RAFIK HARIRI UNIVERSITY

SELF-BALANCING ROBOT

By

MOHAMMAD SAYDEH

A capstone research project submitted partial fulfillment of the requirement for the degree of Master of Science in Mechatronics Engineering of the college of Engineering Specialty of Mechanical and Mechatronics Engineering Department, at Rafik Hariri University.

MESHREF –LEBANON

DECEMBER 2021

RAFIK HARIRI UNIVERSITY

SELF-BALANCING ROBOT

By

MOHAMMAD SAYDEH

Approved by

DR. HASSAN HARIRI Signature

____
_

College of Engineering

Date of Thesis Research Defense [December 2021]

RAFIK HARIRI UNIVERSITY

THESIS RESEARCH RELEASE FORM

I. [MOHAMMAD AHMAD SAYDEH]

- ☐ Authorize the RAFIK HARIRI UNIVERSITY to supply copies of my project to libraries or individuals upon request

- ☐ Do not authorize the RAFIK HARIRI UNIVERSITY to supply copies of my project to libraries or individuals for a period of two years starting with the date of the project defense

____

Signature

____

Date

ACKNOWLEDGEMENTS

I would like to express my gratitude first and foremost for my family's endless support since the beginning of my Engineering studies. I want to also thank my project supervisor Dr Hassan Hariri for suggesting this topic and for his help comments and support. Last but not least, I want to thank my advisor Dr Bassam Moslem and the MME department chairperson Dr Nadim Diab for their help and understanding particularly during my Masters journey.

ABSTRACT

This project represents the work done to model and simulate a two-wheeled self-balancing robot TWSBR using Proportional-integral-derivative (PID) and Linear-quadratic regulator (LQR). These robots are commercially present, yet they still attract several control enthusiasts to study their stability, using different algorithms, due to their sustainable design and advantages. In this book, the commercial OSOYOO TWSBR is modeled on Matlab Simscape, and simulated using PID and LQR controllers on Matlab Simulink. The goal of the controllers is to monitor and maintain the robot's tilting angle and position using its displacement force. After modeling and tunning, the controllers' outcomes are evaluated and compared using the control system evaluation parameters and control effort. LQR yielded the most promising results; however, it required more sensors to access the robot's states. Therefore, the LQR with minimum order observer controller is developed for sensors' cost reduction. The latter competed with LQR but was less stable and took longer time to reach steady state. For future work the goal is to control the real robot using these 3 different controllers and compare their results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background Information

When talking about control systems, we are talking about autonomous robots, self-driving cars, drones, transportation means, heavy industries, home appliances, medical devices, cell phones, computers etc. The control theory has been changing lives since the beginning of time, and it is considered as a solution for various complex and dynamic systems including self-balancing robots. One of the most important and considerable milestones in the robotics field's history is the invention of self-balancing robots. Since the 1980s, these robots attracted researchers, from different countries, to study their systems [1]. Despite their non-linearity instability and complex yet broad control systems, self-balancing robots have found their application in multiple fields due to their sustainable design. These fields include military or civilian self-transport alternatives or assistants, dangerous and normal workplaces, research, and testing of various applicable control algorithms [1,2]. What is in common between all these self-balancing robot operators, is that they are all based on the inverted pendulum theory. The TWSBR combines the inverted pendulum theory with two wheeled robot and the dynamics control. Self-balancing robots existed in the market during the past two decades, starting from the famous Segway, as shown in Figure 1, which is independently controlled [3]. The LegWay robot, as shown in figure 2, from Lego company was also designed to move on irregular or inclined surfaces but using remote control operation [3]. Other self-balancing robot Balanduino, as shown in figure 3 which is open source, and other robots developed by researchers [4]. In addition to their considerable price, these robots are eco-friendly since they are electrically powered, and time and space efficient due to their small size and extra flexibility. However, each of these robots, whether commercially present or built by robotics enthusiasts, is built using a unique control algorithm, PID and LQR, Fuzzy Logic Controller (FLC), etc. And in research testing the control theory applied on such robots is done via simulation prior any code implementation on the microcontroller. This is due to the time cost and risk that simulation saves during multiple parameters tuning and results variation.

1.2 Motivation

Controlling TWSBRs is a newly and continuously evolving field. These types of robots are proved to be sustainable in terms of design and environmental effect. In addition, few researchers worked on simulating different control algorithms on a moving self-balancing robot and then compare the results and decide on the appropriate control method. For the robot’s model little to no work involved modeling OSOYOO 2WD Balance Car Robot, on MATLAB Simscape, and implement it as a plant in these different control systems.

Figure 1: Segway Self Balancing Human Transporter

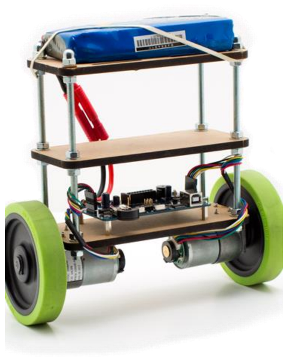


Figure 1: Balanduino

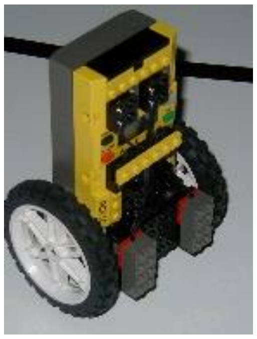


Figure 2: Legway

CHAPTER 2

LITERATURE REVIEW

2.1 Principle of Inverted Pendulum

TWSBR is based on the fundamental of an inverted pendulum, where the wheels are simplified as a cart on wheels and the robot's chassis is compared to the pole, as represented in Figure 4. In this section the dynamics of the inverted pendulum will be discussed.

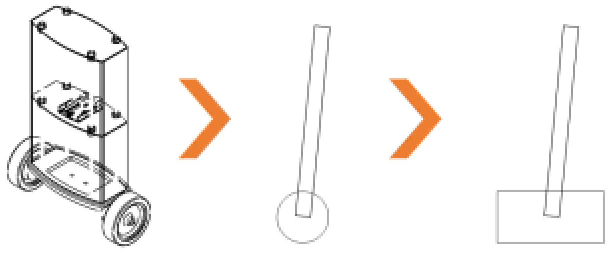


Figure 3: TWSBR: Simplified [9]

Other than TWSBRs, the inverted pendulum concept has plenty applications like human walking robots, earthquake resistant building design and missile launchers [7]. The widest and most common example of an inverted pendulum is the human being. The inverted pendulum as shown in Figure 4 is a non-linear dynamic system, and despite its complexity, many researchers have worked on developing and enhancing its control system.

2.2 Previous Work

Previous work to be reviewed in this project fall in the scope of simulating single or multiple control methods on a TWSBR and compare the results if applicable. The work cited will be listed and described according to their ascending dates.

In this work two control systems are involved in maintaining robot's balance: PID and LQR. These methods are implemented to control two subsystems: self-balance, during motion, and yaw rotation, which involves regulating steering angle during turning. It must be mentioned that PID was applied to both subsystems and LQR for self-balancing only. After simulation, the two methods were compared, and it was found that the LQR outperformed PID. PID involved using single feedback signal, either tilt angle or tilt angle rate from accelerometer or gyroscopes, for control value generation. LQR had the advantage of combining all output state variables including tilt angle rate, tilt angle and position to get the control value. This usually allows the system to achieve stabilization in the desired time even if one variable was inaccurate. For this reason, LQR achieved the steady state faster than PID [3].

When comparing controllers in this work, researcher included, in addition to the LQR and PID controllers, the FLC. These controllers were implemented on a two-wheeled inverted pendulum mobile robot. After experiments it was found that the FLC outperformed LQR and PID by having a less overshoot and faster response; however, it consumed higher energy. And comparing LQR and PID alone both methods took the same rise time and energy; however, LQR showed a faster response and less overshoot than PID [5].

Rahman et al. also compared LQR PID and FLC on a TWSBR using Robot Operating System ROS and Gazebo. It was found that when plotting real time pitch angles with respect to time, PID gave the most stable performance, however faster response was recorded with LQR. On the other hand, the FLC was the most non-stable, due to its inefficient tuning [6].

This work involved the design development and analysis of TWSBR using a novel control method involving PID. In most common controllers and feedback systems, state estimation is done via the Inertial measurement unit IMU, which includes motion measuring sensors, and feedback is received from the actuator using tachometers or encoders. This work is based on generating the feedback after an on-board state estimation algorithm, by that eliminating the need of actuators. This robot demonstrated robust performance by taking a very fast response to correct any disturbances from equilibrium. The controller gains played an important role in reaching stability with minute undesirable errors, where they were adjusted according to the robot's height and mass distribution. In addition to the gains, moment of inertia, motor and IMU parameters proved to affect the conserving stability. It must be noted that choosing precise sensors and actuators could highly eliminate imperfections in stability [4].

Another work involved experimental comparison between the conventional PID controller and the optimal LQR controller. These controllers were simulated using the

MATLAB/Simulink platform and then were digitally implemented on the TWSBR. The aim was to maintain the robot in vertically balanced despite disturbances. Comparison was done on the level of stabilization adequate dynamic response and control robustness. Results showed that LQR is simpler since it requires 2 gains for control; however, the PID needs computational calculation. Both controllers meet the specification: setting time less than 200 ms and robot's vertical posture less than 5 degrees tilt. On the other hand, the PID had the overshoot issue due to its high integral side gain, but with smaller steady state error. LQR had a better transient response minimizing overshoots and compensating steady state error, making the reaction to body falling and imbalances faster and smoother. This makes the general LQR controller practice better than PID controller [2].

Thwin et al., applied the LQR algorithm into the TWSBR system, and they stated that LQR is an automatic method of getting the optimal state-feedback controller. In this work the system's full state information regarding the cart's position and velocity are received from a dc motor encoder, and the pendulum's angular position and velocity are attained from an IMU sensor. The pitch angle was determined using a complimentary filter that assessed the gyroscope and accelerometer signals. The robot was implemented sing the Arduino Mega controller, which can be compatible with MathWorks and Simulink (using the Rensselaer Arduino Support Package). They concluded that Q and R values must be carefully tuned due to their sensitive effect on the control system, and that sensors must be carefully selected, and their signals must be filtered and conditioned to remove any noise affecting the control's performance [7].

Another most recent work about TWSBR was done to apply control methods on the robot so it can self-balance. This system utilized signals from an accelerometer and gyroscope and send them to the microcontroller and in parallel the latter gives rotation instructions to the motor drive of the wheel so that the TWSBR can self-balance. As cited by Patil et al. this system better describes an optimal robot able to make sharp and fast turns and navigate in tight areas, which will be helpful in plenty of applications. Controlling this system is done via PID tuning, and its implementation is done on the ATEGA-328P microcontroller. The robot was able to effectively balance itself during forward and backward leaning;

however, its performance can be enhanced by improving the motor speed readings' precision, hence enhancing stability [8].

2.3 Purpose

This project will be based on modeling the self-balancing car "OSOYOO 2WD" model built and simulated on Matlab Simscape. This model will be simulated using different control algorithms the PID and LQR to ensure system stability. The robot must travel and reach a certain distance while maintaining the tilting angle the closest to zero in the least amount of time and with the least control effort. The output responses, settling time, overshoot control effort and other metrics will be assessed and compared among the 2 simulation models. Based on this assessment the most optimal model will be chosen for future real implementation on the robot's kit. According to literature review, still no similar work involved using this model. This work will add a great value to the huge field of controlling TWSBRs which outperform normal robots due to their sustainable architecture.

2.4 PID

A Proportional P, Derivative D and Integral I control technique is one of the most known feedback mechanisms used in control plants. Briefly the purpose of this controller is to forcefully match the feedback to a setpoint, in other words to set the angular position of the pendulum, which is continuously acquired, to π. PID systems as well are known as SISO Single Input Single Output systems. The three adjustments in PID aid the plant in compensating system changes. It must be noted that PID controllers provide their best performance when included in light-weight systems. It is also suggested for an often load and setpoints varying structures. The three working pillars in PID control must be individually modified or tuned to trigger the adjustment factor [11].

