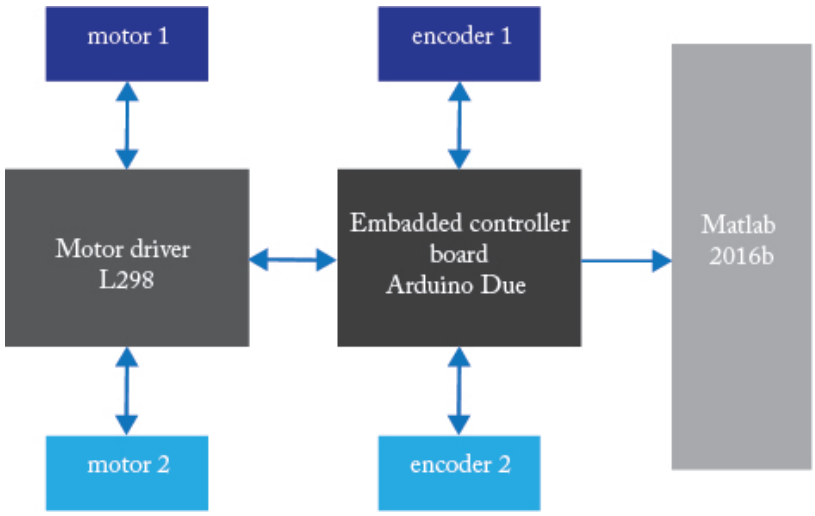


Figure 2.5: Block diagram of the robot

2.1.4 Motor and encoder

The robot that we are going to design is a differential drive robot with two wheels, so we require two motors for its locomotion. Each motor consists of quadrature encoders to get the motor rotation feedback. The quadrature encoder will give the motor rotation feedback as square pulses; we can decode the pulse to get the number of ticks of the encoder, which can be used for feedback. If we know the wheel diameter and the number of ticks of the motor, we can compute the displacement and the angle that the robot traversed. This computation is very useful for navigation of the robot. The following figure 2.6 shows the image of the motor we have. The motor (PGM35-3657E) comes with an integrated quadrature encoder with a resolution of 14x4 counts per revolution of the motor shaft. those encoders based on hall effect sensing [3].

Figure 2.6: The PGM35-3657E motors [3]

How Hall Effect Encoders Work

Hall effect encoders use magnetic phased arrays that contain hall sensor elements arranged in a pattern to match a magnetic wheel. A signal is produced as the sensor passes over the magnetic field which is then interpolated to the desired resolution. Inside a magnetic hall effect encoder, a Hall-array sensor averages the signal over multiple detectors to deliver robust, high-resolution performance that is insensitive to misalignment, shock, and vibration.

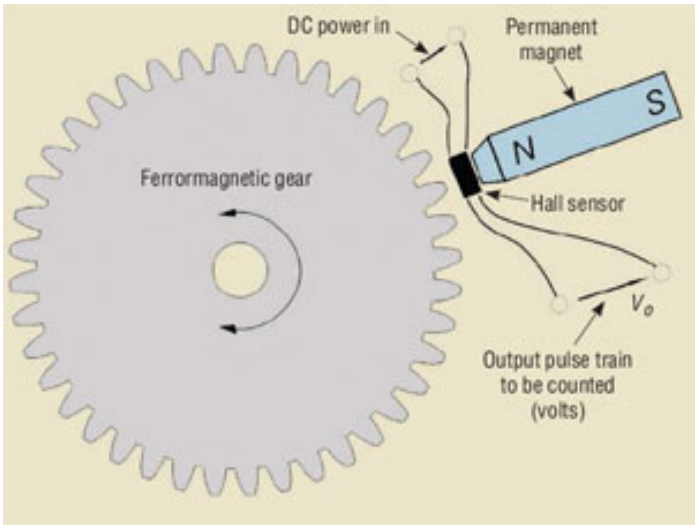


Figure 2.7: Hall Effect Encoder

Benefits of Hall Effect Encoder

Magnetic hall effect encoders can be extraordinarily robust. Because magnetic encoders are based on an inductive effect, they do not require bearings, which removes a point of failure from the system. Typically, the electronics inside the hall effect encoder are encapsulated so that they are not exposed to the elements. Our integrated encoder has 2 hall effect sensor in able to give a feedback of the motor direction. as shows the following figure 2.8.

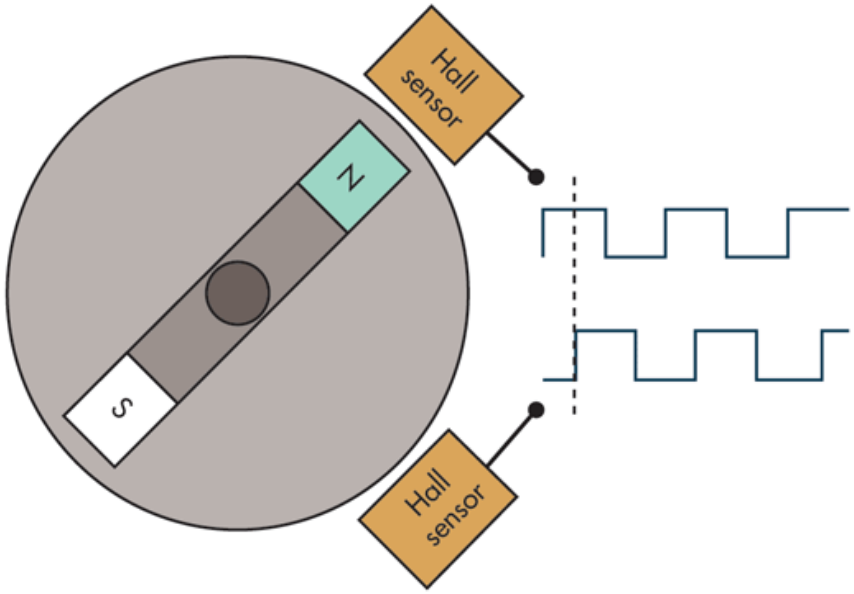


Figure 2.8: Hall Effect Encoder explanation

the motor that we are using has 6 pins with different colors. The pin description of this motor is given in the following figure 2.9.

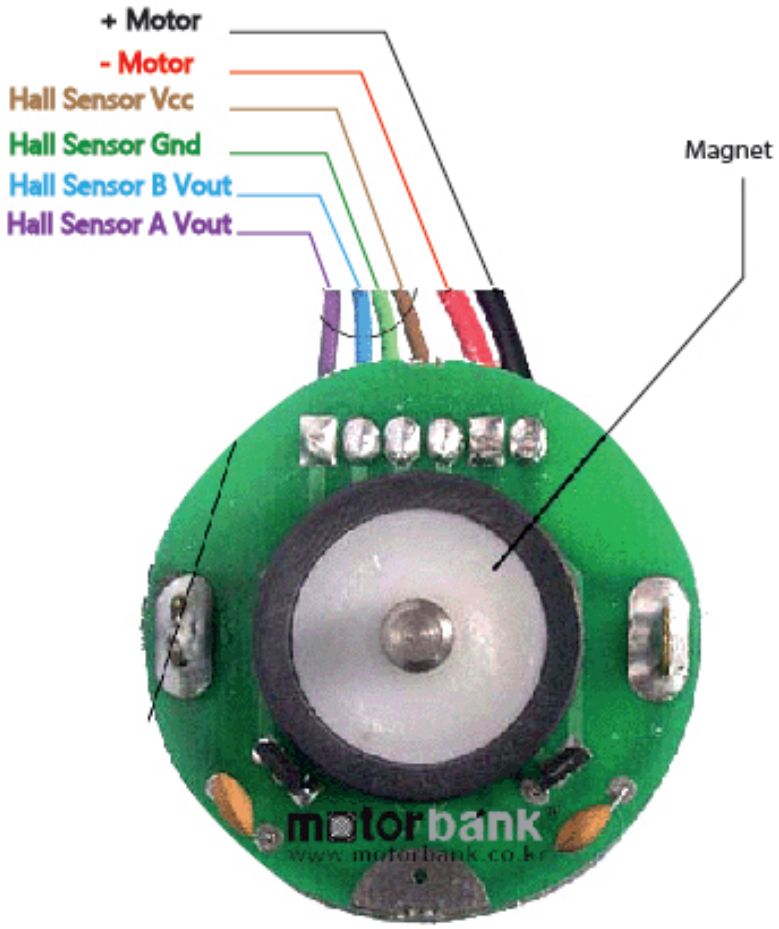


Figure 2.9: motor-encoder

2.1.5 Motor driver

A motor driver or motor controller is a circuit that can control the speed of the motor. Controlling motors means that we can control the voltage across the motor and can also control the direction and speed of the motor. Motors can rotate clockwise or counter clockwise, if we change the polarity of motor terminal. H-bridge circuits are commonly used in motor controllers. H-bridge is an electronic circuit that can apply voltage in either direction of load. It has high current handling properties and can change the direction of current flow. The following screen shot 2.10 shows a basic H-bridge circuit using switches.

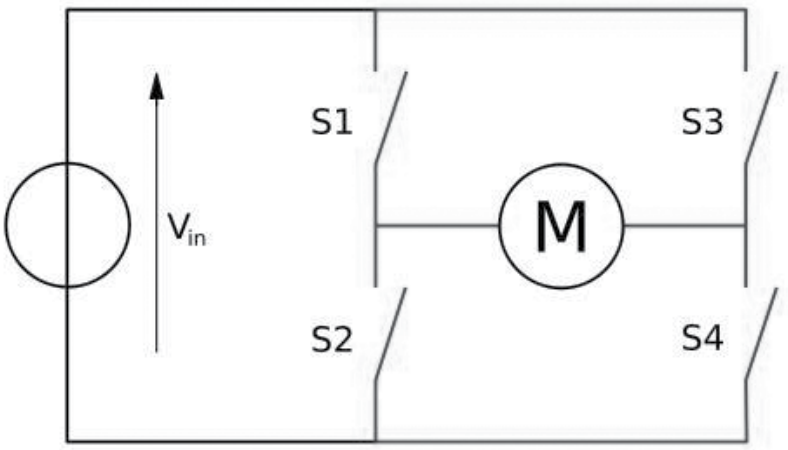


Figure 2.10: H-bridge circuit[7]

The direction of the motor (figure 2.1), depending on the four switches, is given as follows:

S1	S2	S3	S4	the direction of the motor
1	0	0	1	Motor moves right
0	1	1	0	Motor moves left
0	0	0	0	Motor free runs
0	1	0	1	Motor brakes
1	0	1	0	Motor brakes
1	1	0	0	Motor shoots through
0	0	1	1	Motor shoots through
1	1	1	1	Motor shoots through

Table 2.1: the direction of the motor depending on the H-bridge switches

We have seen the basics of an H-bridge circuit on the motor driver circuit. Now, we can select one of the motor drivers for our application and discuss how it works. The following figure 2.11 shows one of the motor drivers that we will use in our robot

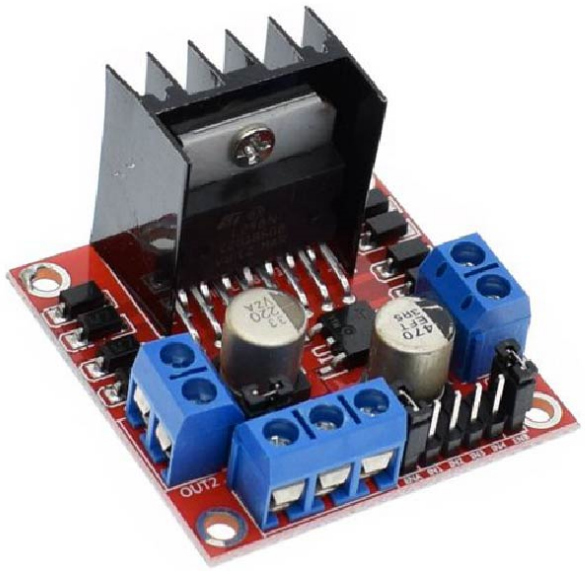


Figure 2.11: L298 drock module

L298N H-bridge

L298N is a high voltage, high current motor driver chip. The chip uses 15-pin package. The main features are: high voltage, maximum operating voltage up to 46V; output current, instantaneous peak current up to 3A, continuous operating current of 2A; rated power 25W. Contains two H-bridge high-voltage and high current full-bridge driver can be used to drive DC motors and stepper motors, relays, coils and other inductive load; using standard logic level signal control; having two enable control terminal, in allows input signal without being affected or disable the device has a logic supply input, the internal logic circuit part of the work at a low voltage; can be an external sense resistor, the amount of change back to the control circuit. Drive motor using L298N chip, the chip can drive a two-phase stepper motors or four-phase stepper motor, can drive two DC motors. **Features**

1. The module uses L298N chip as the main drive, with driving ability, low heat, anti-jamming features.

2. It can be used to take power through the built-in 78M05 drive power part of the work, but in order to avoid damage to the regulator chip, when a drive voltage greater than 12V, please use an external 5V logic supply.

3. Uses high-capacity filter capacitors, freewheeling diode protection, reliability can be improved.

Input pins

The following pins are the input pins of the motor driver, by which we can control mainly the motor speed and direction:

Table 2.2: Input pins

Pin Name	**Function**
1INa, 1INb, 2INa, 2INb	These pins will control the direction of motor 1 and 2 in the following manner: 1. If INA = INB = 0, motor will break 2. If INA = 1, INB = 0, motor will rotate clockwise 3. If INA = 0, INB = 1, motor rotate counter clockwise 4. If INA = INB = 1, motor will break
1PWM, 2PWM	This will control the speed of motor 1 and 2 by rapidly turning them on and off.

Output pins

The output pins of the motor driver will drive the two motors. The following are the output pins:

Table 2.3: Output pins

Pin Name	**Function**
OUT 1A, OUT 1B	These pins connect to motor 1 power terminals
OUT 2A, OUT 2B	These pins connect to motor 2 power terminals

Power supply pins

The following are the power supply pins:

Table 2.4: Power supply pins

Pin Name	Function
VIN (+), GND (-)	These are the supply pins of the two motors. The voltage ranges from 5.5 V to 16 V.
+5 VIN, GND (-)	This is the supply of motor driver. The voltage should be 5 V.

2.1.6 Embedded controller board (ARDUINO DUE)

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button [4]. Unlike most Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board[4].

AVR Arduino microcontroller	
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

ARM Core Benefits

- A 32-bit core, that allows operations on 4 bytes wide data within a single CPU clock. (for more information go to int type page).
- CPU Clock at 84Mhz.
- 96 KBytes of SRAM.
- 512 KBytes of Flash memory for code.

- A DMA controller, that can relieve the CPU from doing memory intensive tasks.

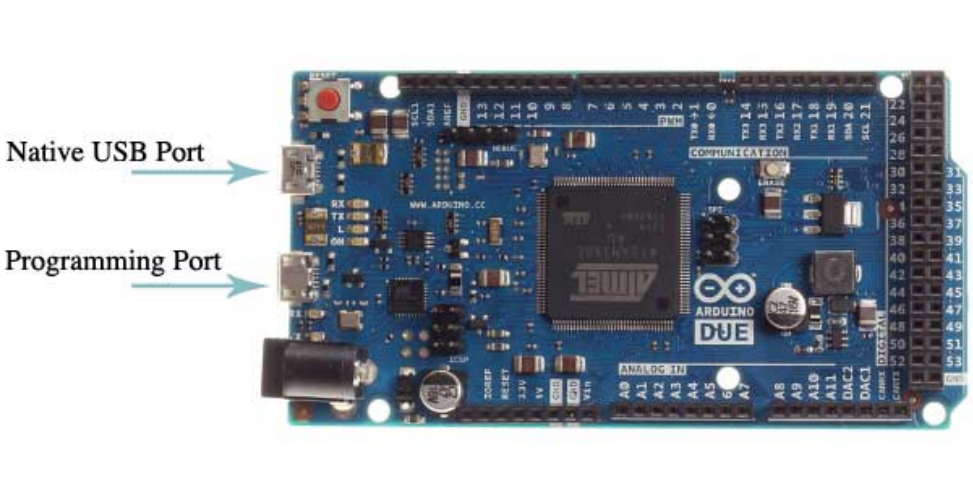


Figure 2.12: DueUSBPorts [4]

2.1.7 Power supply

One of the important hardware component is the power supply. We saw in the specification that the robot has to work for more than 1 hour; it will be good if the supply voltage of the battery is common to the components. Another concern is that the maximum current needed for the entire circuit should not exceed the battery's maximum current, which it can source. The maximum voltage and current distribution of each part of the circuit is as follows: To meet these specifications, we

Table 2.5: The maximum voltage and current distribution of each part of the circuit

Components	**Maximum voltage (volt) and current (Ampere)**
Motors	12 V,0.7 A
Motor driver	12 V ,2 A
Arduino due	6-12 V (external)

are selecting a 14.8V by combining 4 cells of lithium-ion 3.7V rechargeable battery

with 5800 mah capacity in series for our operation. This battery can also source maximum current up to 30 Ampere and with Max Charge Current: 2.15A (1C)

Figure 2.13: lithium-ion 3.7V rechargeable battery

voltage regulator

The motors needs 12V maximum voltage to rotate on maximum speed so we need to drop out the voltage of our battery which is 14.8 to 12V.to do that we used DC-DC Adjustable Power Supply Module LM2596 Voltage Regulator Module with Strap Calibration FZ0783 and Voltage Meter Voltmeter Led Display 2.14 **Description**

- Input voltage range: 5-32.0V (32V to limit the maximum voltage, be sure to use too aggressive otherwise damaged)
- Output voltage range: 0-30.0V
- Output Power: 30W max
- Output Current: 1.5A (max), over 1.5a application please forced cooling
- Output voltage setting resolution: 0.1V
- Voltage measurement accuracy: ± (1 + 2 words)
- Dimensions: 66mm * 36mm * 14mm

- Display color: red
- IN +: Positive Input
- IN-: Negative Input
- OUT +: positive output
- OUT-: output negative

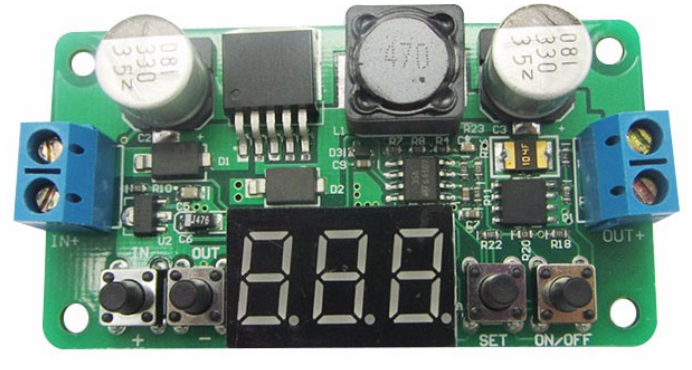


Figure 2.14: Voltage Regulator

2.1.8 Working of the Robot hardware

The main aim of this section was to design the hardware for our robot, which included finding the appropriate hardware components and finding the interconnection between each part. The main functionality of this robot is to perform autonomous path tracking. The hardware design of robot is optimized for autonomous path tracking.

The robot drive is based on differential drive system, which consists of two motors and two wheels. These two motors can move the robot in any pose in a 2D plane by adjusting their velocities and direction.

For controlling the velocity and direction of the wheels, we have to interface a motor controller, which can do these functions. The motor driver we chose should able to control two motors at a time and change the direction and speed.

The motor driver pins are interfaced to a microcontroller board called Arduino Due board, which can send the commands to change the direction and speed of the motor.

Each motor has a rotation feedback sensor called the encoder, which can be used to estimate the robot's position. Encoders values are received on the Due and sent to PC via USB Serial. The PC is interfaced to Arduino Due. The PC has Matlab running on it and it will receive Encoders data and converted to data equivalent to angular and direct velocities. This data can be used to a simulink matlab block for modeling and commanding the robot. The speed commands generated in matlab with a path generation are sent to the Due.The Arduino Due will then process the speed commands and send appropriate PWM values to the motor driver circuit. After designing and discussing the working of the robot hardware, we will discuss the detailed interfacing of each component and the firmware coding necessary for the interfacing.

2.2 Working with Robotic Actuators and Wheel Encoders

2.2.1 Interfacing DC geared motor with Arduino Due

The following figure 2.15 shows the interfacing circuit of two motors using L298 H-Bridge with Arduino Due: The two geared DC motors are connected to OUT1A, OUT1B, and OUT2A, OUT2B of the motor driver. VIN(+) and GND(-) are the supply voltages of the motor. These DC motors can work with 12 Volt supply, so we give 12 Volt as the input voltage. The motor driver will support an input voltage range of 5.5 V to 16 V. The control signals/input pins of motor drivers are on the left-hand side of the driver. The first pins is 1INA and 1INB control the direction of

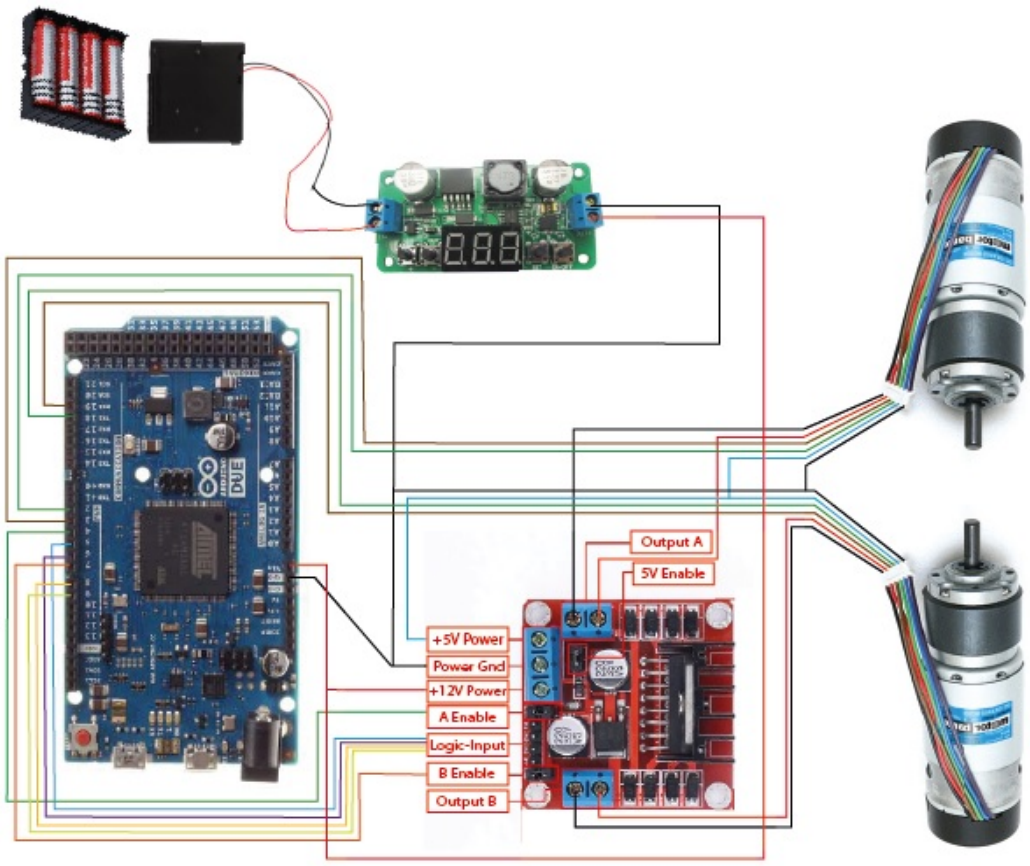


Figure 2.15: all the connections in our new platform

rotation of the motor. The Enable A pin will switch the motor to ON and OFF state and achieve speed control using PWM. The following table 2.6 shows the truth table of the input and output combinations:

Table 2.6: the truth table data with the operating mode of the motor

INA	**INB**	**OUTA**	**OUTB**	**Operating Mode**
1	1	H	H	Brake to Vcc
1	0	H	L	Clockwise (CW)
0	1	L	H	Counterclockwise (CCW)
0	0	L	L	Braker to GND

The two motors work in a differential drive mechanism. The following section discusses differential drive and its operation.

2.2.2 Differential wheeled robot

The robot we have designed is a differential wheeled robot [7]. In a differential wheeled robot, the movement is based on two separately driven wheels placed on either side of the robot's body. It can change its direction by changing the relative rate of rotation of its wheels, and hence, doesn't require additional steering motion. To balance the robot, a free turning wheel The following figure 2.16 shows a typical representation of differential drive:

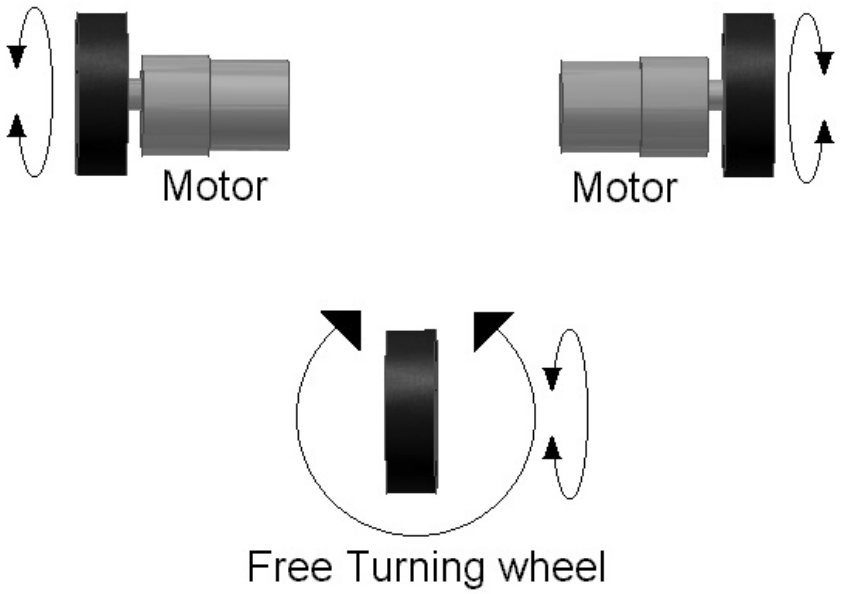


Figure 2.16: representation of differential drive [7]

If the two motors are moving in the same direction, the robot will move forward or backward. If one motor has more speed than the other, then the robot turns to the slower motor side; so to turn left, stop the left motor and move the right motor. The following figure 2.17 shows how we connect the two motors in our robot. The two

motors are mounted on the opposite sides of the base plate and we put two casters in front and back of the robot in order to balance it:

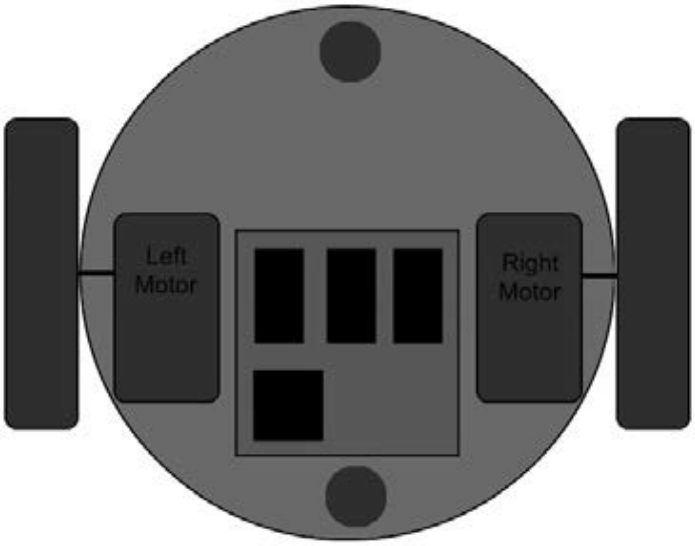


Figure 2.17: connecting the motors in our robot [7]

Next, we can program the motor controller using Arduino Due according to the truth table data then we communicate the arduino with Matlab Simulink to execute our commands and generate the path for the robot.

