Balancing ball on a plate-Final Report

Thesis · July 2020

DOI: 10.13140/RG.2.2.26127.00161

1 author:

Ehsan Lakzaei
Eindhoven University of Technology
1 PUBLICATION **0** CITATIONS

SEE PROFILE

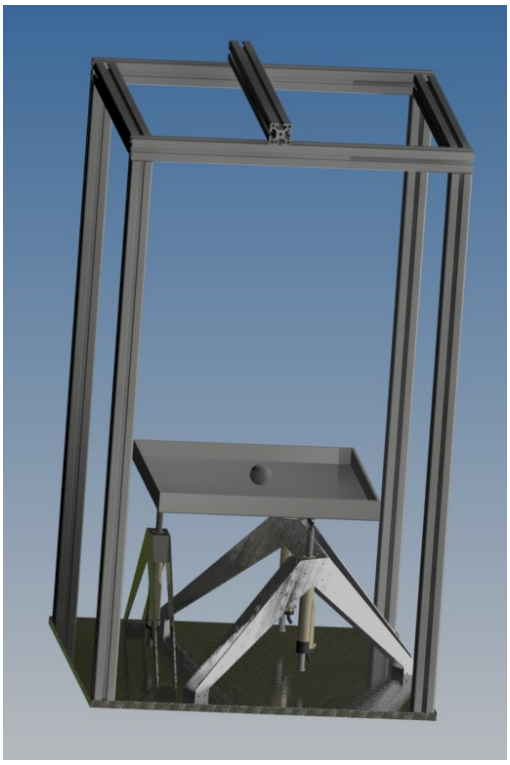
Technische Universiteit Eindhoven University of Technology

Fontys University of Applied Sciences

Balancing ball on a plate

Final Report

Ehsan Lakzaei



Supervisors:
Ing. W.H.A.Hendrix (TU/e)
Dr.Ir. M.C.F Donkers (TU/e)
Mr. J.Lazaroms (Fontys U.o.A.S)

Eindhoven, July 2020

Version.No.	Description	Date
0.0	Added:Chapter one and two	06.02.2020
0.1	Added:Chapter two and three	28.02.2020
0.2	Added:Chapter three and four	15.04.2020
0.3	Added:Chapters four is finished	21.05.2020
0.4	Added: Chapters five and six	31.07.2020
1.0	Finalizing	1.08.2020
1.1	Final draft	2.08.2020

Table 1: Document History

Abbreviation	Definition
TU/e	Technical University of Eindhoven
U.o.A.S	University of Applied Science
CS	Control Systems
EPC	Equipment and Prototype Center
FPS	Frame Per Second
BW	Band-Width
DS1104	An interface (board) to Dspace platform
Mb/s	Mega bits per second
FPGA	Field-Programmable Gate Array
I/O	Input / Output
DoF	Degree of Freedom
R.T.I	Real-Time Interface
W.r.t	With respect to
LDM	Linear DC motor
TF	Transfer Function
S-domain	refers to frequency domain
Z-domain	refers to discrete domain
ZSE	Zero Steady state Error
Hz	Hertz
FIFO	First In First Out

Table 2: Abbreviation

Abstract

This report represents a realization of specifications, Modelling, designing a controller, validation of model, implementing and testing of real-time tracking control for a ball and plate system. The position of the ball is measured with a camera. The image processing algorithm is done by the Raspberry Pi board. dSPACSE board will be the main station to analyze the incoming data and applying it to the system. A detailed dynamic model is derived for the simulation study. Although, This model is neglected the high-order coupling terms to obtain the system as simple as possible, it stays accurate all the time. Simplification and linearization of the model makes this system divided into two parts, since it acts the same upon references, with respect to X and Y axes.The controller is designed firstly in continuous time (Frequency domain). Then, two cases will be compared, namely converting the controller directly to discrete domain and designing the whole system in discrete domain. The validity and precision of the system is then investigated through simulation and experimental studies. The results of experiments illustrate that the designed system works reasonably and they are in line with simulation results.

Acknowledgment

I would like to thank the following people, without whom I would not have been able to complete this project, and without whom I would not have made it through my internship period!
The Control Systems Team at Technische Universiteit, Eindhoven, especially to my supervisor Ing. W.H.A.Hendrix , whose insight and knowledge into the subject matter steered me through this project.
And special thanks to Eng. J.Lazaroms from Fontys University of Applied Science, whose support as part of his supervisor role, allowed my studies to go the extra mile (sorry for all the questions, day and night, Joachim!).
The Equipment And Prototyping Center team at Technische Universiteit, Eindhoven, especially to Eng. E.Dekkers and Eng. P.W.J.H. de Laat for designing the mechanical part and their ideas to make this project happens.
I would like to say a special thank you to my friend, J. Kusters for His support, guidance and overall insights in the field of AI that have made this an inspiring experience for me.
And my biggest thanks to my family (Mieke, Jahangir, Amir, Kimberly, Mehdi) for all the support you have shown me through this tough period.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this report, tracking of a ball on a plate with validation and testing will be illustrated. First, research is conducted to find the best solution for designing such system. Each main components is compared to other alternatives and the one with reasonable specifications is chosen. Next, a detailed model is demonstrated, where this model illustrates the non-linearity of the system. By neglecting high-order terms the model is simplified and decoupled into two separate system with respect to X-Y axes. Moreover, a model for actuators (as the inner-loop controller), which was given from the client, is realized and analyzed. Further, a controller for the outer-loop is designed in frequency domain, based on the behavior of the model with using SISOTOOL application in MATLAB. After validating the controller, two cases is discussed in detail to approach a design in discrete domain. Acting upon various references is demonstrated in SIMULINK as the next step. Then, the transformation of positions of the actuators to the angles and vice versa is in detail explained. Afterwards, The model is investigated in practice and the results were compared to the ones from simulations. After that all parts separately was realized, designed and tested, the total system is implemented for the test procedure in dSPACE platform. Three types of tests are performed, at which, the results of the first controller design has shown that it is not working properly and this led the project to design a new outer-loop controller. Moreover, the actuators are analyzed under these total tests Lastly, a conclusion is made based on the outcomes and problems that this project was faced on and some future works is recommended.

Chapter 2

System design and specifications

This chapter will present the selection of a specific position sensor after having a comparison between some alternatives. Afterwards the specifications of the mechanical design will be explained in detail and at last, the control platform will be described.

2.1 Position Sensor

The position sensor plays one of the most important roles in this project since the position of the ball must be known at any time to be able to control the system and steer the ball to the desired position. At the end, a camera has been elected to be the interface between the mechanical design and control system. The reasons will be explained through the following sections.

2.1.1 Sensor Alternatives

There are generally two alternatives for the position sensor, namely:

- Sensitive Touchscreen
- Camera

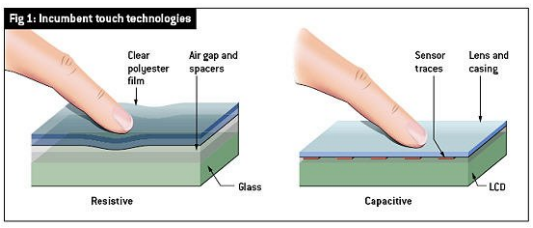


Figure 2.1: Sensitive Touchscreens

Figure 2.2: Camera

Following sections will study both alternatives and then based on advantages and disadvantages, one option will be chosen.

Sensitive Touchscreen

These sensors seem to be popular, especially in smartphones world and they are divided into two types, namely: Resistive Touchscreens and Capacitive Touchscreens as shown in figure 2.1. An example of Resistive Touchscreen is the bottom touchscreen in 3DS Nintendo (gaming apparatus). Capacitive touchscreen on the other hand can be found on almost every new smartphone.

Resistive Touchscreen

It's made by plastic and it bends in when it is touched. it uses a number of layers and works by using the amount of pressure applied to the screen. Input will be registered when the second layer is pushed by the first layer due to pressing the screen. It is sensitive to all types of objects such as pen, nail, fingers and etc.

Capacitive Touchscreen

This type of screens works by using the conductive properties of an object. Usually, these sensors are made of glass and they do not rely on pressure. This makes it more responsive to swiping and pinching. It is operating only by fingers.

Based on properties of these two sensors, Resistive touchscreen has the leverage of interfacing with different objects. Therefore this touchscreen will be further studied and compared to the other option, which is camera.

Camera

This sensor is able to track the ball with limiting properties such as latency, frames per second (fps). Moreover, processing images is also needed for detection of the ball. Although these limits impose some constraints for design of the controllers, overhead cameras could fulfil the necessary requirements.

Resistive Touchscreen vs. Camera

First option is used in [1] and [2]. Second option is chosen in [3], [4] and [5]. Both devices suffer from certain limits that they have in their properties. Table 2.1 shows the most significant factors for comparing these two options. Afterwards, each factor will be explained for both devices and lastly, one option will be chosen.

Case	Resistive Touchscreen	Camera
Cost	–	+
Availability	– –	++
False detection	+–	+–
Operating speed	+	–
Mechanical design	+	–
Instructive concept	–	+

Table 2.1: Resistive Touchscreen vs. Camera

*__Note__: ++ indicates good valid option
+ indicates acceptable option with some sidenotes
+– indicates mediocre option
– – indicates not a valid option

For a 40x40 cm^2 resistive touchscreen, there are not many suppliers. Not only the prices vary in a large range from 50 euro up to 130 euro, but also the size is not precise (mostly 40x30 cm^2 which is the closest to the desired size). So, to order a screen with specific size, it should be customized by the factory and this leads to a higher price. Delivery time is also a disadvantage for this option since it takes in the best cases almost one month. On the contrary, camera has a lower cost and it can be easily found and ordered. Also, delivery time is within 2 days. Therefore, camera wins the argument when it comes to cost and availability.
False detection can occur using both devices and this is more noticeable when plate is moving with an angle. Since resistive touchscreens detect the ball by measuring amount of pressure, ball's mass becomes a critical aspect. considering the plate's angle when it is moving, pressure decreases due to speed of the ball and hence, false data will be passed to the system. In case of having a camera, the ball will be detected by processing images which uses pixels of the background image. When plate is tilting, size of background's pixels will change and therefore, false detection may happen. Thus, neither of alternatives will have an upper hand in this case.
Next important factor is operating speed. First option has a higher operating speed since it is only needed to sense the amount of pressure and after some conversion the location of the ball will be passed to the system whereas camera is limited to sample rate (30 Hz for a common camera) and latency while image processing, which causes adding extra constraints to requirements. Hence, touchscreen is a better choice in terms of speed.
Choosing the first alternative will make the mechanical design simpler since it does not need an extra design while a frame must be designed to mount the camera.
Lastly, using a camera leads to learn a new concept for the author which is image processing whereas working with the touchscreen leads to programming in C/C++ language which it is already experienced in other projects.
All in all, based on table 2.1 and arguments above, camera is chosen for this project.

2.1.2 Camera

In this section, camera and its specification will be realized. One important part of Hardware design is selecting a camera with suitable specification that satisfies the requirement. The role of camera is quite important since it should track the location of the ball and transfer the data to the controller. Four major specifications of the camera are :

- FPS : frame per second
- Resolution : the amount of detail that the camera can capture
- Latency : the time it takes between each frame to be updated
- Band Width : the rate of transferring data to the Host

Conducted research revealed that in most cases, fps and resolution are given in datasheet but latency and band width are not. So, these requirements will be discussed in the following sections.

Latency

Latency is a process in which, capturing of the frame is going to happen in the first sampling time, then it is processed and afterwards data is sent to a controller block to take a proper action. It should be noted that the required time for each of these steps is one fps. Figure 2.3 demonstrates this process.

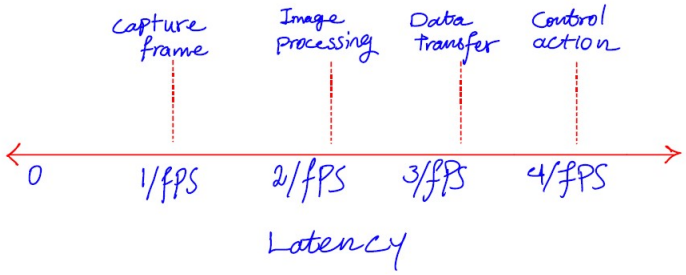


Figure 2.3: Latency Camera

From figure 2.3, the higher sample rate, the lower latency can be concluded. The exact amount of improvement percentage is observed and calculated in [6]: ”By increasing the frame rate, there will be an improvement to latency (e.g. from 30 FPS to 60 FPS) by 14% ”. This percentage shows that Latency is important but not remarkably considerable. So, a middle class camera will satisfy this requirement.

Band Width

By considering that the camera is running in the highest resolution, frame rate and colour depth available. If the camera has a resolution of 1280x720 pixels and FPS = 30 with a colour format of RGB8, then: A single frame requires :

$$\text{frame} = 1280 * 720 * 3 = 2764800 \text{ B}$$

The total required Band Width = 2764800 * 30 * 8 = 663552000 b/s $\rightarrow$ 663.5 Mb/s

A middle class computer will easily satisfy this requirement.

***Note**: Calculation above is based on [7].

2.1.3 Camera Selection

In this section a comparison between four middle class cameras will be done. afterwards a decision is going to be made based on table 2.2.

Specs	**Logitech C310**	**Microsoft LifeCam**	**Renkforce RF-WC-1080p**	**Jeway 5152**
Video Resolution	1280x720	1280x720	1920x1080	640x480
FPS	30	30	30	30
Required Band Width (Mb/s)	~ 650	$\sim$ 650	$\sim$ 1400	$\sim$ 220
Interface	USB 2.0	USB	USB	USB
Sensor type	CMOS	CMOS	OV2735	Not Mentioned
Price (Euro)	60	35	33	30

Table 2.2: Comparison table of four Cameras

Conclusion

According to table 2.2, despite the fact that Renkforce RF-WC-1080p requires a higher bandwidth, it will be the best option, considering a significant difference in terms of FPS with other alternatives and also having a reasonable price among other options. Its bandwidth issue is negligible since almost all computers nowadays can easily handle this requirement.

2.1.4 Image Processing Platform

There are various alternatives in market for image processing purpose, but Raspberry pi, Arduino and FPGA boards tend to be the most common boards. This section will compare these options and the conclusion will be drawn based on comparison arguments.

Raspberry pi vs Arduino

Table 2.3 will cover almost all significant differences between these two options.