As shown in Figure 5, Without any of the PID pillars, the plant will work on processing the input and executing an output, which in non-linear systems will be different by a quantity called steady state error. To match the output to the set or required goal, a controller must be introduced. In case of PID, when introducing the first term which is "P", the controller will work on multiplying the given error at any time t with a certain constant or gain Kp. This, in some systems will help in matching the output to the set point faster, until the difference becomes null. For this reason, the proportional part is called to be used for error elimination [3]. Some system will require an additional term since P alone can sometimes never nullify the steady state error. In this case one can refer to the integrator term, which accordingly will sum the input over time, which will make it the memory of the system. When the steady state error in non-zero the value will be integrated pushing the output to reach the setpoint. In some systems the PI alone can lead to overshoot due to the non-ideal approach it gives, therefore, to predict the future overshoot, the derivative term will be needed to measure the rate of error or how fast the error is increasing or decreasing. If the error is decreasing as we are starting to reach the set point, the derivative will output a negative term which will be summed to the rest of the terms and therefore reducing the control signal to prevent overshoot. Explaining the theory behind PID makes it easy; however, gains must be carefully tunned, and one must compromise based on the application.

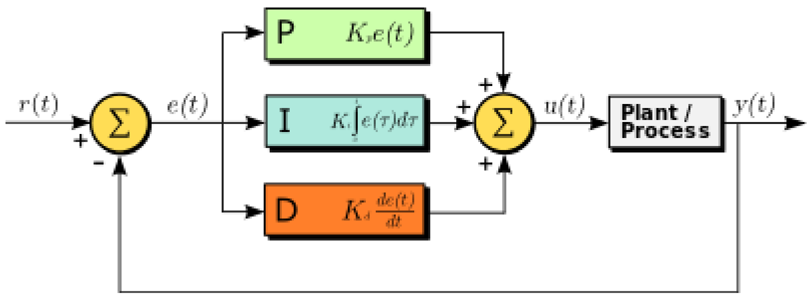


Figure 4: PID controller

2.5 LQR

Linear quadratic regulator controller is another advanced type which is based on using state space systems as shown in equations (1) and (2) and represented in Figure 6.

$$\dot{x}(t) = Ax(t)+Bu(t) \quad (1)$$

$$y(t)=Cx(t)+Du(t) \quad (2)$$

Where $x(\dot{t})$ is the state vector, which varies according to the variable states in the system x(t) and any external input u(t), this relationship is governed by the matrix A, which describes how the internal states are connected to each other’s. The variable states in the system are described by the minimum set of variables needed to fully describe the system and predict future behavior. B is also a matrix which describes how the input enter to the system and which state is it affecting.

The y(t) vector is the output of the system and in other words it is the part the user is interested to know. C matrix describes how the states are connected to produce the output and D acts as standard gain multiplied to the input.

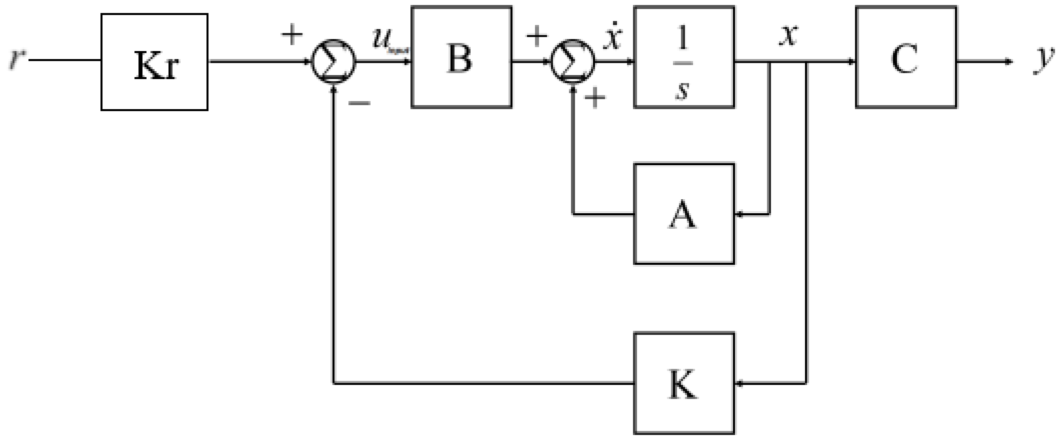


Figure 5: State Space Model Describing the Dynamic System's State-modified [9]

LQR is called the optimal control technique when compared to the pole placement technique. Briefly the pole placement technique involves manually choosing 2 gains: the state vector gain and the scaled reference gain (as a pre-compensator for the steady state error), as shown in Figure 5 “K and Kr”, to move the system into stability by having the poles, which are identical to the eigen values of A matrix, on the left axis side. LQR involves providing optimal gains to reduce the general cost function as in equation 3, and to increase performance with minimum effort.

$$J=\int_0^{\infty}[x^T Q x + u^T R u]dt \qquad (3)$$

And according to Figure 6:

$$u=rkr - Kx \qquad (4)$$

Where J is the cost, which is always a scalar value, x is the state variable, u is the input and t is the time. Q and R determine the weights which heavily influence the cost. Q is a matrix that through it we can choose what variable to penalize, and R is a real constant number.

After setting the state variables of the system, the user can start adjusting the Q and R values based on the design. The optimal gain or the K matrix can be computed numerically on Matlab according to the “lqr” function, the response can be simulated, and re-adjustments can be made. In pole placement the user manually places the poles by adjusting the K values; however, in LQR Q and R are tuned accordingly.

CHAPTER 3

SIMULATION

Simulation in this project is done on 2 different models on the Matlab Simscape environment. Technically, Simscape enables fast development of physical systems' models in Simulink's platform.

3.1 Modeling of the 2WD Robot

The TWSBR model simulation will be done based on the characteristics of the commercial OSOYOO 2WD Balance Car Robot shown in Figure 7. The real robot's hardware is composed of an Arduino Uno controller, the wheels, stepper motors and motor driver, the sensing unit, which is the MPU6050 gyroscope module, Bluetooth module for smartphones connection, and other mechanical and electrical components [13]. Model development will be done using 2 different controllers, with identical robot characteristics. These characteristics include weight and dimensions of the cart and the chassis components of the robot. These metrics were measured after robot disassembly.

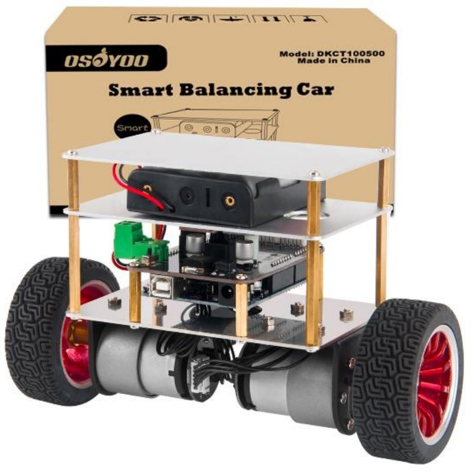


Figure 6: OSYOO 2WD Balance Car Robot Kit [13]

3.1.1 Cart

The cart consisted of the right wheel left wheel and the shaft. The shaft is not included in the assembly of the commercial robot as shown in Figure 7, it was included in the model design for simplicity. Its size was modeled according to the distance between both wheels, and the thickness was based on the motors' dimensions, the weight was also set a matching the weight of the motors and the brackets that connect the wheels to the chassis. Table 1 represents the geometry and mass of the modeled shaft.

Table 1: Shaft Geometry & Mass

Shaft		Unit
Shape	Brick	
Dimensions	[1.5, 1.5, 18]	cm
Mass	345	g

The right and left wheels were modeled as wheel bodies and wheel tires, there geometry and mass are also described in Table 2. These metrics apply symmetrically to both left and right wheels.

Table 2 Left & Right Wheels Geometry & Mass

Left/Right Wheels			Unit
	Wheel Body	Wheel Tire	
Shape	Cylinder		
Radius	2.36	3.2	cm
Length	2.57	2.6	cm

Mass	23.4	25.6	g

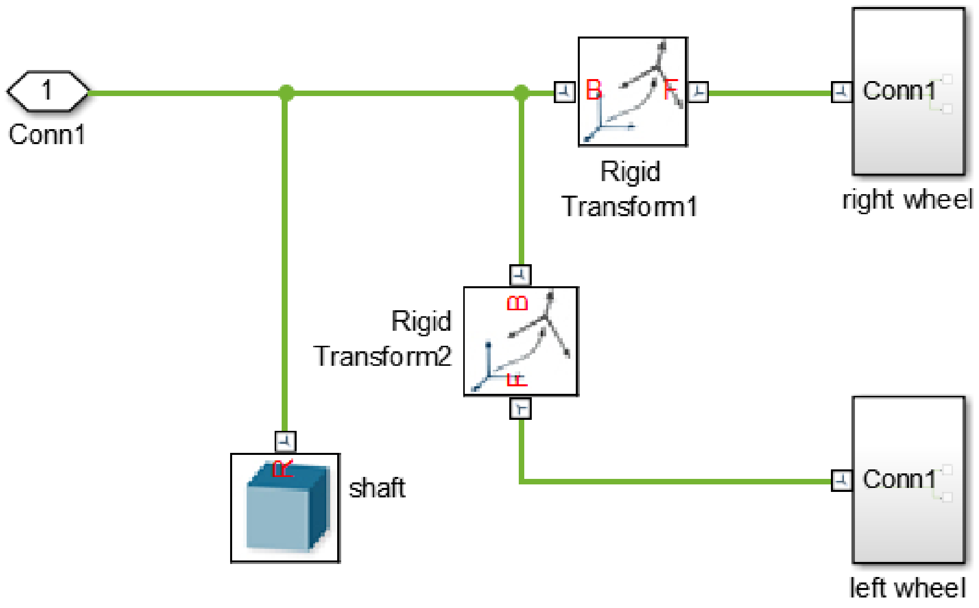


Figure 7: Cart's Model on Simulink

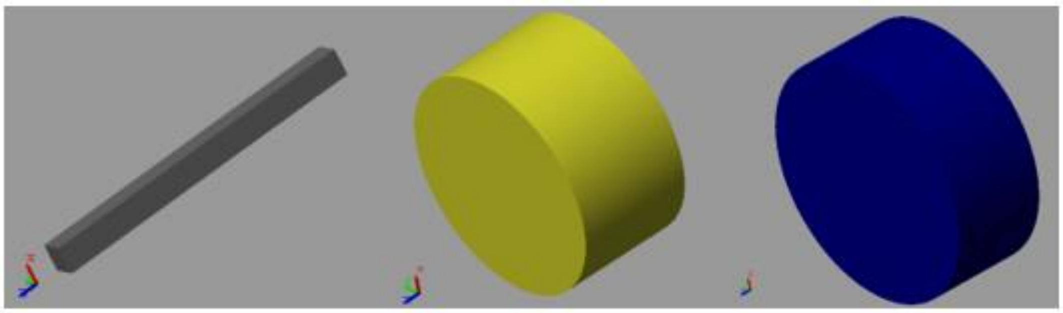


Figure 8: Shaft, Wheel Body and Wheel Tire Simscape Models

Figure 8 shows the implemented the complete cart’s model on Simulink, where the rigid transform is the block that enables the conversion of the wheel’s states tied to the shaft. Figure 9 shows the separate models of the cart’s component as appearing in the Simscape environment.

3.1.2 Chassis

The chassis part of the model is developed from 4 levels and the battery: bottom or base level which is the heaviest since it is holding the rest of the layers, the controller level holding the Arduino, the middle level which holds the battery and finally the top level over the batteries. These levels are separated and held using screwdrivers. Model wise these levels and the battery were developed using Bricks and held using rods. For simplicity the mass of the controller and motor driver were included in the mass of the level considerably holding the controller. Table 3 describes the geometry and mass of each of these levels. Table 4 describes the geometry mass and number per set of the rods separating and holding each level, 3 sets of 4 rods each were needed in this model where 2 sets were of the same geometry and mass. These 2 sets or pillars as shown in Figure 10 separated between middle and controller levels and between middle and top levels.