2.2.3 interfacing the Arduino Due with matlab

In this section we talk about arduino in matlab. we are using Simulink Support Package for Arduino Hardware for simulink to interface the Arduino with matlab [2].

Simulink Support Package for Arduino Hardware

enables you to create and run Simulink models on Arduino boards. The support package includes a library of Simulink blocks for configuring and accessing Arduino sensors, actuators, and communication interfaces. It also enables you to

interactively monitor and tune algorithms developed in Simulink as they run on Arduino. Refer to `https://fr.mathworks.com/help/supportpkg/arduino/ug/install-support-for-arduino-hardware.html` [2] we install the Arduino tolbox for simulink.

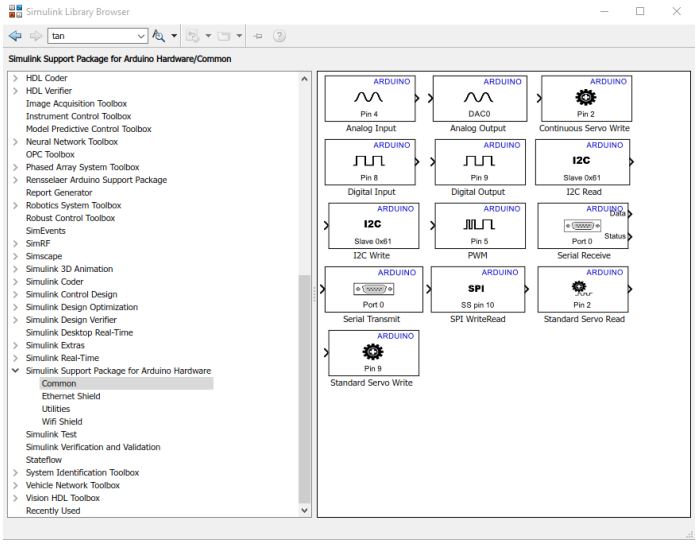


Figure 2.18: Simulink Support Package for Arduino Hardware

Communication between simulink and the Arduino hardware

In order to ensure that our computer can properly communicate with the Arduino board,we build and run the following Simulink diagram using the steps below:

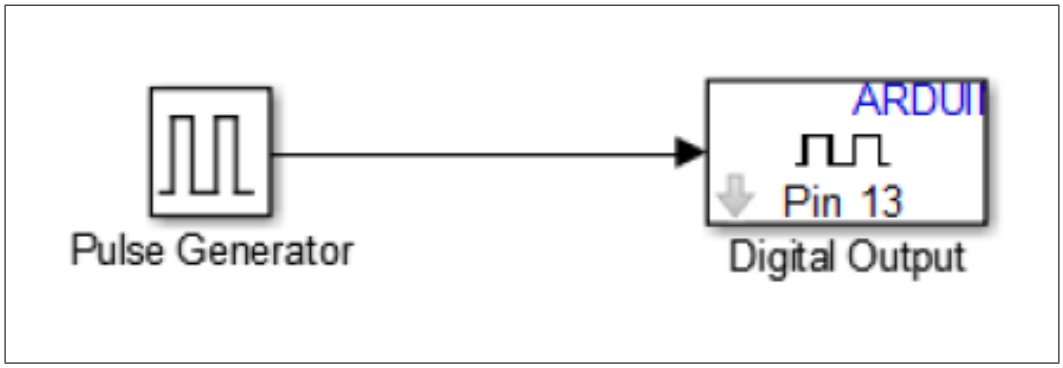


Figure 2.19: example of Simulink diagram for Blinking a LED using arduino

1. Open MATLAB then open a new Simulink model.
2. Find and assemble the required blocks.
   - Pulse Generator is under View → Library Browser → Simulink → Sources
   - Digital Output is under View → Library Browser → Simulink Support Package for Arduino Hardware → Common
3. Change the output pin to 13, which is connected to the on board LED. This is done by double clicking on the Digital Output block.
4. Double click on the Pulse Generator to change the amplitude to 1, the Period to 10, and the pulse width to 50%.
5. Under the tools menu go to 'Run on Target Hardware' and click 'Prepare to Run' (figure 2.20). In the window that pops up we will need to select Arduino Due in the drop down for Target Hardware (figure 2.21).

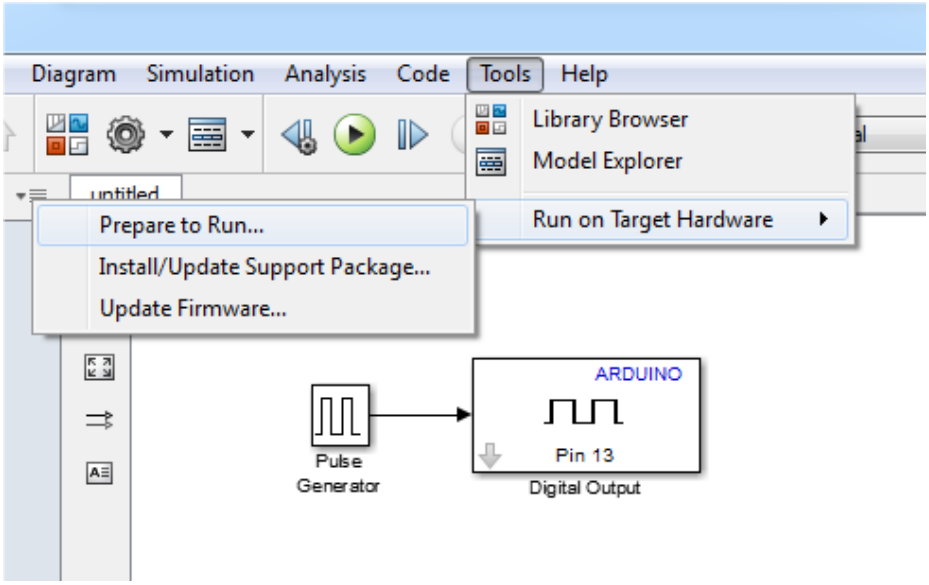


Figure 2.20: working on matlab Step 5-1-1

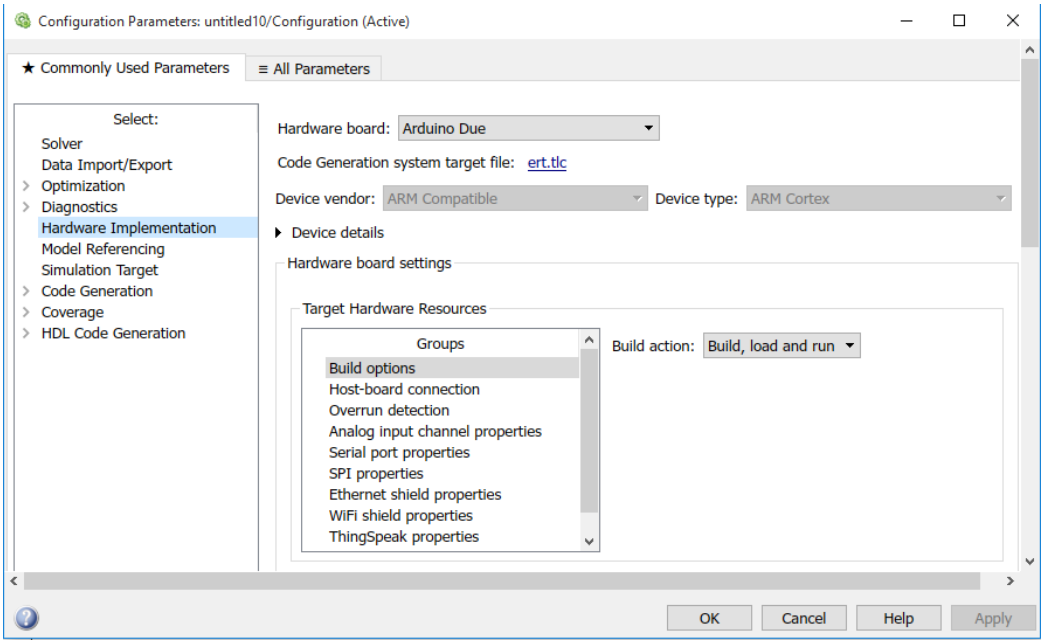


Figure 2.21: working on matlab Step 5-1-2

- Set host COM port: "should be set to "Automatically", click OK. We can specify the COM port your board if for some reason there are issues downloading the code.(figure 2.22)

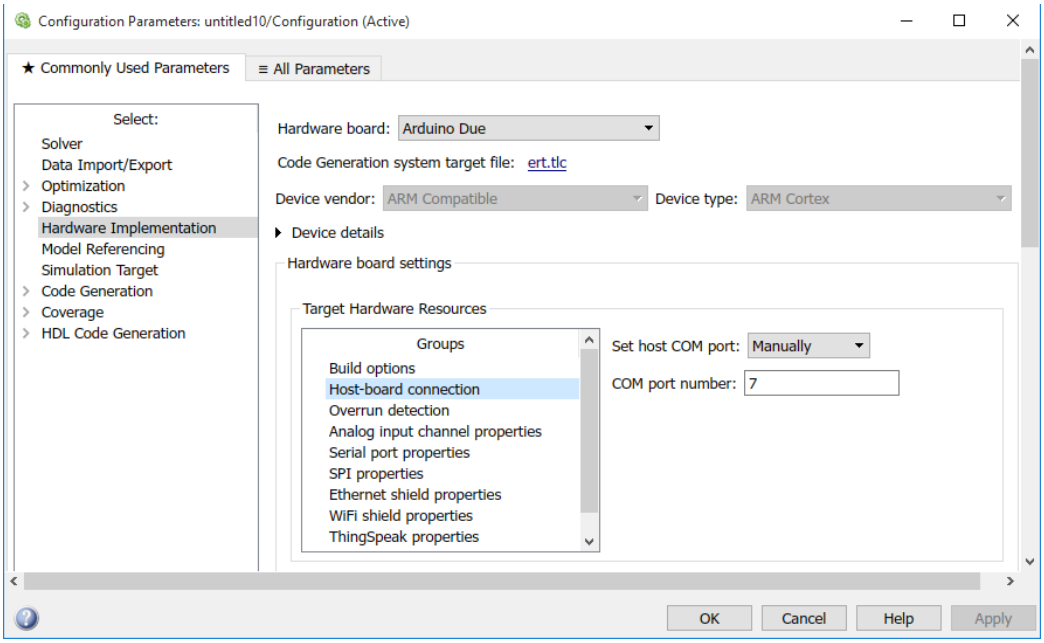


Figure 2.22: working on matlab Step 5-2

- Right click on the background of the Simulink window and select 'Model Configuration Parameters'.
- we go to 'Solver' on the left hand menu and we type 'inf' into the 'Stop time' spot so that the model will keep running. We make sure that the solver options are set to "Fixed-step" and "discrete (no continuous states)". The "Fixed-step size" should be 0.05. This is how fast in seconds the control loop will execute. In this case, every 50 milliseconds it will execute the Simulink code (figure 2.23).

6. Compile and download the code to the board

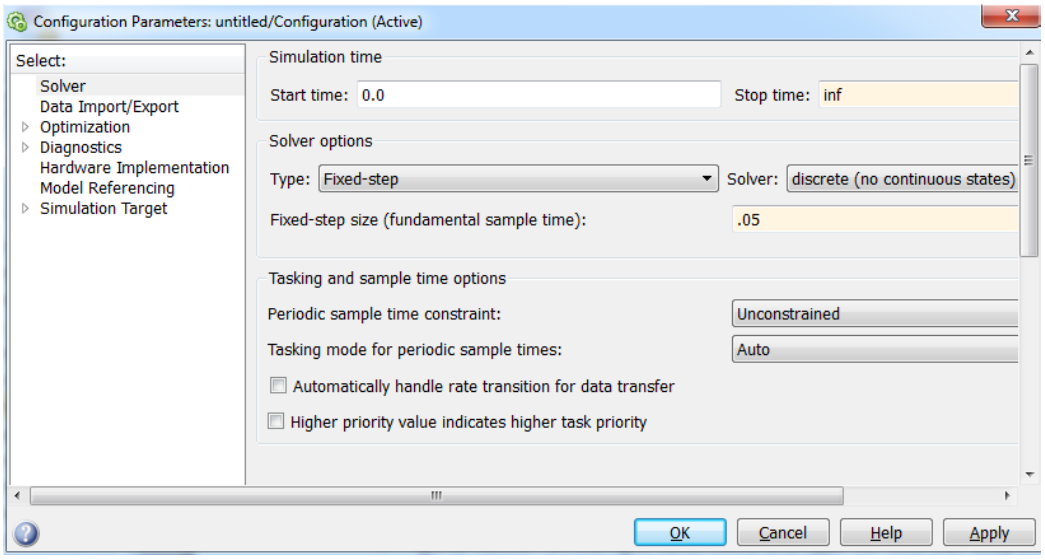


Figure 2.23: working on matlab Step 5-3

:

- Click "Deploy to Hardware Button" or the keystroke Ctrl+B (figure 2.24).

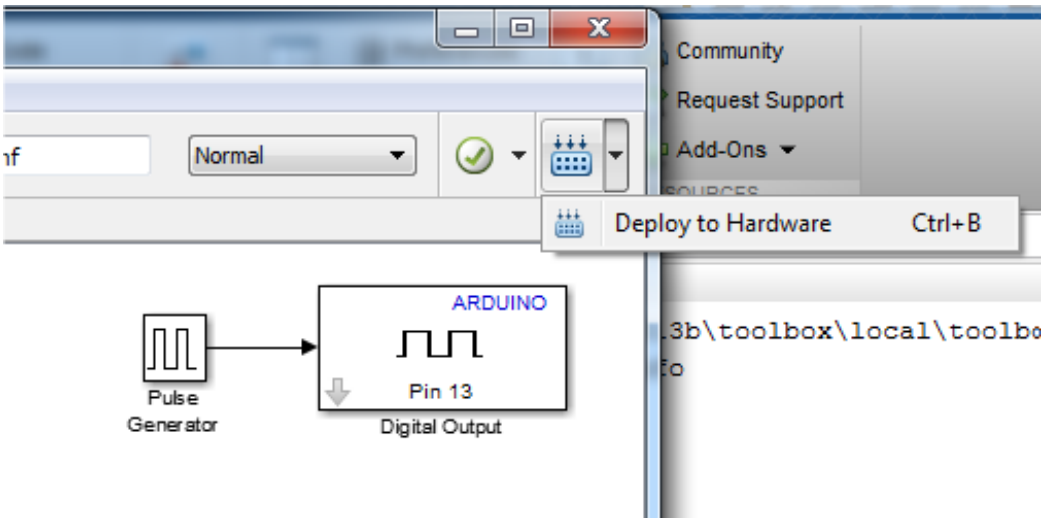


Figure 2.24: Deploy to Hardware Button

The "Deploy to Hardware button" will download the code to the board and will not interact or connect with it. In "Normal" Mode the code is downloaded to the board, and no information is passed between it and your computer unless your program specifically instructs the board to do so. At this point the onboard LED connected to pin 13 should start blinking on and off at a very low frequency.

Communicating with External Mode

- External mode and the green play button allows bi-directional communication to/from the application board to the PC, but can limit how fast your code can execute.
  - The "Deploy to Hardware Button" downloads the code and does not connect or interact with it, it can allow the code to run faster.
- When external mode is selected and the green play button is pressed it downloads additional code to the board to allow this communication. This does increase the program size (requires more memory).
- When you use external mode, you can change parameters while the system is running
  - However, this has an impact on the performance due to the communication bandwidth, the fastest we can run in this mode and still meet the control loop time is approximately 30 milliseconds. Newer USB 3.0 ports and computers may run faster.
  - You can tell how fast we can run our code by examining the running clock in the Simulink diagram (fiugure 2.25) after we press the play button. When the code is running the green bar will be flashing and the time in seconds should be advancing: If this clock advances as fast as the normal

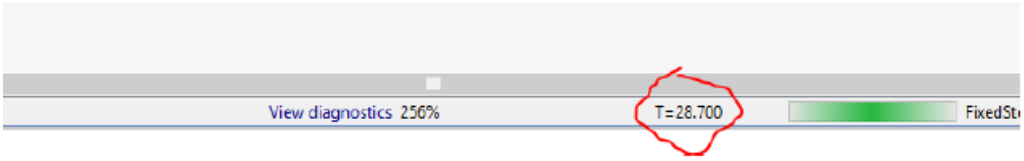


Figure 2.25: ining the running clockin the Simulink diagra

real word clock your code is "keeping up". If, for example, it can take 1 minute for the this timer to reach 10 seconds your code is not keeping up with real time and your requested sample time is to fast – it can not compute everything it needs to do fast enough.

- The code can be run much faster (in the orders of 1 or 2 milliseconds) if external mode is not used, by using "Deploy to Hardware" instead of the green run button.

2.2.4 Interfacing quadrature encoder with Arduino Due

The quadrature encoder has 4 pins. The Vcc pin is the 5 V power for the encoder. The Gnd pin is the ground pin. The 2 other pins are channel A and channel B to count the interruption for a (CW) direction or (CCW) direction.

The pins of the channels are connected to the interrupts pins in Arduino. The due does not have a problem about the number of the interrupt pins because all the pins are interrupt pins. In this case we choses pins (18,19) for the encoder1 and pins (2,3) for the other. As shows the figure 2.15

2.3 designing the schematic and the pcb circuit interface for the robot

in this section we discuss the schematic and the pcb design circuit board that we made to interface all electronics components of our robot. For this design we used

eagle as schematic and pcb design software. The following figure shows the schematic of the design:

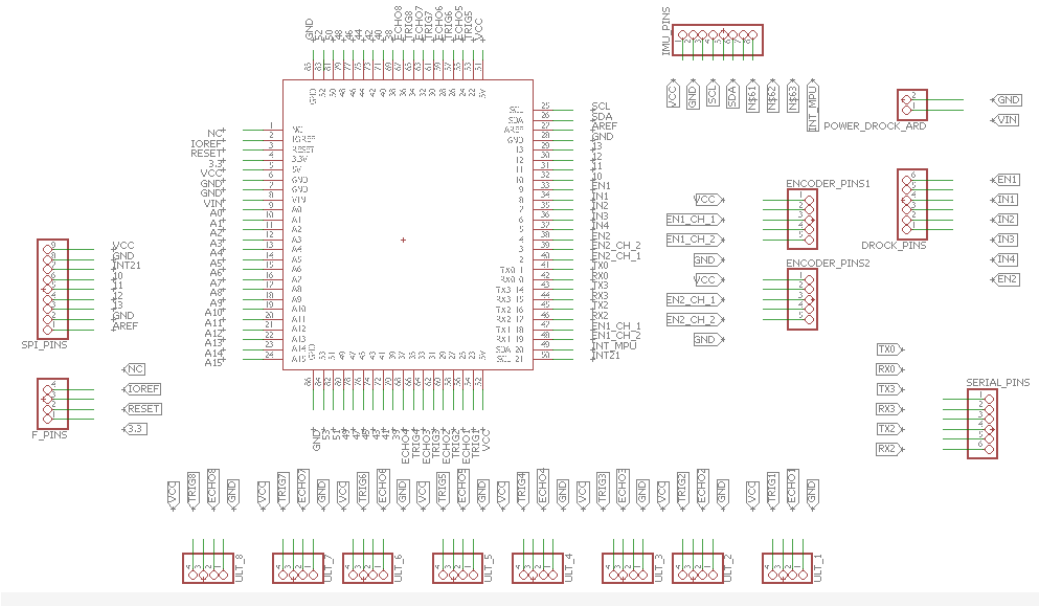


Figure 2.26: schematic circuit design of our board

The following figure shows the PCB board design:

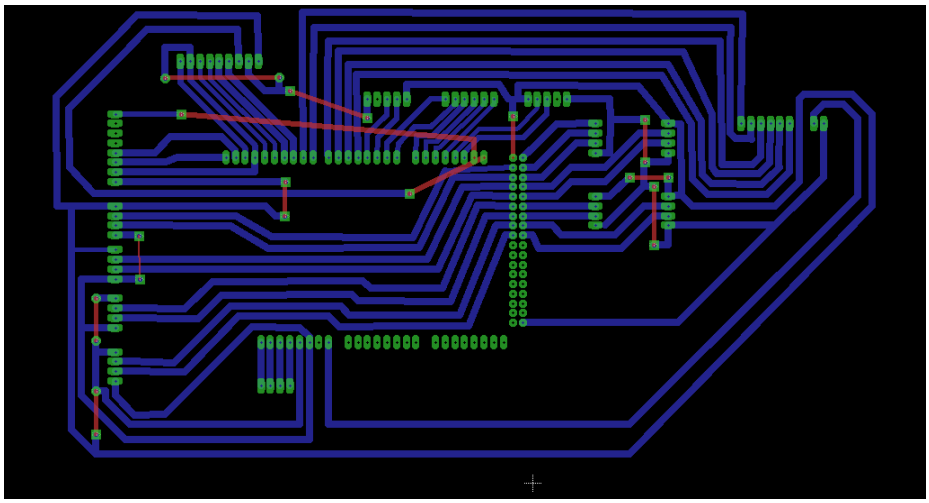


Figure 2.27: PCB board designed by eagle

after we designed our pcb board we fabricate the board then we solders all the electronics part on the board.we test the interconnection on the board to make sure

that there is no short circuit on our board. The following figure shows our final board for our robot:

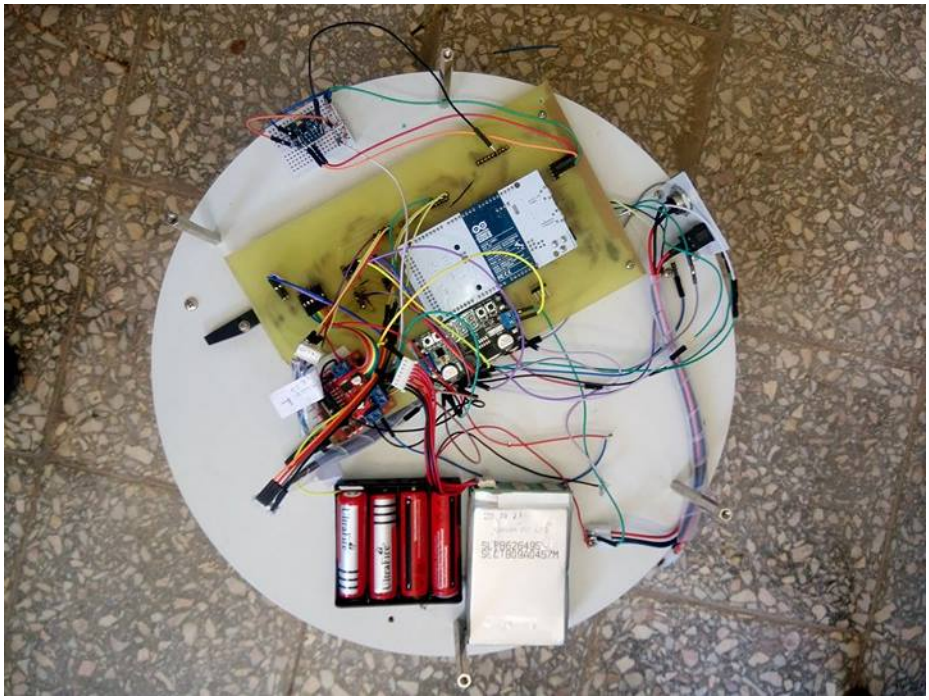


Figure 2.28: the final PCB board manufactured

Chapter 3

Modeling Differential-Drive Wheeled Mobile Robots

Deriving a precise mathematical model is a crucial part of designing control system for any physical plants such as mobile robots. In this chapter dynamics and kinematics of a differential drive robot are derived and differences between the two models and limitations of the kinematic model are explored. The pure rolling nature of the wheels causes a reduction in the local mobility of the robot. This limitation is expressed as a non-holonomic constraint which is further discussed.

3.1 Non-Holonomic Constraint

Wheeled vehicles are generally subjected to a constraint. For instance, a car can reach any final configuration in its plane, but it can never move sideways. Hence, depending on the goal configuration, it requires to perform a series of maneuvers (such as parallel parking) to reach the desired state.

First, holonomic and non-holonomic systems have to be defined. Let's consider a mechanical system with generalized coordinates $q \in C$, where C is the configuration

space of the proposed system and coincides with $\mathbb{R}^n$. For such system, a constraint is called *Kinematic* when it only involves generalized coordinates (q) and velocities ($\dot{q}$). Kinematic Constraints are usually defined in *Pfaffian* Form :

$$v_i^T(q)(\dot{q}) = 0 \qquad i = 1, ..., k < n \tag{3.1}$$

where v_i's are k linearly independent vectors.

If all of the kinematic constraints defined by Equation (3.1) are integrable to a form of

$$h_i(q) = m_i \qquad i = 1, ..., k < n$$

where, m_i is the integration constant, then they are considered to be *holonomic* constraints and the system subjected to them is called a *holonomic* system. Joints in a robotic manipulator are common example of such constraints.

Each holonomic constraint causes a loss of accessibility of the system in its configuration space. Hence, for a system with k holonomic constraints, the accessible configurations are reduced to a $(n - k)$ dimensional subset of C.

A *non-holonomic* system on the other hand, is subjected to at least one nonintegrable (i.e. *non-holonomic*) constraint. Although such constraint limits the local mobility of the system, due to its non-integrable nature, the accessibility to C is not affected. Hence, generalized coordinates are not reduced. However, generalized velocities in a system subjected to k non-holonomic constraint belongs to a $(n - k)$ dimensional subspace.