No	**Raspberry Pi**	**Arduino**
1	A mini computer with Raspbian OS which runs multiple programs at a time	A micro-controller that is a part of a computer which runs one program at a time
2	Requires complex tasks such as installing libraries and software for interfacing electronics components	Very simple to interface sensors and other electronic components
3	Expensive	available at low cost
4	Ethernet port and USB Wi-Fi dongles make it easy to connect to internet	Needs external hardware
5	Four USB ports which makes it easier to connect different devices	One USB port
6	Various language programming platforms can be used, such as: C,C++,Python and etc	C/C++

Table 2.3: Raspberry Pi vs Arduino

Raspberry pi vs FPGA

FPGA boards can handle parallel tasks easily since each independent processing task is assigned to a dedicated section of the chip and can not be interrupted by other logic blocks. These type of boards will offer various features such as multiple I/O ports, high speed in processing data, highly structured methods to manoeuvre different projects. However, there are some drawbacks that leads to not use such boards, for instance:

- FPGAs are expensive in comparison to Raspberry Pi
- Working with these boards needs a detailed and precise plan to define a project in sub-blocks, and then each sub-parts must be implemented separately which it takes a lot of time to do so. Hence, FPGAs are more complex to work with than Raspberry Pi

Conclusion

Although in table 2.3, it is mentioned that Arduinos are available in lower cost, it should be noted that the proper board to use in this project could be Arduino Mega which does not have significant difference with Raspberry Pi in terms of cost. Next, running multiple programs at a time is important in case of integration all programs on the board which leads to using Raspberry Pi. Lastly, interface features and language programming platforms are other advantages of Raspberry Pi. Hence, this board is going to be used as a platform for image processing.

Raspberry Pi

Raspberry Pi 3 model B+ is the chosen model, since it was available by author ad it satisfies all requirements mentioned above. Specifications of chosen Raspberry Pi can be found in table 2.4.

Specs	Details
SoC	Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
GPU	Broadcom Videocore-IV
RAM	1GB LPDDR2 SDRAM
Networking	Gigabit Ethernet (via USB channel) 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
Bluetooth	Bluetooth 4.2, Bluetooth Low Energy (BLE)
GPIO	40-pin GPIO header, populated
Ports	HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI) Display Serial Interface (DSI)

Table 2.4: Raspberry Pi 3 Model B+ Specifications

Image processing will be done by using this board. To do so, there are some libraries which are necessary packages to be imported, namely:

- numpy : nummerical computation in Python, which contains a n-dimensional array
- argparse : Used to obtain command line arguments into program

- cv2 : OpenCV-Python library
- imutils : collection of OpenCV functions
- time
- deque : a list-like data to maintain a list of past locations of the ball N(x,y)
- VideoStream : used for live video streaming

It should be noticed that the focus on this project will not be on the codes for detecting the ball (which can be found in Appendix) but merely the way that the positions are sent to dSpace platform (serial communication).

2.2 Mechanical Construction

In this section, mechanical design will be presented. Each part will be in detail explained w.r.t their specifications and requirements for this project.

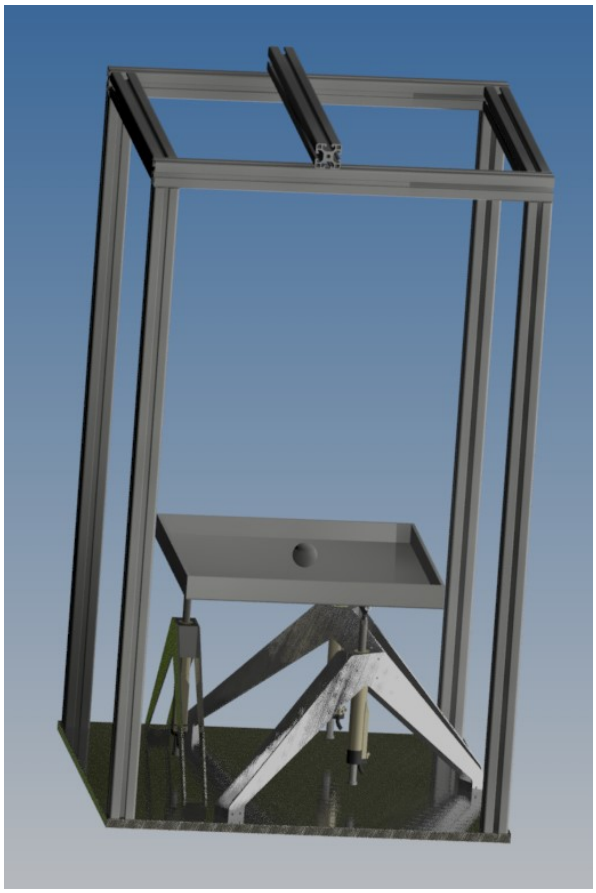


Figure 2.4: Total Design

2.2.1 Plate Actuation

Alternative plate actuation solutions of this project are proposed in this section. It should be noted that this report will not explain every possible model, but merely shows the differences and based on some desire prospective goals, one design will be chosen.
Generally, there are two different popular alternatives for this project, in particular:

- Table with pivot points
- Stewart Platform

In figures 2.5 and 2.6, both designs have been illustrated.

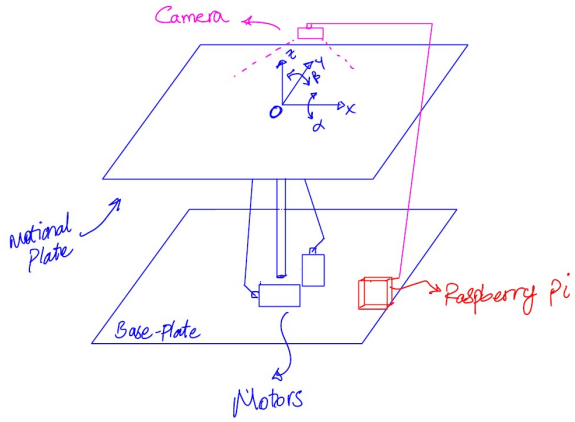


Figure 2.5: 2DOF Table with pivot points

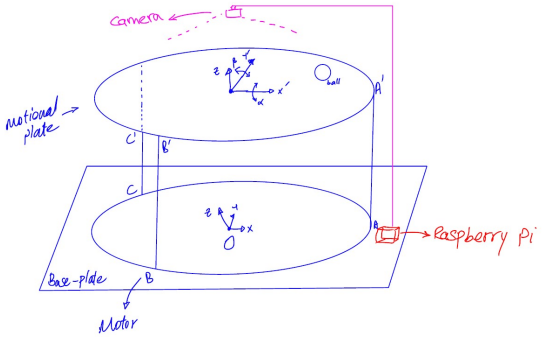


Figure 2.6: 3DOF Stewart Platform

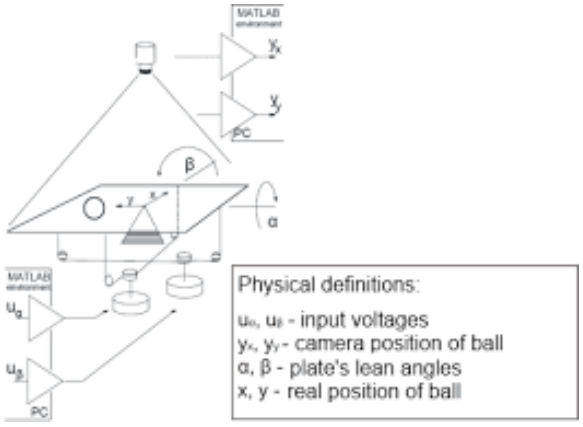


Figure 2.7: One pivot point

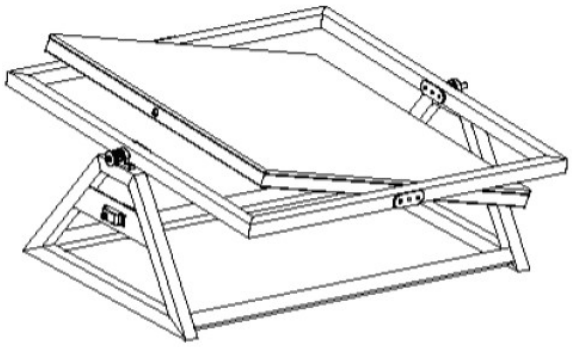


Figure 2.8: Inner plate inside an outer plate

In the first solution the mechanical construction limits the movement of the plate in only 2 degrees of freedom (DOF), namely two rotation angles around a mechanical pivot point. This alternative is relatively simple and effective which are the advantages of this set-up, since it uses two motors to control the axes. In [8] (figure 2.7), one pivot is chosen to be at the center and two stepper motors are connected to the motional plate via steel wires using pulleys to control the axes, whereas [9] (figure

2.8) used the inner plate rotating in an outer frame design. The challenge in [8] is elastic deformation in wires for connection. The non-linearity of the system would be more observable due to this problem, especially in fast inclination changes. Challenge in [9] is an assumption of having a symmetrical system for actuation since one motor (at least) should be directly connected to the inner plate in set-up for motion without gearbox or any transmission.
The second solution is using the Stewart Platform. This model is used in vast majority of robotic world and it is a parallel manipulator type. Here the original idea is to allow a 6DOF movement of the plate. The main advantage of such system is that the motion plate is being controlled by multiple arms and this results to a better controllability of the system with higher precision in comparison to the first solution. However the extra control over the model in exchange for its simplicity is the disadvantage of such model. Both [10] and [11] have explained: firstly how a 6DOF Stewart Platform works and then they have decreased the degree of freedom to 3 (which 2DOF in two rotation angles and 1DOF in translation in Z-direction) to obtain a simpler system.
After consulting with the supervisors and experts in Equipment and Prototyping Center (EPC) at TU/e, the second option has been considered as the solution. The motivations are:

- Suitable actuators for this design are available at TU/e Control Systems Group
- An educative concept for the students due to computation part of angles translation to positions and vise versa
- More control on system with higher precision
- A basis for future projects such as: Octo Bouncer

2.2.2 Motion Plate

***Note1**: An edge is also added to the motion plate to prevent falling the ball off the plate. The height of this edge must be at least equal to the diameter of the ball.

***Note2**: Mass plays a crucial role for the calculation of the applied torque on motors. Therefore, the lighter the plate the lower torque on the motors. A lighter plate will lead to less stiffness, which will affect the dynamics of the rolling ball. So the design choice is a trade off between weight and stiffness.

Specs	**Type/Value**	**Note**
Material	Perspex	Density : 1180 $Kg/(m)^3$
Dimension	0.39x0.39x0.005	W-L-H $(m)^3$
Weight	0.89739	mass = W*L*H*Density Kg
Color	White	Simpler design with regarding to image processing

Table 2.5: Motion Plate

2.2.3 Base Plate and Frame

***Note1**: Dimension is larger than the motion plate due to making some space for a Raspberry Pi board and Camera holder basis on the plate.

***Note2**: Since the base plate must prevent vibrating as much as possible, it should be heavy. Vibrating is an essential factor for controlling the motion plate and having less disturbance for the Camera.

Specs	Type/Value	Note
Material	Aluminum	Density : 2700 $(Kg/(m)^3)$
Dimension	0.63x0.63x0.015	W-L-H $(m)^3$
Weight	16.074	mass = Volume*Density (Kg)

Table 2.6: Base Plate

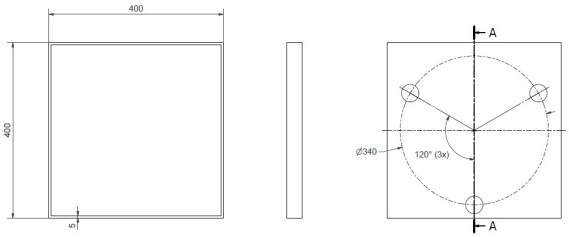


Figure 2.9: Motion Plate design in AUTOCAD

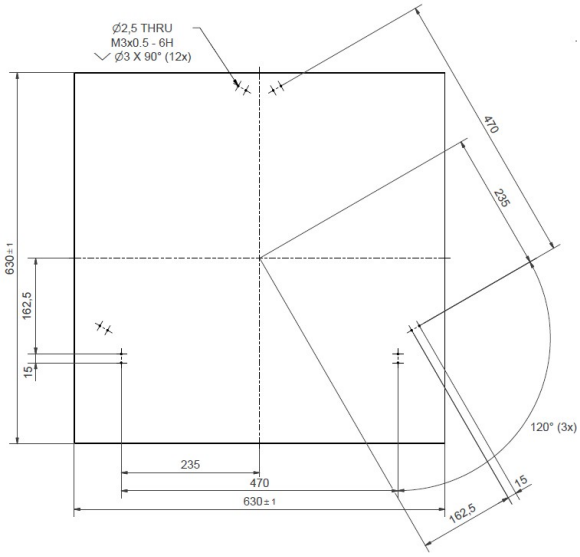


Figure 2.10: Base Plate design in AUTOCAD

2.2.4 Ball

Figure 2.11 demonstrates the chosen ball and table 2.7 will illustrate the specifications.

Figure 2.11: Ball

Specs	Type/Value	Note
Material	Rubber	Density : 1522 $Kg/(m)^3$
Diameter	$38.1 * (10)^-3$	(m)
Weight	0.0441	mass = Volume*Density (kg)
Color	Black	Simpler design with regarding to image processing

Table 2.7: Ball

2.2.5 Actuators

Actuators are given from client. Specifications can be found in table 2.8.