Table 3: Geometry & Mass of the Chassis's Levels

Levels						Unit
	Base	Controller	Middle	Top	Battery	
Shape	Brick					
Dimensions	[12.58,8.2,0.16]	[8.22 6.19 1]	[12.58,8.2,0.16]	[12.58,8.2,0.16]	[6,4,1.1]	cm
Mass	107.8	69.6	63	63	105	g

Table 4: Rods Geometry Mass & Numbers

Rods			Unit
	1st Set	2nd Set	
Shape	Cylinder		
Radius/rod	0.5	0.5	cm
Length/ rod	4.4	2.3	cm

Mass/ rod	6.1	2.9	g
Number/Set	4	8	Rods

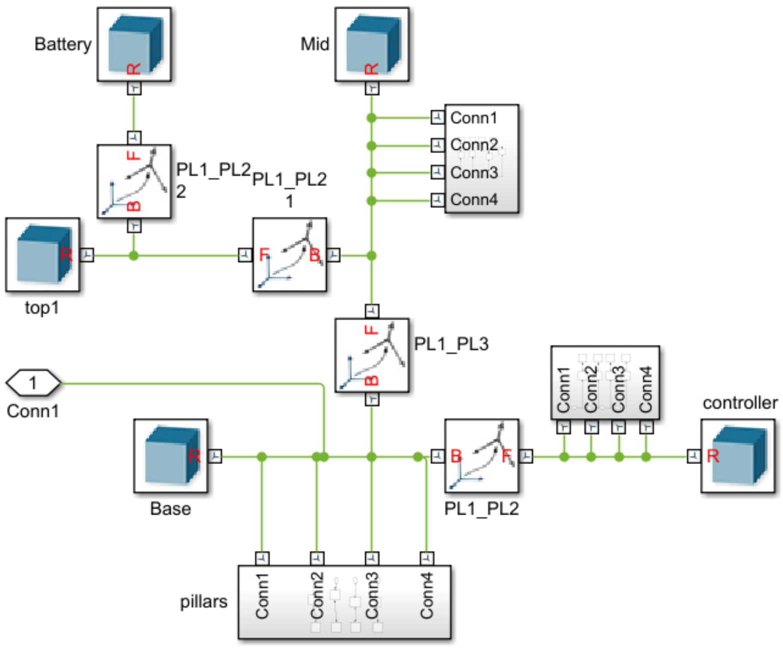


Figure 90: represents the chassis's model implemented on Simulink

Figure 11 shows how the levels and rods are shaped in Simscape, since the base middle and tope levels have the same dimensions, the middle and top levels' figures weren't included for simplicity. The second level shown in figure 12 is the controller layer which is thicker but lighter than the base level due to its smaller dimensions. The last radius in the last rod appears bigger; however, according to Table 4 both have the same radius: 0.5 cm. this difference is due to last rod's shorter length.

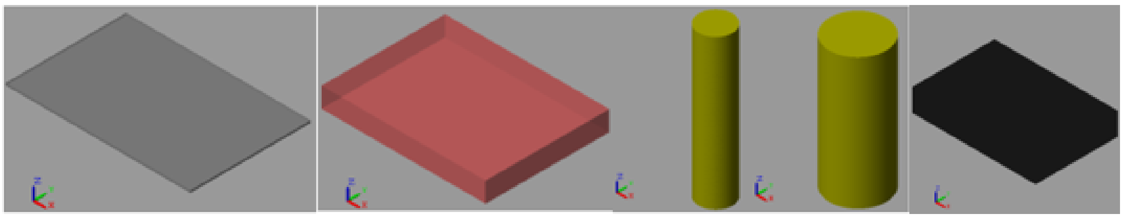


Figure 10: Levels controller battery and rod's Simscape models

Figure 12 represents the chassis's model on Simulink. The levels are connected and via the pillars. As reflected in figure 12 the base is connected through the shorter pillars to the controller level and at the same time to the middle level which holds the battery that are covered by the top level.

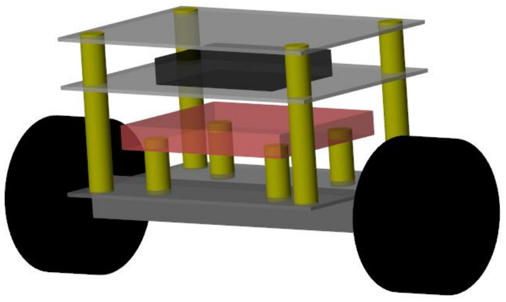


Figure 11: Simscape Robot's Model

3.2 Simulation: Common Blocks

After describing the model of the robot's body, the basic and common blocks in the robot's plant, as shown in figure 13 in all controllers' diagrams to be simulated will be explained. But before that one must decide and discuss the model's input output and control signal which are almost common in all models. The goal is to control the robot's displacement and tilting angle by a control signal identical to the displacement force. The control signal is introduced to the "f" pin in the prismatic joint which presents the displacement input force, this signal i.e. the output of the controller is determined according to the difference between the reference signal and the feedback from the sensing side, one of it is from "p" the sensing output in the prismatic joint which gives information about the lateral displacement. The prismatic joint is provided by world base connection to the World Frame through "B". It is described as the system's rail which gives it the ability to move by having the cart connected to its Frame "F" through a rigid transform. This joint is connected through another rigid transform to the revolute joint. It gives the latter its base which in return is translated to the chassis through the "F" pin. The revolute joint block includes a gyroscope which enables angle sensing from "q". Like "p"," q" which is the feedback signal providing information about the chassis's angle. One last important common block is the scope that is used to visualize and record the control effort to continuously maintain the robot's equilibrium, this control effort is identical to the displacement force that the controller provides to correct and maintain the robot's position.

3.3 Simulation: PID

Since PID controller is known to be a SISO system, 2 controllers will be introduced in this model. First, the PID model's input is divided between the 2 PID controllers as presented in Figure 14. The input "reference for displacement" is a unit step function where its value is set to 1 given that the robot must move this distance along the x axis and fed to the PID displacement controller which provides the robot with the external force." q" and "p" sensor represent the feedback of PID controllers to produce the needed control action "f". In this way angle and displacement are controlled using the displacement force commands according to the error signals (feedback from sensors-input signal) continuously feeding the PID controllers.

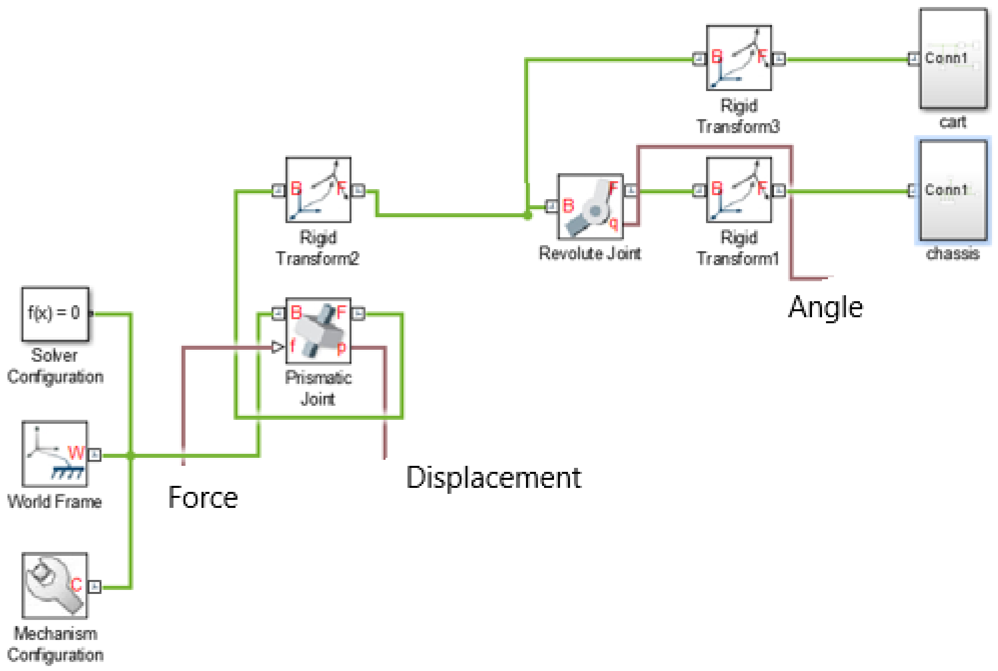


Figure 12: Robot's Plant

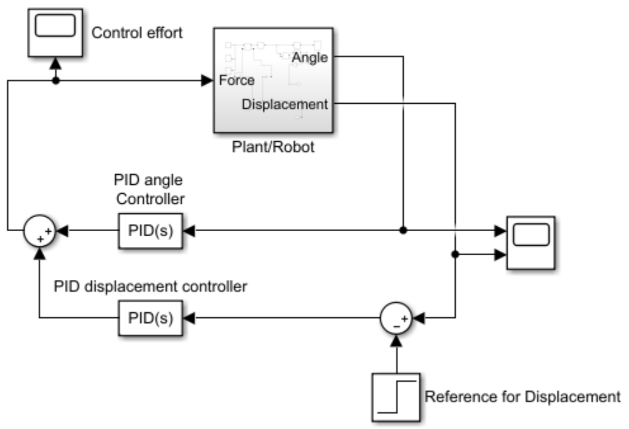


Figure 13: PID Controller

For performance evaluation the following metrics will be assessed: the rise time, overshoot undershoot, settling time, steady state error, and control effort. The rise time is the time needed to first cross the steady state. Overshoot correspond to the exceeded plant's performance and undershoot quantizes the still needed effort to reach steady state. Settling time represents the time the system stabilizes with a constant steady state error. The steady state error is the difference between the target state and the actual state as time goes to infinity.

As a description to figure 15, the robot during the first 4.78 seconds, reached the targeted 1 m distance but deviated from its equilibrium position by 3.24 degrees and during stabilization attempt it redeviated by 0.78 degrees to then reach steady state which is 0 degrees at 15.8 s with a steady state error 0.15 %. For displacement after reaching the target distance the robot walked an excessive distance of 0.23 meters and then returned to its steady state at 23.2 s with a steady state error of 0.3%. Subsequently the control effort will be typically as in figure 16, which translates the robot's quick attempt to correct the angle deviation at the start.