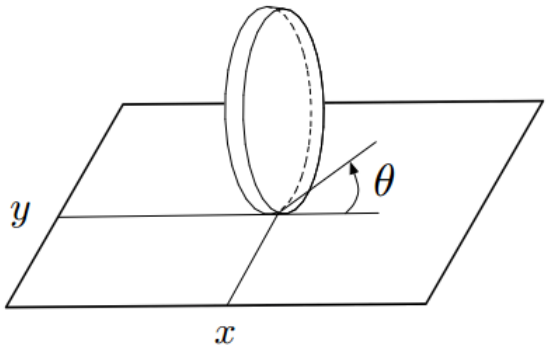


Figure 3.1: Pure rolling disk and its generalized coordinates in 2D plane

Wheels are typical sources of non-holonomic constraints. Consider the disk in Figure 3.1 with generalized coordinates $q = [x \quad y \quad z]^T$, assuming the disk can only roll on the touching plane without slipping to the sides (i.e. there is no velocity component for the contact point perpendicular to the plane containing the disk). This can be defined as:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3.2}$$

Rewriting Equation 3.2 in *pfaffian* form will result in

$$[\sin\theta \quad -\cos\theta \quad 0]\dot{q} = 0 \tag{3.3}$$

As it can be seen, Equation (3.3) is not integrable causing the nature of the wheel to be non-holonomic. Also, it should be emphasized that this constraint implies no loss in accessibility of the wheel configuration space, meaning that wheel can reach any goal configuration $q_f = [x_f \quad y_f \quad z_f]^T$ starting from any initial state $q_i = [x_i \quad y_i \quad z_i]^T$.

Figure 3.2: Mecanum wheel can move sideways and is holonomic

This kinematic constraint applies to all wheel-based systems, making them non-holonomic. However, it should be noted that not all wheels are non-holonomic. Configuration of caster wheel proposed in mic or Mecanum wheels (as shown in Figure 3.2), which are commonly used in omnidirectional robots, are exempt from this constraint and in fact are considered, holonomic.

3.2 Robot Kinematics

Reordering k kinematic constraints in Equation (3.1) into matrix form $V^T(q)\dot{q} = 0$, shows that the generalized velocities $(\dot{q})$ belongs to null space of $V^T(q)$, which is $(n-k)$ dimensional and agrees with what was stated earlier in this chapter.

Choosing a basis for $\mathcal{N}(V^T(q))$ denoted by $[b_1(q)...b_{n-k}(q)]$ a kinematic model of the constrained mechanical system is given by:

$$\dot{q} = \sum_{i=1}^{n-k} b_i(q)u_i = B(q)\mathbf{u} \tag{3.4}$$

where $\mathbf{u} = [u_1...u_{n-k}]^T \in \mathbb{R}^{n-k}$ is the input vector and $q \in \mathbb{R}^n$ is the state vector.

The basis for nullspace of $V^T(q)$ is not unique and typically, it can be chosen such that inputs u_i represent a physical concept. However, these inputs should not directly represent forces or torques, hence the name *kinematic model*.

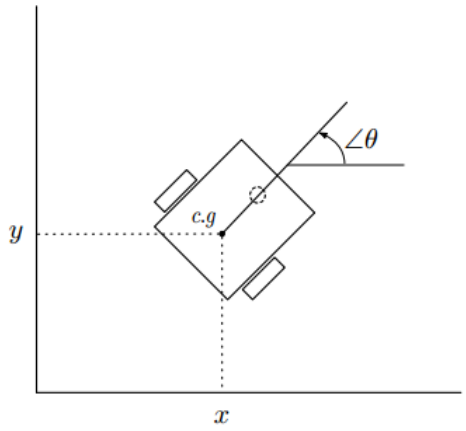


Figure 3.3: Generalized coordinates for a mobile robot

Consider the mobile robot in Figure 3.3. Using generalized coordinate vector $q = [x \quad y \quad \theta]$ the robot's posture can be defined on its whole configuration space. The wheels driving the robot make it non-holonomic and imposes the pure rolling constraint on the system which as discussed before, is expressed as

$$V^T(q)\dot{q} = [\sin\theta \quad -\cos\theta \quad 0]\dot{q} = 0 \tag{3.5}$$

a basis for $\mathcal{N}(V^T(q))$ is then chosen as

$$B(q) = [b_1(q) \quad b_2(q)] = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

Using this basis and based on Equation (3.4) the kinematic model will be

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w \tag{3.7}$$

where, the inputs have clear physical interpretation, v and w are the linear velocity and angular velocity of the robot, respectively, as shown in Figure 3.3.

There exists a one to one relation between formerly mentioned velocities and actual velocity inputs, which are angular speed of two wheels denoted by w_L and w_R for left and right wheels, respectively and is governed by:

$$v = \frac{r(w_R + w_L)}{2} \qquad w = \frac{r(w_R - w_L)}{l} \tag{3.8}$$

where, r is the radius of the wheels and l is the distance between the wheels as shown in Figure 3.4.

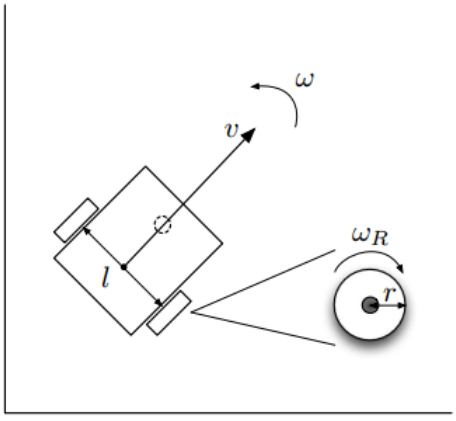


Figure 3.4: Linear and Angular velocity of the robot

3.3 Robot Dynamics

Inputs in a kinamtic model do not directly represent actual inputs (i.e. forces and/or torques). In another words, we are neglecting dynamics of a system when dealing just with a kinematic model. Consequently, It is important to derive the dynamic model and explore its characteristics.

There are two methods for dynamic model derivation. Newton-Euler method describes the system in terms of all the forces and momentum acting on the system based of direct interpretations of Newtons Second Law of Motion.

On the other hand, *Lagrange* method incorporates the concepts of *Work and Energy* to indirectly derive the equations of motion. Here, Lagrange method is chosen due to its more systematic nature and automatic elimination of workless and constraint forces.

Lagrangian of a system is defined as the difference between its kinetic and potential energy

$$\mathcal{L}(q,\dot{q}) = \mathcal{T}(q,\dot{q}) - \mathcal{U}(q) = \frac{1}{2}\dot{q}^T I(q)\dot{q} - \mathcal{U}(q) \tag{3.9}$$

where, $\mathcal{T}(q,\dot{q})$ and $\mathcal{U}(q)$ are the kinetic and potential energy, respectively and $I(q)$ is the inertia matrix of the mechanical system.

Lagrange-Euler equations representing the dynamics are expressed as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T - \left(\frac{\partial \mathcal{L}}{\partial q}\right)^T = 0 \tag{3.10}$$

This general form of Lagrange equation applies to holonomic system. In case of a non-holonomic system we have to replace Equation (3.10) by

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T - \left(\frac{\partial \mathcal{L}}{\partial q}\right)^T = S(q)\tau + V(q)\lambda \tag{3.11}$$

where, $S(q)$ is a (n by m) matrix mapping the ($m = n - k$) external inputs τ to generalized forces, $V(q)$ is the transpose of $V^T(q)$ in Equation (3.5) governing the nonholonomic constraint. $\lambda \in \mathbb{R}^m$ is the vector of the Lagrange multipliers representing the forces required to impose such constraint in the configuration plane. $V(q)\lambda$ is the reaction forces at generalized coordinate plane.

Based on Equation (3.9) and Equation (3.11), the dynamical model of a non-holonomic mechanical system is obtained as

$$I(q)\ddot{q} + n(q, \dot{q}) = S(q)\tau + V(q)\lambda \tag{3.12}$$

$$V^T(q)\dot{q} = 0 \tag{3.13}$$

$$n(q, \dot{q}) = \dot{I}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T I(q)\dot{q}\right)\right)^T + \left(\frac{\partial \mathcal{U}(q)}{\partial q}\right)^T \tag{3.14}$$

where $n(q, \dot{q})$ given in Eq (3.14) represents vector of centripetal and coriolis terms [5].

Let I be the moment of inertia around the central vertical axis and m the mass of the differential drive mobile robot in 3.3. Using the Lagrange representation in

Equation (3.12) and Equation (3.13), the dynamic model of the robot is then derived.

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tau_l \\ \tau_a \end{bmatrix} + \begin{bmatrix} \sin\theta \\ -\cos\theta \\ 0 \end{bmatrix} \lambda \tag{3.15}$$

$$[\sin\theta \quad -\cos\theta \quad 0]\dot{q} = 0 \tag{3.16}$$

Where, τ_l and τ_a represent the linear force and angular torque of the mobile robot, respectively. The robot is in inertial frame coriolis and centripetal term $n(q, \dot{q})$ is non-existence [5].

The relations between linear velocity (v), angular velocity (w) and the generalized velocities ($\dot{q}$) are

$$\begin{cases} v = \sqrt{\dot{x}^2 + \dot{y}^2} \\ w = \dot{\theta} \end{cases} \tag{3.17}$$

Using derivatives of Equation (3.17), the dynamic model represented in matrix form in Equation (3.15) can be rewritten in a more familiar form.

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\cos\theta \\ w = \dot{\theta} \end{cases} \tag{3.18}$$

$$\begin{cases} \dot{v} &= \frac{\tau_l}{m} \\ \dot{w} &= \frac{\tau_a}{I} \end{cases} \tag{3.19}$$

Where, Equations (3.18) are the kinematic models and Equations (3.19) integrate the dynamics of the robot.

It should be noted that the constraint equation (Equation (3.16)) is valid in any case. Similar to linear and angular velocity of the robot and wheels' angular velocity, angular torque τ_a and linear torque τ_l are related to the torques of each wheel by Equation (3.20):

$$\tau_l = \frac{(\tau_R + \tau_L)}{r} \qquad \tau_a = \frac{l(\tau_R - \tau_L)}{r} \tag{3.20}$$

where, τ_R and τ_L respectively represent right and left wheel torques.

Such toques and velocities are produced by the actuators driving each wheel. It is important to appreciate the fact that these actuators have their own internal dynamics and can not realize speed commands instantaneously.

3.4 Actuator Dynamics

DC motors are widely used in robotic applications and are the main type of actuators used in mobile robots. Consequently, it is important to analyze and integrate their dynamics into robot's model.

3.4.1 Linear state-space model

Under the assumption of a homogenous magnetic field,the direct current (DC) motor is modeled as a linear transducer from motor current to electrical torque. The classical model of the DC motor is composed of a coupled electrical and a mechanical subsystem as shown in figure 3.5.

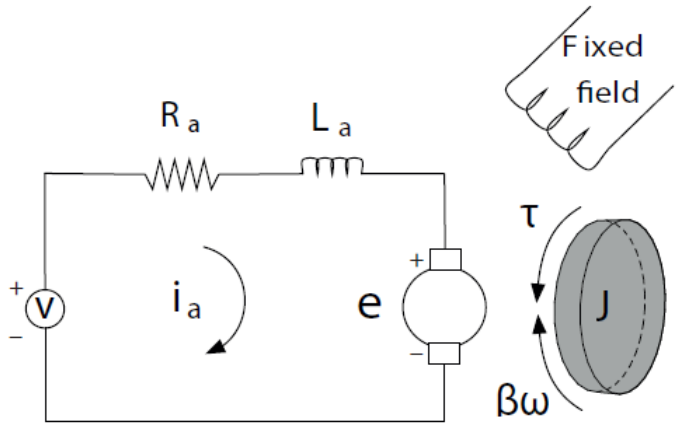


Figure 3.5: Circuit equivalent of a DC motor with a free body attached

The angular velocity w is controlled by the input voltage v_a with a constant voltage drop attributed to the brush and rotor resistance, and a back-electromotive force (EMF) caused by the rotary armature. The motor inductance contributes proportional to the change in motor current i_a. The motor current couples the electrical component with the mechanical one, as it generates the driving torque. This torque is antagonized by the motor inertia and structure damping. The motor dynamics are described by:

$$v_a(t) = L_a \frac{di_a}{dt} + R_a i_a(t) + K_b w(t) \tag{3.21}$$

$$K_m i_a(t) = J \frac{dw}{dt} + \beta w(t) \tag{3.22}$$

where K_m, K_b and β denote the motor torque, the back EMF and the damping constants. J denotes the mechanical inertia including the motor armature and shaft. L_a and R_a represent the inductance and the total connection resistance of the motor.

Using Equations (3.21) and (3.22), the transfer function from input voltage to angular velocity is

$$\frac{W(s)}{V_a(s)} = \frac{K_m}{(L_a s + R_a)(Js + \beta) + K_b K_m} \tag{3.23}$$

Closed loop block diagram of DC motor model expressed in Equation (3.23) is shown in Figure 3.6, angular displacement can also be found by integrating $W(s)$.

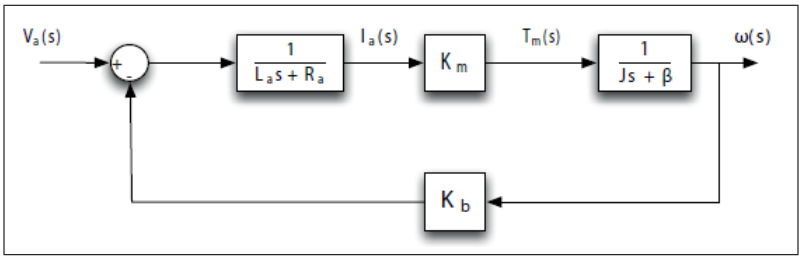


Figure 3.6: DC Motor block diagram

However, for many applications this structure is not sufficient. The main drawback of the linear state-space model is a negligence of nonlinear effects, whose properties can significantly affect the dynamic behavior of a modeled system. To complete the representation of essential physical phenomena effecting in the motor structure the frictional nonlinearity must be included.

3.4.2 System nonlinearity

According to Paduart et al. (2006)) the linear state-space model with a multivariable nonlinear input function $f(\mathbf{x}(t), u(t))$ assumes the general form:

$$\dot{\mathbf{x}} = \mathbf{A}x + \mathbf{B}u + \mathbf{H}f(\mathbf{x}, u), \tag{3.24}$$

In which A is the system matrix, B is the input vector, and the coupling vector H links the nonlinearity with the linear part. In context of permanent magnet DC motor Coulomb friction constitutes the major source of nonlinear behavior (Knudsen and Jensen (1995)). Additional nonlinearities emerge from the inhomogeneity of the stator magnetic field and transfer characteristics of the amplifier and IO elements as well as motor cogging and ripple effects (see Proca et al. (2003)).

For many applications a static friction model that includes Coulomb and viscous parts suffices to capture the main frictional phenomena.

The linear viscous friction is already comprehended in the damping term in equation (3.22). Considering the nonlinear Coulomb friction which depends on the rotation direction and introducing the state vector $x = [i,\ w]^T$ results in:

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{R_m}{L} & -\frac{K_b}{L} \\ \\ \frac{K_m}{J} & -\frac{\beta}{J} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{1}{L} \\ \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ \\ \frac{T_f}{J} \end{bmatrix} \operatorname{sgn}(x_2) \tag{3.25}$$

in which T_f denotes the Coulomb friction coefficient. The overall model has six independent parameters.

We will use the linear model of the DC motor to derive the linear state-space actuator + mobile robot *dynamics model of the robot* ($T_f = 0$).

3.5 Linear state-space *actuator + mobile robot dynamics* model

In order to derive a precise model of the *actuator + mobile robot* dynamics, the DC Motor model derived in section 3.4 along with derived dynamics in Equation (3.19) are used. This model is illustrated in Figure 3.7. In this model, DC motors are considered non identical.

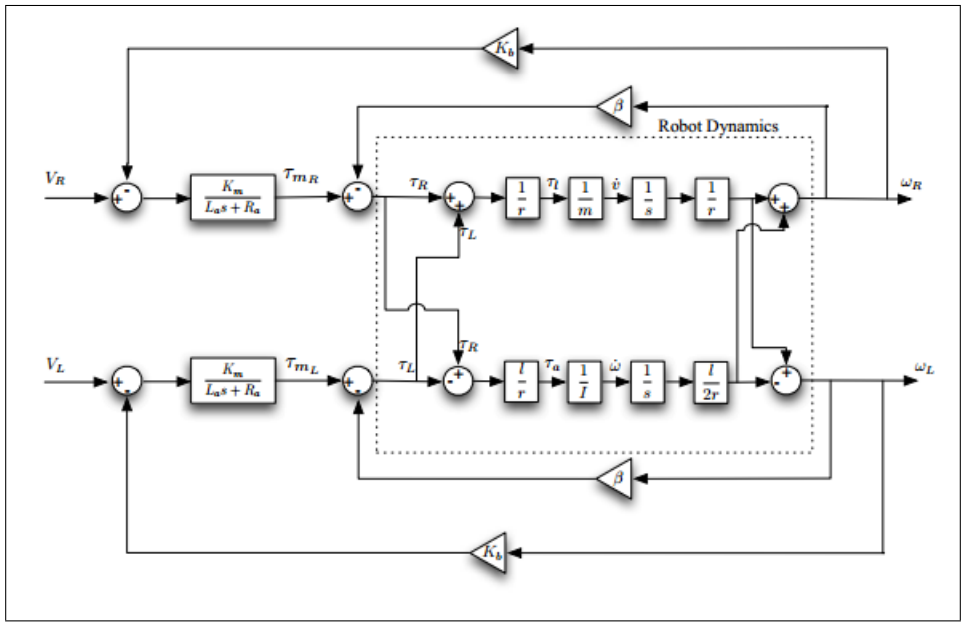


Figure 3.7: Actuator and body dynamics block diagram from V_R & V_L to w_R & w_L

We choose $\mathbf{x} = [i_{a1}, i_{a1}, v, w]^T$ to be the state vector. from the block diagram shown in figure 3.7 we have :

$$\begin{cases} \dot{v} &= \frac{1}{m}\left[\frac{1}{r}K_{m1}i_{a1} + \frac{1}{r}K_{m2}i_{a2} - \frac{1}{r}\beta_1 w_R - \frac{1}{r}\beta_2 w_L\right] \\ \dot{w} &= \frac{1}{I}\left[\frac{l}{r}K_{m1}i_{a1} - \frac{l}{r}K_{m2}i_{a2} - \frac{l}{r}\beta_1 w_R + \frac{l}{r}\beta_2 w_L\right] \end{cases} \tag{3.26}$$

and from equation (3.8) we have

$$\begin{bmatrix} w_R \\ w_L \end{bmatrix} = M_i \begin{bmatrix} v \\ w \end{bmatrix} \tag{3.27}$$

where M_i is a transformation matrix defined as:

$$\begin{bmatrix} \frac{1}{r} & \frac{l}{2r} \\ \frac{1}{r} & -\frac{l}{2r} \end{bmatrix}$$

combining equations (3.26) and (3.27) we get :

$$\begin{bmatrix} \dot{v} \\ \dot{w} \end{bmatrix} = J_{2\times2}T_{2\times2}K_{m2\times2}i_a - J_{2\times2}T_{2\times2}\beta_{2\times2}M_i v_w \tag{3.28}$$

where $J_{2\times2}$, $T_{2\times2}$, $K_{m2\times2}$, $\beta_{2\times2}$, i_a and v_w matrices are defines as :

$$J_{2\times2} = \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & \frac{1}{I} \end{bmatrix}, \qquad T_{2\times2} = \begin{bmatrix} \frac{1}{r} & \frac{1}{r} \\ \frac{l}{r} & -\frac{l}{r} \end{bmatrix}, \qquad K_{m2\times2} = \begin{bmatrix} K_{m1} & 0 \\ 0 & K_{m1} \end{bmatrix}$$

$$\beta_{2\times2} = \begin{bmatrix} \beta_1 & 0 \\ \\ 0 & \beta_2 \end{bmatrix}, \qquad i_a = \begin{bmatrix} i_{a1} \\ \\ i_{a2} \end{bmatrix}, \qquad v_w = \begin{bmatrix} v \\ \\ w \end{bmatrix}$$

and from (3.21) and (3.27) we get :

$$\dot{i}_a = -L_{a2\times2}R_{a2\times2}i_a - L_{a2\times2}K_{b2\times2}M_i v_w + L_{a2\times2}V \tag{3.29}$$

where $L_{a2\times2}$, $R_{a2\times2}$, $K_{b2\times2}$, V_a, matrices are defines as :

$$L_{a2\times2} = \begin{bmatrix} \frac{1}{L_{a1}} & 0 \\ \\ 0 & \frac{1}{L_{a1}} \end{bmatrix}, \qquad R_{a2\times2} = \begin{bmatrix} R_{a1} & 0 \\ \\ 0 & R_{a2} \end{bmatrix}$$

$$K_{b2\times2} = \begin{bmatrix} K_{b1} & 0 \\ \\ 0 & K_{b1} \end{bmatrix}, \qquad V = \begin{bmatrix} V_R \\ \\ V_L \end{bmatrix},$$

Then the state-space model of the *actuator + mobile robot* dynamics is derived choosing $\mathbf{y} = [w_R,\ w_L]^T$, and $\mathbf{u} = [V_R,\ V_L]^T$:

$$\begin{bmatrix} \dot{i}_a \\ \\ \dot{v}_w \end{bmatrix} = \begin{bmatrix} -L_{a2\times2}R_{a2\times2} & -L_{a2\times2}K_{b2\times2}M_i \\ \\ J_{2\times2}T_{2\times2}K_{m2\times2} & -J_{2\times2}T_{2\times2}\beta_{2\times2} \end{bmatrix} \begin{bmatrix} i_a \\ \\ v_w \end{bmatrix} + \begin{bmatrix} L_{a2\times2} \\ \\ 0 \end{bmatrix} \begin{bmatrix} V_R \\ \\ V_L \end{bmatrix} \tag{3.30}$$

$$\begin{bmatrix} w_R \\ \\ w_L \end{bmatrix} = \begin{bmatrix} 0 & M_i \end{bmatrix} \begin{bmatrix} i_a \\ \\ v_w \end{bmatrix} \tag{3.31}$$

Where :

$$A = \begin{bmatrix} -\frac{R_{a1}}{L_{a1}} & 0 & -\frac{K_{b1}}{rL_{a1}} & -\frac{K_{b1}l}{2rL_{a1}} \\ \\ 0 & -\frac{R_{a2}}{L_{a2}} & -\frac{K_{b2}}{rL_{a2}} & \frac{K_{b2}l}{2rL_{a2}} \\ \\ \frac{K_{m1}}{rm} & \frac{K_{m2}}{rm} & -\frac{(\beta_1+\beta_2)}{(rm^2)} & \frac{l(\beta_1-\beta_2)}{(2rm^2)} \\ \\ \frac{K_{m1l}}{rI} & -\frac{K_{m2}}{rI} & \frac{l(\beta_2-\beta_1)}{(rI^2)} & -\frac{l^2(\beta_1+\beta_2)}{(2rI^2)} \end{bmatrix}$$

and :

$$B = \begin{bmatrix} \frac{1}{L_{a1}} & 0 \\ \\ 0 & \frac{1}{L_{a2}} \\ \\ 0 & 0 \\ \\ 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 0 & \frac{1}{r} & \frac{l}{2r} \\ \\ 0 & 0 & \frac{1}{r} & -\frac{l}{2r} \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ \\ 0 & 0 \end{bmatrix}$$

Chapter 4

Parameter Estimation

4.1 Parameter Estimation

In order to design a control law based on the dynamic model developed earlier, we have to estimate the model parameters.

The model presented in Figure 3.7 contains both the motors' parameters witch are:

- The motor torque K_m.
- The back EMF K_b.
- The damping constant β.
- The inductance L_a.
- The total connections resistance R_a.

and the robot parameters witch are :

- The inertia of the robot I, witch contains the moment of inertia of the body about the central vertical axis, and the wheels with the motors, about the wheels axis.

- The mass of the robot m with prior approximation (r and l are measured)

After consulting the robot information delivered by the manufacturer, we didn't find any parameter value, thus we decided to estimate all this parameters starting with the motors' ones. to do that we used the Parameter Estimation tool of the SIMULINK Design Optimization toolbox.

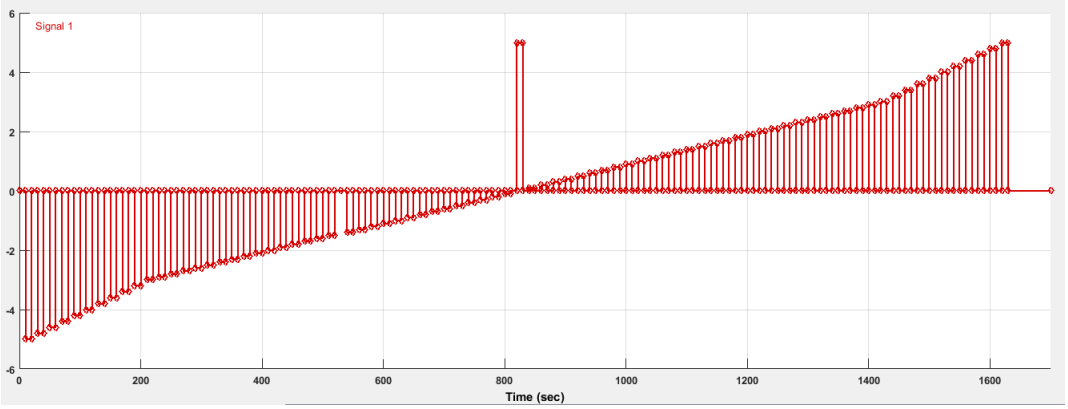


Figure 4.1: The Input signal

As shown in Figure 4.1 the voltage input of the motors is supplied through the motors' drivers witch are controlled by the PWM signal (average value of range $[0, 5]$) sent by the ARDUINO Due, the drivers amplify the input PWM signal to a DC voltage of range $[0, 10]$ with polarity inversion capability according to the logic pins inputs, the amplifying function is not linear though, it should be known experimentally.