Specs	Type/Value	Note
Model	P01-23x80/30x90	Series : 100
Weight	0.118	Slider mass (Kg)
Force Constant	11	(N/A)
Viscous Friction Constant	16.5	-
Supply Voltage	48	V
Current Range	[-3 , 3]	A

Table 2.8: Linear DC Motor

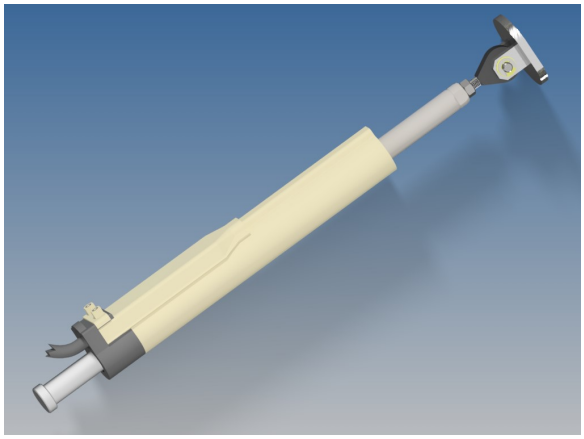


Figure 2.12: Linear DC Motor View1

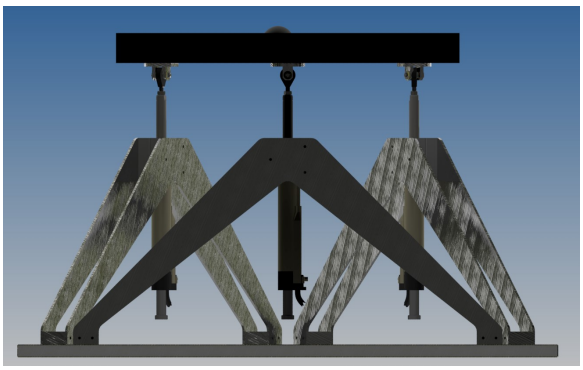


Figure 2.13: Linear DC Motors view 2

2.2.6 Camera Positioning

There are two options to use the camera in this project, namely:

- Static camera
- Moving camera

As mentioned earlier, in case of a static camera, pixels size will change when the plate is tilting and this might lead to false detection. To prevent such situation, a projection correction can be done. On the

other hand, a moving camera will not suffer from this drawback , since the background will be always the same with regarding to the position of camera. However, the challenge in this case is designing a mechanical frame that prevents any vibration (since vibration causes blurred captures and it results failure in tracking and detecting the ball [12]). Hence, a stationary camera is a better solution for this project (despite extra step (Projection correction) which is needed for the ball detection).There are two frames proposed for this project, namely:

- A four legs frame (in figure 2.4)
- A two/four legs frame in a triangle/pyramid shape

First option is chosen due to freedom it provides to adjust the location of camera.

2.3 Control Platform

Figure 2.14: dSPACE Controller Board

2.3.1 Application area

The real-time hardware based on PowerPC technology and its set of I/O interfaces make this board an ideal solution for developing controllers in various fields, such as drives, robotics, and aerospace.[13]

2.3.2 Key benefits

The DS1104 RD Controller Board is a cost-effective entry-level system with I/O interfaces and a real-time processor on a single board that can be plugged directly into a PC. It upgrades your PC to a development tool for rapid control prototyping and is ideal for developing smaller control applications or for education purposes. Real-Time Interface (RTI) provides Simulink® blocks for graphical I/O configuration. The board can be installed in virtually any PC with a free PCI or PCIe slot.[13]

2.3.3 Using Real-Time Interface

With Real-Time Interface (RTI), you can easily run your function models on the DS1104 RD Controller Board. You can configure all I/O graphically, insert the blocks into a Simulink block diagram, and generate the model code via Simulink® Coder™. The real-time model is compiled, downloaded, and started automatically. This reduces the implementation time to a minimum.[13]

Chapter 3

Modelling

The goal of this chapter is to approach a mathematical model for the ball and plate system. This model, not only should describe the system with a high accuracy, but also, simple enough for integrating to a real design. In [14], [3],[15] and [9], it is shown that although actuators and ball-plate models are highly related, both models can be separately analyzed. Therefore, the same approach is going to be used for this project.

3.1 System's Overview

The complete system model is shown in figure 3.1 and the control block scheme is illustrated in figure 3.2.The system includes two separated parts with different DOFs. Motion Plate has two DOF, since it can move w.r.t X and Y-axis and Motors are having 3DOF, due to their possible movement.

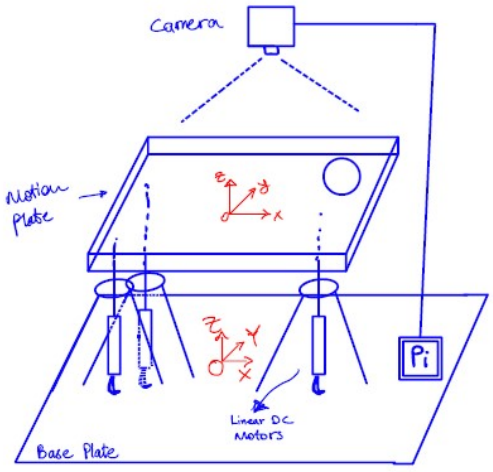


Figure 3.1: System's Overview

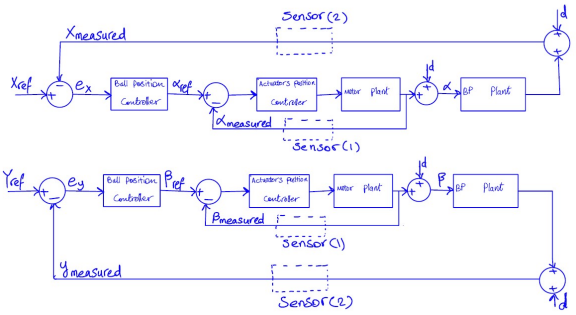


Figure 3.2: Total System - Controller Scheme

Where: BP Plant : $G_{P_x}(s) = \frac{X(s)}{\alpha(s)} = \frac{Kg}{s^2}$ is the ball and plate's transfer function

Motor Plant: $\frac{X(s)}{I(s)} = \frac{\frac{K_m}{m}}{s \cdot (s + \frac{C_v}{m})}$ is the motor's plant transfer function

Actuators Position Controller : represents the inner-loop controller

Ball Position Controller : stands for the outer-loop controller
Sensor (1): measures the angle made by motors
Sensor (2): measures the position of the ball after processing the image
Disturbance (d): Any interference on the position of the ball will be the disturbance
X_{ref}, Y_{ref} : The desired co-ordinates which ball should move towards that point.

***Note**: Motor's inputs are positions and not angles and this problem will be explained later in this chapter, but the concept is that motors will provide the desired angles. So, it is safe to say in figure 3.2 that angles are the inputs.

The overall control scheme can be explained as follows:
While the controller in outer-loop computes the angle by which the plate should move to balance the ball, the inner-loop controller moves the plate by that angle.

3.2 Ball-Plate Model

To derive dynamic equations of this system, following assumptions are made to prevent complexity:

- No slippery movement between the ball and the plate (proof in .1,section .1.2).
- Ball can not jump, hence there is always contact between the ball and the plate(proof in .1,section .1.4.
- Friction forces are negligible.
- The ball has a homogeneous spherical shape.

3.2.1 System Variables

Figures 3.3 and 3.4 demonstrate two different views of the plate and they will cover all necessary variables, which are shown in table 3.1.

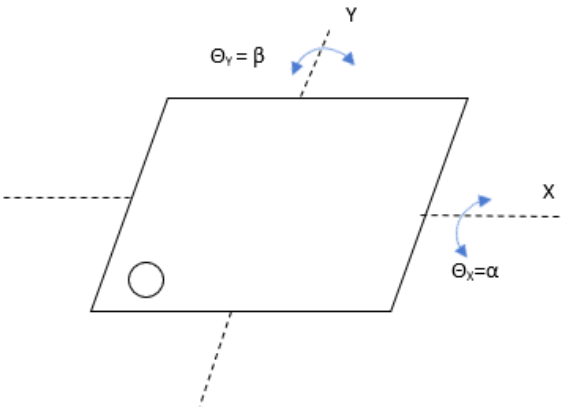


Figure 3.3: Top View of System

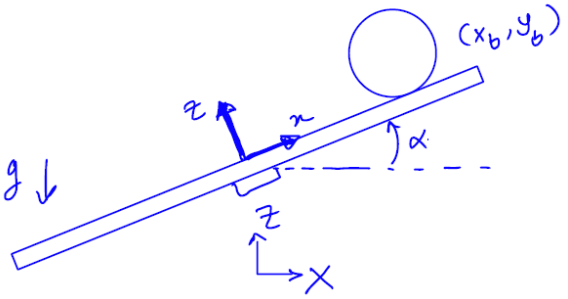


Figure 3.4: Side view of System

Variable	Description
Z , X	Axis of the world frame
z , x	Axis of the motion frame
x , y	Displacement of the ball along the x-axis and y-axis, respectively.
α, β	The angle between the x-axis and Y-axis of the plate and horizontal plane, respectively.
τ_x, τ_y	Torque exerted on the plate in X-axis and Y-axis, respectively.
m	mass of the ball
r	radius of the ball
I_b, I_p	Inertia of the ball and plate
$\dot{x}$, $\dot{y}$	Velocity of the ball with respect to X and Y axis $\frac{m}{s}$
$\dot{\alpha}$, $\dot{\beta}$	Rotational velocity of the plate $\frac{1}{rad}$
$\ddot{x}$, $\ddot{y}$	Acceleration of the ball with respect to X and Y-axis $\frac{m}{s^2}$
$\ddot{\alpha}$, $\ddot{\beta}$	Angular velocity of the plate $\frac{1}{rad^2}$

Table 3.1: System Variables

3.2.2 Energy Analysis

As it can be seen in figure 3.1, there are two co-ordinates systems. The world frame is denoted by oXYZ and it is placed at the center of base plate with Z-axis pointed upwards vertically. The motion frame has its own co-ordinate system, denoted by oxyz, with z-axis perpendicular to the motion plate itself and the origin is placed at the center of the plate.
Based on assumption that is made in previous section, there are generally two types of energies that can describe the motion of the ball and plate, namely:

- Kinetic Energy
- Potential Energy

Lagrange equation can describe the behaviour of the system with regarding to energies:

$$L = T - V \tag{3.1}$$

where T is the sum of Kinetic energy of the ball and the plate, and V is the sum of Potential energy of the ball and the plate. Hence,

$$T = T_b + T_p \tag{3.2}$$

$$V = V_b + V_p \tag{3.3}$$

By considering the fact that the plate does not have Potential energy, equation 3.1 becomes:

$$V = V_b \tag{3.4}$$

Kinetic Energy

As mentioned in previous section, Kinetic energy is divided into two parts. General equations for Kinetic energy of the ball and the plate are as following:

$$T_b = \frac{1}{2} \cdot m \cdot (V_s)^2 + \frac{1}{2} \cdot I_b \cdot \omega^2 \tag{3.5}$$

$$T_p = \frac{1}{2} \cdot (I_p + I_b) \cdot (\dot{\alpha}^2 + \dot{\beta}^2) + \frac{1}{2} \cdot m \cdot (x\dot{\alpha} + y\dot{\beta})^2 \tag{3.6}$$

Where V_s and ω are the velocity and angular velocity of the ball, respectively.
Based on [16] and figure 3.5, an equivalent equation for Kinetic energy of the ball can be found as follow:

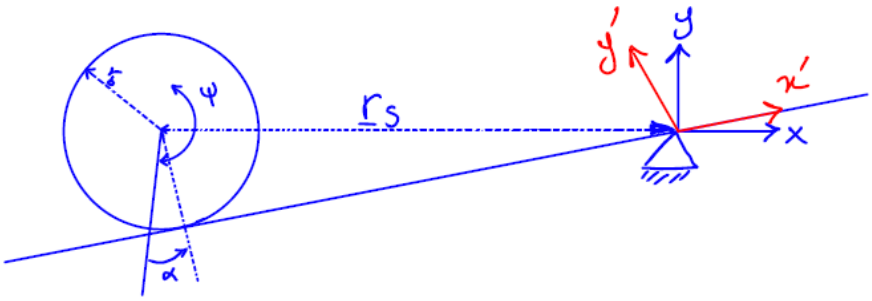


Figure 3.5: The definition of the position vector of the ball

Where x and y are static co-ordinate system (inertial system), x' and y' are the moving reference system and ψ is the angle of the ball with regarding to the x'.
It is essential to note that because of a symmetric system, some of calculations will be done with respect to one axis then it will be applied to the other axis as well.
To formulate V_s which is the velocity of the ball in 3.5, with respect to inertial system, the following correlation is valid:

$$\underline{V_s} = \underline{V_s'} + \underline{\omega} \times \underline{r_s} \tag{3.7}$$

Where $\underline{\omega}$ is the angular velocity vector of the ball and $\underline{r_s}$ is the position vector and they are defined with regarding to x' as follow:

$$\underline{r_s} = [-x', r]^T$$

$$\underline{V_s'} = \frac{d'\underline{r_s}}{dt} = [-\dot{x'}, 0]^T$$

$$\underline{V_s} = \frac{d\underline{r_s}}{dt} = \begin{bmatrix} -\dot{x'} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\alpha} \end{bmatrix} \times \begin{bmatrix} -x' \\ r \\ 0 \end{bmatrix} = \begin{bmatrix} -\dot{x'} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -\dot{\alpha} \cdot r \\ -x'\dot{\alpha} \\ 0 \end{bmatrix} = \begin{bmatrix} -\dot{x'} - \dot{\alpha} \cdot r \\ -x'\dot{\alpha} \\ 0 \end{bmatrix}$$

$$\underline{V_s^2} = \begin{bmatrix} -\dot{x'} - \dot{\alpha} \cdot r \\ -x'\dot{\alpha} \\ 0 \end{bmatrix} \cdot \begin{bmatrix} -\dot{x'} - \dot{\alpha} \cdot r \\ -x'\dot{\alpha} \\ 0 \end{bmatrix} = \dot{x'^2} + 2\dot{x'}\dot{\alpha}r + (\dot{\alpha}r)^2 + (x'\dot{\alpha})^2 \tag{3.8}$$

To formulate ω (w.r.t x'), it should be noted that $\omega_{x'}$ is a combination of rotation of the ball itself and rotation of the plate. This leads to the following:

$$\omega_{x'} = \dot{\psi} + \dot{\alpha} = \frac{\dot{y'}}{r} + \dot{\alpha} \tag{3.9}$$

Symmetric system with respect to the moving reference system will deliver the following equations:

- x':

$$\omega_{x'} = \dot{\psi} + \dot{\alpha} = \frac{\dot{y}'}{r} + \dot{\alpha}$$

$$\underline{V_s^2} = \dot{x}'^2 + 2\dot{x}'\dot{\alpha}r + (\dot{\alpha}r)^2 + (x'\dot{\alpha})^2 \tag{3.10}$$

- y':

$$\omega_{y'} = \dot{\Psi} + \dot{\beta} = \frac{\dot{x}'}{r} + \dot{\beta}$$

$$\underline{V_s^2} = \dot{y}'^2 + 2\dot{y}'\dot{\beta}r + (\dot{\beta}r)^2 + (y'\dot{\beta})^2 \tag{3.11}$$

Adding 3.10 and 3.11 will give the following equations:

$$\omega_{x'}^2 + \omega_{y'}^2 = \frac{\dot{x}'^2}{r^2} + \frac{\dot{y}'^2}{r^2} + \frac{2}{r}(\dot{\alpha}\dot{y}' + \dot{\beta}\dot{x}') + \dot{\alpha}^2 + \dot{\beta}^2 \tag{3.12}$$

$$\underline{V_{s_{x'}}^2} + \underline{V_{s_{y'}}^2} = \dot{x}'^2 + \dot{y}'^2 + r^2(\dot{\alpha}^2 + \dot{\beta}^2) + x'^2\dot{\alpha}^2 + y'^2\dot{\beta}^2 + 2r(\dot{x}'\dot{\alpha} + \dot{y}'\dot{\beta})$$

*Note: from this step, to make calculations easier, x' will change to x and y' to y since the angle between the two co-ordinates is very small.