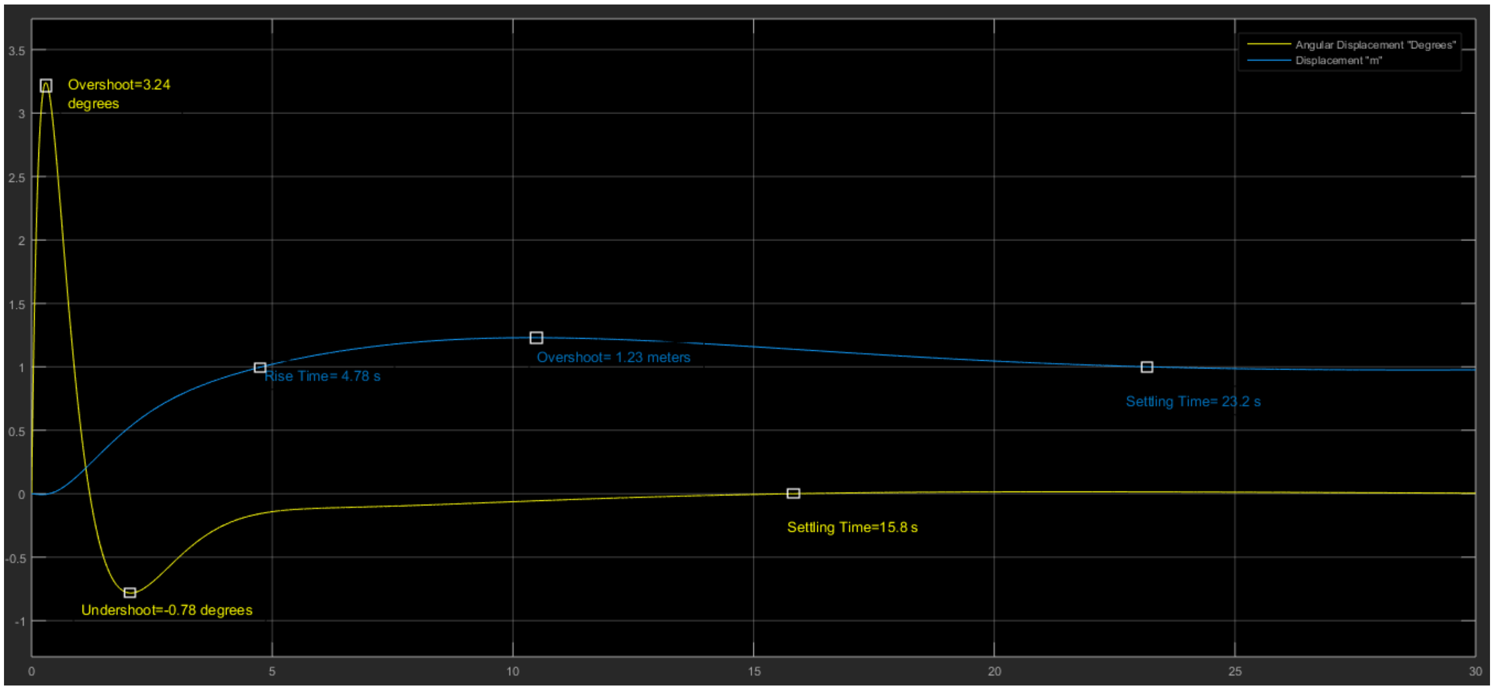


Figure 14: PID: Displacement and tilting angle responses in meter and degrees, respectively, in function of time in seconds

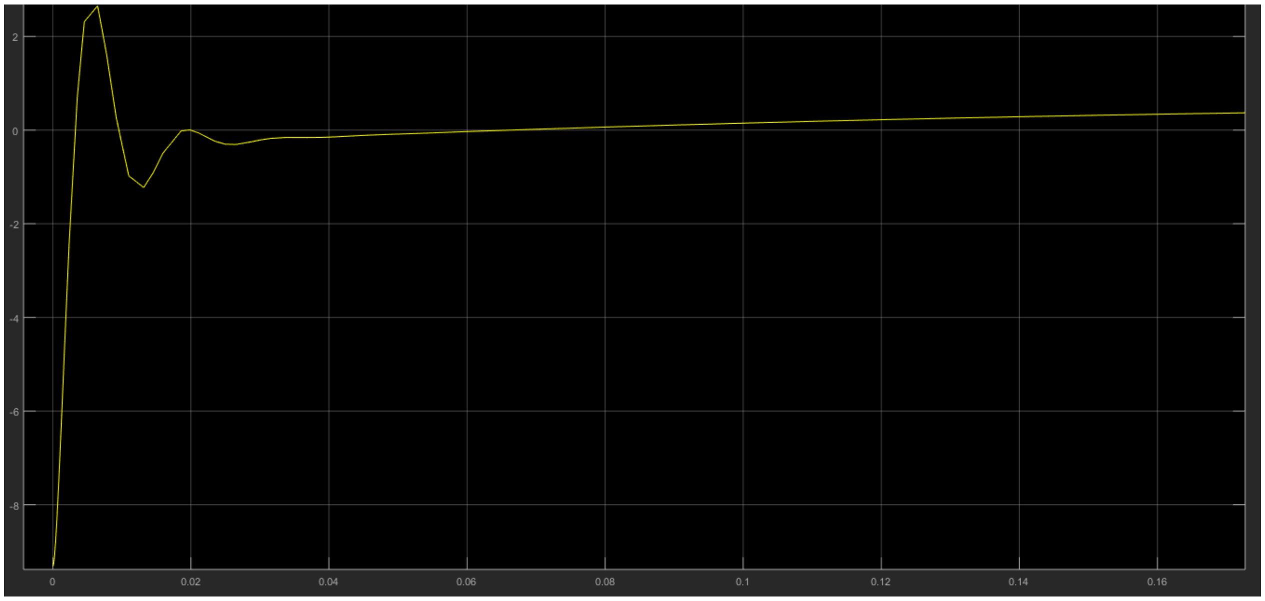


Figure 15: PID: Control Effort with respect to time

3.4 Simulation: LQR

This model, as shown in figure 17, involves using state variables due to the controller's concept of operations. The state variables here included in addition to the displacement, and angle, the velocity and angular velocity. Therefore, the prismatic joint will output an additional signal: velocity labeled as "v" and the revolute joint will output the angular velocity "w". these metrics are determined using sensors in this model. It must be mentioned that this model is a SIMO system, therefore one step input of a unity final value is fed to the controller. This system consisted of feed forward and feedback controls. The first is essential in reducing the impacts of the measured interruptions that in our case may exist in displacement values, whereas the feedback control compensates for error in measurements. The velocity angular displacement and angular velocity are carried to a feedback system where they are multiplied by their tuned gains given by the "lqr" function in the Matlab script. The matrix is then subtracted from the feedforward vector to produce the control signal. The displacement variable was fed to the feedforward control since in this project we are controlling the displacement. For this reason, the "p" value is subtracted from the step signal and introduced to the feedforward control. Angle and displacement response are evaluated since they are the assigned outputs and control effort is measured.

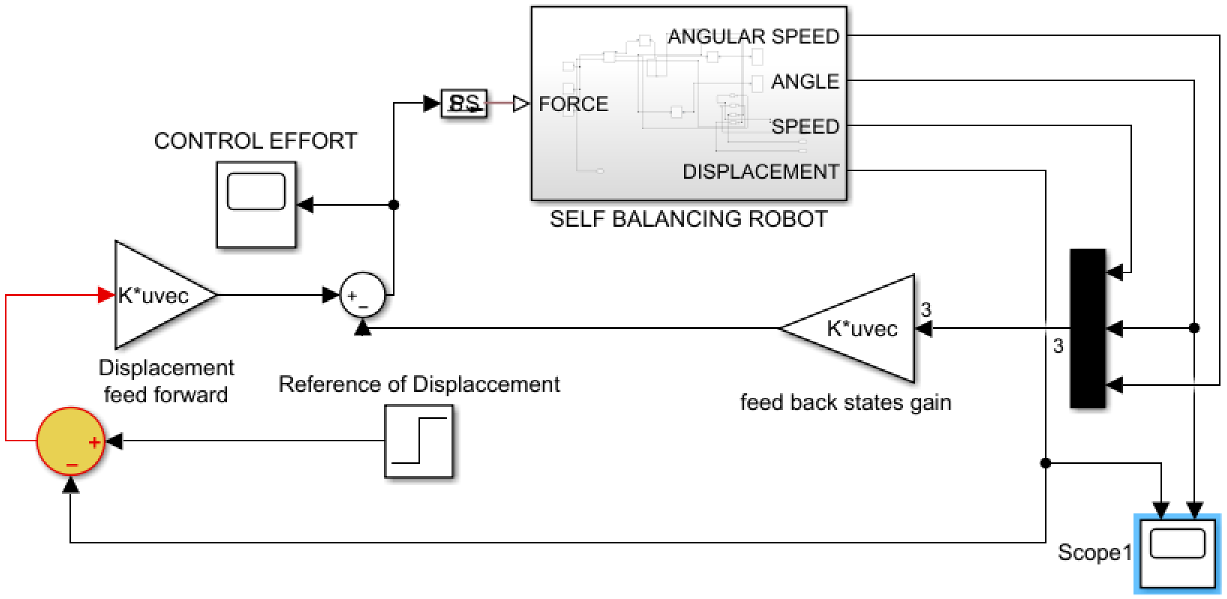
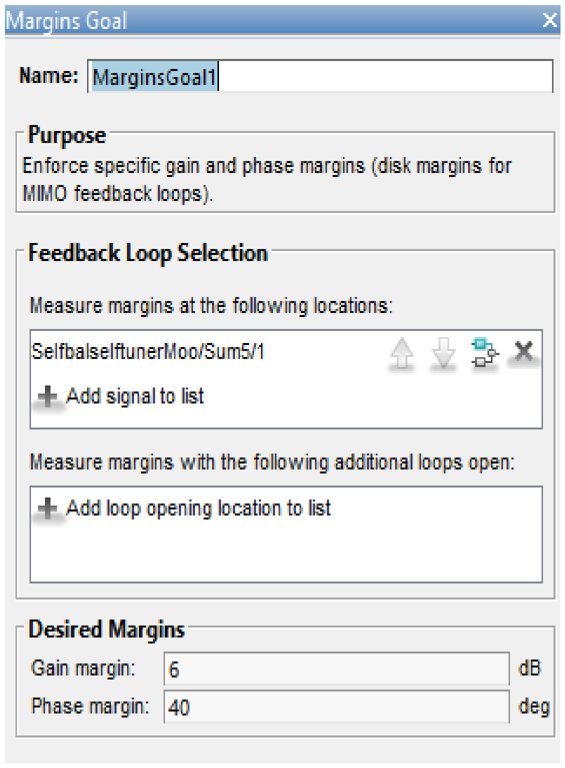


Figure 16: LQR block diagram

This performance of the LQR controller is governed by goals and constraints from self-tunning. These include the margin goal, which enforces phase and gain values or margins, and it is implemented by selecting the desired signal, which is in the feedback loop, for measuring. The step tracking goal is also involved in self-tunning and from its name is related to the step response. The purpose is to make the closed-loop step response match the desired one. This is achieved by setting a certain time constant and the desired overshoot to the values the user desire to achieve. The tracking goal includes following the step signal commands with a certain preset performance. The pole goal involves constraining the dynamics of the system by setting the region of the pole location. These self-tunning goals are further described in figures 18 19 20 and 21


Margins Goal

Name: MarginsGoal1

Purpose

Enforce specific gain and phase margins (disk margins for MIMO feedback loops).

Feedback Loop Selection

Measure margins at the following locations:

SelfbalselftunerMoo/Sum5/1

Add signal to list

Measure margins with the following additional loops open:

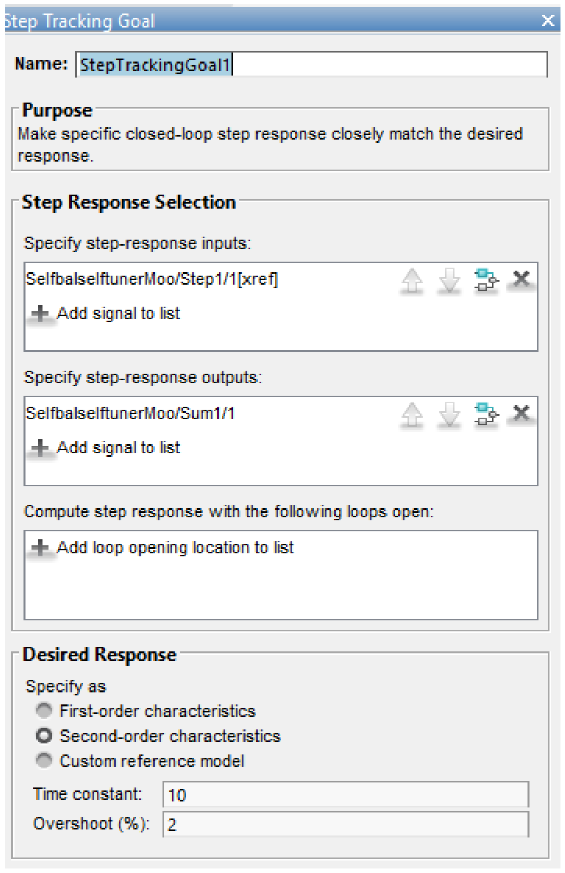
Add loop opening location to list

Desired Margins

Gain margin: 6 dB

Phase margin: 40 deg

Figure 17: Margin Goal


Step Tracking Goal

Name: StepTrackingGoal1

Purpose

Make specific closed-loop step response closely match the desired response.