4.1.1 Experiment Setup

At the first stage we will estimate the motors' parameters separately, and the drivers' $Input_{PWM}/Input_{DC}$ functions, to do that we proceed as follow:

1 Excite the motor with a sufficiently rich $Input_{PWM}$ signal (the wheel over the ground)

The input signal used is shown in Figure 4.1 , it's a stairs like signal with 10 seconds at each stair and 10 seconds of 0 value between the stairs, having values from -5 to 5 with 0.2 difference from -5 to 3 and from 3 to 5 and with 0.1 difference from -3 to -3 to increase the resolution before entering the saturation, the signal is recorded to workspace variable and sent to the driver as a PWM signal of average value in the range $[0, 5]$ with 2 digital signals to encode the polarity of the voltage applied the driver, the SIMULINK block is shown in Figure 4.2

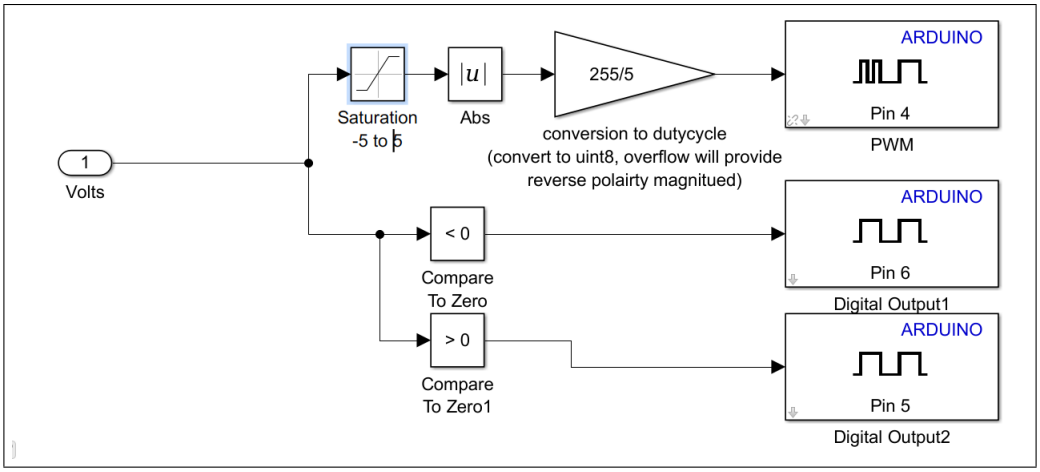


Figure 4.2: The SIMULINK motor driver block

2 Measure the motor angular velocity and filter it

As mentioned in chapter 2, the motor is occupied with quadrature (2 channels) Hall effect encoder witch gives 14 pulse par channel per revolution. Counting each channel for the rising and the falling edge will give us 14×4 counts per revolution, and giving that the reduction gears have a ratio of 50, the total counts per revolution will be 2800, but after some calibration works in witch we compare the measured distance according to the encoder and the real one, we figured out that the encoder gives around 2360 counts per revolution.

To measure the distance traveled by the wheel we have to read the 2 channels signals and count the rising and the falling edges of each, and for that we use the *Encoder block* of the *Rensselear* ARDUINO *Support package* library witch support a range of most used sensors and modules. The library is dedicated to the ATMEGA micro-controllers that the ARDUINO boards (nano, uno, mega,...) are based on, and because the *Encoder block* use interrupt hardware pins and giving the fact that we use ARDUINO Due in the project for its higher performance witch is based on the Cortex-M3 micro-controller, we decided to tweak the source code so that we can use it for our hardware.

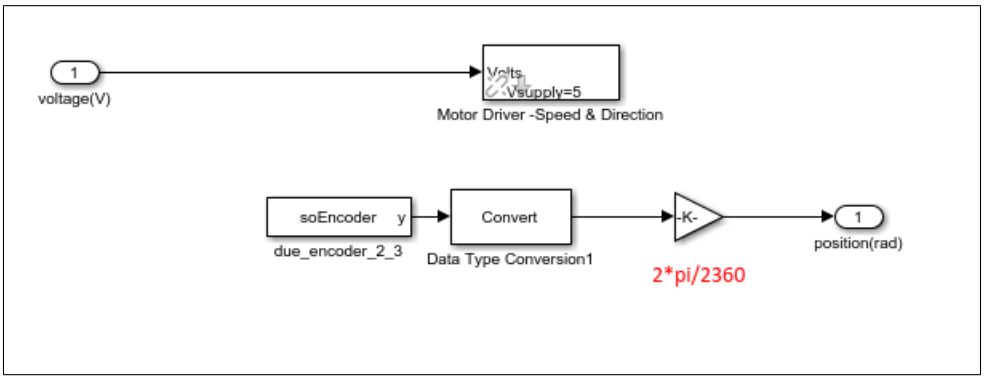


Figure 4.3: The voltage to angular velocity subsystem

the voltage to angular velocity subsystem in the Figure 4.3 consists of the motor driver block introduced in Figure 4.2, the encoder block and the conversion gain of value $(2 \times pi)/2360$ giving the angular position of the wheel, then to get the angular velocity we add a discrete derivative block, the full I/O model is shown in Figure 4.4

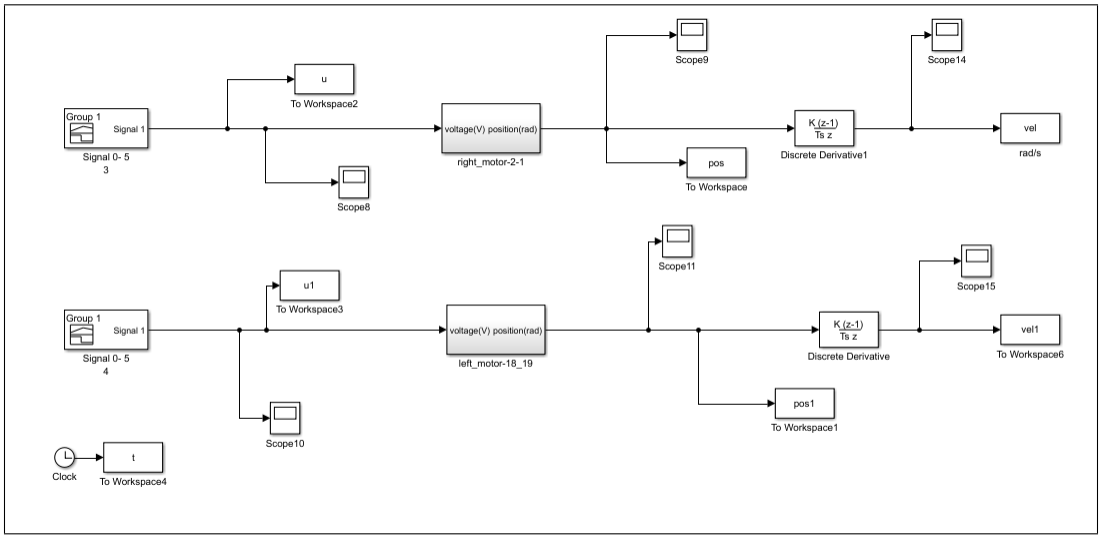


Figure 4.4: The I/O model

The angular velocity response of one motor to the input signal in the Figure 4.1 is shown in the figure below :

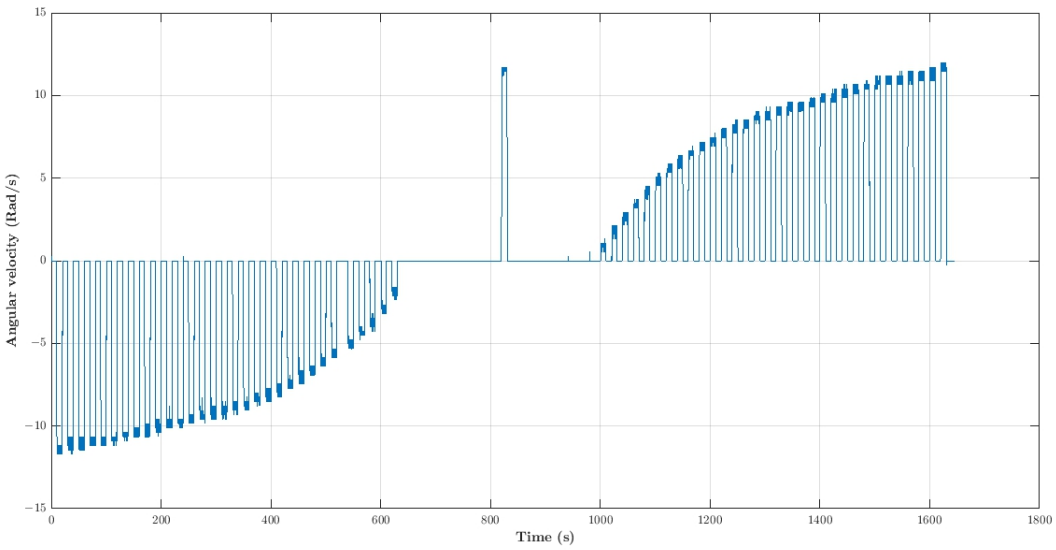


Figure 4.5: The angular velocity response of the motor

As shown in Figure 4.5 low voltage pulses generate zero velocities, therefore the existence of *coulomb friction* characterized by the parameter T_f in (3.25) witch will be included in the identified model later.

Because of the discrete and noisy nature of the position signal (counting signal), derivating it will produce a spiky signal, therfore it needs to be filtered to be used later, so we use the MATLAB *smooth* function with *'loess'* method and an appropriate *span* value. In Figure 4.6 we compare a smoothed portion of the velocity signal with the unsmoothed correspondent.

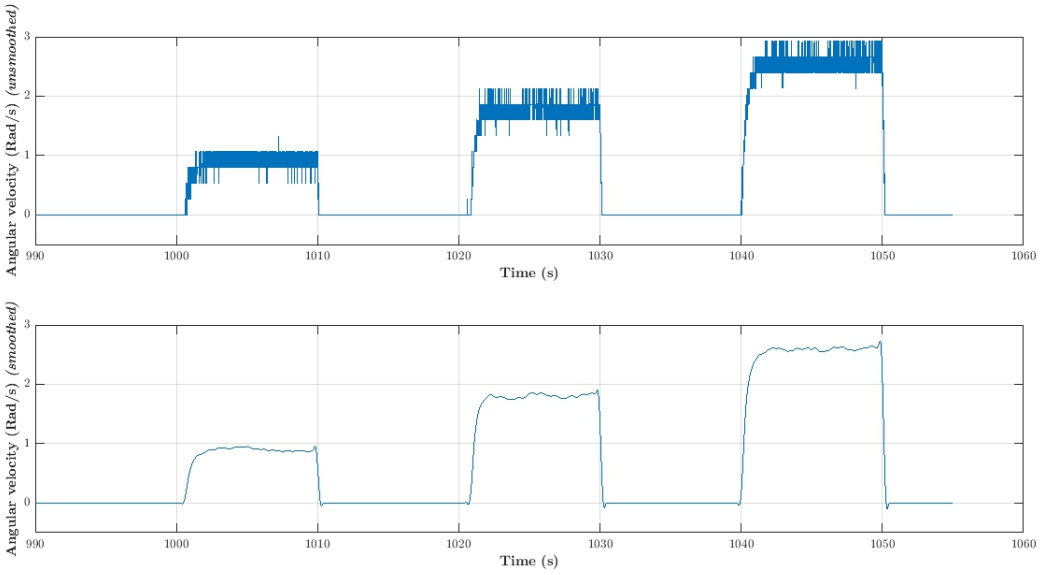


Figure 4.6: smoothed and unsmoothed velocity signal

3 Measure the real voltage input of the motor

During the experiment we read the voltage supplying the motor by a multimeter for each pulse (10 seconds is sufficient to read the steady-state voltage value), we did that to extract the effect of the motor's driver witch we call $Input_{PWM}/Input_{DC}$ map function, the result is shown below:

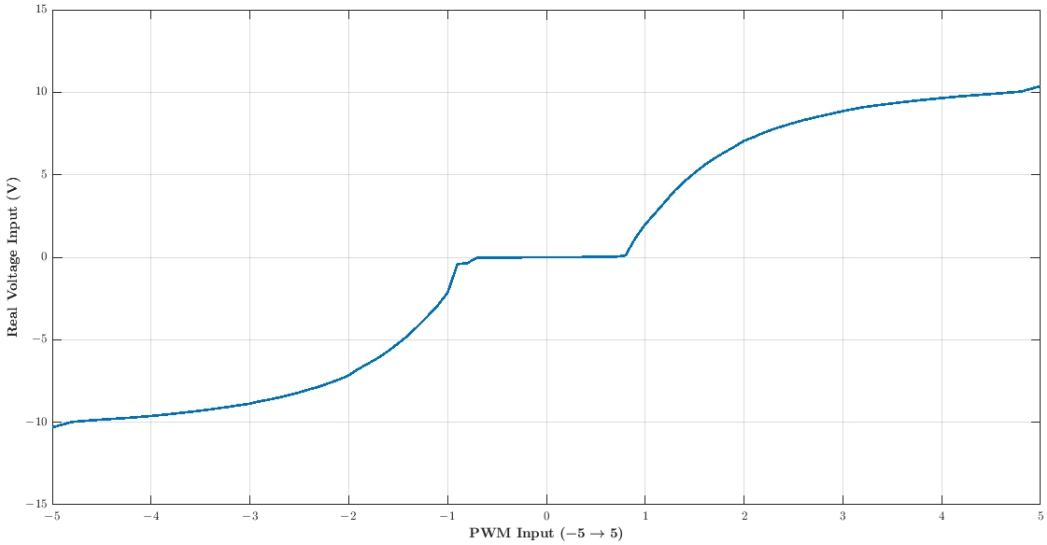


Figure 4.7: $Input_{PWM}/Input_{DC}$ map function

4 Create the SIMULINK motor model containing the parameters to estimate

Based on the motor equations (3.21) and (3.22) we create a SIMULINK model with the exception that we include the *coulomb friction* T_f parameter, this is done by substituting the β gain with a Coulomb & Viscous Friction block with T_f and β parameters $(y = sign(w) * (\beta * abs(w) + T_f)$.

The model also include a *loockup table* block to map from $PWM(-5 \rightarrow 5)$ Input to the real voltage Input of the motor, the model is shown in Figure 4.8. we also substitute K_m with $0.95 * K_b$ considering 95% of electrical to mechanical energy conversion efficiency.

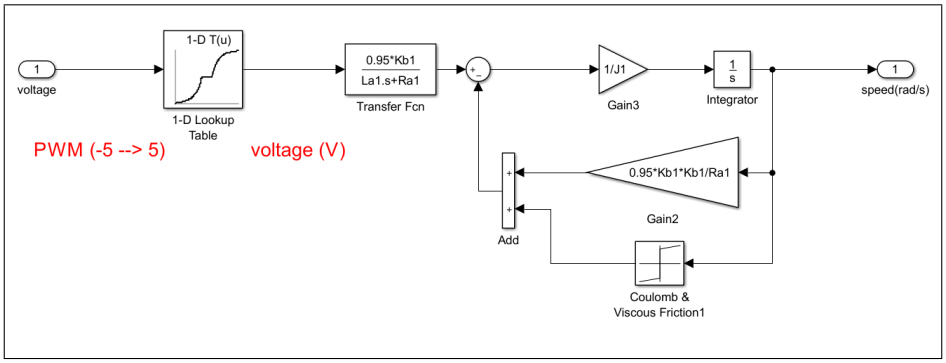


Figure 4.8: The SIMULINK parameterized motor model

5 Estimate the model parameters using Parameter Estimation tool of the SIMULINK Design Optimization toolbox

At this point we have a parametriazed model and both the $Input_{PWM}$ and the $Output_{velocity}$ (measured data). The tool increases model accuracy so that the model response converges to the real system response, the tool formulates the estimation problem as an optimization one, with the sum squared error as a cost function, by default the *Trust-Region-Reflector least square algorithm* is used.

The tool GUI after estimation is shown in the figure below

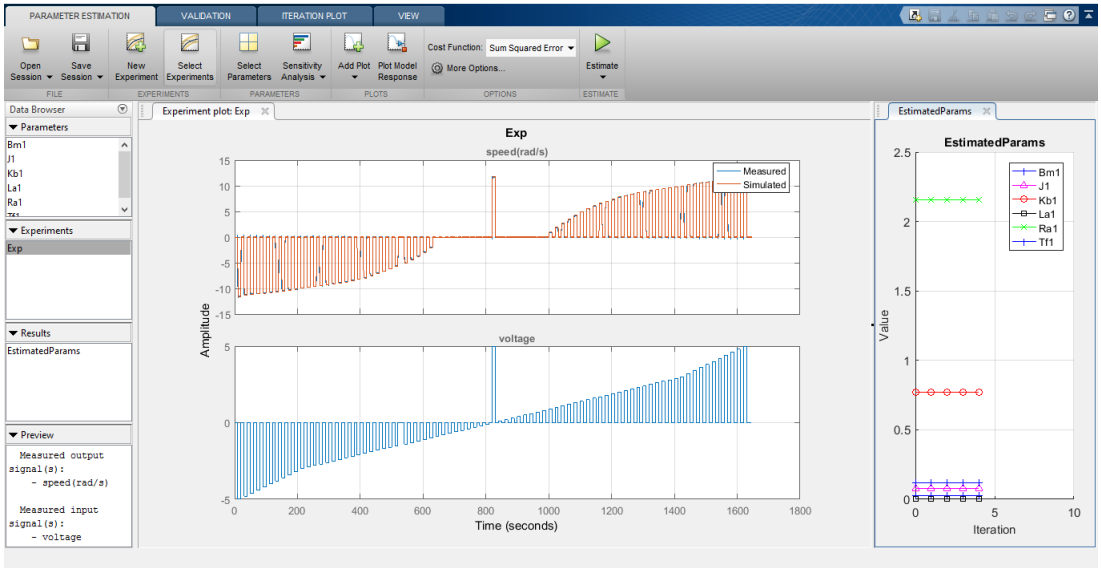


Figure 4.9: The Parameter Estimation tool GUI

As shown in Figure 4.9, after estimation, the model response in red matches the recorded response in blue.

First click *New Experiment* and enter the experimental data as fellow

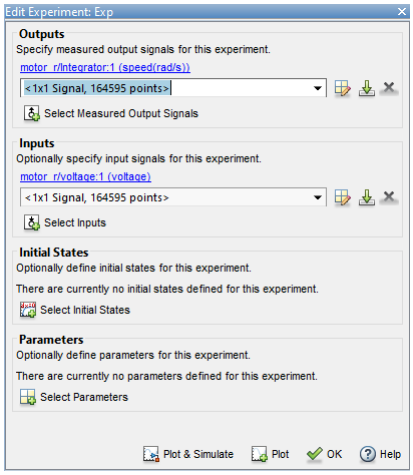


Figure 4.10: Enter the experimental data

Then click *Select parameters* to choose the model parameters to estimate and provide the min and max limits and the initial guesses as shown bellow

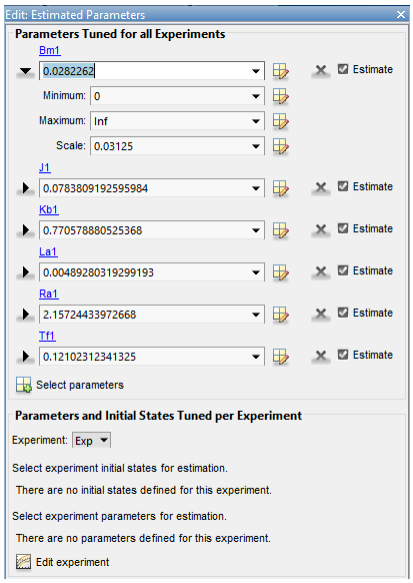


Figure 4.11: Parameters to estimate

Now after estimating the motors' parameters, we still have to estimate the robot mass and inertia m and I, for that, this time with the robot on the ground we repeat the steps 1,2,4 and 5 but this time on the robot model shown in Figure 4.12

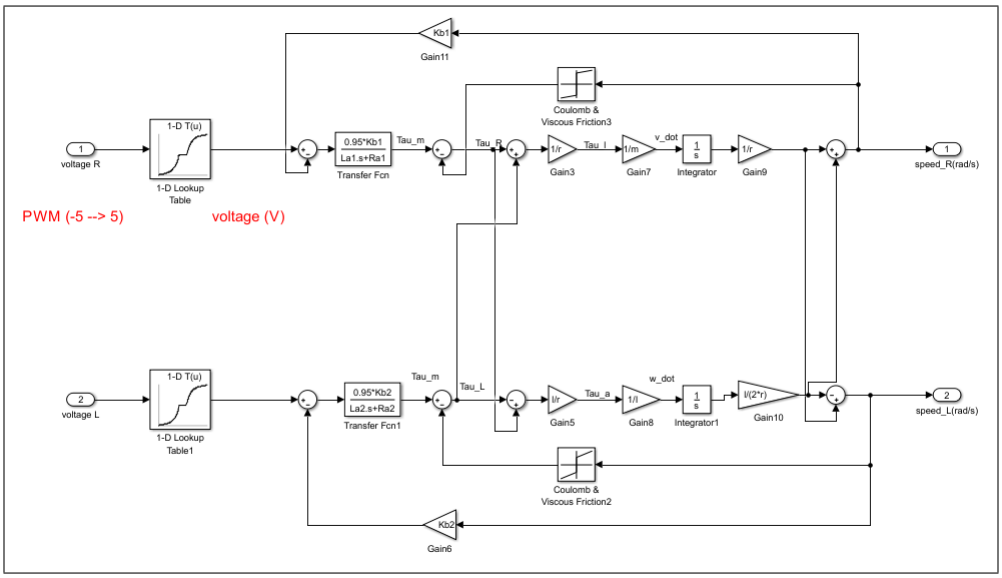


Figure 4.12: The SIMULINK robot parameterized robot model

And using the input signals in Figure 4.13 for the right motor and in Figure 4.14 for the left motor

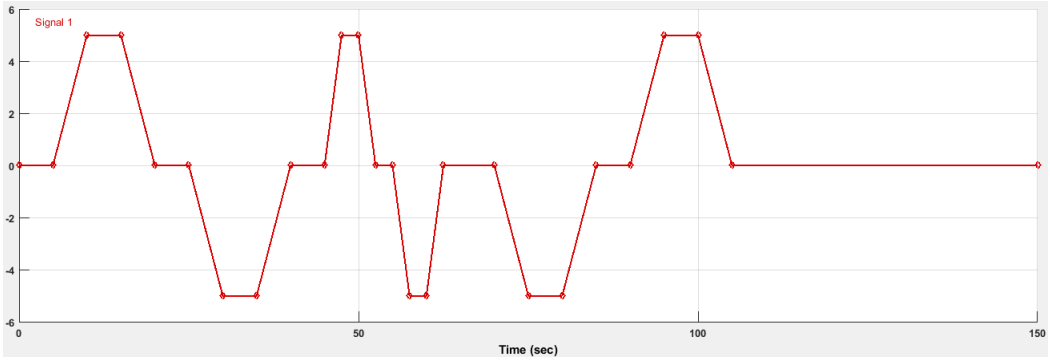


Figure 4.13: Right motor input signal

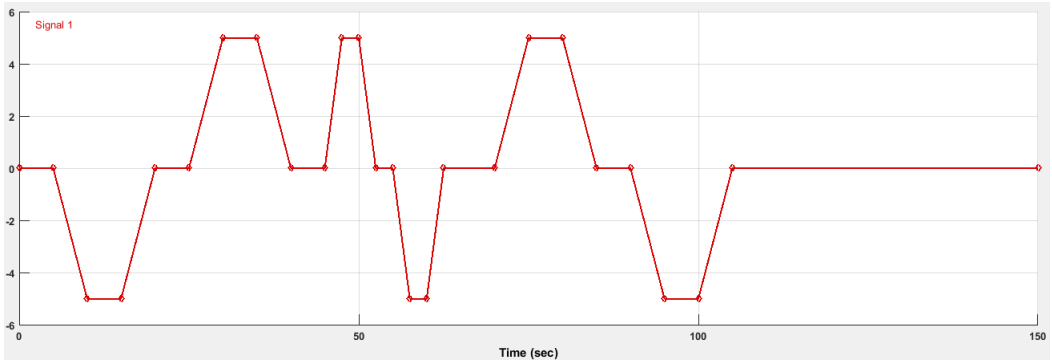


Figure 4.14: Left motor input signal

We designed the input signals to move the robot around its self and in line path. The parameter estimation GUI after estimation is shown below

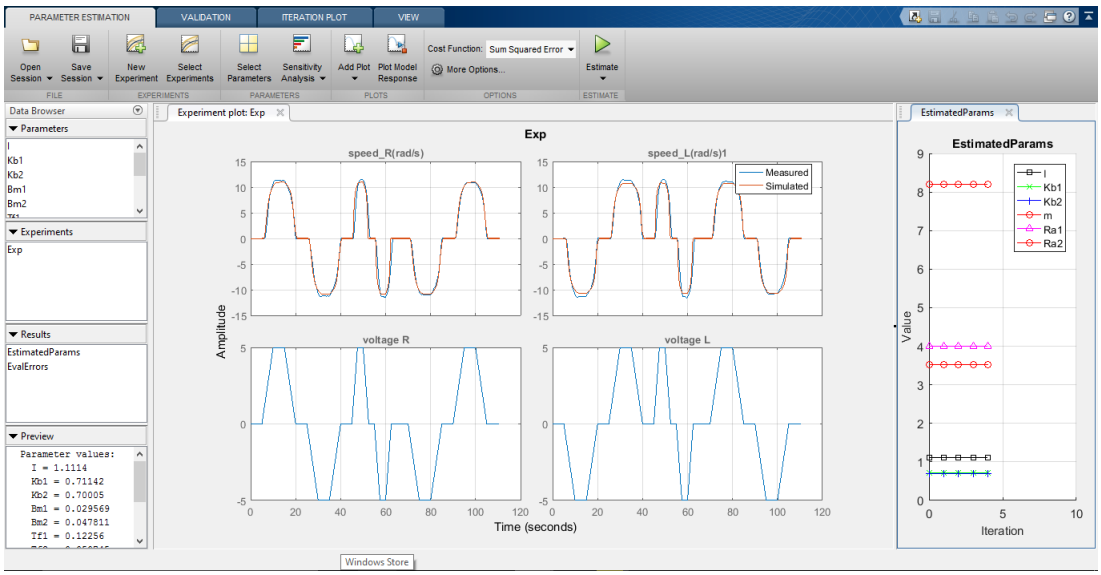


Figure 4.15: The Parameter Estimation tool GUI of the second experiment

Using the 2 experiments results we present the estimated parameters in the Table 4.1.1

Table 4.1: Dynamic Model Parameter Description and their estimated Values

Parameter	Description	Nominal Value
K_{m1}	Torque Constant (Right motor)	0.6758 $N.m/Amp$
K_{m2}	Torque Constant (Left motor)	0.6650 $N.m/Amp$
K_{b1}	Back EMF Constant (Right motor)	0.7114 $V/(rad/sec)$
K_{b2}	Back EMF Constant (Left motor)	0.700 $V/(rad/sec)$
β_1	damping Constant (Right motor)	0.0286 $N.m.s$
β_2	damping Constant (Left motor)	0.0445 $N.m.s$
T_{f1}	Coulomb friction coefficient (Right motor)	0.1226 $N.m$
T_{f2}	Coulomb friction coefficient (Left motor)	0.0507 $N.m$
L_{a1}	Armature Inductance (Right motor)	5.1×10^{-3} H
L_{a2}	Armature Inductance (Left motor)	9.6×10^{-3} H
R_{a1}	Armature Resistance (Right motor)	4 ohm
R_{a2}	Armature Resistance (Left motor)	3.5327 ohm
m	Mass	8.2 Kg
I	Inertia	1.1114 $Kg.m^2$
r	Wheel Radius	0.065 m
l	Distance between the wheels	0.21 m

Chapter 5

Robot Control Design and Simulation

This chapter is dedicated to address the control design and simulation of the mobile robot, the control structure contains 2 nested control loops:

- The inner loop (velocity loop) dealing with the dynamic model of the robot to track the desired linear and angular velocities.
- The outer loop (position loop) this time using the kinematic model, we design a controller to track a reference trajectory.