Substituting equations 3.12into equation 3.5 will lead to:

$$T_{ball} = T_{ball_x} + T_{ball_y} =$$

$$\frac{1}{2}m(\dot{x}^2+\dot{y}^2+r^2(\dot{\alpha}^2+\dot{\beta}^2)+x^2\dot{\alpha}^2+y^2\dot{\beta}^2+2r(\dot{x}\dot{\alpha}+\dot{y}\dot{\beta}))+\frac{1}{2}I_b(\frac{1}{r^2}(\dot{x}^2+\dot{y}^2)+\frac{2}{r}(\dot{\alpha}\dot{y}+\dot{\beta}\dot{x})+\dot{\alpha}^2+\dot{\beta}^2$$

$$T_{plate} = \frac{1}{2}\cdot(I_p + I_b)\cdot(\dot{\alpha}^2 + \dot{\beta}^2) + \frac{1}{2}\cdot m\cdot(x^2\dot{\alpha}^2 + 2xy\dot{\alpha}\dot{\beta} + y^2\dot{\beta}^2) \tag{3.13}$$

Potential Energy

As mentioned earlier the total Potential energy is the energy from the ball and can be described as figure 3.6:

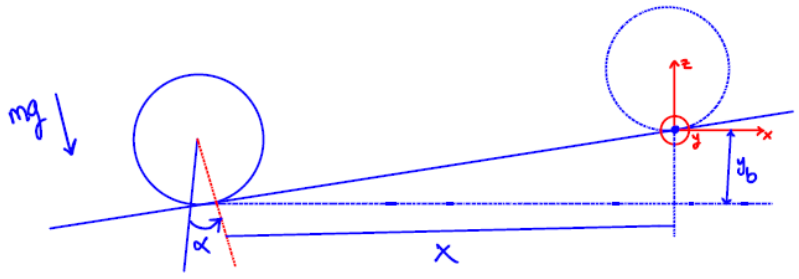


Figure 3.6: Potential Energy of the ball

$$V_b = -mg(xsin(\alpha) + ysin(\beta)) \tag{3.14}$$

Where g is the gravity force and equal to 9.81 $\frac{m}{s^2}$.

3.2.3 Force Analysis

The generalized co-ordinates are the two ball co-ordinates [x,y] and the two plate inclination [α, β], where [x,y]=[0,0] is the center of the plate.
The generalized forces are divided into two parts, namely:

- The external forces on the ball
- The external forces on the plate

According to figure 3.7, there is no external force on the ball itself except the gravity, these terms will be zero for both direction x and y. On the contrary, there are forces in form of torque acting on the plate on each direction. These forces are provided by actuators.

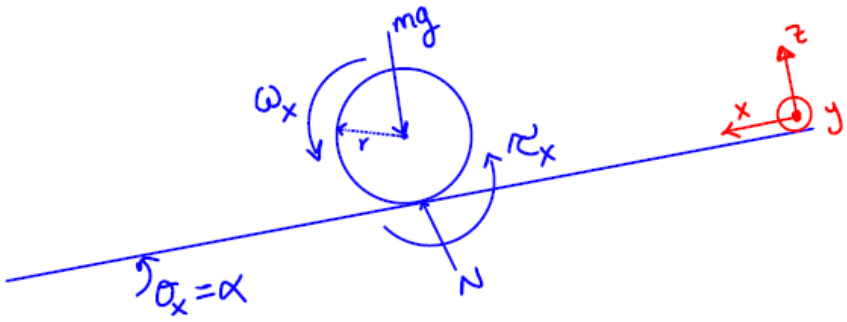


Figure 3.7: Forces on System

3.2.4 System Equations

To describe dynamical equations of the model, the general form of Euler-Lagrange equation is used:

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}}) - \frac{\partial L}{\partial q} = Q$$

Considering equation 3.1:

$$\frac{d}{dt}(\frac{\partial (T-V)}{\partial \dot{q}}) - \frac{\partial (T-V)}{\partial q} = Q$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{q}}) - \frac{\partial T}{\partial q} + \frac{\partial V}{\partial q} = Q \tag{3.15}$$

Where q is the generalized co-ordinates and Q is the generalized forces and based on previous section, these forces are defined as follow:

$$q = \begin{bmatrix} x \\ y \\ \alpha \\ \beta \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 \\ 0 \\ \tau_x \\ \tau_y \end{bmatrix} \tag{3.16}$$

Dividing equation 3.15 to sub-parts, makes the calculations easier:

- $\frac{\partial T}{\partial q} : \frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}, \frac{\partial T}{\partial \alpha}, \frac{\partial T}{\partial \beta}$
- $\frac{\partial V}{\partial q} : \frac{\partial V}{\partial x}, \frac{\partial V}{\partial y}, \frac{\partial V}{\partial \alpha}, \frac{\partial V}{\partial \beta}$
- $\frac{\partial T}{\partial \dot{q}} : \frac{\partial T}{\partial \dot{x}}, \frac{\partial T}{\partial \dot{y}}, \frac{\partial T}{\partial \dot{\alpha}}, \frac{\partial T}{\partial \dot{\beta}}$
- $\frac{d}{dt}(\frac{\partial T}{\partial \dot{q}}) : \frac{d}{dt}(\frac{\partial T}{\partial \dot{x}}), \frac{d}{dt}(\frac{\partial T}{\partial \dot{y}}), \frac{d}{dt}(\frac{\partial T}{\partial \dot{\alpha}}), \frac{d}{dt}(\frac{\partial T}{\partial \dot{\beta}})$

Above equations and considering equations 3.2, 3.3 will lead to the following:

$$\frac{\partial T}{\partial x} = m\dot{\alpha}(2x\dot{\alpha} + y\dot{\beta}) \quad \frac{\partial T}{\partial \alpha} = 0$$

$$\frac{\partial T}{\partial y} = m\dot{\beta}(2y\dot{\beta} + x\dot{\alpha}) \quad \frac{\partial T}{\partial \beta} = 0$$

$$\frac{\partial V}{\partial x} = -mgsin(\alpha) \quad \frac{\partial V}{\partial \alpha} = -mgxcos(\alpha)$$

$$\frac{\partial V}{\partial y} = -mgsin(\beta) \quad \frac{\partial V}{\partial \beta} = -mgycos(\beta)$$

$$\frac{\partial T}{\partial \dot{x}} = (m + \frac{I_b}{r^2})\dot{x} + mr\dot{\alpha} + \frac{I_b}{r}\dot{\beta} \quad \frac{\partial T}{\partial \dot{y}} = (m + \frac{I_b}{r^2})\dot{y} + mr\dot{\beta} + \frac{I_b}{r}\dot{\alpha}$$

$$\frac{\partial T}{\partial \dot{\alpha}} = (mr^2 + 2mx^2 + 2I_b + I_p)\dot{\alpha} + mr\dot{x} + mxy\dot{\beta} + \frac{I_b}{r}\dot{y}$$

$$\frac{\partial T}{\partial \dot{\beta}} = (mr^2 + 2my^2 + 2I_b + I_p)\dot{\beta} + mr\dot{y} + mxy\dot{\alpha} + \frac{I_b}{r}\dot{x}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{x}}) = (m + \frac{I_b}{r^2})\ddot{x} + mr\ddot{\alpha} + \frac{I_b}{r}\ddot{\beta}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{y}}) = (m + \frac{I_b}{r^2})\ddot{y} + mr\ddot{\beta} + \frac{I_b}{r}\ddot{\alpha}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\alpha}}) = (mr^2 + 2mx^2 + 2I_b + I_p)\ddot{\alpha} + mr\ddot{x} + \frac{I_b}{r}\ddot{y} + mxy\ddot{\beta} + 4mx\dot{x}\dot{\alpha} + my\dot{x}\dot{\beta} + mx\dot{y}\dot{\beta}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\beta}}) = (mr^2 + 2my^2 + 2I_b + I_p)\ddot{\beta} + mr\ddot{y} + \frac{I_b}{r}\ddot{x} + mxy\ddot{\alpha} + 4my\dot{y}\dot{\beta} + mx\dot{y}\dot{\alpha} + my\dot{x}\dot{\alpha}$$

Substituting above equations in equation 3.15 and considering equations 3.16 will point the system equations:

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{x}}) - \frac{\partial T}{\partial x} + \frac{\partial V}{\partial x} = 0$$

$$(m + \frac{I_b}{r^2})\ddot{x} + mr\ddot{\alpha} + \frac{I_b}{r}\ddot{\beta} - 2mx\dot{\alpha}^2 - my\dot{\alpha}\dot{\beta} - mgsin(\alpha) = 0 \tag{3.17}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{y}}) - \frac{\partial T}{\partial y} + \frac{\partial V}{\partial y} = 0$$

$$(m + \frac{I_b}{r^2})\ddot{y} + mr\ddot{\beta} + \frac{I_b}{r}\ddot{\alpha} - 2my\dot{\beta}^2 - mx\dot{\alpha}\dot{\beta} - mgsin(\beta) = 0 \tag{3.18}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\alpha}}) - \frac{\partial T}{\partial \alpha} + \frac{\partial V}{\partial \alpha} = \tau_x$$

$$(mr^2 + 2mx^2 + 2I_b + I_p)\ddot{\alpha} + mr\ddot{x} + \frac{I_b}{r}\ddot{y} + mxy\ddot{\beta} + 4mx\dot{x}\dot{\alpha} + my\dot{x}\dot{\beta} + mx\dot{y}\dot{\beta} - mgxcos(\alpha) = \tau_x \tag{3.19}$$

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\beta}}) - \frac{\partial T}{\partial \beta} + \frac{\partial V}{\partial \beta} = \tau_y$$

$$(mr^2+2my^2+2I_b+I_p)\ddot{\beta}+mr\ddot{y}+\frac{I_b}{r}\ddot{x}+mxy\ddot{\alpha}+4my\dot{y}\dot{\beta}+mx\dot{y}\dot{\alpha}+my\dot{x}\dot{\alpha}-mgycos(\beta) = \tau_y \tag{3.20}$$

Equations 3.17 and 3.18 describe the ball's motion and demonstrate how acceleration of the ball depends on its position on the plate and on angles and angular velocity of the plate.
Equations 3.19 and 3.20 express the plate dynamics and how it depends on external forces, the position and the velocity of the ball.

Matrix form of system

Based on equations 3.17,3.18,3.19 and 3.20, it is possible to form all relations in a matrix form. The general form is as following:

$$M(q,\dot{q})\ddot{q} + C(q,\dot{q})q + G(q) = Q$$

Where, M(q , $\dot{q}$) stands for Inertia Matrix and it clarifies that its elements (co-ordinates itself or the first derivative of co-ordinates) depend on the second derivative terms in above equations.
C(q , $\dot{q}$) is called Coriolis Matrix and G(q) is the Gravity Matrix.

$$q = \begin{bmatrix} x \\ y \\ \alpha \\ \beta \end{bmatrix} \quad \dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} \quad \ddot{q} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\alpha} \\ \ddot{\beta} \end{bmatrix} \quad Q = \begin{bmatrix} 0 \\ 0 \\ \tau_x \\ \tau_y \end{bmatrix}$$

Defining Constants
$a_1 = m + \frac{I_b}{r^2}, \quad a_2 = mr, \quad a_3 = \frac{I_b}{r}, \quad a_4 = mg, \quad a_5 = m, \quad b_1 = mr^2, \quad b_2 = 2I_b + I_p$

Re-state equations 3.17,3.18,3.19 and 3.20 using defined constants:

$$a_1\ddot{x} + a_2\ddot{\alpha} + a_3\ddot{\beta} - 2a_5x\dot{\alpha}^2 - a_5y\dot{\alpha}\dot{\beta} - a_4sin(\alpha) = 0 \tag{3.21}$$

$$a_1\ddot{y} + a_2\ddot{\beta} + a_3\ddot{\alpha} - 2a_5y\dot{\beta}^2 - a_5x\dot{\alpha}\dot{\beta} - a_4sin(\beta) = 0 \tag{3.22}$$

$$(b_1 + 2a_5x^2 + b_2)\ddot{\alpha} + a_2\ddot{x} + a_3\ddot{y} + a_5xy\ddot{\beta} + 4a_5x\dot{x}\dot{\alpha} + a_5y\dot{x}\dot{\beta} + a_5x\dot{y}\dot{\beta} - a_4xcos(\alpha) = \tau_x \tag{3.23}$$

$$(b_1 + 2a_5y^2 + b_2)\ddot{\beta} + a_2\ddot{y} + a_3\ddot{x} + a_5xy\ddot{\alpha} + 4a_5y\dot{y}\dot{\beta} + a_5x\dot{y}\dot{\alpha} + a_5y\dot{x}\dot{\alpha} - a_4ycos(\beta) = \tau_y \tag{3.24}$$

Based on equations 3.21, 3.22, 3.23 and 3.24;

$$M(q,\dot{q}) = \begin{bmatrix} a_1 & 0 & a_2 & a_3 \\ 0 & a_1 & a_3 & a_2 \\ a_2 & a_3 & b_1 + 2a_5x^2 + b_2 & a_5xy \\ a_3 & a_2 & a_5xy & b_1 + 2a_5y^2 + b_2 \end{bmatrix}$$

$$C(q,\dot{q}) = \begin{bmatrix} 0 & 0 & 4a_5x\dot{\alpha} + a_5y\dot{\beta} & a_5y\dot{\alpha} \\ 0 & 0 & a_5x\dot{\beta} & 4a_5y\dot{\beta} + a_5x\dot{\alpha} \\ -2a_5x\dot{\alpha} - a_5y\dot{\beta} & -a_5x\dot{\beta} & 4a_5x\dot{x} & a_5x\dot{y} + a_5y\dot{x} \\ -a_5y\dot{\alpha} & -2a_5y\dot{\beta} - a_5x\dot{\alpha} & a_5y\dot{x} + a_5x\dot{y} & 4a_5y\dot{y} \end{bmatrix}$$

$$G(q) = \begin{bmatrix} -a_4 sin(\alpha) \\ -a_4 sin(\beta) \\ -a_4 x sin(\alpha) \\ -a_4 y sin(\beta) \end{bmatrix}$$

As it can be seen from matrices above, the system with this configuration is very complicated. High orders of each variable is coupled with each-other. The best way to deal with such a complex system is linearization.