Step Response Selection

Specify step-response inputs:

SelfbalselftunerMoo/Step1/1[xref]

Specify step-response outputs:

SelfbalselftunerMoo/Sum1/1

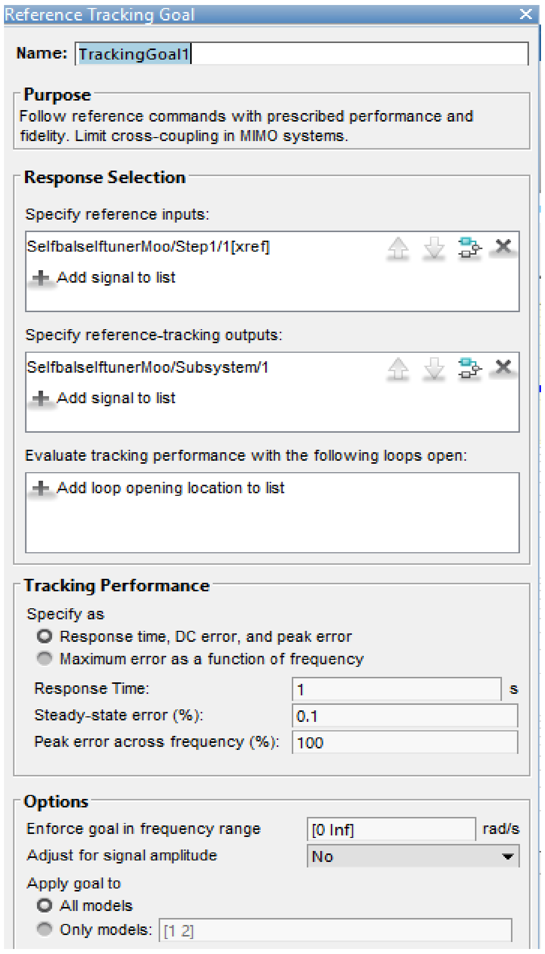
Compute step response with the following loops open:

Desired Response

Specify as
- First-order characteristics
- [✓ Second-order characteristics]
- Custom reference model

Time constant:	10
Overshoot (%):	2

Figure 18: Step Tracking Goal



Reference Tracking Goal

Name: TrackingGoal1

Purpose
Follow reference commands with prescribed performance and fidelity. Limit cross-coupling in MIMO systems.

Response Selection
Specify reference inputs:
SelfbalselftunerMoo/Step1/1[xref]
Add signal to list

Specify reference-tracking outputs:
SelfbalselftunerMoo/Subsystem/1
Add signal to list

Evaluate tracking performance with the following loops open:
Add loop opening location to list

Tracking Performance
Specify as
[✓ Response time, DC error, and peak error]
Maximum error as a function of frequency

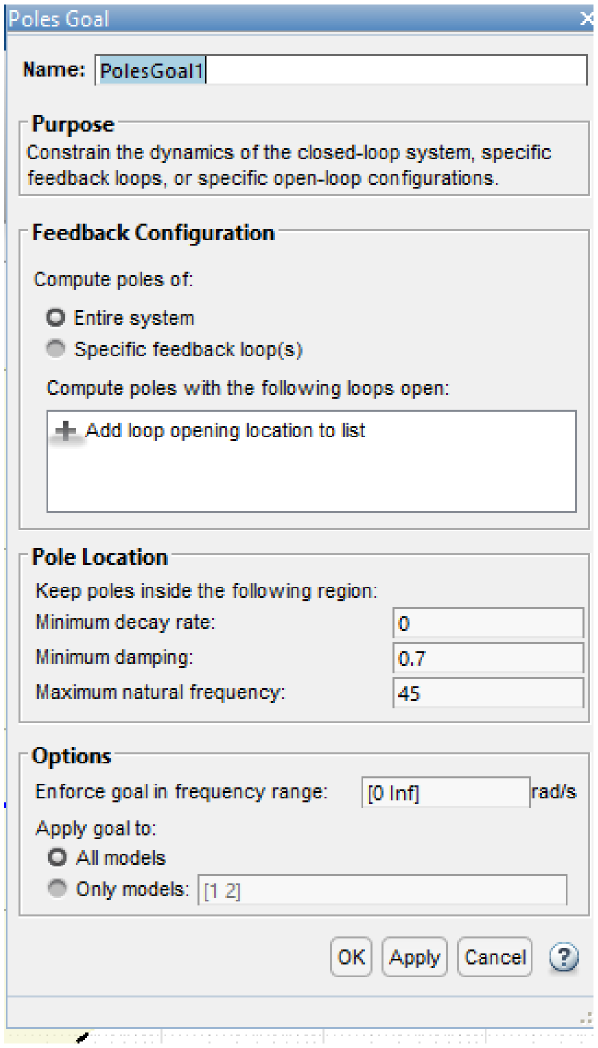
Response Time:	1	s
Steady-state error (%):	0.1	
Peak error across frequency (%):	100	

Options

Enforce goal in frequency range	[0 Inf]	rad/s
Adjust for signal amplitude	No	

Apply goal to
[✓ All models]
Only models: [1 2]

Figure 19: Tracking goal



Poles Goal

Name: PolesGoal1

Purpose

Constrain the dynamics of the closed-loop system, specific feedback loops, or specific open-loop configurations.

Feedback Configuration

Compute poles of:

[✓ Entire system]

Specific feedback loop(s)

Compute poles with the following loops open:

Add loop opening location to list

Pole Location

Keep poles inside the following region:

Minimum decay rate:	0
Minimum damping:	0.7
Maximum natural frequency:	45

Options

Enforce goal in frequency range: [0 Inf] rad/s

Apply goal to:

[✓ All models]

Only models: [1 2]

Figure 20: Poles Goal

The results are described in figures 22 and 23.

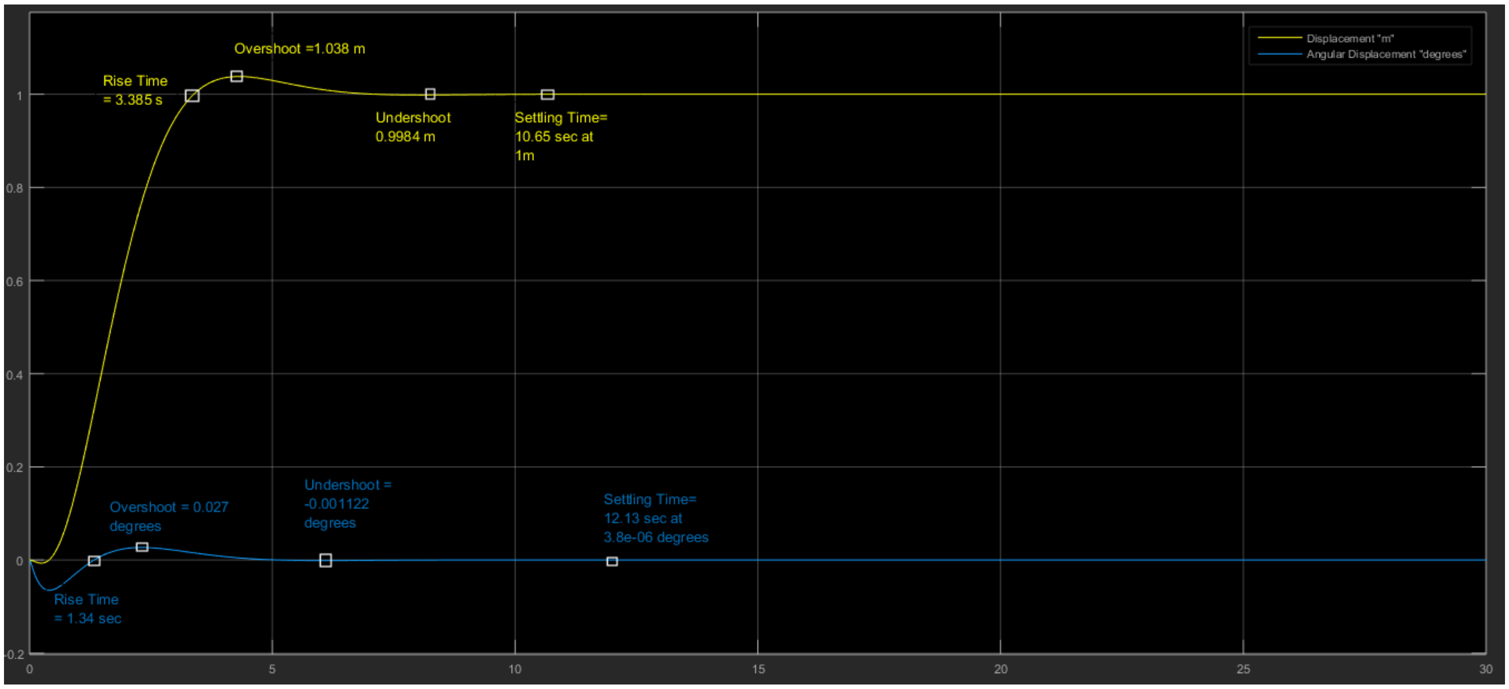


Figure 21: LQR: Displacement and tilting angle responses in meter and degrees, respectively, in function of time in seconds

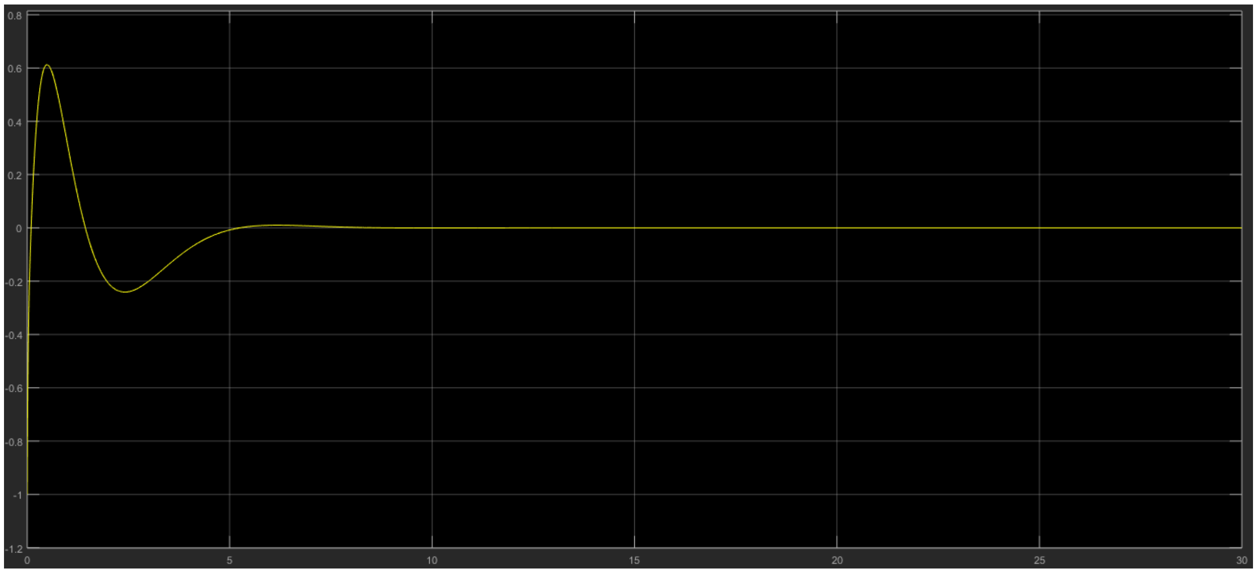


Figure 22 LQR: Control Effort with respect to time

3.5 Simulation: LQR with Minimum-Order Observer MOO

The LQR controller has shown a significant enhancement in performance in comparison with the PID controller. However, it requires additional sensors which can be computationally and cost-wise expensive to include in real life application or implementation on the real robot. Therefore, the LQR controller was modified to be an MOO and was tested on the model for performance assessment. The LQR with MOO will be further elaborated after setting the inverted pendulum model involved in this work.