When working with light weight robots or assuming that the motors are powerful enough, a control structure based only on the kinematic model is sufficient, in the other cases where the robot dynamics are not negligible, the dynamics model and the controller (inner loop) are essential.

In our case, regarding the robot mass and inertia we consider the 2 control loops, the overall control scheme is shown in the Figure 5.1

First the trajectory planning provide the reference trajectory to fellow $[x_d, y_d, \dot{x}_d, \dot{y}_d]$ then the outer loop controller (kinematic controller) generates the desired linear and angular velocities. The dynamic controller drive the robot to the desired velocities by giving the adequate voltages to the motors , lastly the kinematic model use the

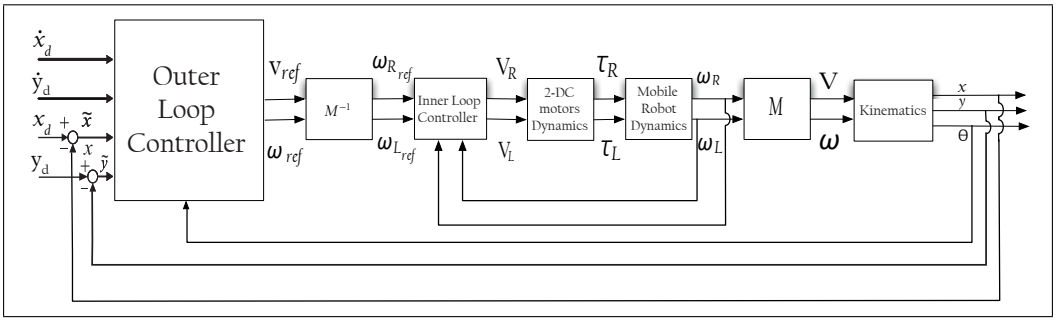


Figure 5.1: The overall control scheme of the mobile robot

measured right and left wheels' angular velocities to estimate the robot pose (position and orientation).

5.1 Dynamic control design

In this section we design a centralized LQG servo controller for the mobile robot dynamics. The plant is defined in equations (3.30) and (3.31), in order to achieve zero steady-state error to step reference command, two integrator have to be augmented to the plant output, but first we will briefly discuss the LQG problem and solution.

5.1.1 LQG control

The LQG control is an optimal control in witch the plant must be linear and known, and that the measurement noise and input noise are stochastic with known statistical proprieties. That is, we have a plant model

$$\dot{x}(t) = Ax(t) + Bu(t) + B_w w_n(t) \tag{5.1}$$

$$y(t) = Cx(t) + v_n(t) \tag{5.2}$$

where $w_n(t)$ and $v_n(t)$ are the input and measurement noise respectively, witch usely assumed to be uncorrelated zero-mean Gaussian stochastic processes with constant

power spectral density matrices W_n and V_n respectively. That is, $w_n(t)$ and $v_n(t)$ are white noise processes with covariances

$$E\{w_n(t)w_n(\tau)^T\} = W_n\delta(t-\tau) \tag{5.3}$$

$$E\{v_n(t)v_n(\tau)^T\} = V_n\delta(t-\tau) \tag{5.4}$$

and

$$E\{w_n(t)v_n(\tau)^T\} = 0 \ , \ E\{v_n(t)w_n(\tau)^T\} = 0 \tag{5.5}$$

where E is the expectation operator and $\delta(t-\tau)$ is a delta function.

The LQG control problem is to find the optimal control $u(t)$ witch minimizes

$$J = E\left\{\lim_{T\to+\infty}\frac{1}{T}\int_0^T [x^TQx + u^TRu]dt\right\} \tag{5.6}$$

Where Q and R are appropriately chosen constant weighting matrices (design parameters) such $Q = Q^T \geq 0$ and $R = R^T > 0$. The name LQG arises from the use of Linear model, an integral Quadratic cost function, and Gaussian white noise processes to model noises.

The solution to the LQG problem, known as the Separation Theorem is surprisingly simple an elegant. It consists of first determining the optimal control to a deterministic linear quadratic regulator (LQR) problem, it means the above LQG problem without $w(t)$ and $v(t)$. The solution to this problem can be written in terms

of a simple state feedback law

$$u(t) = -K_r x(t) \tag{5.7}$$

where K_r is a constant matrix witch is independent of W_n and V_n, the statistical properties of the plant noise.

The next step is to find an optimal estimate $\hat{x}(t)$ of the state $x(t)$, so that $E\left\{[x(t) - \hat{x}(t)]^T[x(t) - \hat{x}(t)]\right\}$ is minimized.

The optimal state estimate is given by the Kalman filter and is independent of Q and R. The required solution of the LQG problem is then found by replacing $x(t)$ by $\hat{x}(t)$, to give $u(t) = -K_r\hat{x}(t)$. We therefore see that the LQG problem and its solution can be separated into two distinct parts, as illustrated in Figure 5.2.

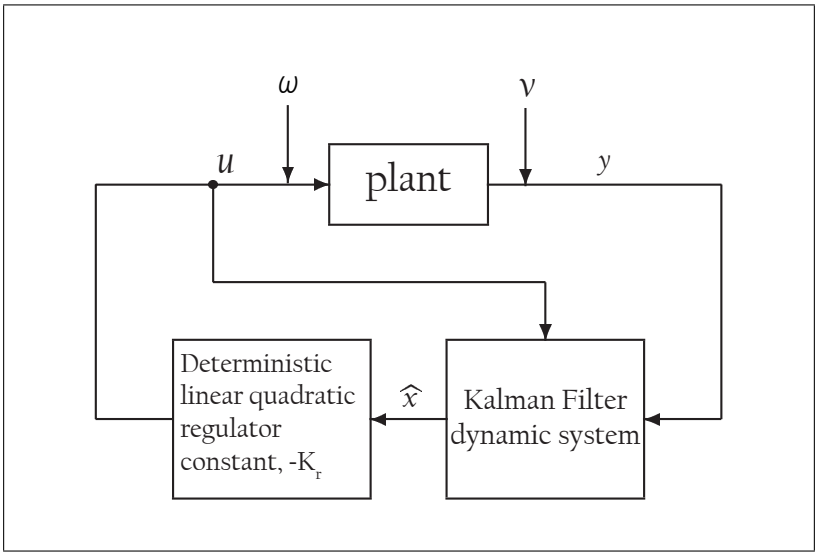


Figure 5.2: The Separation Theorem

We will now give the equations necessary to find the optimal state feedback matrix K_r and the Kalman Filter.

Optimal state feedback

The LQG problem, where all the states are known, is the deterministic initial value problem: Given the system $\dot{x}(t) = Ax(t) + Bu(t)$ with a given non-zero initial state $x(0)$, find the input $u(t)$ witch take the system back ti the zero state ($x = 0$) in an optimal manner, i.e by minimizing the deterministic cost

$$J_r = \int_0^{+\infty} \left(x^T Q x + u^T R u\right) dt \tag{5.8}$$

The optimal solution is for any initial state $u(t) = -K_r x(t)$ where

$$K_r = R^{-1} B^T P \tag{5.9}$$

and $P = P^T \geq 0$ is the unique positive-semi-definite solution of the stationary algebraic Riccati equation

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{5.10}$$

Kalman filter

The Kalman filter has the structure of an ordinary state-estimator or observer, as shown in figure 4.2, with

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K_f(y - C\hat{x}(t)) \tag{5.11}$$

The optimal choice of K_f, witch minimize $E\left\{[x(t) - \hat{x}(t)]^T [x(t) - \hat{x}(t)]\right\}$, is given by

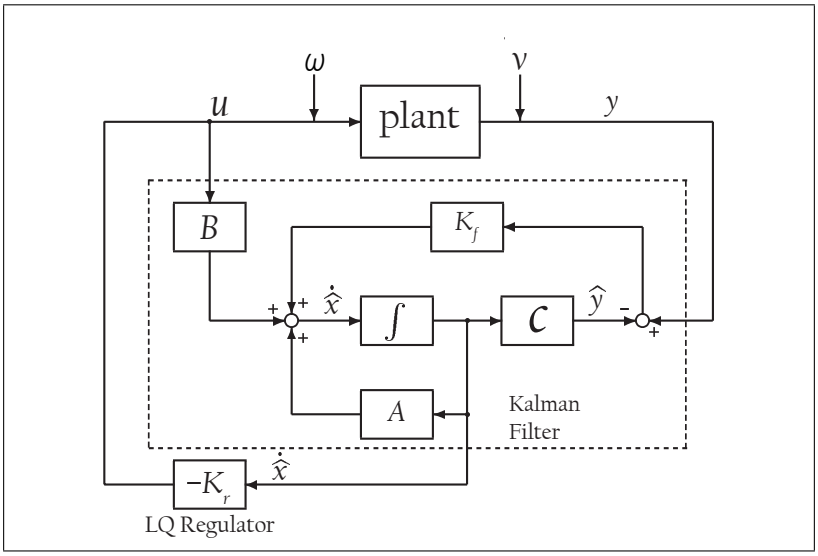


Figure 5.3: The LQG controller and noisy plant

$$K_f = MC^T V_n^{-1} \tag{5.12}$$

where $M = M^T \geq 0$ is the unique positive-semi-definite solution of the stationary algebraic Riccati equation

$$MA^T + AM - MC^T V_n^{-1} CM + B_w W_n B_w^T = 0 \tag{5.13}$$

LQG: Combined optimal state estimation and optimal state feedback

The LQG control problem is to minimize J in equation (5.6). The structure of the LQG controller is illustrated in figure 5.3. The optimal gain matrices K_f and K_r exist, and the LQG-controller system is internally stable if stability and convergence conditions of the Riccati equations are verified witch are:

- (A, B_w) and (A, B): controllable or stabilizable
- (A, C_q) and (A, C): observable or detectable $(Q = C_q^T C_q)$

Now after discussing the LQG problem and solution, we will present our LQG servo controller with integral action. The control structure is as follow

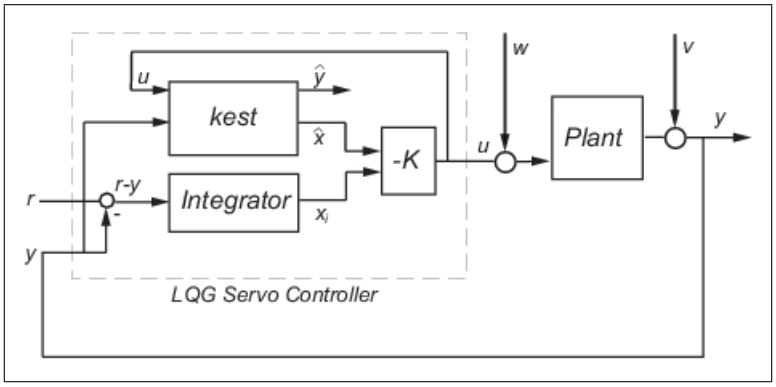


Figure 5.4: The LQG servo controller and noisy plant

where:

- $r = [w_R^d, w_L^d]^T$ denotes the reference right and left wheels' velocities.
- $y = [w_R, w_L]^T$ denotes the measured right and left wheels' velocities.
- $u = [V_R, V_L]^T$ denotes the right and left motors' voltages.
- $x = [i_{a1}, i_{a2}, v, w]^T$ denotes the motors' currents and the linear and angular robot velocities.
- $x_i = [\xi_1, \xi_2]^T$ denotes the 2 integrator output states.

This servo controller ensure that the output y track the reference command r while rejecting input and measurement white noises $w_n(t)$ and $v_n(t)$. In the next two sections we will explain the steps followed to create our LQG servo controller

5.1.2 Constructing the Optimal State-Feedback Gain for Servo Control

Providing the state-space plant model `sys` and the weighting matrices Q and R, to construct the optimal gain, we type the following command:

```
Kr= lqi(sys,Q,R,N)
```

This command computes the optimal gain matrix `Kr`, for which the state feedback law $u = -K_r z = -K_r[x; x_i]$ minimizes the following quadratic cost function for continuous time:

$$J_z = \int_0^{+\infty} \left(z^T Q z + u^T R u \right) dt \tag{5.14}$$

The matrix `Kr` is computed by solving the algebraic Riccati equation:

$$\hat{A}^T P + P\hat{A} - P\hat{B}R^{-1}\hat{B}^T P + Q = 0 \tag{5.15}$$

where

$$\hat{A} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{5.16}$$

are the *augmented system* matrices.

5.1.3 Constructing the Kalman State Estimator

Fist we construct the noisy plant `sys_noise`, with $B_w = B$ (we suppose that the noise is added directly to the input)

```
sys_noise = ss(A,[B B],C,0)
```

then providing the power spectral density matrices W_n and V_n (adjustable) we compute the state space model `kalmf` of the Kalman filter by typing the command

```
[kalmf,L,P,M] = kalman(sys_noise,Wn,Vn)
```

the Kalman filter takes u and y as inputs, and provide $\hat{x}$ and $\hat{y}$ as outputs.

5.1.4 Simulation Results

Computer simulations were performed using MATLAB and SIMULINK, the MATLAB code is used to define the system and perform control law and kalman filter calculation then the SIMULINK model is used for simulation, the SIMULINK model is shown in Figure 5.5, where the MATLAB code is presented in appendix B.

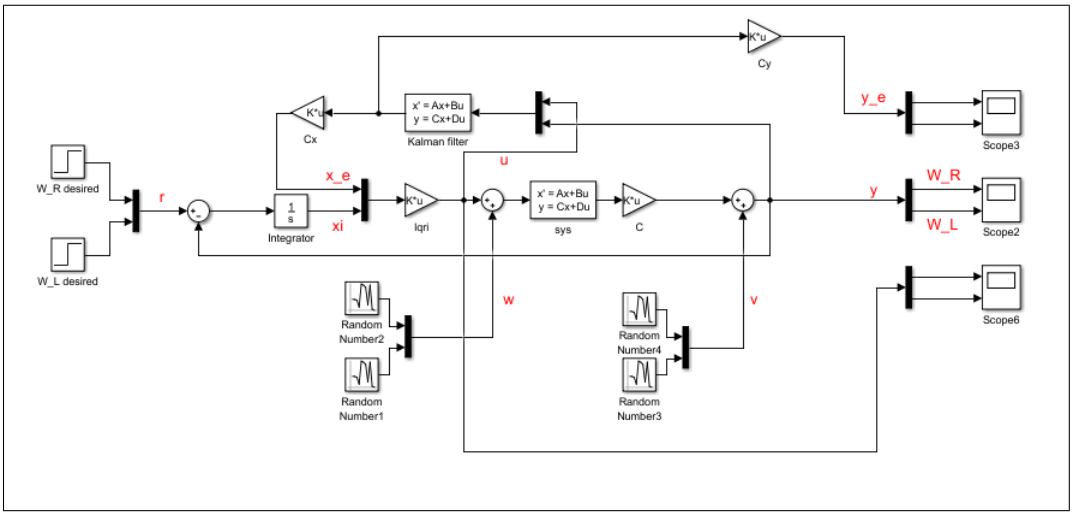


Figure 5.5: The SIMULINK model of the system and the LQG servo Controller

We chose the weighting matrices to be $Q = diag(0.01, 0.01, 0.0237, 0.0065, 0.36, 0.36)$ and $R = 0.1diag(0.005, 0.005)$, and we modeled the noises signals by a *Random Number* blocks with different *seed* numbers and $W_n = diag(0.1, 0.1)$ and $V_n = diag(0.1, 0.1)$. the step block have value of 10.

first we will introduce the open loop response of the SIMULINK model.

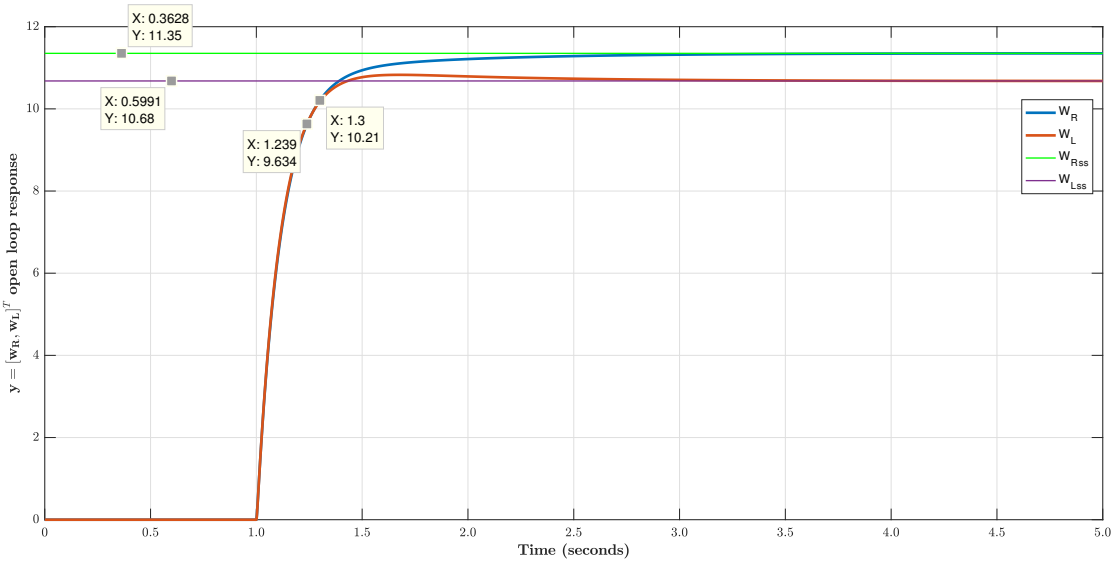


Figure 5.6: The open loop response (without noise)

as shown in figure 5.6 the system is stable with no overshoot and a rise time of around 0.21 seconds.

Next we will present the closed loop response and the controller effort where there is no noise added to the system.

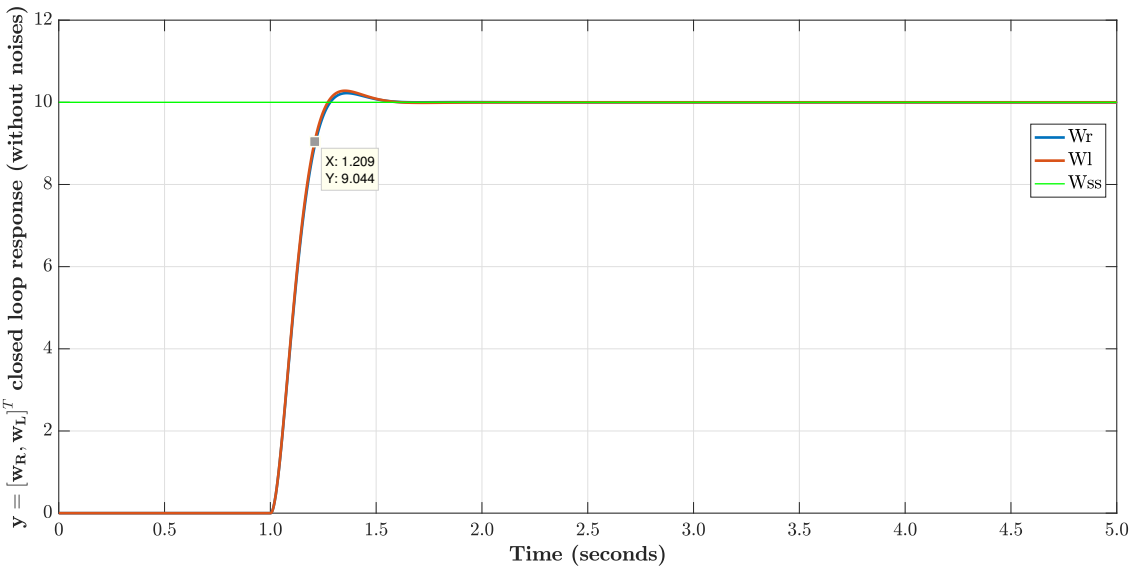


Figure 5.7: The closed loop response (without noise)

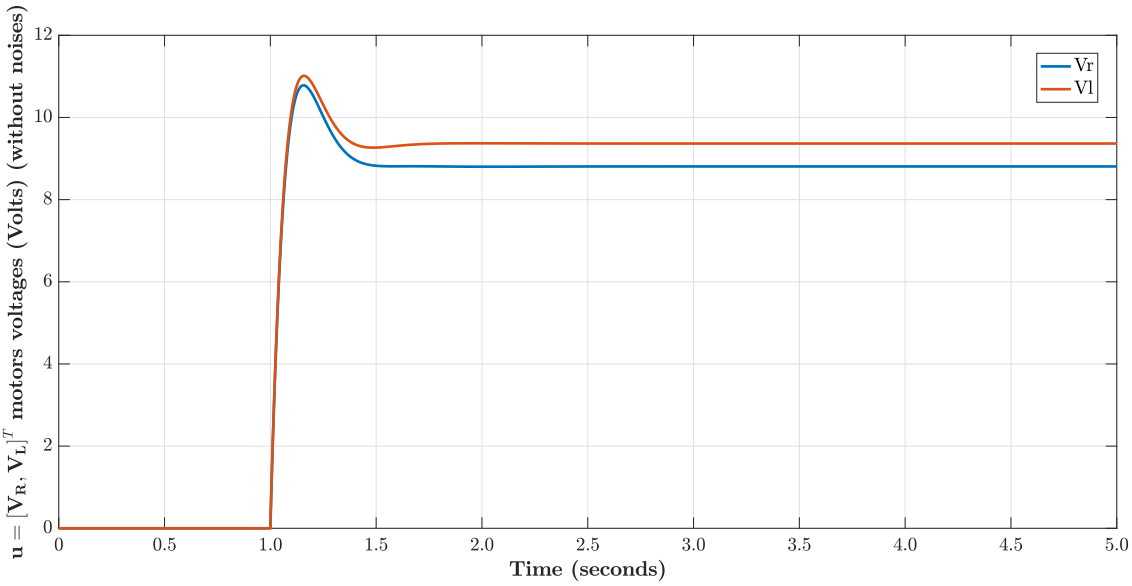


Figure 5.8: The controller effort (without noise)

Finley we present the closed loop response either $y = Cx$ in Figure 5.9 and the filtered $\hat{y} = C\hat{x}$ in Figure 5.10 and the controller effort in figure 5.11, this time considering the input and measurement noises.

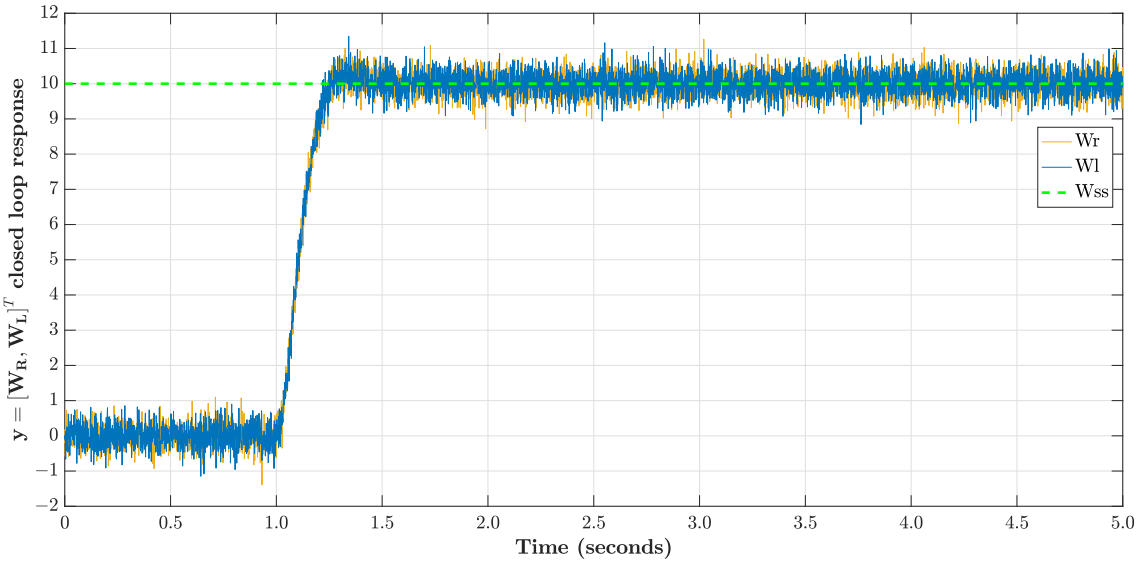


Figure 5.9: The closed loop response (with noise)

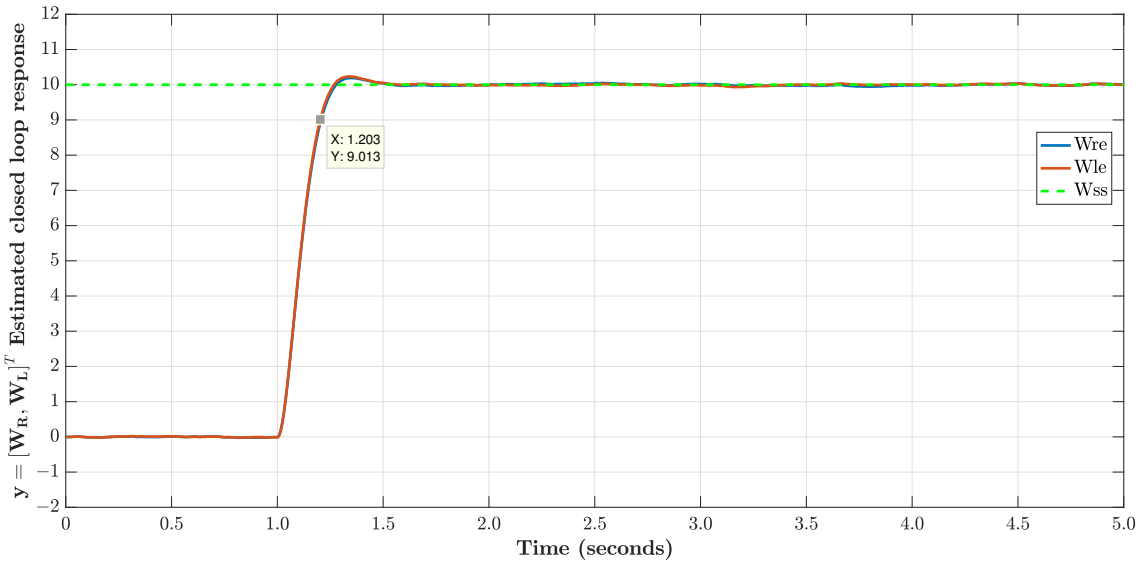


Figure 5.10: The estimated closed loop response (with noise)

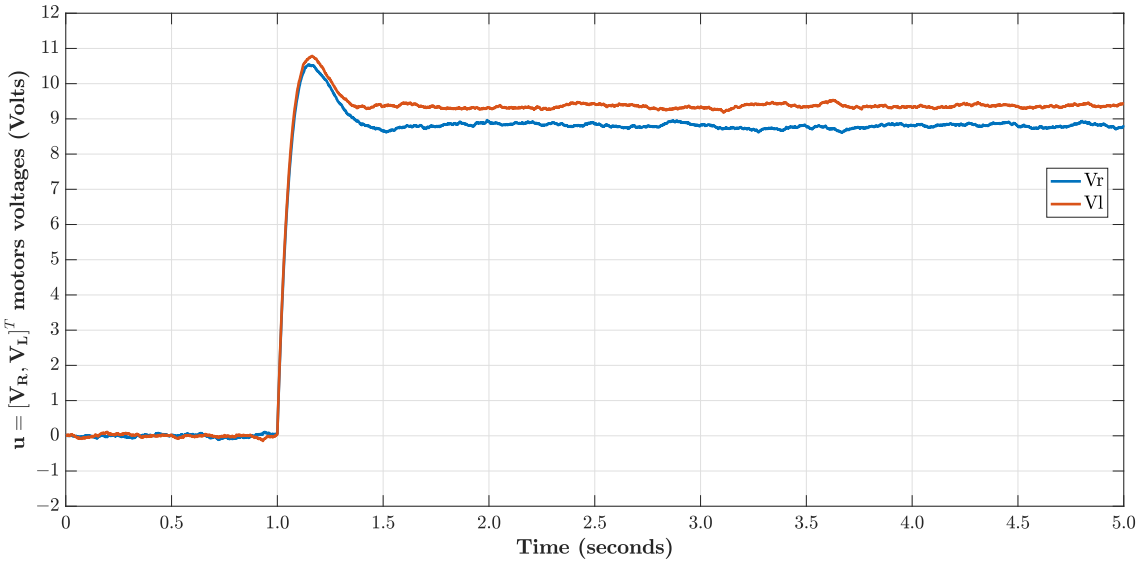


Figure 5.11: The controller effort

We see that the closed loop response track the reference step signal with zero steady state error even in presence of noises, and the rise time still around 0.21 seconds.