3.2.5 Linearization

The way that [8] and [17] have tackled this problem is different with each-other. in [8], it is assumed that α and β are very small $\langle -5°, 5° \rangle$ and therefore :

$$\dot{\alpha}\dot{\beta} \cong 0 \quad \dot{\alpha}^2 = \dot{\beta}^2 \cong 0 \quad sin(\alpha) \cong \alpha \quad sin(\beta) \cong \beta \quad I_{b,hollow-spherical} = \frac{2}{5}mr^2$$

Where [17], keeps the quadratic terms for $\dot{\alpha}$ and $\dot{\beta}$ to obtain more accuracy. However, both of references have ignored the crossing terms and the affect of angular velocity of the plate with respect to axes for simplicity of the design.
This project will also follow some of these steps to keep the model as simple as possible. It is essential to note that the actuators till the plate and the only affect that the plate might have on actuators performance is the mass of the motion plate. Therefore, it is safe to say that angles α and β can be directly the input to the system. Thus, equations 3.23 and 3.24 can be omitted in the final system representation block, on behalf of simplification. However, the mass of the plate will be taken into account in motor's model.
Based on above assumptions and mentioned strategy which is stated at the beginning of this section and by considering constants firstly and then change them back to the original terms, following equations will be the expression of system equations after linearization:

$$a_1\ddot{x} - a_4\alpha = 0 \rightarrow \ddot{x} = \frac{m}{m + \frac{I_b}{r^2}} g\alpha \rightarrow \ddot{x} = Kg\alpha \tag{3.25}$$

$$a_1\ddot{y} - a_4\beta = 0 \rightarrow \ddot{y} = \frac{m}{m + \frac{I_b}{r^2}} g\beta \rightarrow \ddot{y} = Kg\beta \tag{3.26}$$

$$(b1 + b2)\ddot{\alpha} = \tau_x \rightarrow \ddot{\alpha} = \frac{1}{mr^2 + 2I_b + I_p}\tau_x \tag{3.27}$$

$$(b1 + b2)\ddot{\beta} = \tau_y \rightarrow \ddot{\beta} = \frac{1}{mr^2 + 2I_b + I_p}\tau_y \tag{3.28}$$

Where $I_p = \frac{m(h^2+w^2)}{12}[kg.m^2]$
$I_{b,hollow-spherical} = \frac{2}{5}mr^2[kg.m^2]$ which leads to $K = \frac{5}{7}$
It is clear that based on equations 3.25 and 3.26, the dynamic equations are identical in the x-axis and y-axis.

3.2.6 System Representation

There are two ways to represent system equations 3.25 and 3.26, namely:

- State Space model
- Laplace domain

Both methods will be represented in the following sub-sections.

State Space

Based on equations 3.25, 3.26,3.27 and 3.28 and the conclusion made for torques acting on the plate in previous section, the states are as follow:

$$X = [x1, x2, x3, x4, x5, x6, x7, x8]^T = [x, \dot{x}, \alpha, \dot{\alpha}, y, \dot{y}, \beta, \dot{\beta}]^T$$

$$u = [u_x, u_y]^T = [\alpha, \beta]^T$$

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Kg & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Kg & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \\ y \\ \dot{y} \\ \beta \\ \dot{\beta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{3.29}$$

$$y = \begin{bmatrix} x1 \\ x5 \end{bmatrix} \qquad D = 0$$

Where u_x is the input and it passes the angle α to the ball and plate system. From equations 3.27 and 3.28, it can be concluded that $u_x = \tau_x$ and $u_y = \tau_y$.
These matrices can be divided into two parts, since x, α and y, β are acting independently. So, the following representation is only with x-axis consideration:

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & Kg & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} u_x \end{bmatrix} \tag{3.30}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \end{bmatrix} \qquad D = 0$$

Laplace

Considering equation 3.25 and 3.27, it is possible to calculate the transfer function, which are as following:

$$s^2 X(s) = Kg\alpha(s) \rightarrow G_{P_x}(s) = \frac{X(s)}{\alpha(s)} = \frac{Kg}{s^2} \tag{3.31}$$

$$s^2\alpha(s) = \frac{1}{b1+b2}U_x(s) \rightarrow G_{I_x}(s) = \frac{\alpha(s)}{U_x(s)} = \frac{1}{b1+b2}\frac{1}{s^2} \rightarrow G_{I_x}(s) = \frac{1}{mr^2+2I_b+I_p}\frac{1}{s^2} \tag{3.32}$$

Due to a symmetric system:

$$G_{P_y}(s) = \frac{Y(s)}{\beta(s)} = \frac{Kg}{s^2} \tag{3.33}$$

$$s^2\beta(s) = \frac{1}{b1+b2}U_y(s) \rightarrow G_{I_y}(s) = \frac{\beta(s)}{U_y(s)} = \frac{1}{b1+b2}\frac{1}{s^2} \rightarrow G_{I_y}(s) = \frac{1}{mr^2+2I_b+I_p}\frac{1}{s^2} \tag{3.34}$$

Block-diagram of ball and plate
As mentioned earlier, due to omitting equations 3.23 and 3.24, represented transfer function of Ball-Plate model is as following:

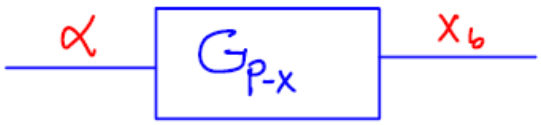


Figure 3.8: Transfer Function Ball and Plate with respect to x-axis

Where G_{P_x} is equation 3.31.

3.3 Actuator Model

In this section, actuator's model will be described. This project uses three Linear DC Motors to govern the movement of motion plate.

3.3.1 System Variables

As it is mentioned earlier, the torque generated by actuator, will provide the external force on the plate. It is also referred that the inertia of the plate must be taken into account (Instead of using equations for the inertia of the plate for the sake of simplification). This is done by distributing the mass of the motion plate between three motors. Figure 3.9 demonstrates the block diagram of actuators (Linear DC Motors) and specifications which associate with this figure can be depicted from table 3.2.

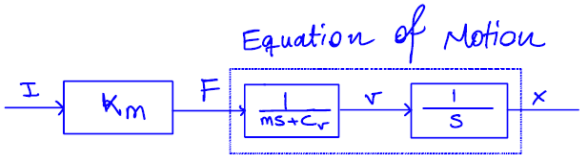


Figure 3.9: LDM Block Diagram

Variable	Description	Unit	Value
I	Current (Input)	[A]	-
F	Force	[Kg.m.s$^-$2]	-
X	Displacement of the slider(Output)	[m]	-
K_m	Thrust Constant	[N.A]	11
m	Slider mass plus $1/3$ plate's mass	[Kg]	$0.118 + (1/3) \cdot 0.682$
C_v	Viscous Friction Constant	-	16.5

Table 3.2: LDM variables

3.3.2 System Equations

Actuator's plant in figure 3.2 can be determined according to figure 3.9 as below:

$$Motor's \ \ Plant : \frac{X(s)}{I(s)} = \frac{\frac{K_m}{m}}{s(s + C_v)} \tag{3.35}$$

There are two important points that must be considered from equation 3.35, namely:

- Input/Output of actuators
- Complexity

In previous sections, it has been realized that the angles α and β can be directly the input to the motion plate' plant and equation 3.35 shows that the current in [A] is the input and displacement in [m] is the output. This problem is solved by using Rotation matrix and Tait-Bryan angles concept which will be discussed in detail in chapter 4.
Complexity can also be seen in equation 3.35, meaning that the actuator's plant becomes a second order tf and therefore, Refer to figure 3.2, it is clear that the inner-loop controller must have at least three poles which makes the closed-loop tf for the inner-loop a fifth order tf. Thus, design the outer-loop controller will be really difficult due to having many poles. To overcome this situation the closed-loop tf of the inner-loop can be simplified to a first order tf with having (almost)the same behavior. This is possible since the inner-loop is operating in (much)higher frequencies than the outer-loop. The solution will be demonstrated in the next chapter.

Chapter 4

Controller Design

In chapter 3, the theoretical model has been described. However, due to actuator's I/O, some conversion should be added to inner-loop block. First part of this chapter is a demonstration of how the system will look like considering conversion blocks. In the second part, the design of the inner-loop controller will be explained. Next, the translation of position to angles and vise versa will be explained. Afterwards, the outer-loop controller's design is going to be illustrated. Lastly, a verification of total system will be shown.

4.1 Control architecture

The design of controllers start from the inner-loop which includes actuators. This part is one of the major differences between the theoretical model and the model in practice. The reason is that linear DC motors (as actuators) have position in $[m]$ as reference-input and the output will be the position after applying the controller. However, as mentioned and shown in chapter 3 in figure 3.2, the reference input which goes to the motor is an angle and the output will be also an angle after applying the controller. The solution to this problem can be found in the following sections. The total plan to design the system are as follows:

First step is converting the given inner-loop controller and motor's plant to a first order tf based on discussion in motor's model in chapter3. The reason here is to simplify the open-loop blocks for designing the outer-loop controller. Otherwise, it would be very difficult to design a controller for a double integrator as the plate's plant and a fifth order tf as the equivalent block for the motor.
Second step would be using root-locus method to design the outer-loop controller in figure 4.2.
Third step is changing the inner-loop back to the given model, because the behavior of designed equivalent first order tf is almost the same and it can be replaced with the original model.
And lastly, a verification will take place to check whether the designed controller for the outer-loop works the same as in the second step.

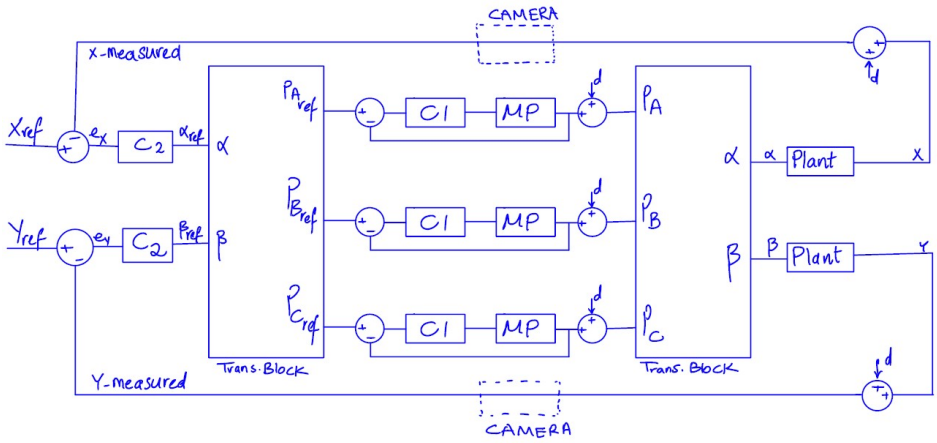


Figure 4.1: Total System in detail

Figure 4.1 illustrates the total system in detail and it is divided in six parts, namely:

C2 : outer-loop controller

Translation Block: Translating the angles to position

Inner-Loop : which includes **C1** as the inner-loop controller and **MP** the plant tf of the motor(equation 3.35)

Translation Block : Translating position to angles

Plant : which represents the plate's plant(equations 3.31and 3.33)

Further, d represents the disturbance and e stands for the error.

Following is the figure 4.2 that shows also the total system after calculating the closed-loop tf of the inner-loop.

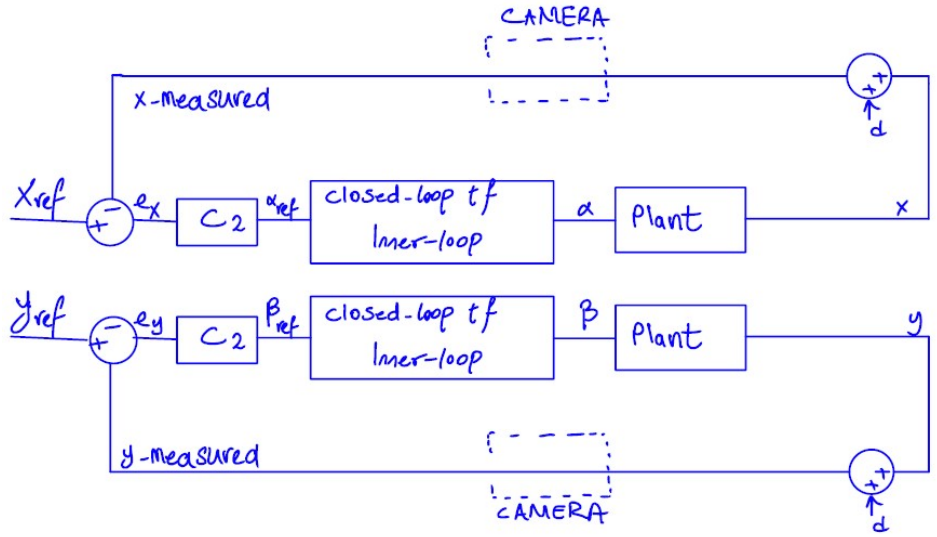


Figure 4.2: Total System with regarding to the outer-loop layer

4.2 Actuator position control (inner loop)

The motor's plant for this project can be described as a gain $MP = \frac{K}{\tau}$ where K and τ will be defined based on a given controller's specification by client.

4.2.1 Given Controller from Client

In this section, the given model for the linear motor will be realized.