3.5.1 Inverted Pendulum Model

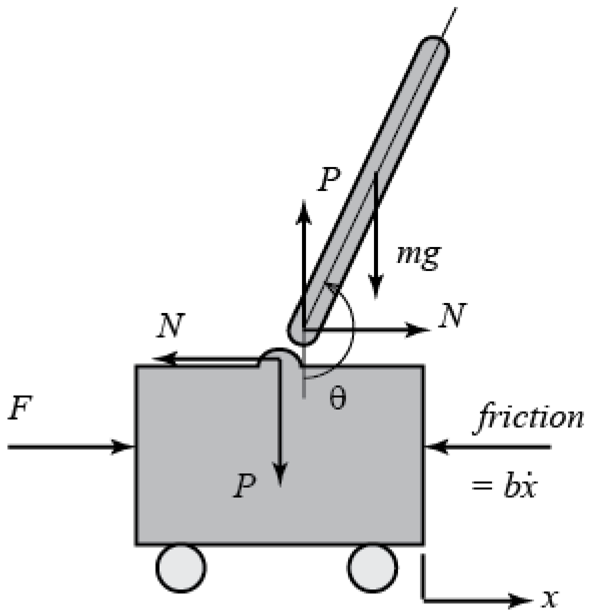


Figure 23: Inverted Pendulum Free Body Diagram [10]

Theoretically, the inverted pendulum consists of a mobile cart-rail system with a swaying pole connected as shown in figure 24. It is usually modeled as a rigid pole fastened to a rigid cart through a frictionless joint. At rest, the pole will fall, due to gravitational force. The non-linear model of the inverted pendulum is given by the MathWorks tutorial [10], where the equations are obtained by the Newton's second law method separately first on the cart's during horizontal direction, where:

$$\Sigma\vec{F} = M\vec{a} \tag{5}$$

$$M\ddot{x}+b\dot{x}+N=F \tag{6}$$

Adding the forces in the horizontal direction of the pendulum will yield following:

$$N = m\ddot{x}+ ml\,\ddot{\theta}\cos\theta - ml\,\dot{\theta}^2\sin\theta \tag{7}$$

Substituting (6) in (7) we get:

$$(M + m)\ddot{x} + b\dot{x} - ml\,\dot{\theta}^2 \sin\theta + ml\,\ddot{\theta}\cos\theta = F \tag{8}$$

For the equation of motion of the pendulum in the vertical direction, we get:

$$P\sin\theta + N\cos\theta - mg\sin\theta = ml\ddot{\theta} + m\ddot{x}\cos\theta \tag{9}$$

To remove the P and N terms in the above equation, we can add the moments of the pendulum's centroid:

$$-Pl\sin\theta - Nl\cos\theta = I\ddot{\theta} \tag{10}$$

Combining (9) and (10), we get the following

$$(I + ml^2)\,\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta \tag{11}$$

Where "I" is the mass moment of inertia of the pendulum; M and m are masses of the cart and pendulum respectively; b is friction's coefficient of the cart; l is the length to the pendulum's center of mass; F is the external force applied to the cart; x is the cart's position; theta ϴ, the pendulum angular distance from the negative y-axis [10].

These sets of equations need to be linearized, since the control techniques apply to linear systems only. Linearization will be done by setting the equilibrium position along the positive y-axis where $\theta = \pi$, if the system will remain in this equilibrium with no more than 20 degrees deviation [10]. Therefore, $\theta = \pi + \emptyset$, where $\emptyset$ equals 20 degrees. By assuming this small deviation, we can set the following approximations:

$$\cos\theta = \cos(\pi + \emptyset) \approx -1 \tag{12}$$

$$\sin\theta = \sin(\pi + \emptyset) \approx -\emptyset \tag{13}$$

$$\dot{\theta}^2 = \dot{\emptyset}^2 \approx 0 \tag{14}$$

By substituting the mentioned approximations, we obtained the following linear equations:

$$(I + ml^2)\,\ddot{\emptyset} - mgl\,\emptyset = ml\ddot{x} \tag{15}$$

$$(M + m)\ddot{x} + b\dot{x} - ml\,\ddot{\emptyset} = F \tag{16}$$

To get the linearized system equations' transfer functions, the equations 15 &16 must be subjected to Laplace transform assuming zero initial conditions [10]. Since transfer functions demonstrate the relationship between one single input single output, we will obtain after calculations [10], two transfer functions, where the input in both is the external force applied labeled as U, and the output is angular position labeled as ϕ, and the horizontal position labeled as X in the first (17) and second transfer functions (19) as shown below [10].

$$\frac{\phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(l+ml^2)}{q}s^2 - \frac{mgl(M+m)}{q}s - \frac{bmgl}{q}} [\frac{rad}{N}] (17)$$

Where:

$$q = [(M+m)(I+ml^2)-(ml)^2] \qquad (18)$$

the second transfer function is represented by the cart's position with respect to the external force:

$$\frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(l+ml^2)}{q}s^3 - \frac{mgl(M+m)}{q}s^2 - \frac{bmgl}{q}s} [\frac{m}{N}] (19)$$

The above linearized equations (17) and (19), can also be expressed in state space representation, where the system can be labelled as single input multiple output SIMO. Therefore, in in this representation, one can attempt to control both the pendulum's angle and the cart's position [10]:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\emptyset} \\ \ddot{\emptyset} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \emptyset \\ \dot{\emptyset} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{(I+ml^2)}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

(16)

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \emptyset \\ \dot{\emptyset} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

(20)

Robots total weight coefficient of friction b, length to chassis center of mass l and mass moment of inertia I of the chassis are given in Table 5. I, l and m were calculated by assuming that the chassis is presented as a rectangular prism as shown in figure 25. The dimensions and mass of this prism are calculate based on the prior values previously tabulated, were the mass, height, l and I are presented in the below equations, 21 22 23 and 24respectively:

m=mass(layers)+mass(1st set of rods)+mass(batteries)+mass(2nd set of rods)*2 (21)

h= thck (base)+L (1st set of rods)+thck (middle layer)+L (2nd set of rods)+thck (top layer) (22)

where thck is the thickness and L is the length

d and w are the length and the width of the top bottom or middle layer respectively.

$$l=h/2 \tag{23}$$

$$I=\frac{1}{12}m(h^2+d^2) \tag{24}$$

I was chosen according to our designated axis of rotation "x"

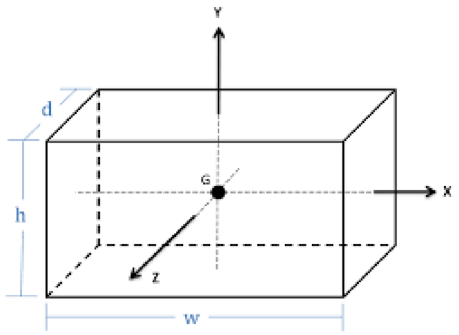


Figure 24: Rectangular prism mimicking the chassis [14]

Table 5: Robot's Characteristics

	Label	Value	Unit
Mass of the Cart	M	0.55	Kg
Mass of the Chassis	m	0.4696	Kg
Length to chassis center of mass	l	3.59	cm

Coefficient of friction of the cart	b	0.1	N/m/sec
Mass moment of inertia of the Chassis	I	0.00046487	Kg.m^2

3.5.2 LQR with MOO

LQR controller can be classified into a system including an observer, which will be discussed next, and into a system without observer represented in the above-mentioned theory.

LQR with observer or full-state feedback system is demonstrated in Figure 26.

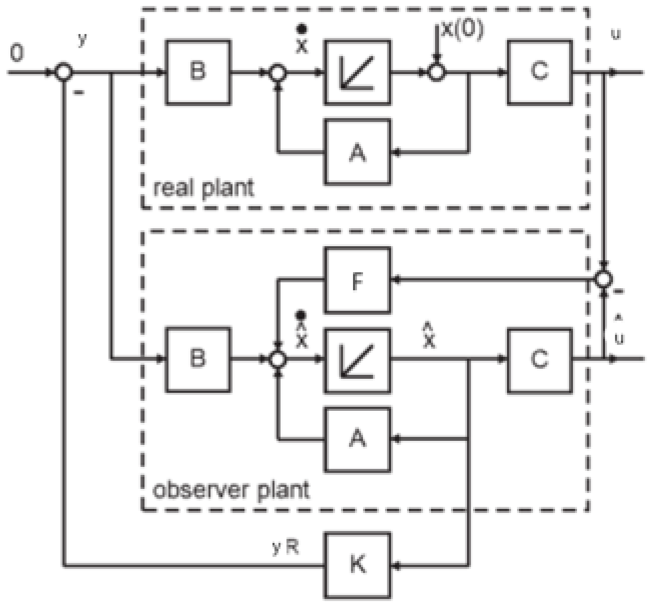


Figure 25: LQR (real plant) with Observer Design [12]

In the ideal LQR design it is assumed that all state variables are known to get the feedback. However, this is not the case in many real existing and non-ideal systems, either due to inability to ideally measure all state variables or due the sensors’ cost [12]. In systems including observer, it will be shown how all state variables are calculated using one or few measured states: this case is called Minimum-Order Observer [12]. State estimation can be calculated according to the below equation which can be acquired by looking at the observer plant in Figure 26 excluding till now the signal coming from F:

$$\dot{\hat{x}} = A\hat{x}+By \tag{25}$$

Where $\hat{x}$ is the estimated true state: *x*. This observer will properly operate if the initial condition *x (0)* is known, which will enable setting $\hat{x}(0)$ equal to it. In real cases the initial condition is rarely well known, which will lead the error expressed in (26) to converge too slowly to zero.

$$\tilde{x}=x-\hat{x} \tag{26}$$

Hence,

$$\dot{\tilde{x}}=A\tilde{x} \text{ or } \tilde{x}(0) = x(0)-\hat{x}(0) \tag{27}$$

The velocity of the error as it converges to zero is identical to the dynamics of A, but its convergence to zero velocity can never meet the true state. To achieve faster convergence, F feedback is introduced as seen in Figure 26, which corrects the model by an added signal using the multiplied or amplified state(s) error value. By looking now at Figure 26, we can infer the following equation:

$$\dot{\hat{x}} = A\hat{x}+By+F(u-C\hat{x}) \text{ Or } \dot{\hat{x}} = (A-FC)\hat{x}+By+Fu \tag{28}$$

$\dot{\hat{x}}$ will be known as "ethatilda", and A-FC will be known as the Â, B will be B̂. F calculation design or tunning will be done in the same manner as finding K, either by manual pole placement with coefficient comparison or with LQR method.

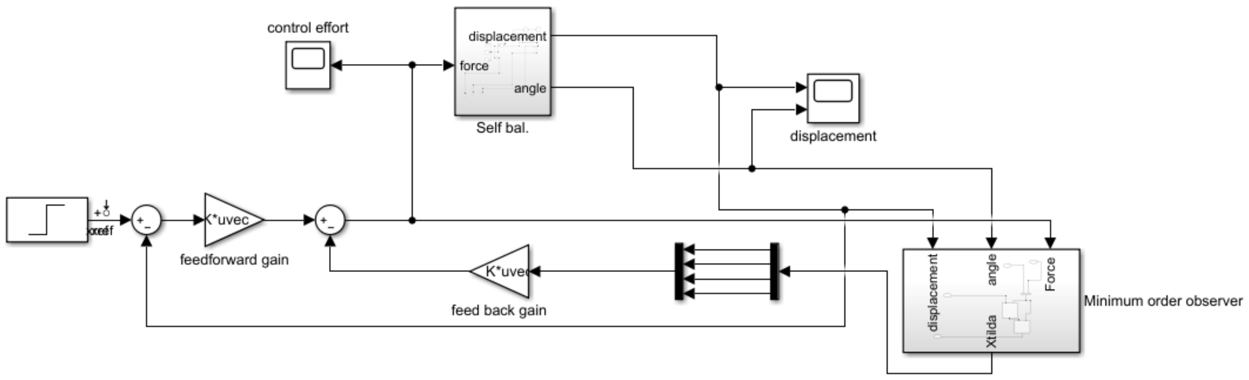


Figure 26: LQR MOO block diagram

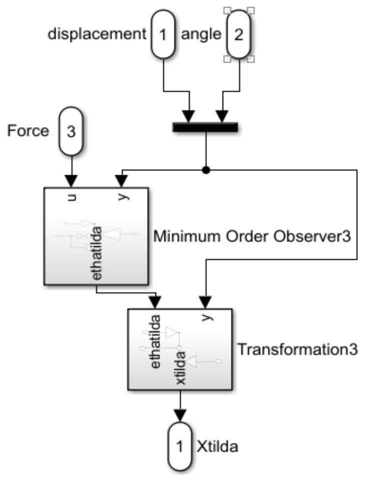


Figure 27: MOO block diagram

Transformation block is translated into:

$$\tilde{x} = \hat{C}\hat{x} + \hat{D}y \quad (29)$$

Figure 27 represents the LQR with MOO block diagram followed by the MOO subsystem, where its output, “xtilda” represents the estimated states. The performance of this controller was also governed by the same constraints and goals as in figures 18 19 20 and 21. The results of this controller are shown in figures 29 and 30.