Now that we designed the inner loop controller, we can move forward to the next section where we add the kinematic model and the outer loop controller.

5.2 The kinematic controller

In this section, the kinematic controller of the differential drive mobile robot presented in [10] is reviewed and then applied to our system, than we present the simulation results.

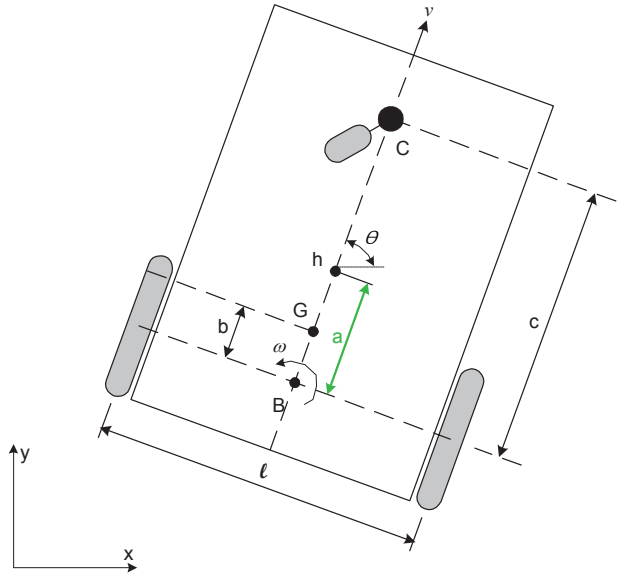


Figure 5.12: The point of interest location

5.2.1 Design of the kinematic controller

The design of the kinematic controller is based on the *modified* kinematic model of the robot

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -a\sin\theta \\ \sin\theta & a\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{5.17}$$

whose output are the coordinates of the point of interest, thus meaning $\mathbf{h} = [x\ y]^T$ as in the figure 5.12 ($a \neq 0$),

hence

$$\dot{\mathbf{h}} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\theta & -a\sin\theta \\ \sin\theta & a\cos\theta \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \mathbf{\Lambda} \begin{bmatrix} v \\ w \end{bmatrix} \tag{5.18}$$

Therefore, the inverse kinematics is given by

$$\dot{\mathbf{h}} = \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{1}{a}\sin\theta & \frac{1}{a}\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \mathbf{\Lambda^{-1}} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{5.19}$$

and the kinematic control law proposed to be applied to the robot is given by

$$\dot{\mathbf{h}} = \begin{bmatrix} v_{ref}^c \\ w_{ref}^c \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{1}{a}\sin\theta & \frac{1}{a}\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x}_d + l_x \tanh\left(\frac{k_x}{l_x}\tilde{x}\right) \\ \dot{x}_d + l_y \tanh\left(\frac{k_y}{l_y}\tilde{y}\right) \end{bmatrix} \tag{5.20}$$

Here, $\tilde{x} = x_d - x$, and $\tilde{y} = y_d - y$ are the current position errors in the axes X and Y, respectively, $k_x > 0$ and $k_y > 0$ are the gains of the controller, $l_x \in \mathbb{R}$, and $l_y \in \mathbb{R}$ are saturation constants, and (x, y) and (x_d, y_d) are the current and the desired coordinates of the point of interest $\mathbf{h}$, respectively. The objective of such a controller is to generate the references of linear and angular velocities for the dynamic controller, as shown in Figure 5.1.

5.2.2 Stability analysis

In this analysis it is supposed a perfect velocity tracking, allowing to equate (5.19) and (5.20) under the assumption of $v \equiv v_{ref}^c$ and $w \equiv w_{ref}^c$, which means that the dynamic effects of the robot are ignored. Then, the closed-loop equation, in terms of the velocity errors, is

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{bmatrix} + \begin{bmatrix} l_x & 0 \\ 0 & l_y \end{bmatrix} \begin{bmatrix} \tanh\left(\frac{k_x}{l_x}\tilde{x}\right) \\ \tanh\left(\frac{k_y}{l_y}\tilde{y}\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.21}$$

Now, defining the output error vector $\tilde{\mathbf{h}} = [\tilde{x}\ \tilde{y}]^T$, Equation (5.21) can be written as

$$\dot{\tilde{\mathbf{h}}} = -\begin{bmatrix} l_x \tanh\left(\frac{k_x}{l_x}\tilde{x}\right) & l_y \tanh\left(\frac{k_y}{l_y}\tilde{y}\right) \end{bmatrix}^T$$

which has a unique equilibrium point at the origin.

To conclude on the stability of such equilibrium point, the Lyapunov candidate function

$$V = \frac{1}{2}\tilde{\mathbf{h}}^T\tilde{\mathbf{h}} > 0$$

is considered, whose first time derivative

$$\dot{V} = \tilde{\mathbf{h}}^T\dot{\tilde{\mathbf{h}}} = -\tilde{x}l_x \tanh\left(\frac{k_x}{l_x}\tilde{x}\right) - \tilde{y}l_y \tanh\left(\frac{k_y}{l_y}\tilde{y}\right)$$

is negative definite.

Hence, one can straightforwardly conclude that the system characterized so far, when controlled through the kinematic controller here proposed, has an asymptotically stable equilibrium at the origin, which means that $\tilde{x}(t) \to 0$ and $\tilde{x}(t) \to 0$ as $t \to \infty$.

Note : Considering the case in which the reference remains constant for a while, the robot tends to reach such a reference point and to stop there. Nevertheless, it should be also guaranteed that the robot orientation given by the variable θ keeps bounded. It can be seen from (5.19) *that* $w^c_{ref} = 0$ *when* $\tilde{x} = 0$, $\dot{x_d} = 0$, $\tilde{y} = 0$ *and* $\dot{y_d} = 0$. *Under the assumption that* $w \equiv w^c_{ref}$, *it can be concluded that* $\dot{\theta} = w = 0$, *and* $\theta(t) \to \theta_{constant}$.

5.2.3 Simulation Results

As in the dynamic controller section we present the SIMULINK model, this time performing very much like the diagram in Figure 5.1. The SIMULINK model shown in figure 5.13 contains a trajectory planning block witch offer Fixed point reference and three type of reference trajectories ; circle, eight and line trajectories, in Figure 5.14 we show the block window. Also it contains the kinematic controller as in equation (5.20) and the kinematic model as in equation (5.17) with integrator at the outputs, and the dynamic model with the velocity controller presented in figure 5.5.

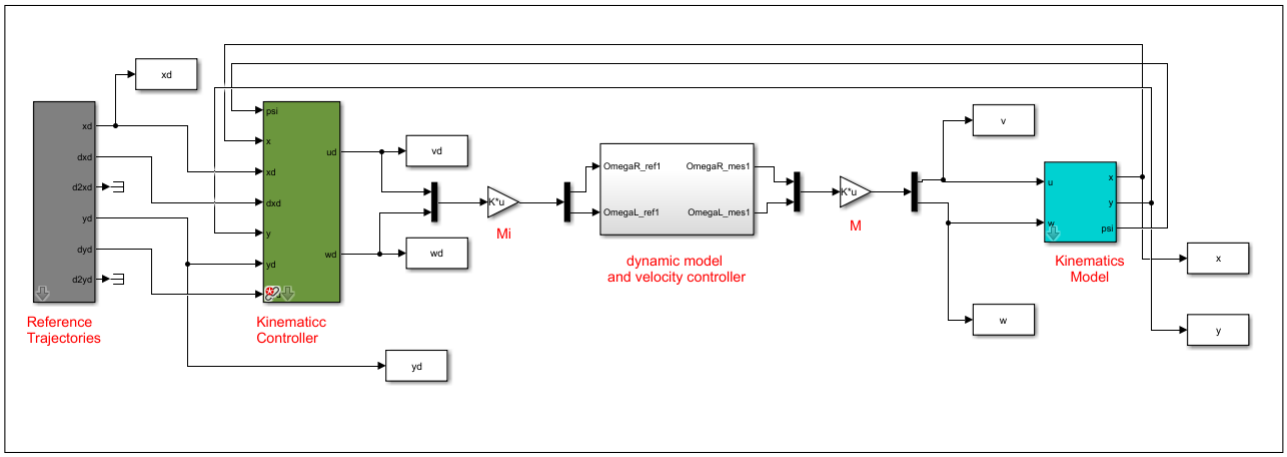


Figure 5.13: The controller effort

The reference trajectorie, the kinematic controller and the kinematic model blocks are used from MathWorks File Exchange [9]

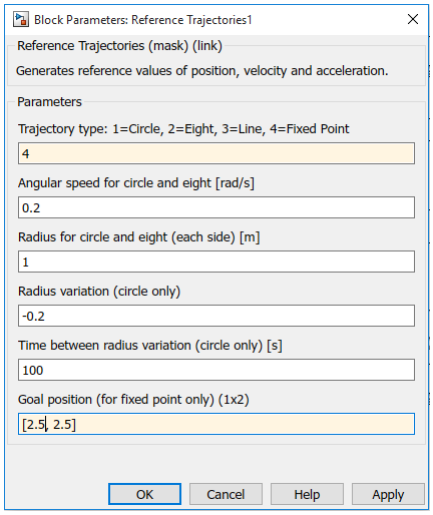


Figure 5.14: The reference trajectories block window

Fixed point reference

Configuring the reference trajectory block to give a fixed point reference of coordinates $\mathbf{h_{ref}} = (2.5, 2.5)$ with no defined trajectory, the goal is that the interest point $\mathbf{h}$ reach the fixed point without any constraint on the robot heading, starting from $\mathbf{h_0} = (0, 0)$ and using the following kinematic controller parameters (used in all simulations): $k_x = 1,\ k_y = 1,\ l_x = 0.25,\ l_y = 0.25$ with auto mode and variable-step solver options. we present bellow the simulation results:

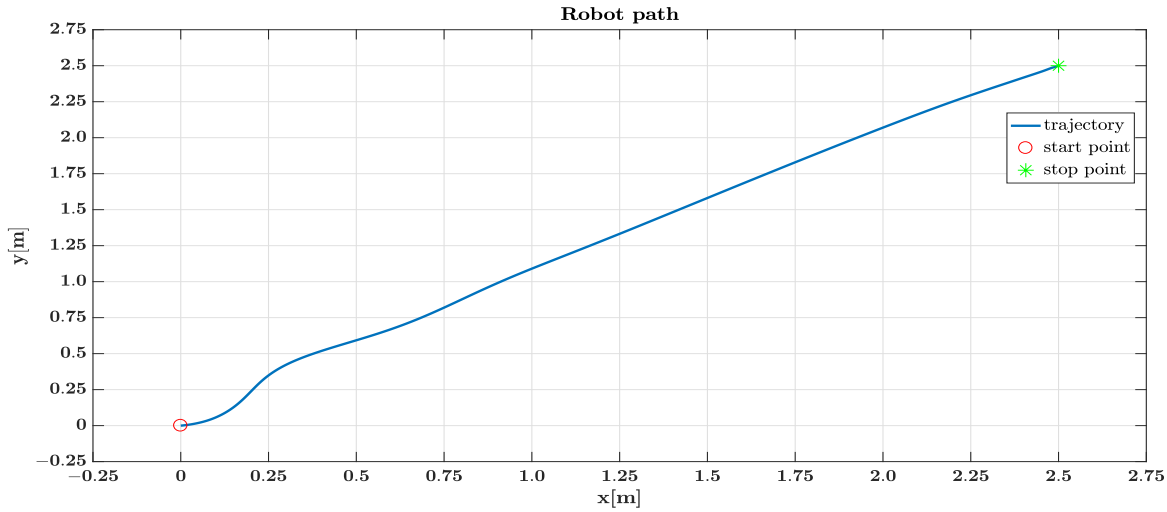


Figure 5.15: Fixed point reference : Robot path

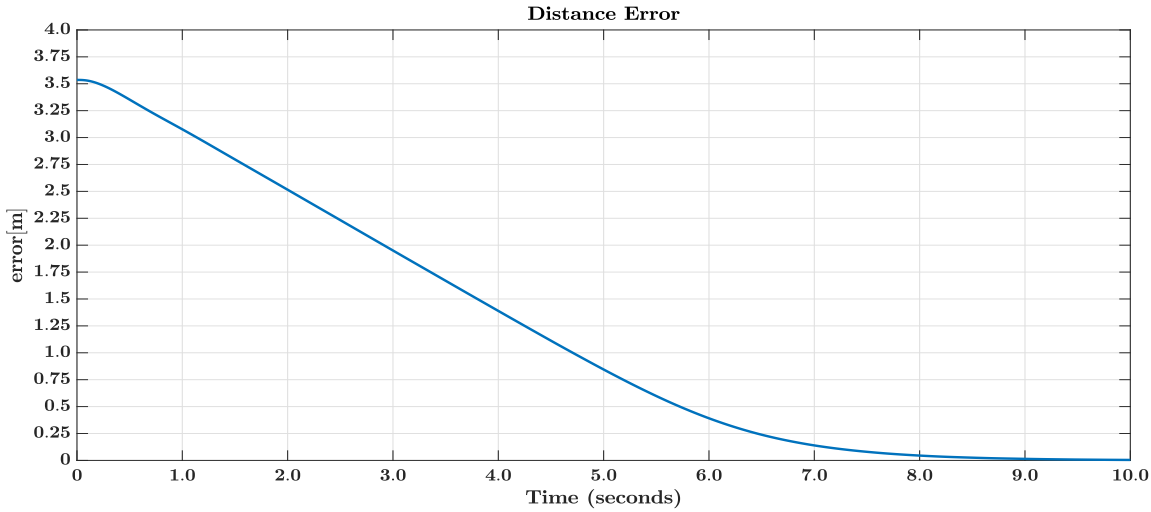


Figure 5.16: Fixed point reference : Evolution of distance error

Figure 5.15 and 5.16 show the path followed by the robot and the evolution of the distance error (the instantaneous distance between the reference position $\mathbf{h_{ref}}$ and the robot actual position $\mathbf{h}$), we see that the distance error decrease to zero as the simulation progresses and the robot takes a line trajectory to the fixed point.

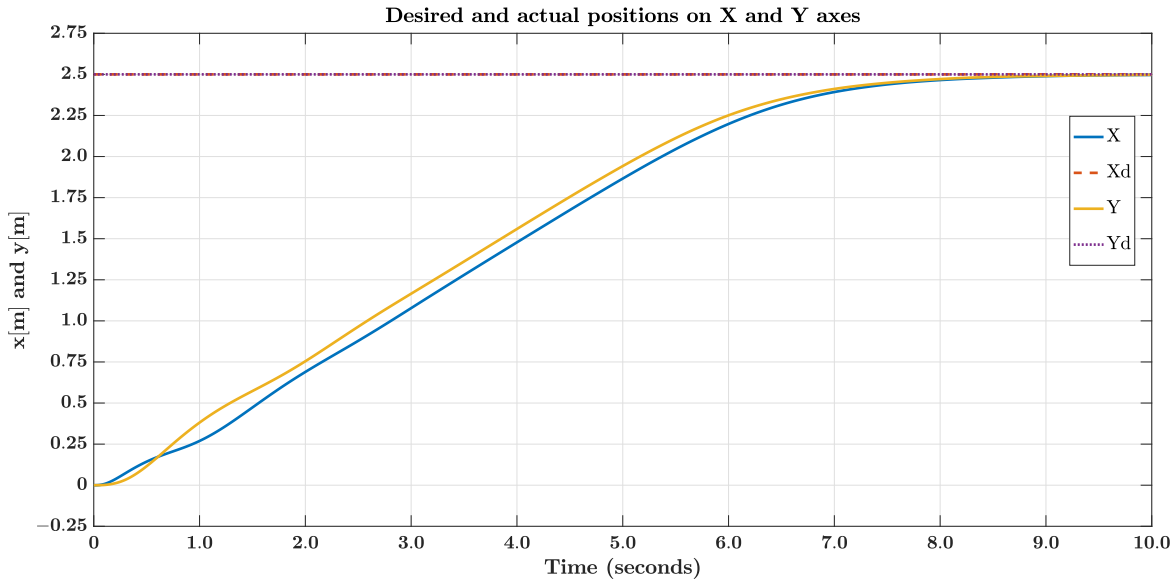


Figure 5.17: Fixed point reference : desired and actual positions

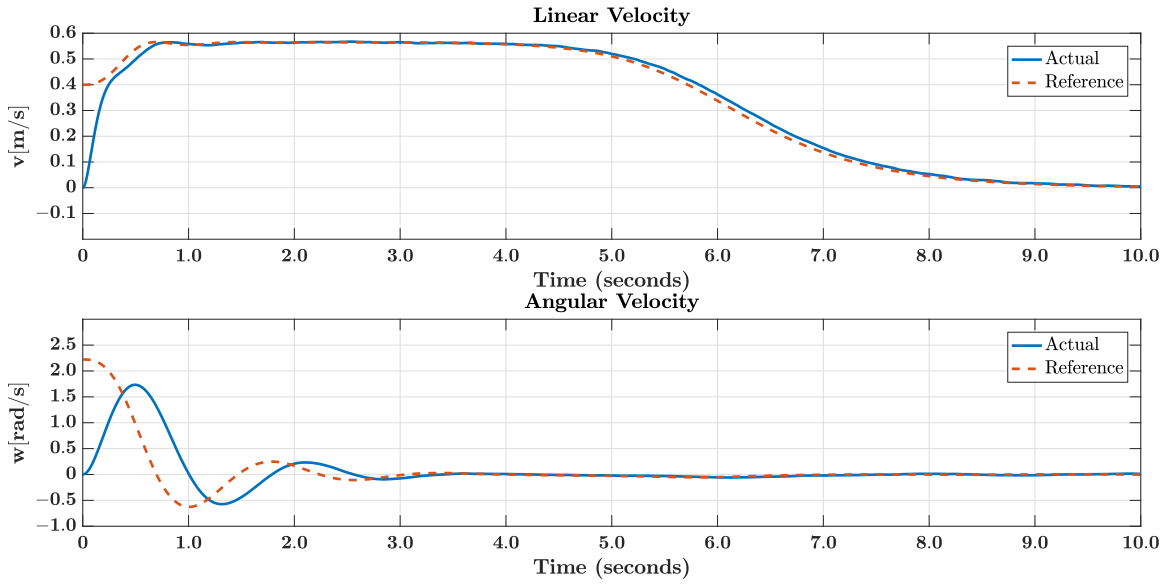


Figure 5.18: Fixed point reference : linear and angular velocities

From Figure 5.17, we see that the robot takes about 9 *seconds* to reach the stop point. Considering a line trajectory between the start and the stop points with initial error distance of about 3.5 m (Figure 5.16) give us an average velocity of about 3.9 m/s. Figure 5.18 shows us the reference linear and angular velocities generated

by the kinematic controller and the dynamic model response that ensure minimal tracking error.

Circle trajectory

the circle trajectory can be run indefinitely while occupying a finite space, for this simulation we choose a circle trajectory with 1 m radius.

From Figure 5.19 we see clearly that the position controller had succeed to track the reference circle trajectory giving around zero error distance, this is also shown in Figure 5.19. In Figure 5.20 we see the position controller effort (v and w), this time with a time varying x and y references and in blue the actual linear and angular velocities

Figure-eight trajectory

A figure-eight trajectory is convenient because it contains an equal number of left and right hand turns, so we used it for this third simulation as a reference trajectory with 1 m radius.

As with circle reference trajectory, Figure 5.22 shows how the robot tracks the figure-eight trajectory, but this time with a small distance error at and after each turn, this is due to a small phase in the angular velocity response shown in figure 5.24.

In the next chapter we will present the experimental results of the 3 previous simulations and for the velocity control, and we will discuss the results.