Linear Plant Model

$$H_p(s) = \frac{X(s)}{I(s)} = \frac{\frac{K_m}{m}}{s \cdot (s + \frac{C_v}{m})} \tag{4.1}$$

where K_m : stands for force constant
m : stands for the mass of slider of the motor plus $1/3$ of the plate's mass ($1/3$ factor due to taking the role of moment of inertia of the plate into account by dividing the plate's mass between three motors)
C_v : Viscous friction constant
$X(s)$: position in $[m]$
$I(s)$: Current in $[A]$

Controller

The controller is designed based on a desired bandwidth at which motors operate fast enough for this project. To achieve this goal and considering equation 4.1 which has a pole at zero and a pole at $\frac{-Cv}{m}$ and $\frac{K_m}{m}$ as gain, following controller is considered:

$$H_c = K_c \cdot \frac{(s + B/3)(s + B/6)}{s(s + 3B)(s + 6B)} \tag{4.2}$$

Where B: desired frequency ($300\ \ rad/s \sim 50\ \ Hz$)
K_c: controller gain
This controller assures zero steady state error due to the integrator and 0dB open-loop magnitude at desired frequency. Moreover, the location of zeros are a factor $1/3$ and $1/6$ of the bandwidth value (in rad/s) and the location of poles are factor 3 and 6 of the bandwidth value (refer to equation 4.2). To determine the required gain, it is needed to calculate system's bandwidth which is as following:

$$B = \frac{C_v}{m} = 139.8\ \ rad/s \sim 22\ \ Hz$$

The desired bandwidth is $300\ \ rad/s$ which is determined after some experiments of how smooth motor operates in different bandwidths. To increase system's bandwidth, it is essential to calculate the magnitude of plant at desired frequency. The magnitude can be found either by reading it from bode plot of plant and convert it from dB to magnitude or by using the following equation:

$$P_{mag} = \frac{\frac{K_m}{m}}{B\sqrt{(\frac{-C_v}{m})^2 + B^2}} \rightarrow 20\log(P_{mag}) = ...dB$$

The required controller gain that makes the open-loop magnitude 0dB at desired frequency can be calculated as following:

$$K_c = \frac{1}{P_{mag} \cdot C_{mag}}$$

Where $C_{mag} = \frac{1}{B \cdot 18}$ w.r.t equation 4.2.
All things considered, the designed controller transfer function is:

$$C1(s) = \frac{5.752e06(s+100)(s+50)}{s(s+900)(s+1800)} \tag{4.3}$$

And the closed loop transfer function is:

$$H1(s) = \frac{5.362e08(s+100)(s+50)}{(s+2032)(s^2+137s+5090)(s^2+670.7s+2.592e05)} \tag{4.4}$$

Figure 4.3 shows 0dB open-loop magnitude at desired frequency is achieved.

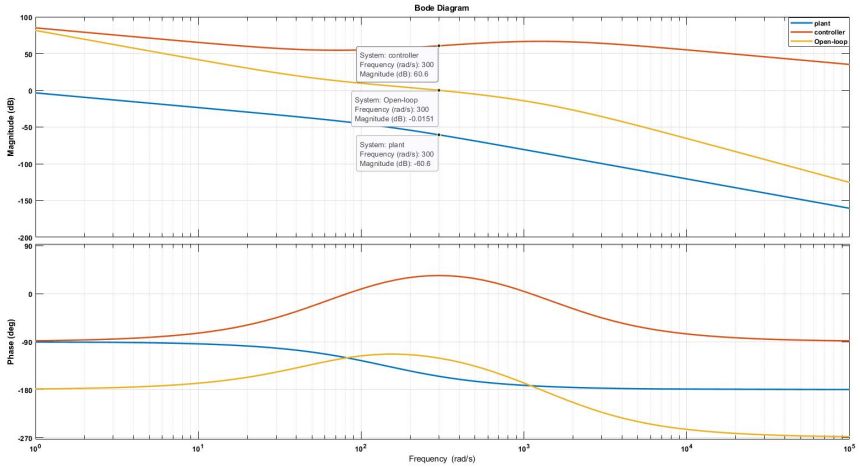


Figure 4.3: Inner-loop bode plot

As stated earlier, inner-loop transfer function should be converted to a first order transfer function:

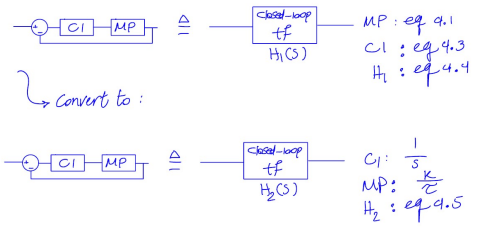


Figure 4.4: Inner loop Convert

To convert the controller tf and hence the closed-loop tf to the desired controller (shown in figure 4.4),

it is needed to check what the specifications of given controller are. One simple way to find out the specs, is how the closed-loop tf will react upon a step input.

Step Info	Description	Unit
Rise time	0.0033	seconds
Settling time	0.0207	seconds
Overshoot	38%	-

Table 4.1: Step info of the original Closed - loop tf of the motor

Base on table 4.1, the values for K and τ for a first order transfer function, are calculated as follows:
To obtain the settling time :

$$4 \cdot \tau = Ts = 0.0207 \rightarrow \ \tau = 0.005175 \text{ and } K = 1$$

hence:

$$H2(s) = \frac{K}{\tau s + 1} = \frac{1}{0.005175s + 1} \tag{4.5}$$

to verify whether equations 4.1 and 4.5 have a similar behavior, a step input is given to both functions and following figure is the result:

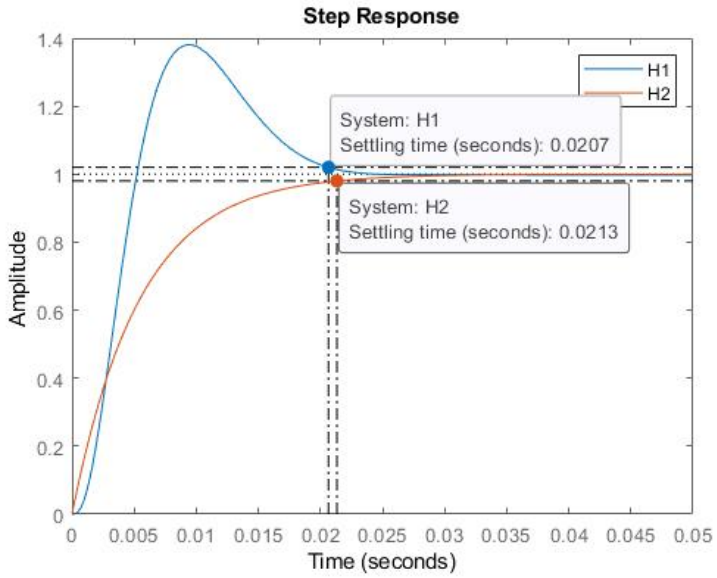


Figure 4.5: Closed-loop tf comparison for the inner-loop

Based on figure 4.5, it can be seen that both tfs have the same behavior in terms of settling time.
So, now that the equivalent closed-loop tf is designed and evaluated. it can be placed instead of closed-loop tf block in figure 4.2. But before jumping to design of outer-loop controller, it is needed to explain the translation of angles to position and vise versa, since input/output of the actuators are not the same as designed model (4.2. Afterwards, the outer-loop design will be discussed.

4.3 Plate Rotation and actuator translation

Based on the decision that has been made in chapter 2, this project is going to use a 3 DOF Stewart Platform as a design.

Background and Kinematic Analysis

Parallel Manipulators

Industrial applications are the ones which use this idea in many different ways. One of the first suggested application was a tire testing machine [18]. After almost three decades, when the potential of parallel manipulators came to the spot light, the applications were oriented toward this idea. Some of those projects were Camera-Orienting device [19] and Satellite Positioning [20].
The Stewart Platform is a specific type of Parallel Manipulators which is designed at first with six legs and 6 DOF, but after conducted research, it is realized that the DOF can be decreased to 5,4 or 3. 6 DOF allows different possible position and orientation of the motion plate, which is only limited by the length of its legs and the range of motion of its joints. Figure 4.6 demonstrates the most common Stewart Platforms:

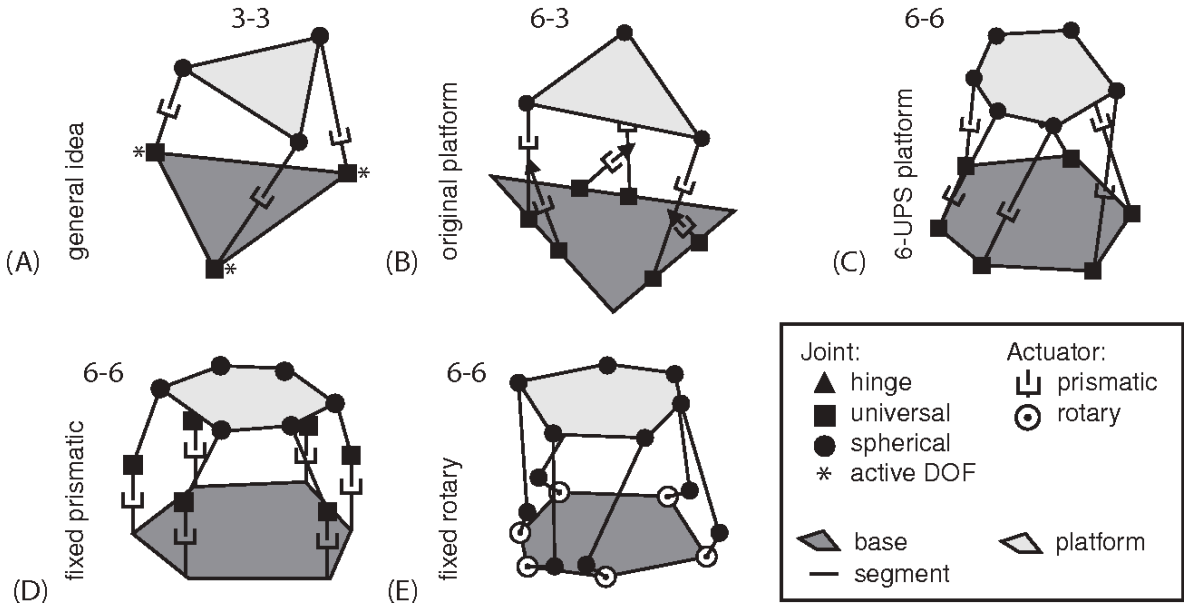


Figure 4.6: Stewart Platform types

Grübler formula [10], can be applied to calculate the number of the DOF for a closed loop parallel Kinematic Machine:

$$F_e = \lambda(l - j - 1) + \sum_{i=1}^{j} f_i - I_d \tag{4.6}$$

Where: F_e represents the effective DOF of the assembly or mechanism
λ stands for the DOF of the space in which the mechanism operates
l is the number of links
j describes the number of joints
f_i shows the DOF of the i joint
I_d is the idle or passive DOFs.

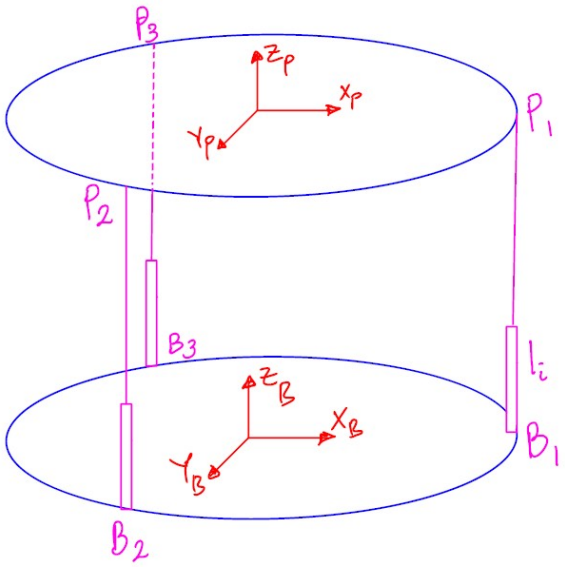


Figure 4.7: Kinematic Model of 3DOF Stewart Platform

Based on figure 4.7, each leg includes two links and both motion plate and base plate considered as one link. Hence, there are eight links in total. Six universal joints and three prismatic joints are present which makes the total number of joints to nine. The robot operates in 6 DOF in the space. Lastly, The number of DOF for each universal joint is two and each prismatic joint has one DOF which DOF becomes in total fifteen. Using 4.6, the total DOF can be calculated as following:
F_e = 6 (8-9-1) + 15 = 3
Hence, this specific model has three DOF.

Inverse Kinematic

The kinematics of Stewart Platform describes how the joint coordinates is related to the end effector. Two methods are known to find the position of the end effector and joint coordinates depending which one is given (known). Inverse Kinematic is being used when the end effector position is known and joint coordinates is unknown. In contrast, Forward Kinematic will give the end effector position when joint coordinates are known.
In general, Inverse Kinematic is a common way for these types of Parallel Kinematic machines since a non-linear closed form solutions to the Inverse Kinematic can be easily found. Contrarily, Forward Kinematic is the more intuitive way in Serial Kinematic Machines.

Converting α and β to the position of motors

Based on figure 4.8, this conversion can be done as following:

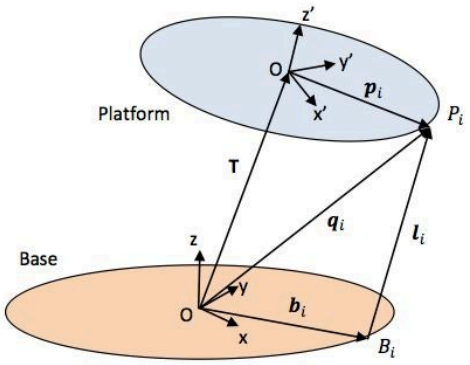


Figure 4.8: Stewart Platform component

Where:

T represents the vector that describes the length and direction from the origin of the base to the origin of motion plate

b_i is the vector of constants that shows the distance from the origin of the base plate to the actuator shafts

l_i illustrates the length of the motor to the motion plate

p_i is the distance from the origin of the motion plate to the end-effector pivot point

i represents actuators

q_i stands for Translation Vector ${}^{B}\vec{q} = [q_x, q_y, q_z]$ where q_x, q_y and q_z are the x, y and z components and it shows the translation from base plate to motion plate

B_i stands for the place where motor is located on the base plate

P_i is the location of end-effector pivot point on the motion plate

If the plates considered as a circle then P_i and B_i are known since motors are placed at 120°from one another and their location on the base plate and joints on the motion plate can be calculated with help of geometry.