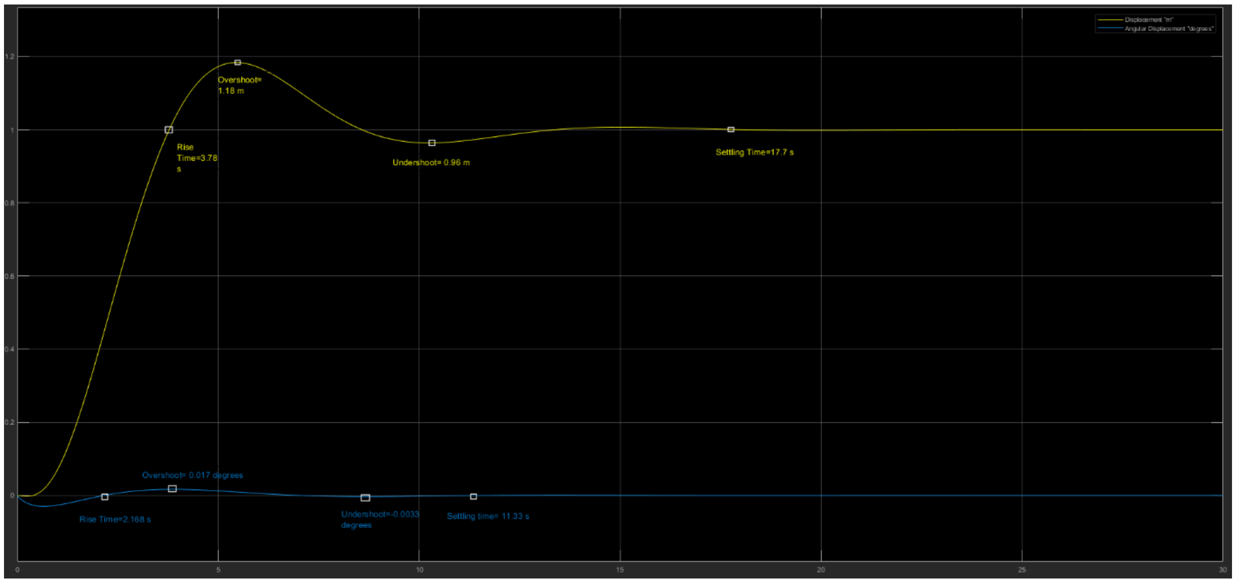


Figure 2928 :LQR MOO: Displacement and tilting angle responses in meter and degrees, respectively, in function of time in seconds

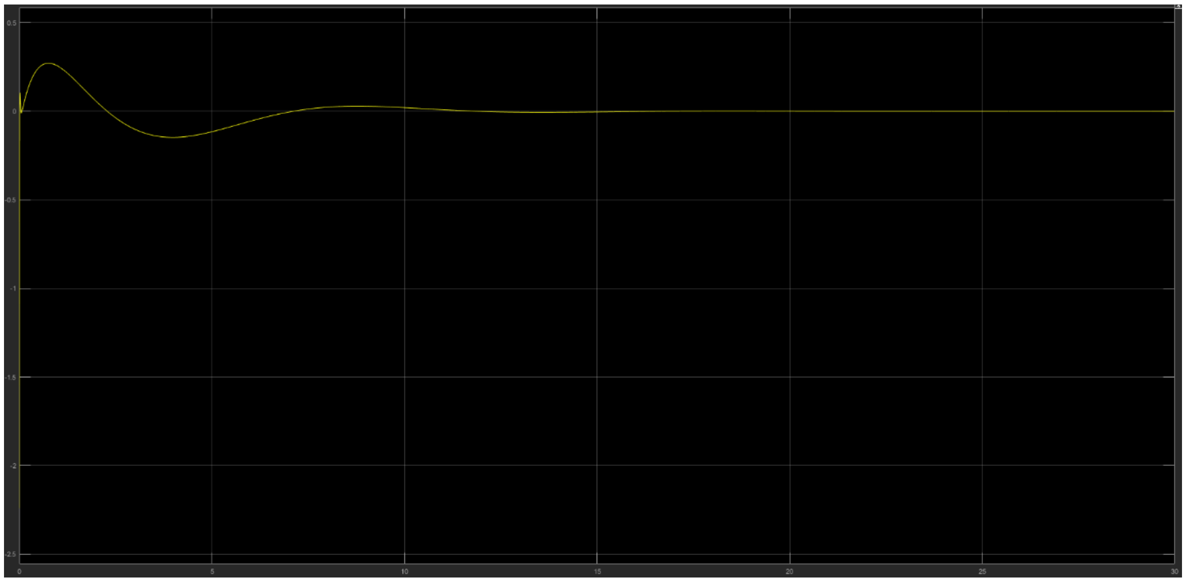


Figure30: LQR MOO: Control Effort with respect to time

CHAPTER 4

COMPARISON AND DISCUSSION

Comparison and discussion will be done between the 3 controllers in terms, of control metrics and performances for the displacement, angle, and control effort responses.

4.1 Displacement

First, Figure 31, shown the displacement response for each controller overlayed on the same graph, along with Table 6 to explicitly highlight the difference.

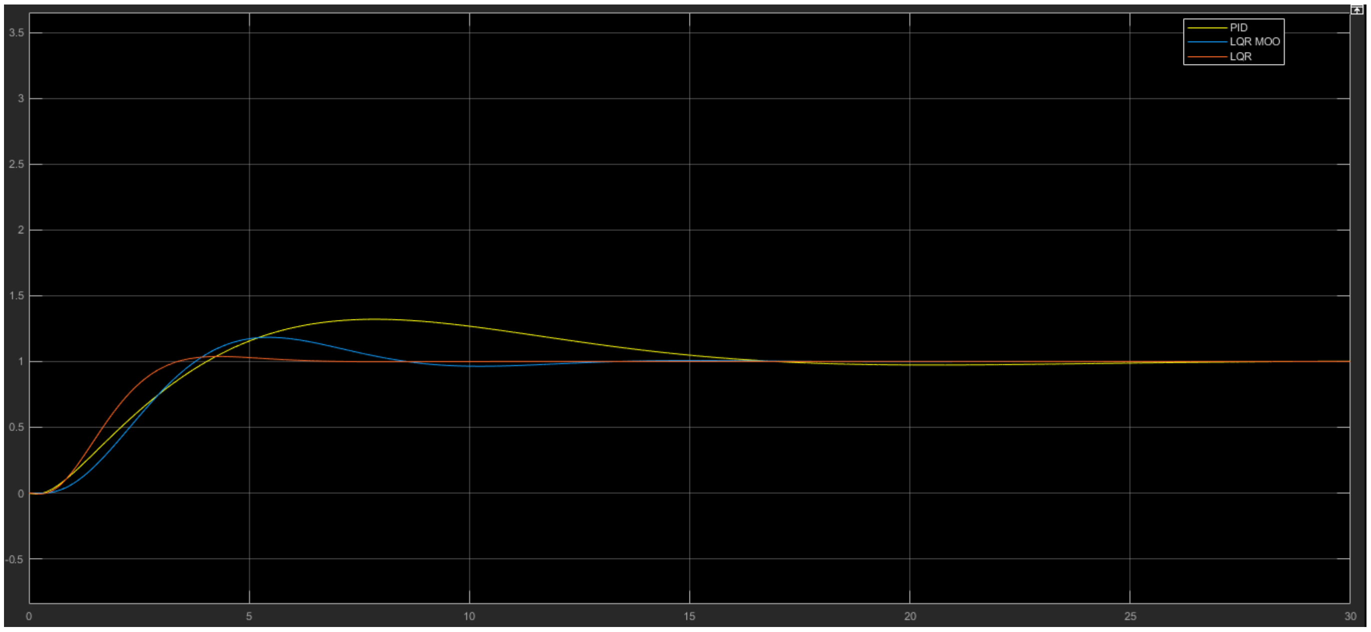


Figure 29: Displacement response with respect to time for PID (yellow), LQR (red) & LQR with MOO (blue)

Table 6: Displacement: Control metrics for PID LQR & LQR with MOO

Displacement	PID	LQR	LQR with MOO
Rise Time	**4.78 s**	**3.85 s**	**3.78 s**
Settling Time	**23.2 s**	**10.65 s**	**17.7 s**

Overshoot	**1.23 m**	**1.038 m**	**1.18 m**
Undershoot	**-**	**0.9984 m**	**0.96 m**
Steady State Error	**0.3 %**	**0 %**	**0.1%**

From figure 31, one can directly tell that the PID response was the slower first in terms of rise and settling times, in addition to the high overshoot which is visually clear. From the above snip alone, one cannot determine the steady state error however from figure 15 the steady state error was calculated and found to be 0.3 % as tabulated. There was no significant undershoot in the PID response. On the other hand, LQR response was faster in terms of settling time with no steady state error and very low overshoot and undershoot, however it may have taken a slightly more rise time than LQR with MOO. The shortest rise time in LQR with MOO was followed by a higher overshoot and under shoot and a longer settling time with a higher steady state error when compared to LQR alone. However, when compared to PID the displacement response in LQR with MOO was more stable.

4.2 Angle

Figure 32 shows the tilting angle response for the 3 controllers also recorded from the same scope, along with the control metric as in Table 7.

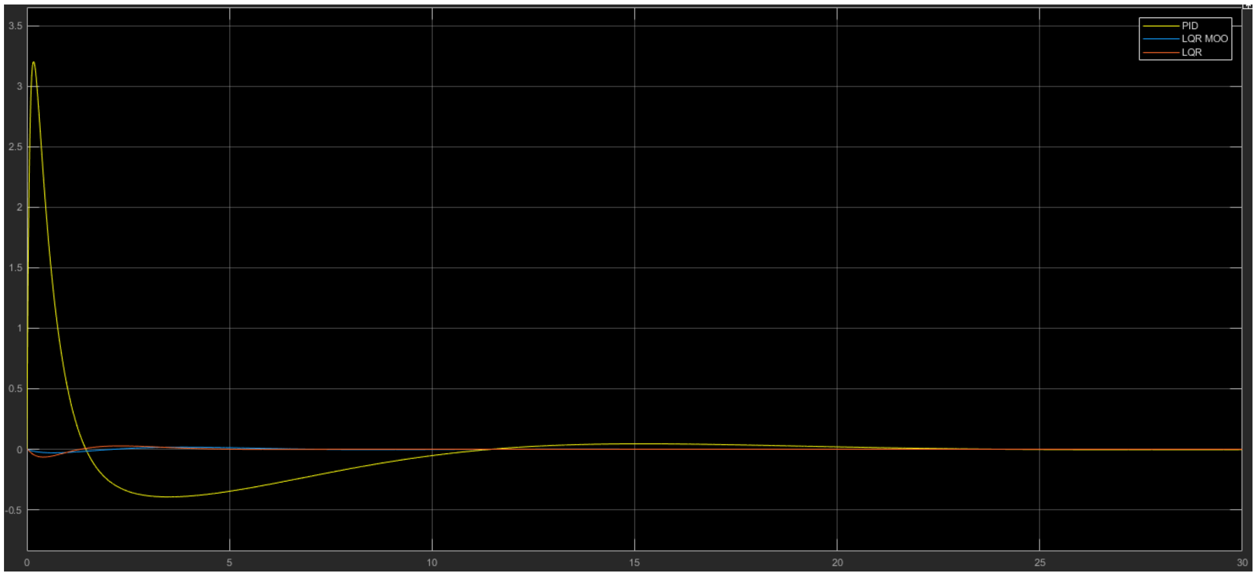


Figure 30: Angular response with respect to time for PID (yellow), LQR (red) & LQR with MOO (blue)

Table 7 Angle: Control metrics for PID LQR & LQR with MOO

Angular Displacement	PID	LQR	LQR with MOO
Rise Time	0	1.34 s	2.168 s
Settling Time	15.8 s	12.13 s	11.33 s
Overshoot	3.24 degrees	0.027 degrees	0.017 degrees
Undershoot	-0.78 degrees	-0.001122 degrees	-0.0033
Steady State Error	0.15 %	≈0 %	≈0 %

Visually, PID angular response was the poorest, due to the significant very high overshoot and undershoot, and the longest settling time. Comparing between the 2 versions of LQR, the performance was somehow close. The LQR rise time and undershoot were less than LQR with MOO, however, the latter took a slightly less time to settle and made a slightly less overshoot.