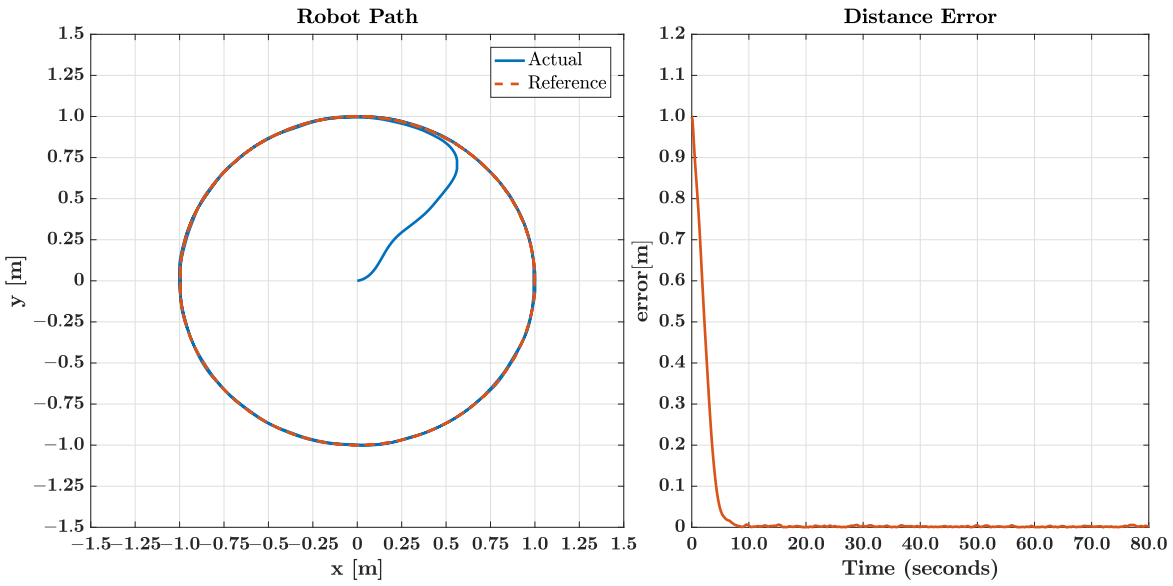


Figure 5.19: circle trajectory reference : Robot path and distance error

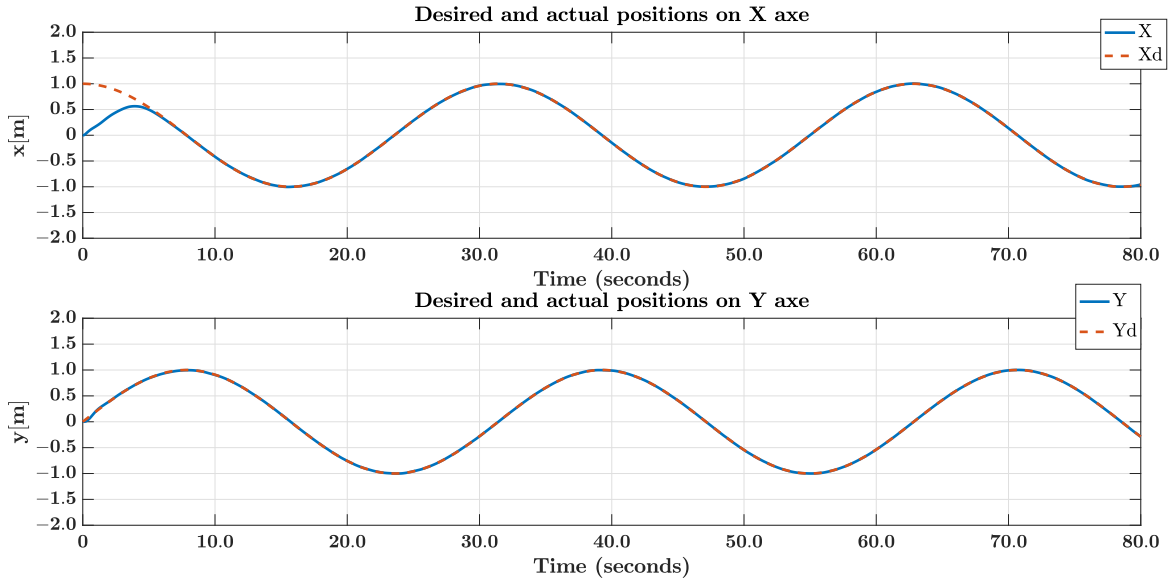


Figure 5.20: circle trajectory reference : desired and actual positions

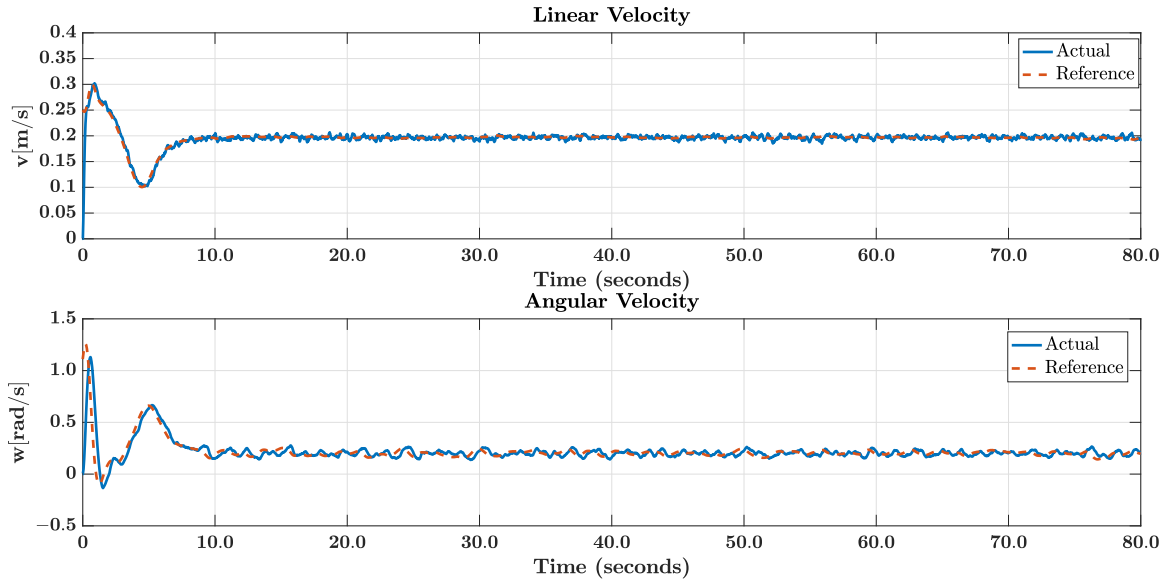


Figure 5.21: circle trajectory reference : linear and angular velocities

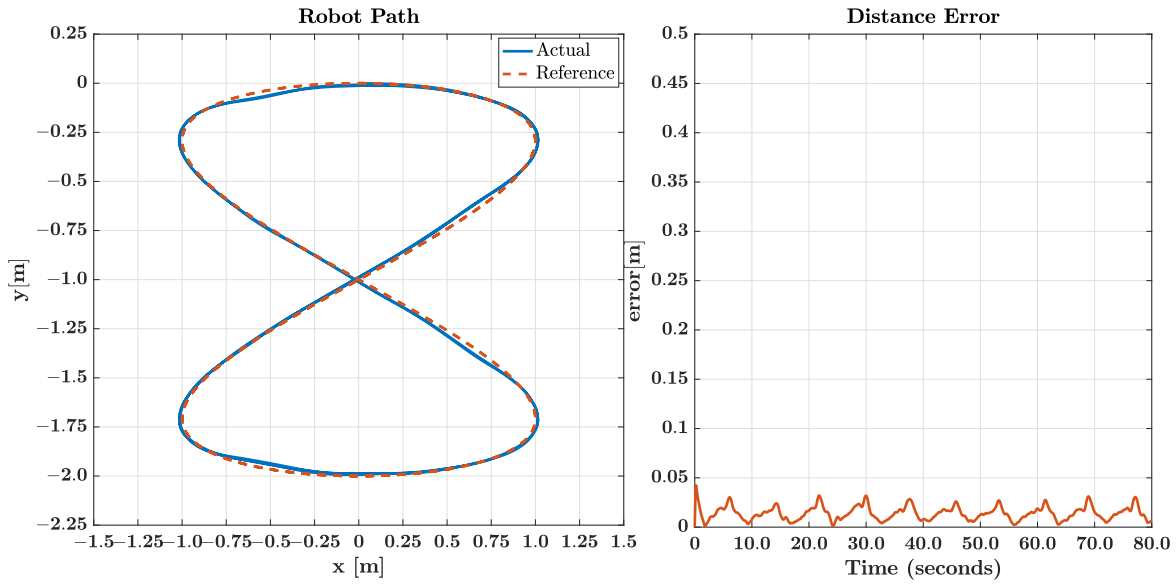


Figure 5.22: eight trajectory reference : Robot path and distance error

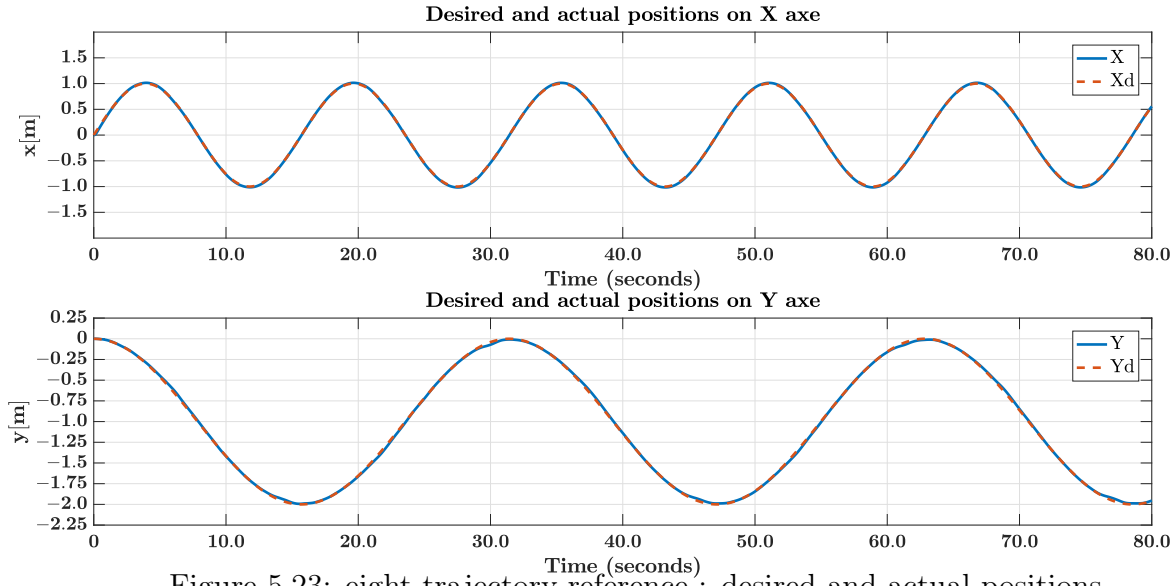


Figure 5.23: eight trajectory reference : desired and actual positions

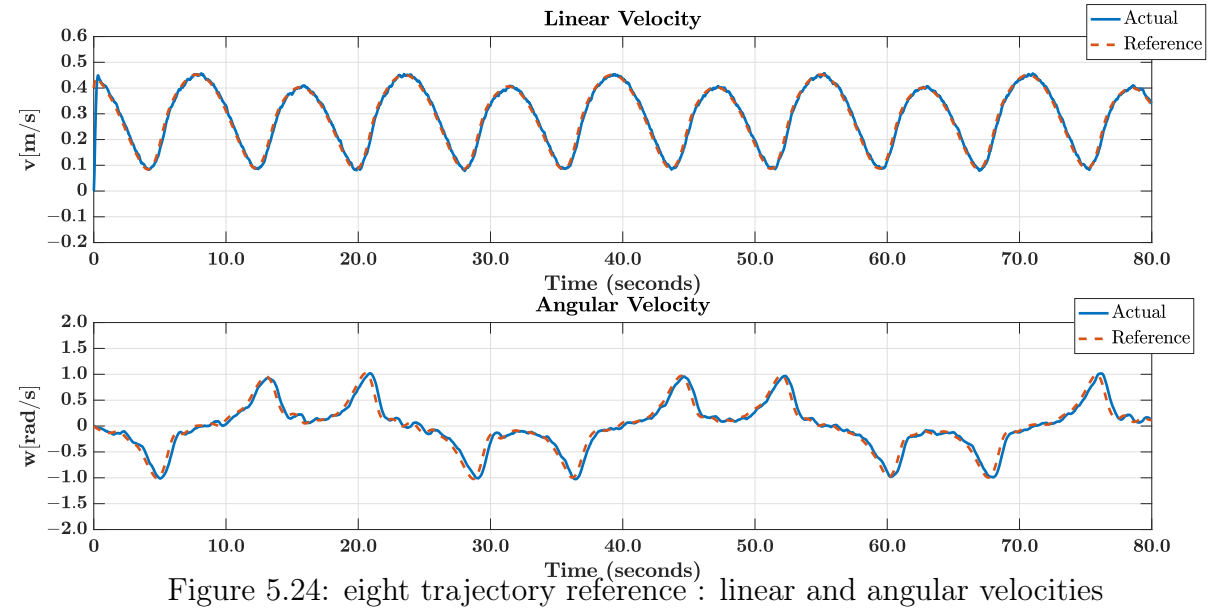


Figure 5.24: eight trajectory reference : linear and angular velocities

Chapter 6

Experimental Results and Discussion

To illustrate application relevance of the proposed control scheme, we are going to present the experimental results for the main three mobile robot tasks:

Velocity control of the mobile robot is a very fundamental problem. This is because underneath any position control there is the assumption of perfect velocity tracking.

Point to Point stabilization in nature is a simpler problem, where the robot only has to start from an initial point and reach a destination point. In this class of tasks the behavior of the robot between the initial and final point, and also the final orientation of the robot is not explicitly controlled.

trajectory tracking is the highest level problem which consists of a robot following a predefined path with a time law; implicitly putting a velocity constraint on the robot, in our case defined by the pulsation of sinusoids defining the circle and figure-eight trajectories (Appendix A).

Than we will analyze and discuss the results.

6.1 Experimental results

In the four experiments; velocity control, fixed point reference, circle and figure-eight reference trajectories, we used our differential drive robot platform (SIRIUS) shown in Figure 6.1. The final SIMULINK model compiled and deployed to the the robot processing unit (ARDUINO Due) is the same as in 5.5 for velocity control and in 5.13 for the rest with replacing the state-space dynamic model by the real system block shown in 6.2

Figure 6.1: The differential drive robot platform

In the last three experiments, we started from the initial point $\mathbf{h_0} = (0, 0)$ and we used the following kinematic controller parameters: $k_x = 1$, $k_y = 1$, $l_x = 0.25$, $l_y = 0.25$. The circle and figure-eight trajectories are used with

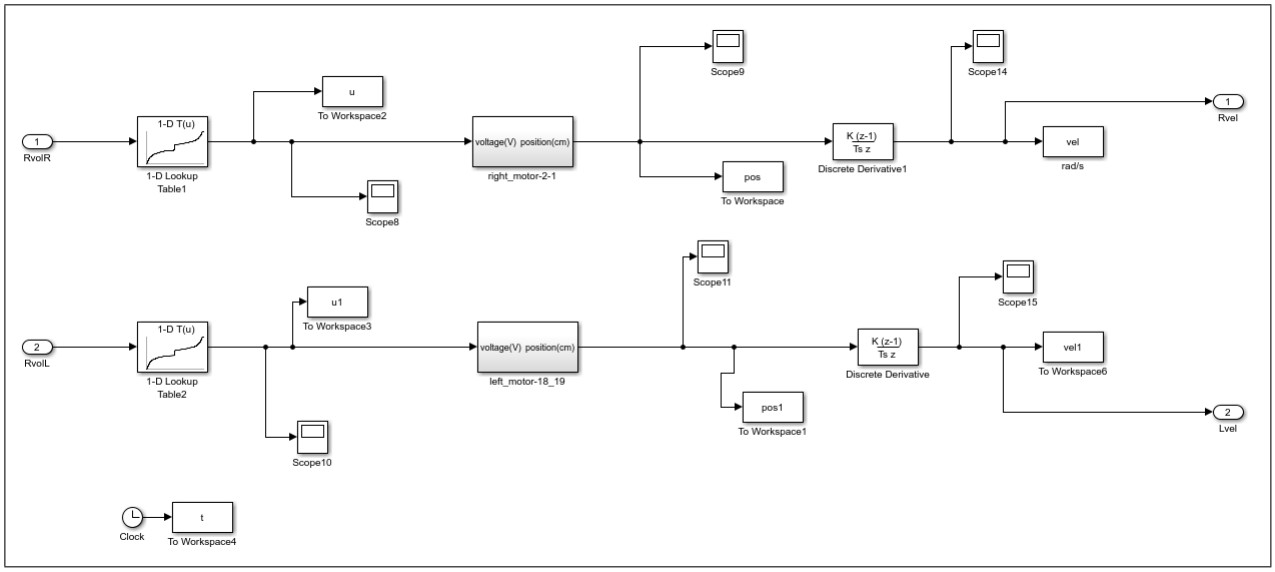


Figure 6.2: The real system SIMULINK block

base sinusoids pulsation of 0.2 Rad/s to avoid robot limitations. In all experiments we used the following LQG controller parameters: the weighting matrices $Q = diag(0.01, 0.01, 0.0237, 0.0065, 0.36, 0.36)$ and $R = 0.1diag(0.005, 0.005)$, and the noises covariances $W_n = diag(0.1, 0.1)$ and $V_n = diag(0.1, 0.1)$. Fixed-step auto solver selection is used, with fundamental sample time of 0.04 s to ensure the real time of the experiment. The 2 models are implemented in *external mode* so that we can record the experimental data.

Note : All velocity data are filtred with the command :

```
smooth(vel_data,0.03,'loess');
```

for *velocity control*, the right and left motors' desired velocities are step functions, with step time of 4 *seconds* and final value of 10 Rad/s. The velocity response is shown in figure 6.3.

The Experimental results of *fixed point reference* with $\mathbf{h_{ref}} = (2.5, 2.5)$ m are shown in Figures 6.4 to 6.7 , for *circle reference* trajectory with 1 m radius are in

Figures 6.8 to 6.10 and for *figure-eight reference* trajectory with 1 m radius are in Figures 6.11 to 6.12

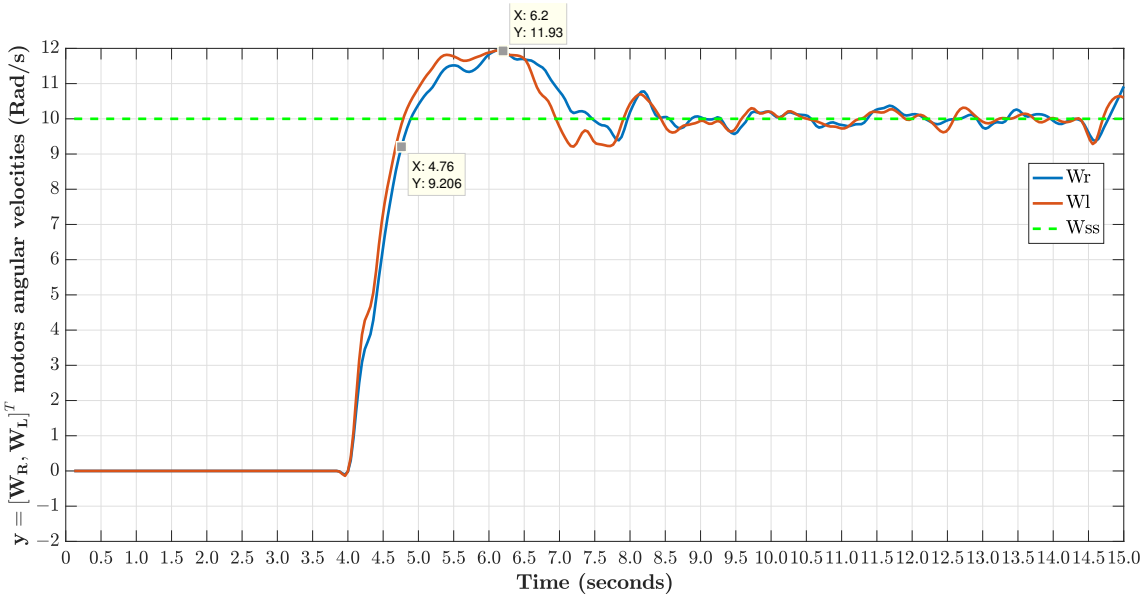


Figure 6.3: The measured closed loop response

From Figure 6.3 we see that the real system response present a considerable overshot of around 20%, and a rise time of 0.76 *seconds*, this result is relatively good but not as good as simulation results, even with the same LQG parameters (Q, R, W_n, V_n), we didn't have much time to further improve the response by adjusting the Q and R matrices due to the mobility and time consuming nature of the experiments. The Q and R matrices are empirically selected according to the good rule of thumb presented in Appendix C, and The W_n and V_n matrices from comparing the measured and the simulated velocity noises (we adjust the noises covariances in simulation so that they give the same effect on the response as real

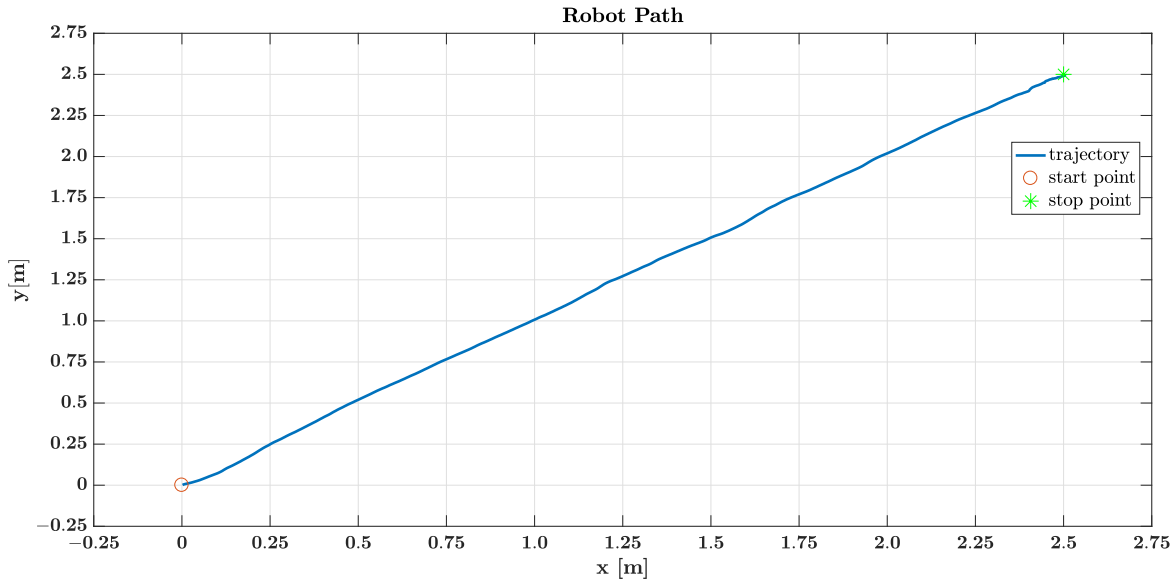


Figure 6.4: Fixed point reference : Robot path

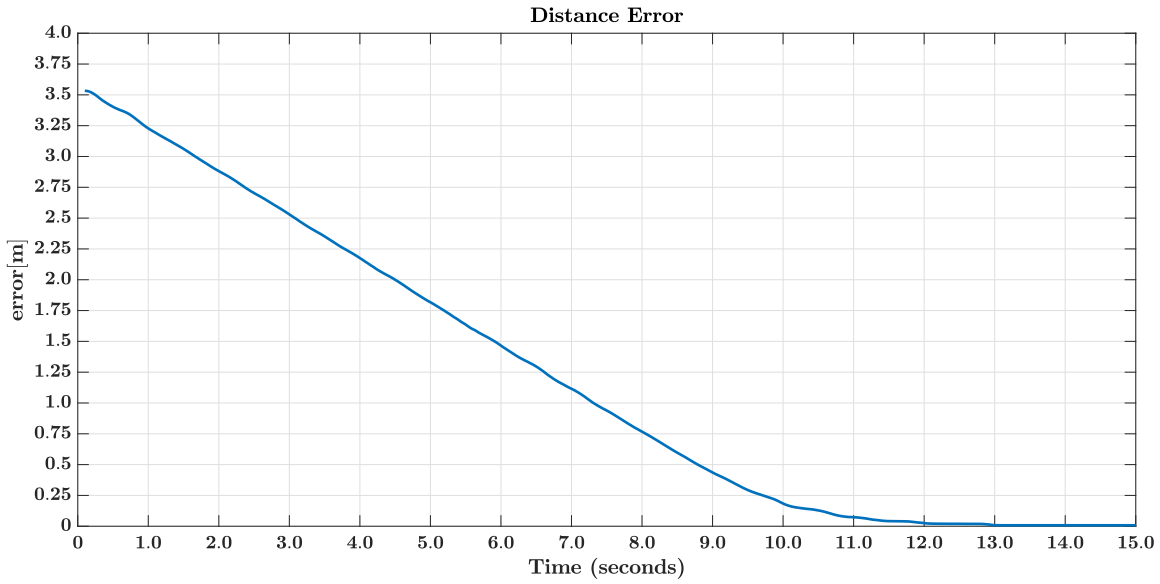


Figure 6.5: Fixed point reference : Evolution of distance error

noises did).

Figure 6.4 shows that the path of the robot is a diagonal line and this is because of the position of the stop point ($x_{ref} = y_{ref}$), else ($x_{ref} \neq y_{ref}$) the robot takes a curved path, Figure 6.5 shows that the distance error vanishes in around

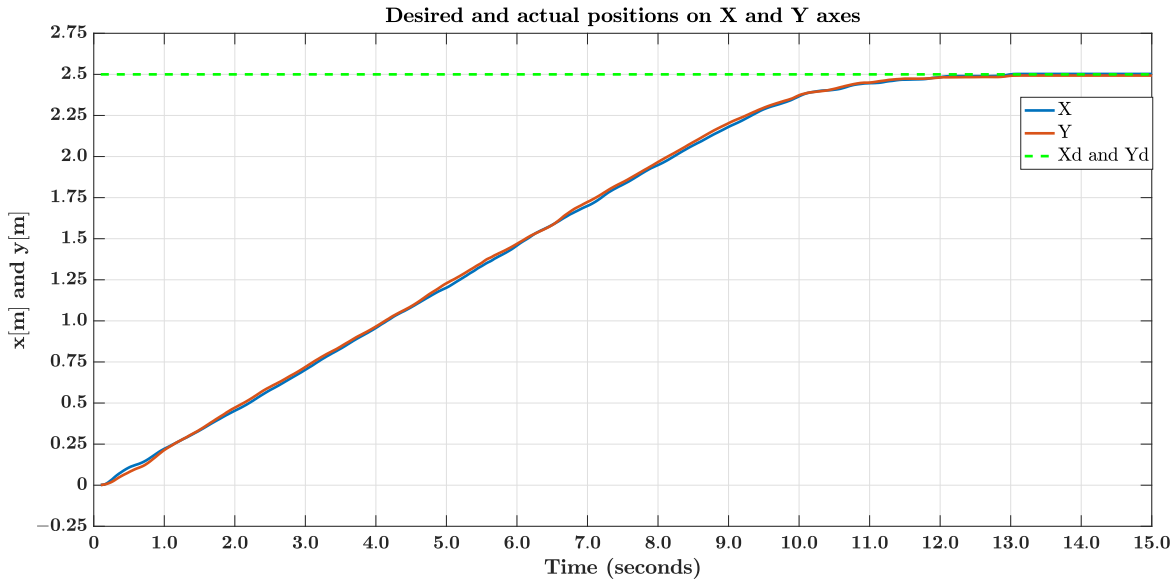


Figure 6.6: Fixed point reference : desired and actual positions

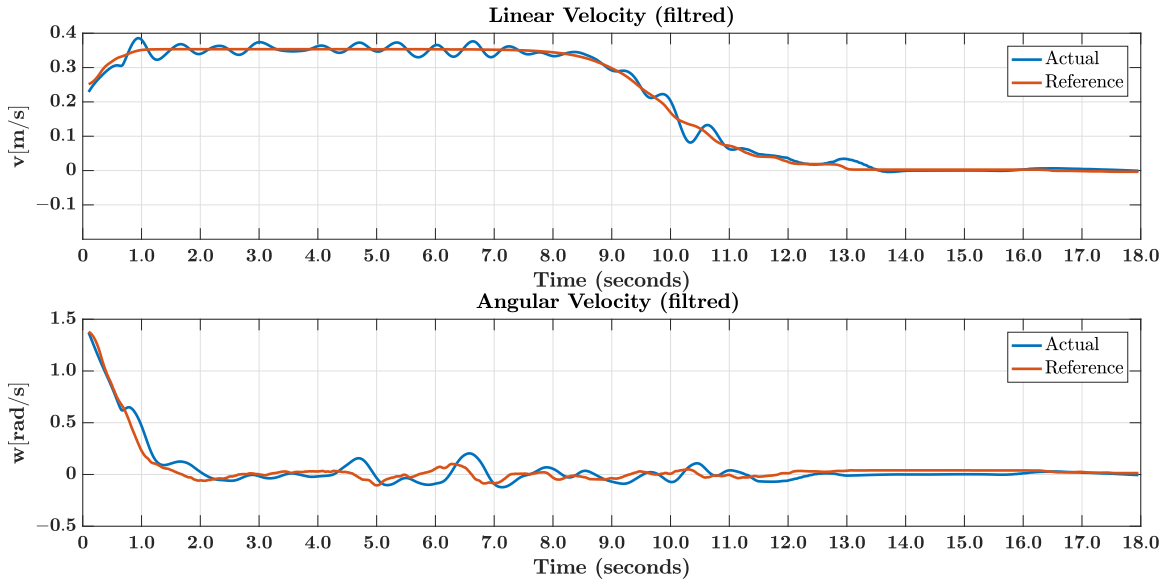


Figure 6.7: Fixed point reference : linear and angular velocities

12 *seconds* instead of 9 *seconds* in simulation because of the non exigence of trajectory reference thus velocity reference thus the time the robot takes to reach the stop point.

Due the proprioceptive nature of the localization sensor of our robot; the encoders, witch present reading drift, we tried to minimize it by calibrating the encoders, but

a certain error still unpredictable due to the build quality of the encoders, because of that real final position of the robot is around 10 *cm* distant from the physical stop point.