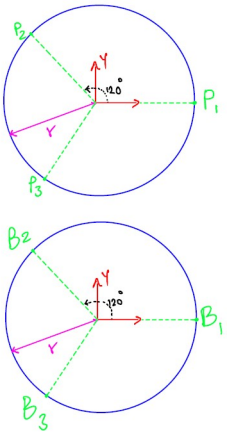


Figure 4.9: Motor's Position on Base Plate and Motion Plate

Based on figure 4.9 and given that $r = 0.17[m]$, P_i and B_i can be calculated as following:

$$ {}^{P}\vec{p_1} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \quad {}^{P}\vec{p_2} = \begin{bmatrix} \frac{-1}{2}r \\ \sqrt{\frac{3}{2}}r \\ 0 \end{bmatrix} \quad {}^{P}\vec{p_3} = \begin{bmatrix} \frac{-1}{2}r \\ -\sqrt{\frac{3}{2}}r \\ 0 \end{bmatrix} $$

$$ {}^{B}\vec{b_1} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \quad {}^{B}\vec{b_2} = \begin{bmatrix} \frac{-1}{2}r \\ \sqrt{\frac{3}{2}}r \\ 0 \end{bmatrix} \quad {}^{B}\vec{b_3} = \begin{bmatrix} \frac{-1}{2}r \\ -\sqrt{\frac{3}{2}}r \\ 0 \end{bmatrix} $$

The distance between the base plate to the motion plate which is represented by vector T is : $\vec{T} = [0, 0, 0.322]^T$. Next is the calculation of the change in the length of motors that is described by l_i and it is as following:

$$ \vec{l_i} = \vec{T} + ({}^{B}R_P \cdot p_i) - b_i \tag{4.7} $$

Where ${}^{B}R_P$ is the rotation matrix and will transform vectors in the Base frame to the ones in the Motion frames which is defined by using in the following order: rotation w.r.t Z-axis: yaw , rotation w.r.t Y-axis": pitch and rotation w.r.t X-axis: roll(based on [21]):

$$ RotZ = \begin{bmatrix} cos(\Psi) & -sin(\Psi) & 0 \\ sin(\Psi) & cos(\Psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8} $$

$$ RotY = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{4.9} $$

$$ RotX = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{4.10} $$

For a given input (Ψ, β and α) which are taken from the accelerometer readings, p_i vector values will change in the motion frame and this matrix is required to find those.
***Note** : This project does not have a rotation w.r.t Z-axis and therefore the angle $\Psi = 0$.

$$ {}^{B}R_P = RotZ \cdot RotY \cdot RotX \tag{4.11} $$

Back to equation 4.7, all terms have been found and now the change in the length motors can be calculated. The absolute value will be then :

$$ l_ipos = \sqrt{(l_i(1,1))^2 + ((l_i(2,1))^2 + ((l_i(3,1))^2} \tag{4.12} $$

Up until now, the change in actuator's sliders has been realized, meaning that the left block of figure 4.13 is completed. Next step is explaining how the right block of that figure works which is the opposite operation. This means that the angles will be found when the changes of actuator's sliders are known. It should be noted that these angles must be the same as the ones which used as input to the

actuators.

Converting the end effector's position to α and β
By finding the end effector position vectors (p_i) on the motion plate in figure 4.8 and transforming it to the base plate, the angles can be calculated back by using dot and cross product.
The equation to find mentioned vector is as following:

$$\vec{P_i b} = \vec{B_i} + (\vec{l_i} \cdot \Delta\vec{P_i}) - \vec{T} \tag{4.13}$$

in Equation 4.13, three right hand vectors are known from previous section and $\Delta P_{i_{ref}}$ is defined as the difference between the output and input of the motor's slider. This fraction must go to one, since the inner-loop controller steer the slider's position to the desired value. Following equation illustrates how this fraction can be calculated:

$$\Delta\vec{P_i} = \frac{\vec{P_i}}{\vec{P_{i_{ref}}}}$$

So, now that the end effector position vectors are calculated, next step will be finding the angles by using dot and cross products. This is a common problem in Robotic world. The method that is being used, is called "Tait–Bryan angles" referred in [22].
First the Unit Vectors of X-component of equation 4.13 ($p_1 b$) must be defined as following:

$$Unit\vec{X}p = \frac{\vec{p_1 b}}{\sqrt{(\vec{p_1 b} \cdot \vec{p_1 b})}}$$

Next, by operating cross product between unitary vector $UnitXp$ and Z-component of equation 4.13 ($p_3 b$) the normal vector can be found which is the projection of Z-component. Then the unitary vector will be calculated:

$$\vec{Z}p = Unit\vec{X}p \times \vec{p_3 b}$$

$$Unit\vec{Z}p = \frac{Unit\vec{Z}p}{\sqrt{(Unit\vec{Z}p \cdot Unit\vec{Z}p)}}$$

Finally, the cross product between $UnitXp$ and $UnitZp$ will be applied to find the projection of Y-component on the base plate (***Note**: This vector is already a unit vector, since the other two vectors are unity vectors):

$$Unit\vec{Y}p = UnitXp \times UnitZp$$

Refer to [22] and calculated equations above, angles can be found as follow:

$$\beta = \arcsin\left(-UnitXp(3)\right) \tag{4.14}$$

$$\Psi = \arcsin\left(\frac{UnitXp(2)}{\sqrt{1 - (UnitXp(3))^2}}\right) \tag{4.15}$$

$$\alpha = \arcsin\left(\frac{UnitYp(3)}{\sqrt{1 - (UnitXp(3))^2}}\right) \tag{4.16}$$

4.3.1 Validation of transformation

There are two points that are needed to be controlled, namely:

- In/Out Angles
- T Vector

The first point will be proven in the next section in figures: 4.34,4.35,4.43 and 4.37.
Second check point is the T vector. As mentioned earlier, this project does not use rotation w.r.t Z-axis, therefore T vector in output must be precisely the same as input. This leads to the fact that the angle Ψ will remain zero during the whole operations above. To confirm this, it is needed to calculate the vector T back. Since angles are calculated in previous section, the rotation matrix ${}^{B}R_{P}$ can be found by equations 4.8, 4.9, 4.10 and finally 4.11. Afterwards :
$\vec{T_{back}} = l_1 + B1 - ({}^{B}R_{P} \cdot P1)$
Figure 4.10 illustrates that this spec is not violated.

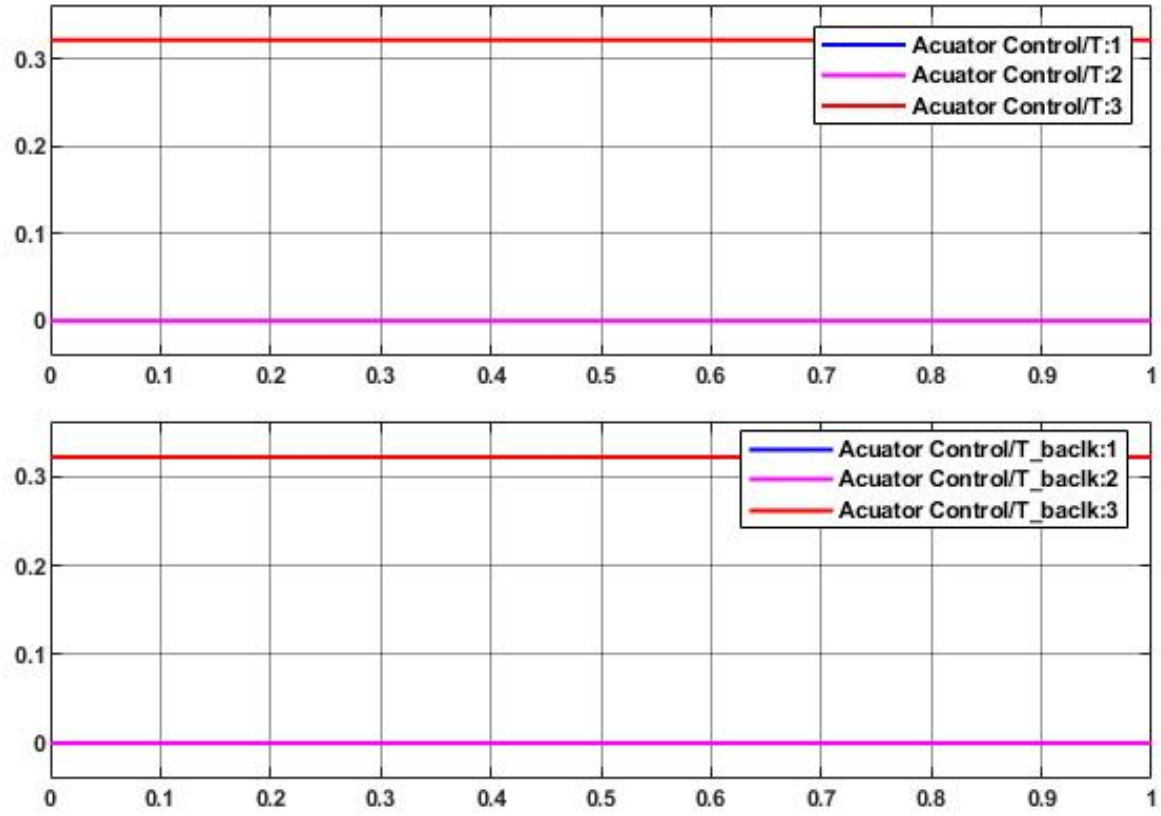


Figure 4.10: T and T-Back

4.3.2 Validation Translation in SIMULINK vs. Autodesk Inventor pro

In this section angles and positions resulted in SIMULINK will be compared to the Mechanical Simulation program which is more close to the practice model. By defining $r = 0.17m$ and $\vec{T} = [0, 0, 0.322]$ following table demonstrates the actuators position.

Alpha (deg)	Beta (deg)	Psi (deg)	Position motorA (m)	Position motorB (m)	Position motorC (m)
0	**2**	**0**	0.3161	0.325	0.325
0	**4**	**0**	0.3101	0.3279	0.3279
3	**0**	**0**	0.322	0.3297	0.3143
7	**0**	**0**	0.322	0.3399	0.3041
9	**6**	**0**	0.3042	0.3538	0.308
10	**10**	**0**	0.2925	0.362	0.3116
0	**10**	**0**	0.2925	0.3368	0.3368
10	**0**	**0**	0.322	0.3476	0.2964

Table 4.2: Translation of Angles to Position

Next, actuators positions are given to the Mechanical simulator and results are illustrated in table 4.3

Position motorA (m)	Position motorB (m)	Position motorC (m)	Alpha (deg)	Beta (deg)	Psi (deg)
0.3161	**0.325**	**0.325**	0	2	0
0.3101	**0.3279**	**0.3279**	0	4	0
0.322	**0.3297**	**0.3143**	3	0	0
0.322	**0.3399**	**0.3041**	6.98	0	0
0.3042	**0.3538**	**0.308**	8.94	6.01	0.47
0.2925	**0.362**	**0.3116**	9.84	10	0.86
0.2925	**0.3368**	**0.3368**	0	10.01	0
0.322	**0.3476**	**0.2964**	10.01	0	0

Table 4.3: Translation of Position To Angles

As it can be seen in table 4.3, results seem most likely the same as table 4.2. However, there are two points that should be noticed, namely:

- rounded values
- Values for Angle Psi

Firstly, due to trigonometry functions which has been used, it can be seen in table 4.3 that some differences but it is negligible since the variation is very small (a fraction of hundred). Secondly, it is observed that the angle Psi does not remain zero fro some cases.Although the value is small, it could be harmful to the design since this change means that there is a rotation w.r.t. Z-axis. However, in practice this problem can be neglected, due to the fact that motion of the motion plate is limited to only X and Y-axis by the design.

4.4 Ball position control (outer loop)

There are two steps to approach a design for the outer-loop controller. Since the position of the ball will be monitored and tracked by a camera and it has a sample frequency of 10 Hz, controller should be designed in discrete domain to take the sample time into account. The method which is going to be used is design the controller in continuous time (S-domain) and then convert it to a controller in discrete domain (Z-domain). However, this method will not guaranty a proper response as it is expected, but in case of failure, the whole design should be designed in discrete domain. Following sections will analyze this issue in detail.

4.4.1 Ball position controller

There are 3 requirements which should be considered for the design, namely:

- Band Width limit
- Zero Steady State Error
- Overshoot less than 25%

Figure 4.1 has a known shape in Control Systems world. It is called Cascade Loop and as it can be seen, it has two loops: one loop within a loop. It is obvious that the inner-loop must be much faster than the outer-loop, since the data in inner-loop must be much faster treated and be ready for the outer-loop. So, because the BW for the actuator control block is 47.5 Hz ($300 rad/s$), the response for the outer-loop seems to be 0.01 times slower. However, to check whether this method really applies to this project, four controllers with different BW has been designed. The result was expected, since the controller with BW = 0.49 Hz worked properly. Another point with regard to BW, is the location of zeroes and poles. The closer the poles to the origin, the slower the response and vise versa. The zeroes and poles locations can be found in the table 4.4.
Zero steady state error is the second requirement and this problem is easily satisfied by adding an integrator to the controller.
Next requirement is having a proper overshoot which also has a relation with the pole locations.In case of higher order systems (higher than two), the two closest poles to the origin are dominant. This means that the other poles will decay very fast and they do not have a major role on how system behaves. Hence, any high order system can be roughly analyzed as a system with two poles. The relation between the poles and overshoot then as follows:
The closer the poles to the x-axis the smaller angle and therefore the smaller overshoot.
Following is a table of four different controllers and a figure of step response to the closed-loop:

Controller	Poles Location	Zero Location	BW (Hz)	ZSE	Overshoot%
C2.1	0, -800, -70	-0.3728, -0.007127	0.31	Yes	15.1
C2.2	0, -1100, -160	-0.01, -0.03	0.49	Yes	1.67
C2.3	0, -1000, -225	-0.5319, -0.007127	1.51	No	6.04
C2.4	0, -150, -15	-0.004533, -0.001779	2.23	Yes	21.8

Table 4.4: Specifications of designed controllers

***Note**: ZSE refers to Zero Steady State Error.

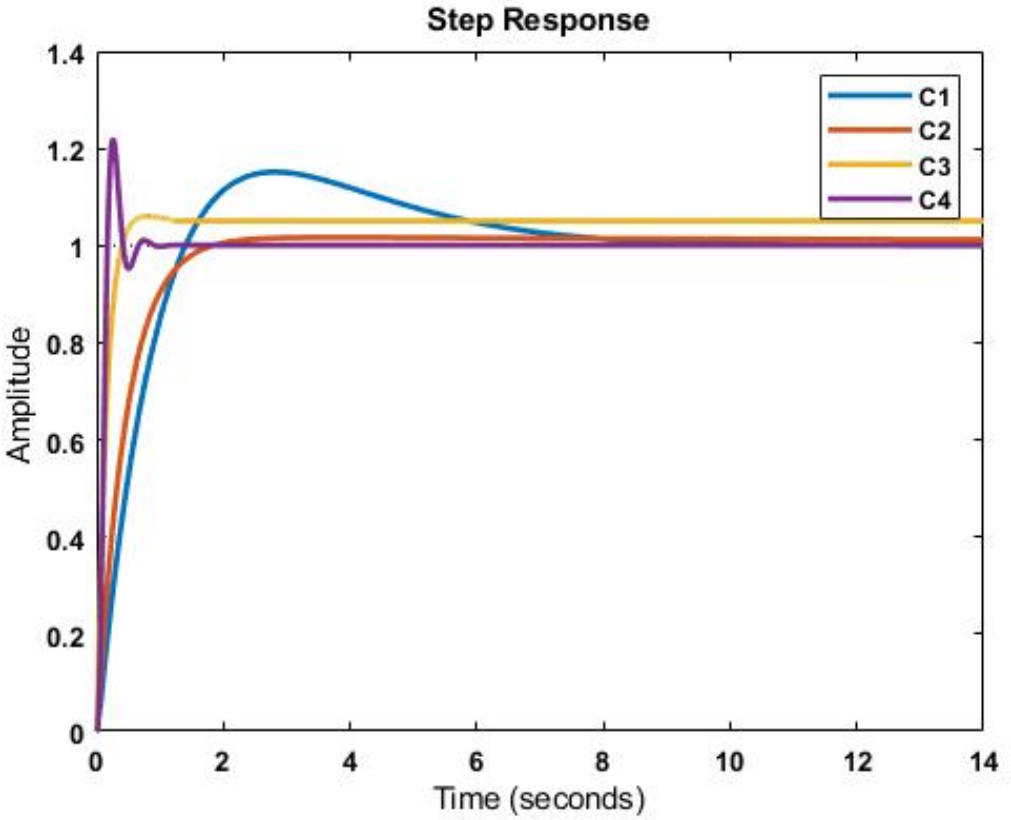


Figure 4.11: Closed-Loop capture of four designed controllers

Considering the table 4.4, figure 4.11 and design arguments, it can be concluded that C2.2 satisfies all the requirements.
Hence, the chosen controller is:

$$C2(s) = \frac{55000 \cdot (s + 0.03) \cdot (s + 0.01)}{s \cdot (s + 1100) * (s + 160)} \tag{4.17}$$

Next step is changing the inner-loop controller to the given model, which will be discussed in the next section.