4.3 Control Effort

The control effort is the reflection of the above-mentioned performances which will also be presented in an overlay of control effort responses as in figure 33, and in Table 8 which recorded the control effort range between 0 and 0.2 seconds.

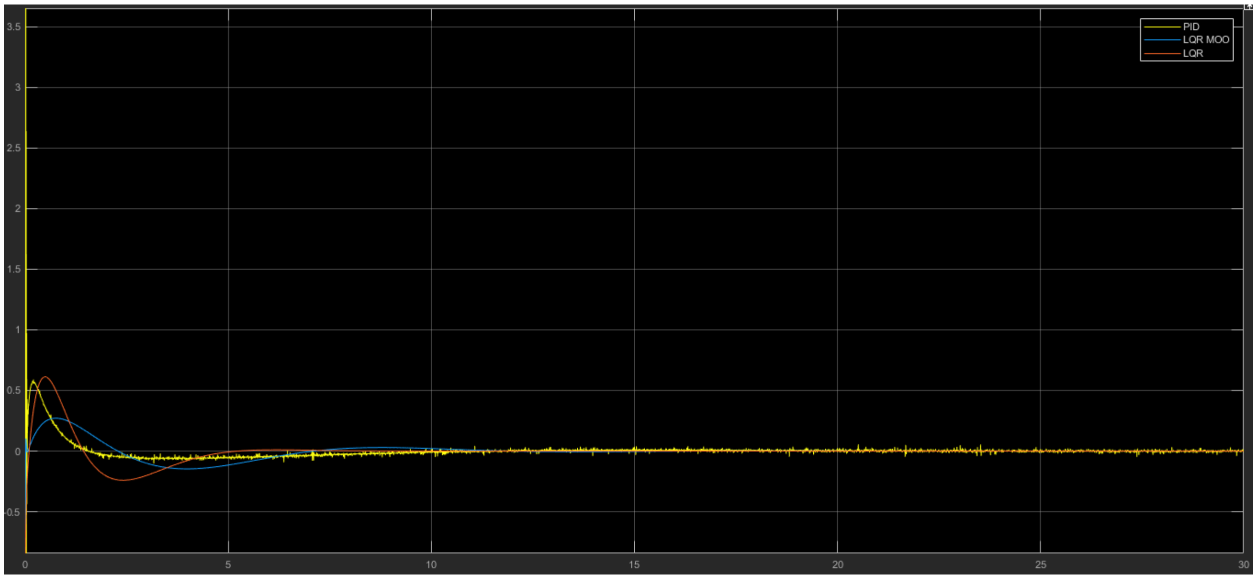


Figure 31: Control effort with respect to time for PID (yellow), LQR (red) & LQR with MOO (blue)

Table 8: Control Effort for PID LQR & LQR with MOO

	PID	LQR	LQR with MOO
Control Effort	[2.66, -9.3]	[0.6, -1]	[0.25, -2.25]

The PID high angular overshoot was corrected by applying a higher control effort as shown in Figure 33 and in table 8, the response remained fluctuating which translates an unstable system. The LQR controller very good performance was reflected by having the shortest dynamic range the LQR with MOO's dynamic range was longer than LQR, this is due to the corrections that were needed to in this controller to stabilize the system at the start.

As shown in figure 33, the control effort for the LQR stabilized before that of PID and MOO. PID’s control effort remained fluctuating, and MOO took the longest time in stabilizing the system (without fluctuation).

CHAPTER 5

CONCLUSION AND FUTURE PERSPECTIVES

4.1 Conclusion

After exhibiting the results of the PID and the LQR controllers on the OSOYOO 2WD self-balancing robot, we further simulated the model on the LQR MOO controller which requires a smaller number of states and is less expensive. The results of these 3 models showed that the LQR yielded the most promising results due to the access on all states and the presence of the feed forward and the feedback controls which further reduced the error, also the application of goals and constraints via the self-tuner played an important role. One drawback of the LQR controller is its high cost, due to the need of placing a lot of sensors to access all states. LQR with MOO solved the cost issue however it didn’t outperform LQR since not all states are measured and therefore the estimation of the rest will not yield same performance quality. This is shown in the higher control effort that was need in LQR MOO, and the longer displacement settling time. This will lead us to a conclusion that LQR outperforms PID, and that LQR controller was the best in stabilizing the system in the least amount of time and with the least control effort among LQR with MOO and PID. The greater conclusion is that one cannot obtain an optimal controller that best performs with the least effort and cost. User must compromise based on his application and choose what to penalize: performance cost or effort.

4.2 Future Perspectives

For future work, real implementation of the documented controllers will be done on the real robot, and result comparison like the one done in this project will be done and both results empirical and real will be compared.

APPENDIX

Non-Observer Matlab Code

```
M = 0.55;
m = 0.29;
b = 0.1;
I = 0.00267;
g = 9.8;
l = 0.053;

p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices
%http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&secti
on=ControlStateSpace
A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);

poles = eig(A)

co = ctrb(sys_ss);
controllability = rank(co)

Q = C'*C;
 Q(1,1) = 1;
 Q(3,3) = 1;
R = 1;
K = lqr(A,B,Q,R);
```

Minimum-Order Observer Matlab Code

```
M = 0.55;
m = 0.29;
b = 0.1;
I = 0.00267;
g = 9.8;
l = 0.053;

p = I*(M+m)+M*m*l^2; %denominator for the A and B matrices
%http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&secti
on=ControlStateSpace
A = [0      1              0           0;
     0 -(I+m*l^2)*b/p  (m^2*g*l^2)/p   0;
     0      0              0           1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (I+m*l^2)/p;
          0;
        m*l/p];
C = [1 0 0 0;
     0 0 1 0];
D = [0;
     0];

states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);

poles = eig(A)
co = ctrb(sys_ss);
controllability = rank(co)
ob = obsv(sys_ss);
observability = rank(ob)
Q = C'*C;
 Q(1,1) = 1;
 Q(3,3) = 1;
R = 1;
K = lqr(A,B,Q,R);
Cnew=[C(:,1) C(:,3) C(:,2) C(:,4)];

Bnew=[B(1,:);B(3,:);B(2,:);B(4,:)];

 Anewi=[A(1,:);A(3,:);A(2,:);A(4,:)];
 Anewf=[Anewi(:,1) Anewi(:,3) Anewi(:,2) Anewi(:,4)];
 Knew=[K(:,1) K(:,3) K(:,2) K(:,4)];

 Jomin=[-40 -40];


Aaa=Anewf(1:2,1:2);
Aab=Anewf(1:2,3:4);
Aba=Anewf(3:4,1:2);
Abb=Anewf(3:4,3:4);
Ba=Bnew(1:2,:);
Bb=Bnew(3:4,:);

Ka=Knew(:,1:2);
Kb=Knew(:,3:4);
%check minimum observability
rank(obsv(Abb,Aab));

Lmin=place(Abb',Aab',Jomin)'


Ahat=Abb-Lmin*Aab;
Bhat=(Ahat*Lmin)+(Aba-Lmin*Aaa);
%note: in simulink what implemented is:(Aba-Lmin*Aaa);

Fhat=Bb-Lmin*Ba;

Chat=[zeros(2,2);eye(2)];
Dhat=[eye(2);Lmin];


Atild=Ahat-Fhat*Kb;
eig(Atild)
eig(A-B*K)
eig(Abb'-Aab'*Lmin')
```

REFERENCES

[1] J. Fang, "The LQR Controller Design of Two-Wheeled Self-Balancing Robot Based on the Particle Swarm Optimization Algorithm," *Hindawi Publishing Corporation Mathematical Problems in Engineering,* vol. 2014, pp. 1-7, 2014.

[2] F. R. Jiménez, I. A. Ruge and A. F. Jiménez, "Modeling and Control of a Two Wheeled Auto-Balancing Robot: a didactic platform for control engineering education," in *18th LACCEI International Multi-Conference for Engineering, Education, and Technology: "Engineering, Integration, and Alliances for a Sustainable Development" "Hemispheric Cooperation for Competitiveness and Prosperity on a Knowledge-Based Economy".*, 2020.

[3] W. An and Y. Li, "Simulation and Control of a Two-wheeled Self-balancing Robot," in *International Conference on Robotics and Biomimetics (ROBIO)*, Shenzhen, 2013.

[4] C. C. SAMAK and T. V. SAMAK, "DESIGN OF A TWO-WHEEL SELF-BALANCING ROBOT WITH THE IMPLEMENTATION OF A NOVEL STATE FEEDBACK FOR PID CONTROLLER USING ON-BOARD STATE ESTIMATION ALGORITHM," *International Journal of Robotics Research and Development (IJRRD)* , vol. 8, no. 2, pp. 1-11, 2018.

[5] A. A. Bature1, S. Buyamin, M. N. Ahmad and M. Muhammad, "A Comparison of Controllers for Balancing Two Wheeled Inverted Pendulum Robot," *International Journal of Mechanical & Mechatronics Engineering IJMME-IJENS,* vol. 14, no. 3, pp. 1-7, 2014.

[6] M. M. Rahman , S. H. Rashid, K. R. Hassan and M. Hossain, "Comparison of Different Control Theories on a Two Wheeled Self Balancing Robot," in *AIP Conference Proceedings 1980, 060005*, 2018.

[7] L. M. M. Thwin and Y. Chan, "APPLICATION OF LQR CONTROL FOR TWO-WHEEL SELF-BALANCING ROBOT," *J. Myanmar Acad. Arts Sci,* vol. XVIII, no. 2C, pp. 263-272, 2020.

[8] S. Patil, D. Bhavsar, S. Mantri and S. Gosavi, "Self Balancing Robot," *International Journal of Engineering Research & Technology (IJERT),* vol. 9, no. 3, pp. 249-252, 2021.

[9] H. HANNA and S. HENRIK, Two-Wheeled Self-Balancing Robot, Stockholm, 2015.

[10] https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling, "Inverted Pendulum: System Modeling," Matlab.

[11] https://www.omega.co.uk/prodinfo/how-does-a-pid-controller-work.html.

[12] R. B¨uchi, State Space Control, LQR and Observer step by step introduction, with Matlab examples, Norderstedt: Herstellung und Verlag: Books on Demand GmbH, 2010.

[13] https://osoyoo.com/2018/08/01/introduction-of-osoyoo-2wd-balancing-car-robot/.

[14] J. Moore, M. Chatsaz, A. d'Entremont, J. Kowalski and D. Miller, "Center of Mass and Mass Moments of Inertia for Homogeneous Bodies".