In Figure 6.7 the linear and angular velocities of the robot contain some fluctuations around the reference velocities, the same in figure 6.3 around the set point velocity, these fluctuations are due to external disturbances and non-modeled dynamics.

In the circle trajectory reference response, Figure 6.8 shows that the robot responds similarly to the simulation and this is illustrated in the distance error graphs similarity, the steady error is in the neighborhood of zero, in figure 6.9 we see the perfect position tracking in X and Y axes along time, witch reflects the success of the kinematic controller.

It should be mentioned that the real path of the robot present drifting error that can be reduced by improving the encoders reliability.

Finally for the figure-eight trajectory, as in the previous experiment , Figures 6.11 and 6.12 show the good trajectory tracking performance of the kinematic controller, with the same small error at and after turns presented also in simulation due to the angular velocity phase.

Figures 6.10 and 6.13 show that also the dynamic controller gives a good tracking performance *with* the help of the kinematic controller, this is seen if we compare the simulation *reference* linear and angular velocities and the experimental *measured* ones. In the two experiments (circle and figure-eight) we see that the kinematic controller

output signals (v_{ref}, w_{ref}) have higher frequency components than the measured linear and angular velocities (v, w), this is not the case in figure 6.7 because in the fixed point reference experiment, there is no velocity reference on the robot so that the kinematic controller can help avoiding the linear and angular velocities fluctuations, and this is why the fluctuations are largely reduced in the circle and figure-eight trajectories experiments, In the last experiment, despite the good performance obtained from the recorded data, in reality the path's right and left turns increased the drifting error, witch unfortunately we can't further reduce regarding its source.

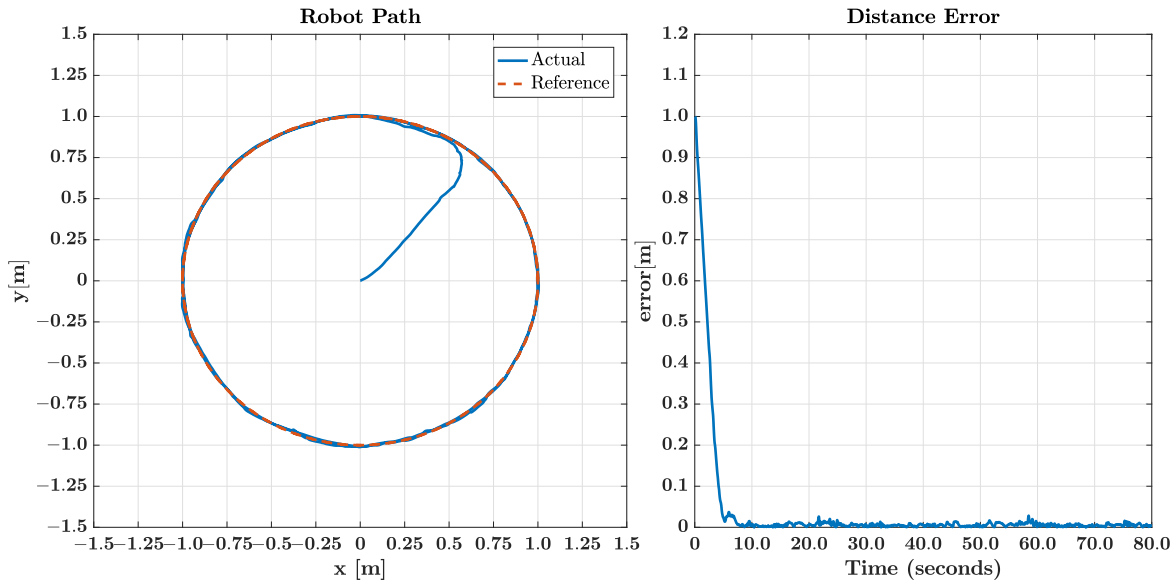


Figure 6.8: circle trajectory reference : Robot path and distance error

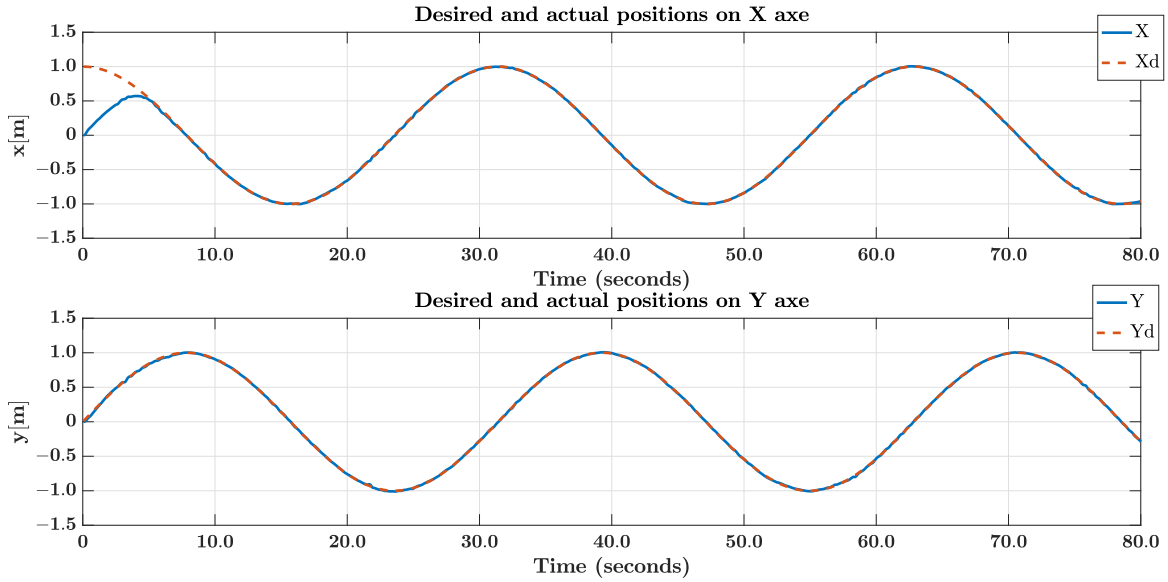


Figure 6.9: circle trajectory reference : desired and actual positions

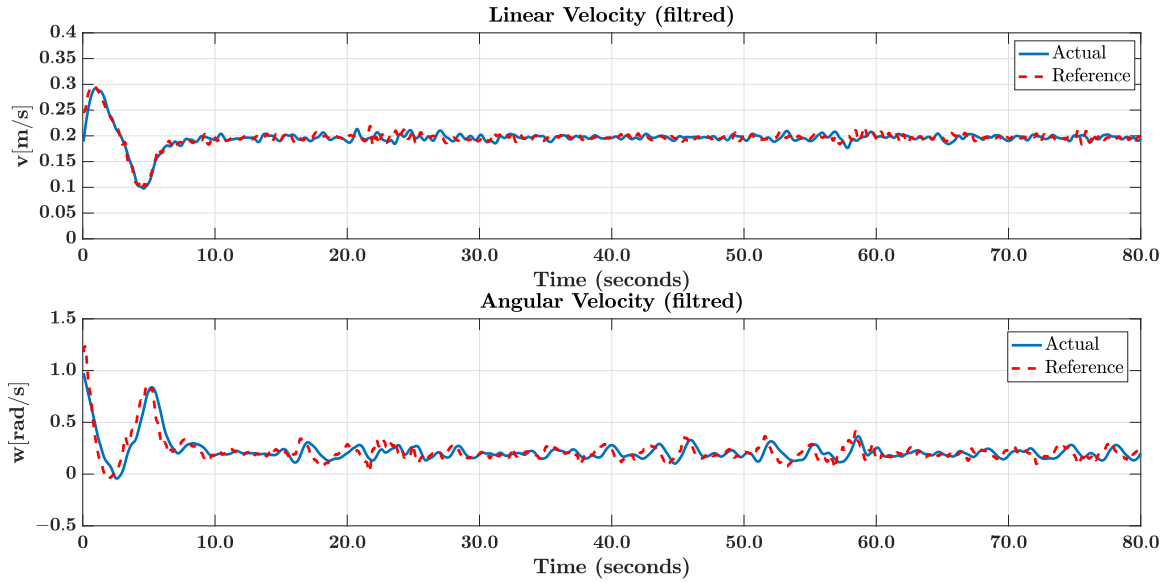


Figure 6.10: circle trajectory reference : linear and angular velocities

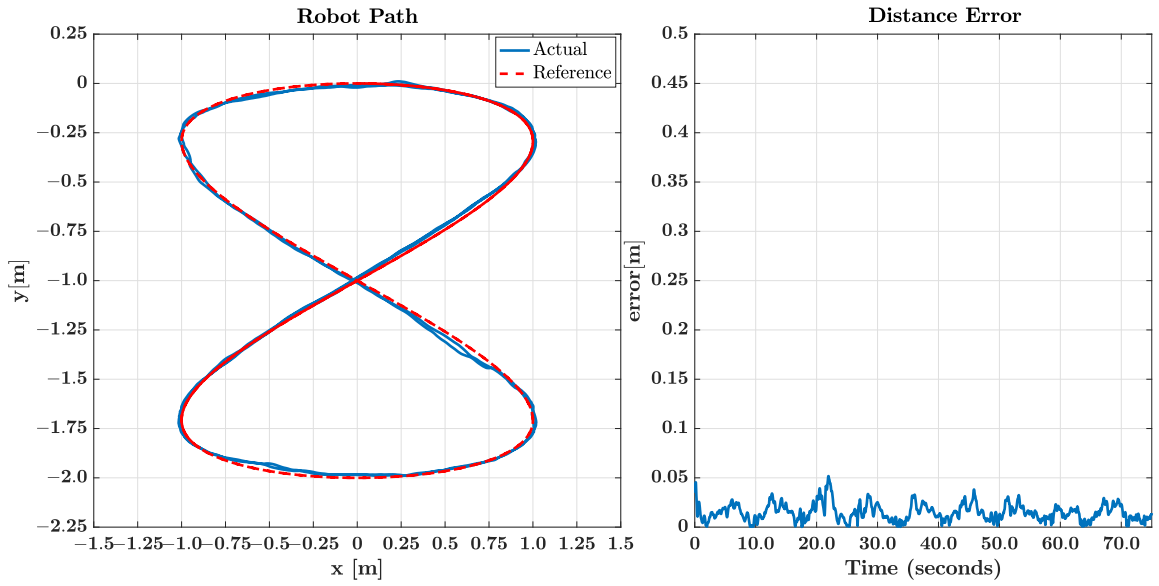


Figure 6.11: eight trajectory reference : Robot path and distance error

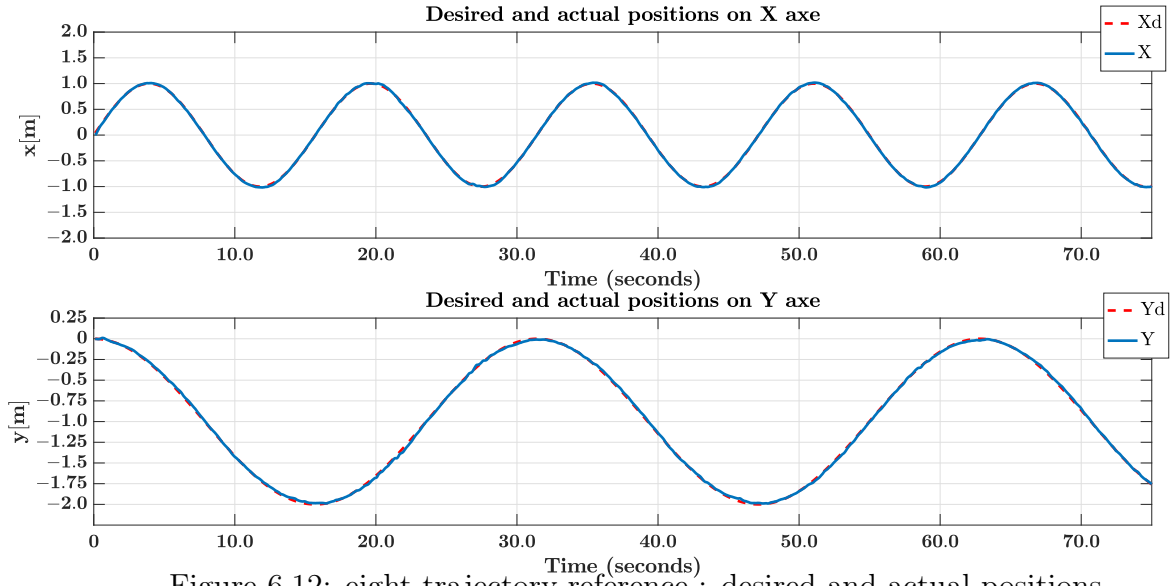


Figure 6.12: eight trajectory reference : desired and actual positions

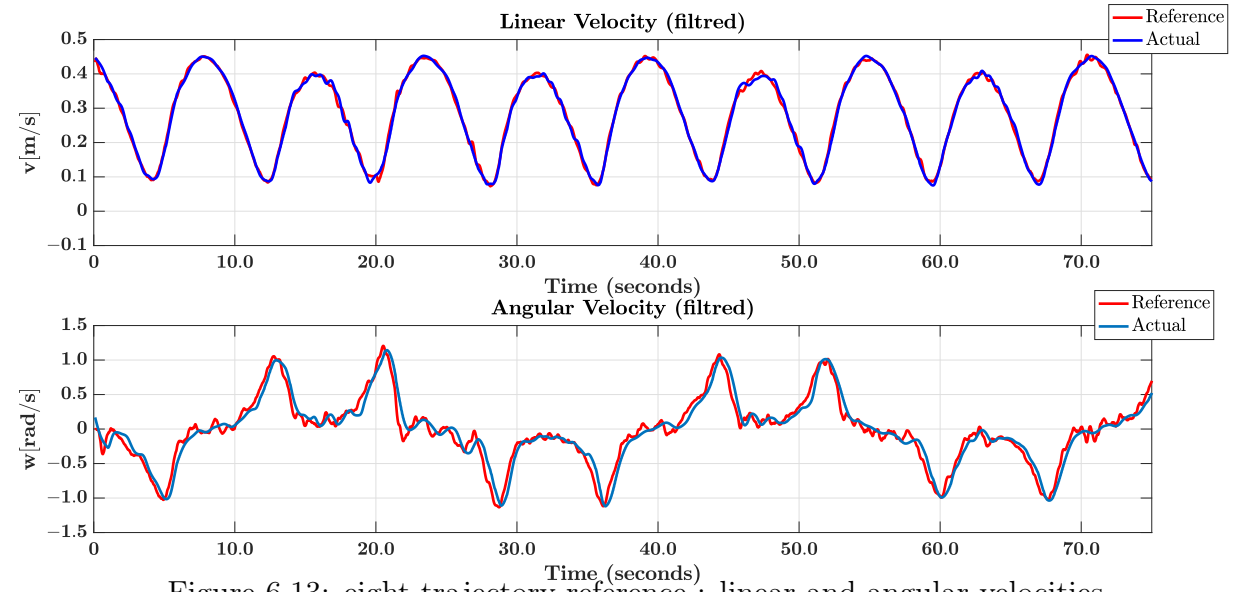


Figure 6.13: eight trajectory reference : linear and angular velocities

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, a practical guide of designing the electrical robot hardware was demonstrated, than a study of mobile robot kinematics and a formulation of a dynamics model witch takes voltages as inputs were presented. An experiment setup for estimating the robot parameters was proposed and performed. Next a centralized LQG servo velocity controller was designed and implemented, and a kinematic controller was reviewed and used for trajectory tracking task. Finally the experimental results was presented.

From the experimental results, we saw that in certain cases, the position loop is faster than the velocity loop, thus the necessity of considering the dynamic model and controller, the proposed LQG servo controller, despite the need of further adjusting for the overshot cancellation, ensured zero steady-state velocity tracking with minimal rise time. The position controller used had succeeded to track the desired trajectories with a small distance error, and the new platform was able to run the overall control structure with respectful sample time in real time.

7.2 Future Work

For future work, the following 2 topics are proposed :

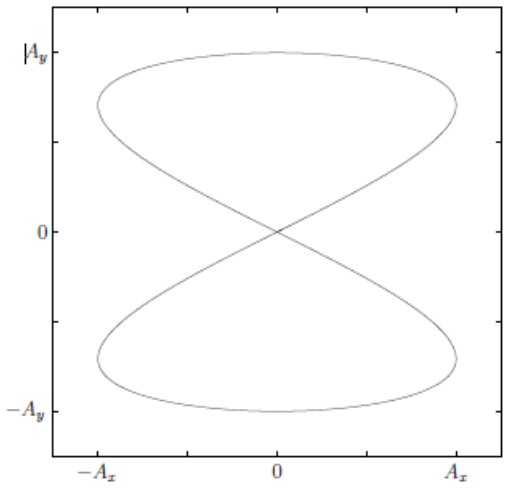
- Improving the velocity loop performance and robustness with a LQG/LTR controller, and a detailed study on the relevance of the inner loop (dynamic model and controller) in different sizes mobile robot platforms by comparing results in case of weather or not the inner loop is considered.

- Add a singel board computer to the robot and generate the control structure as a Robotic Operating System node to work on vision-based autonomous navigation or implementing a Simultaneous Localization And Mapping algorithm.

Appendix A

Generating a Trajectory

A.1 Figure Eight



A figure-eight style trajectory is convenient because it contains an equal number of left and right hand turns, and the trajectory can be run indefinitely while occupying a finite space. To define a figure-eight trajectory, the desired x position should be a sinusoid of frequency 2f, and the desired y position should be a sinusoid of frequency f.

$$x_d = A_x \sin(2ft) \tag{A.1}$$

$$y_d = A_y \sin(ft)$$

where A_x and A_y are the amplitudes in the x and y directions.

In order to feed the trajectory into a controller, the derivative of the desired x and y positions must also be defined.

$$\dot{x}_d = 2fA_x \cos(2ft) \tag{A.2}$$

$$\dot{y}_d = fA_y \cos(ft) \tag{A.3}$$

Appendix B

MATLAB Code

```
% MOBILE ROBOT PLANT SETUP AND LQG CONTROLLER DESIGN
clear all;
close all;
kx=0.1;
ky=0.1;
lx=0.3;
ly=0.1;
% Motors Specification
Kb1 = 0.71142; Kb2 = 0.70005;
Ra1 = 4; Ra2 = 3.5327;
Km1 = 0.95*Kb1; Km2 = 0.95*Kb2;
La1 = 0.0051381; La2 = 0.009607;
Bm1 = 0.028616; Bm2 = 0.044517;
Tf1 = 0.12256; Tf2 = 0.050745;
% Motor Driver lockup table
u_51=[[-5:0.2:-3],[-2.9:0.1:2.9],[3:0.2:5]];
vol_r=[-[10.31,9.98,9.88,9.8,9.72,9.62,9.5,9.37,9.22,9.05,8.87,8.73,...
    8.62,8.49,8.35,8.19,8.01,7.84,7.63,7.41,7.16,6.76,6.43,6.09,5.68,...
    5.22,4.74,4.16,3.56,2.92,2.15,0.42,0.35,0.032,0.027,0.023,0.019,...
```

```
    0.013,0.0068,0.00156],[0,0.0013,0.0018,0.006,0.012,0.018,0.026,...
    0.032,0.109,1.16,2,2.67,3.35,4.05,4.61,5.12,5.59,6.00,6.35,6.69,...
    7.06,7.29,7.55,7.77,7.96,8.14,8.31,8.45,8.59,8.72,8.86,9.09,9.25,...
    9.4,9.53,9.65,9.76,9.85,9.94,10.04,10.37]];
vol_l=[-[10.3,9.95,9.86,9.8,9.74,9.65,9.53,9.42,9.3,9.15,8.98,8.86,...
    8.75,8.64,8.53,8.39,8.25,8.05,7.9,7.6,7.4,7.1,6.88,6.48,6.1,5.7,...
    5.2,4.7,4.13,3.45,2.7,1.75,0.21,0.165,0.12,0.088,0.06,0.036,0.017,...
    0.005],[0,0.005,0.017,0.035,0.061,0.092,0.122,0.160,0.960,1.7,2.6,...
    3.25,4,4.6,5.1,5.6,6,6.44,6.8,7.1,7.4,7.65,7.8,8,8.11,8.25,8.4,...
    8.55,8.68,8.82,8.95,9.15,9.29,9.44,9.56,9.67,9.75,9.82,9.92,10.02,...
    10.36]];
% Robot Specification
r = 0.13/2;                 %wheel radius in Meters
m = 8.2;                    %Mass in Kg
l=0.21;                     %Wheels axis length in Meters
I = 1.1114;                 %Robot inertia
a=0.18;                     %Interest point distance from wheels axis
% State-space model of the dynamic model
A=[-Ra1/La1     0            -Kb1/(r*La1)          -Kb1*l/(2*r*La1)
    0           -Ra2/La2     -Kb2/(r*La2)           Kb2*l/(2*r*La2)
    Km1/(m*r)   Km2/(m*r)    -(Bm1+Bm2)/(m*r^2)    l*(Bm2-Bm1)/(2*m*r^2)
    Km1*l/(r*I) -Km2*l/(r*I) l*(Bm2-Bm1)/(I*r^2) -(l^2)*(Bm2+Bm1)/(2*I*r
        ^2)];
B=[1/La1 0
   0     1/La2
   0     0
   0     0];
C=[0    0    1/r  l/(2*r)
   0    0    1/r  -l/(2*r)];
D=[0 0
```

```
    0 0];
sys= ss(A,B,C,D);
sys.InputName={'RvoltageR','RvoltaeL'};
sys.OutputName={'OmegaMR','OmegaML'};
sys.StateName={'Ia1','Ia2','Lvelocity','Avelocity'};
% LQG parameters design
% LQR
WR_max=10;WL_max=10;        %rad/s
Ia1_max=1;Ia2_max=1;        %A
V_max=(WR_max+WL_max)*r/2;          %m/s
W_max=(WR_max+WL_max)*r/l;          %rad/s
alpha_ia1=0.1;alpha_ia2=0.1;alpha_v=0.1;alpha_w=0.5;
alpha_xi1=0.6;alpha_xi2=0.6;
Vr_max=10;Vl_max=10;        %V
beta_vr=1/sqrt(2);beta_vl=1/sqrt(2);
rho=0.1;
v_q=[(alpha_ia1/Ia1_max)^2,(alpha_ia2/Ia2_max)^2,(alpha_v/V_max)^2,...
    (alpha_w/W_max)^2,(alpha_xi1)^2,(alpha_xi2)^2];
Q=diag(v_q);
v_r=[(beta_vr/Vr_max)^2,(beta_vl/Vl_max)^2];
R=rho*diag(v_r);
[K_lqi,S,e] = lqi(sys,Q,R);
K_x=K_lqi(:,1:4);
% Kalman filter
sys_noise = ss(A,[B B],C,0);
sys_noise.InputName={'RvoltageR','RvoltaeL','volR_noise','volL_noise'};
sys_noise.OutputName={'OmegaMR','OmegaML'};
sys_noise.StateName={'Ia1','Ia2','Lvelocity','Avelocity'};
q1=0.1;q2=0.1;r1=0.1;r2=0.1;
Wn = [q1 0;0 q2];
Vn = [r1 0;0 r2];
[kalmf,L,P,M] = kalman(sys_noise,Wn,Vn);
[An,Bn,Cn,Dn] = ssdata(kalmf);
```

```
82 Cy=[1 0 0 0 0 0;0 1 0 0 0 0];       %extract y_e from kalmf output
83 Cx=[0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0; 0 0 0 0 0 1];    %extract x_e
       from kalmf output
```

Appendix C

Weighting Matrix Selection

- A good rule of thumb when selecting the weighting matrices Q and R is to normalize the signals:

$$Q = \begin{bmatrix} \frac{\alpha_1^2}{(x_1)_{max}^2} & & & \\ & \frac{\alpha_2^2}{(x_2)_{max}^2} & & \\ & & \ddots & \\ & & & \frac{\alpha_n^2}{(x_n)_{max}^2} \end{bmatrix} \tag{C.1}$$

$$R = \begin{bmatrix} \frac{\beta_1^2}{(u_1)_{max}^2} & & & \\ & \frac{\beta_2^2}{(u_2)_{max}^2} & & \\ & & \ddots & \\ & & & \frac{\beta_m^2}{(u_m)_{max}^2} \end{bmatrix} \tag{C.2}$$

- The $(x_i)_{max}$ and $(u_i)_{max}$ represent the largest desired response/control input for that component of the state/actuator signal.

- The $\sum_i \alpha_i^2 = 1$ and $\sum_i \beta_i^2 = 1$ are used to add an additional relative weighting on the various components of the state/control

- ρ is used as the last relative weighting between the control and state penalties $\Longrightarrow$ gives us a relatively concrete way to discuss the relative size of Q and R and their ratio Q/R

Bibliography

[1] *Intelligent Robot Development Kit ED-7273-manual.*

[2] Rensselaer mechatronics, `https://minseg.com/pages/downloads`.

[3] `http://motorbank.kr/`.

[4] `https://www.arduino.cc/en/Guide/ArduinoDue`.

[5] Divya Aneesh. Tracking controller of mobile robot. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, pages 343–349. IEEE, 2012.

[6] Rafael Fierro and Frank L Lewis. Control of a nonholonomic mobile robot: backstepping kinematics into dynamics. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 4, pages 3805–3810. IEEE, 1995.

[7] Lentin Joseph. *Learning Robotics Using Python.* Packt Publishing Ltd, 2015.

[8] Jeffrey W Laut. *A dynamic parameter identification method for migrating control strategies between heterogeneous wheeled mobile robots.* PhD thesis, Worcester Polytechnic Institute, 2011.

[9] Felipe N Martins. An adaptive dynamic controller for autonomous mobile robot trajectory tracking. Available at `http://www.mathworks.com/matlabcentral/fileexchange/44850`.

[10] Felipe N Martins, Wanderley C Celeste, Ricardo Carelli, Mário Sarcinelli-Filho, and Teodiano F Bastos-Filho. An adaptive dynamic controller for autonomous mobile robot trajectory tracking. *Control Engineering Practice*, 16(11):1354–1363, 2008.

[11] Felipe N Martins, Mário Sarcinelli-Filho, and Ricardo Carelli. A velocity-based dynamic model and its properties for differential drive mobile robots. *Journal of Intelligent & Robotic Systems*, 85(2):277–292, 2017.

[12] Pascal Morin and Claude Samson. Motion control of wheeled mobile robots. In *Springer Handbook of Robotics*, pages 799–826. Springer, 2008.

[13] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots.* MIT press, 2011.