4.4.2 substitute the original inner-loop controller

Now that the outer-loop controller is designed (equation 4.17), the given model of inner-loop will be substituted in figure 4.1 and as result after implementation in Simulink, the complete system will look like as follow:

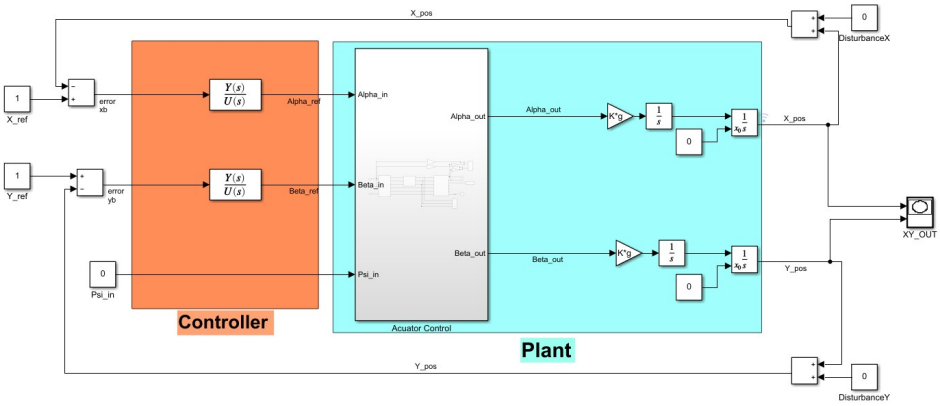


Figure 4.12: Complete System in Simulink

Where: the sub-system includes three main blocks, namely: Angle to Position function,Actuator Control block and Position to Angle function.

$\frac{Y(s)}{U(s)}$: is the numerator and denominator of the outer-loop controller

P_i (i:A,B and C) : is de position of motor's slider4.13

$Plant = \frac{K \cdot g}{s^2}$: is the plate's plant

Following figures are sub-system's blocks:

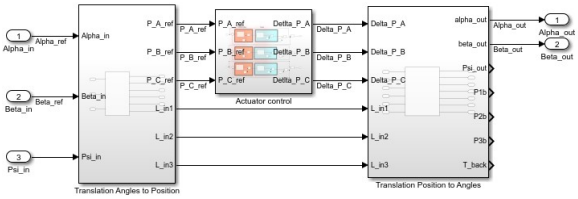


Figure 4.13: Actuator Control Block

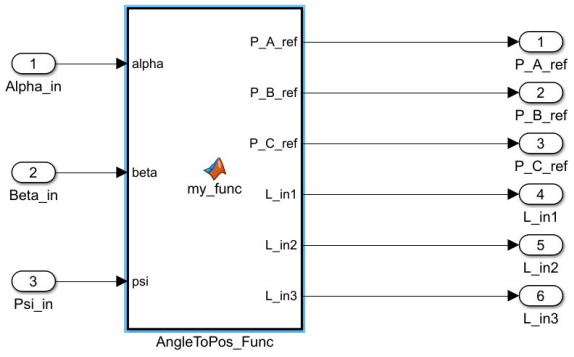


Figure 4.14: AngleToPos function

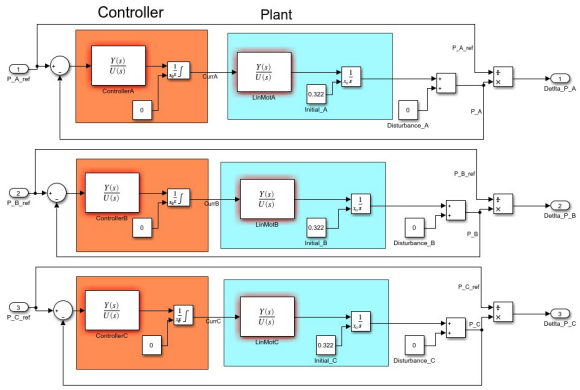


Figure 4.15: Actuator Controller

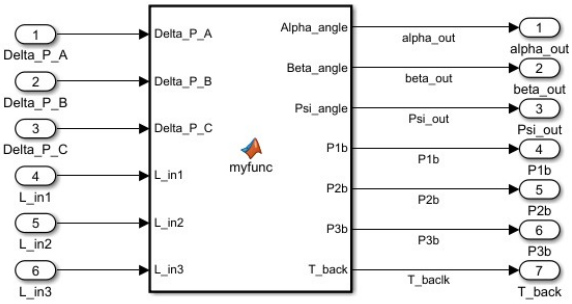


Figure 4.16: PosToAngle function

Figure 4.13 refers the main three blocks of the sub-system, while figures 4.14, 4.15 and 4.16 refer the detail of each blocks. Figures 4.14 and 4.16 as mentioned earlier are translations of position to angles (and vise versa) and are already explained. As can be seen in figure 4.15, there are three similar pairs of given motor model (controller and plant) for each motor. Implemented controller is similar to the original controller minus the integrator for dSpace implementation. The limited integrator is added as separate block, where the saturation limits can be set to avoid integrator wind-up. The range of current which each motor is allowed to be fed by a power supply is between $[-3, 3]$ A. This range is applied to the limited integrator.
Further a disturbance block for each one of them is added, representing the difference between motors in terms of specifications. This means that motors are not precisely the same and by adding the disturbance, this difference will be taken into account when motors are working.

4.4.3 verification of outer-loop controller with given inner-loop controller

To verify the controllers, two types of inputs are given. A rectangle which is a combination of steps, and a circle that includes sine waves with different phase. The inputs could have large amplitudes for the sake of theoretical analysis, but in practice the dimension of the motional plate (40x40 cm^2) should be considered.

Rectangle input

Following figures illustrate how system reacts upon such a reference input:

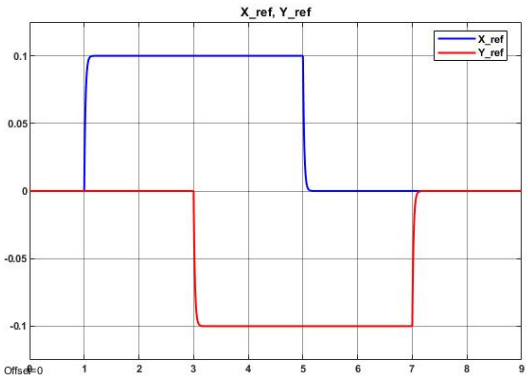


Figure 4.17: Reference Input

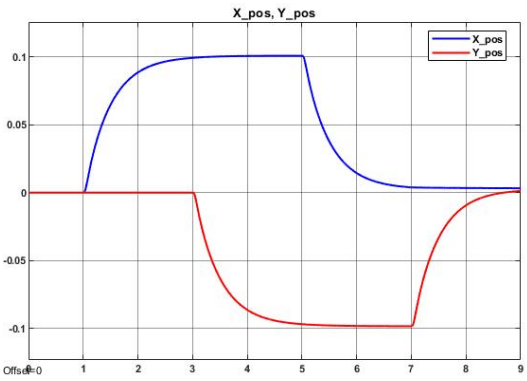


Figure 4.18: Position output

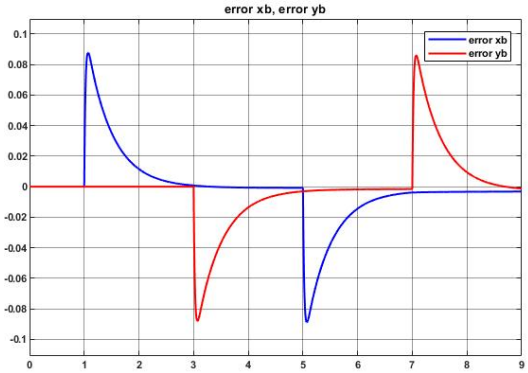


Figure 4.19: Error between the reference and output

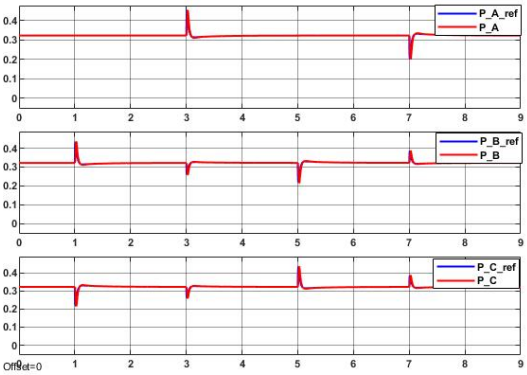


Figure 4.20: Actuator position- reference and output

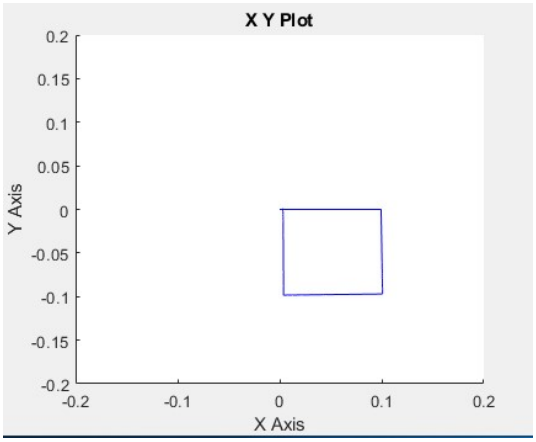


Figure 4.21: XY Plot from the output

Circle

Following figures illustrate how system reacts upon such a reference input:

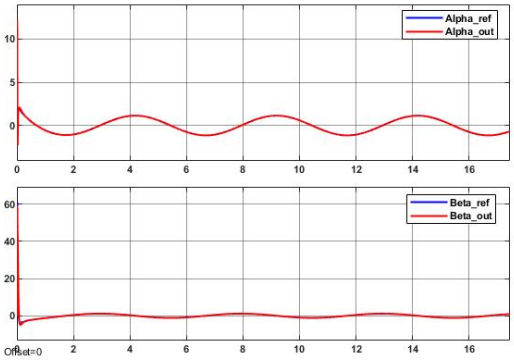


Figure 4.22: Angles-in-out-Circle

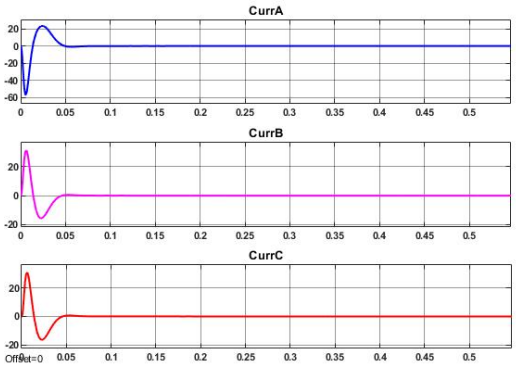


Figure 4.23: Actuator-Currents-Circle

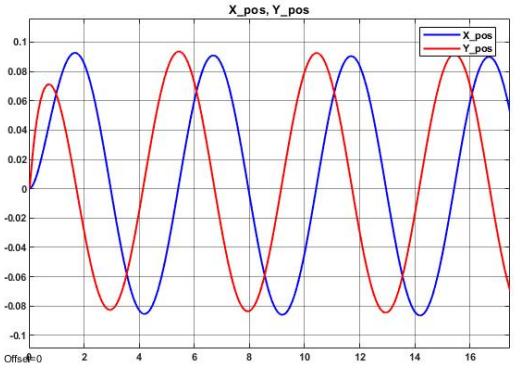


Figure 4.24: Output-position-Circle

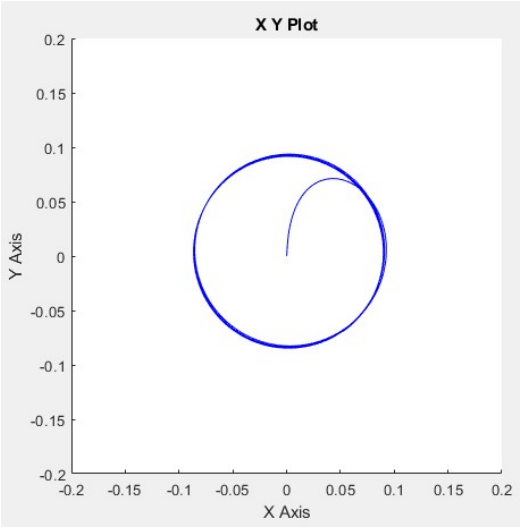


Figure 4.25: XY-Plot-Circle

Analyze

In this section, rectangle as a reference is going to be analyzed since the result of behavior of the system to both references is similar except two cases. First, due to the type of reference (Circle), there is no corners(no sudden changes in the position all the way up/down in comparison to previous position) to check the overshoots and violation of the rules happens only at the initialization time in figure 4.22. Second, for the same reason, violation for the limited range for the currents, also happens at initialization time in figure 4.23. However, based on figure 4.25, it can be observed that the system is able to make such a shape.This problem can be solved by letting actuators start from their operation points which is 322 mm.
Back to the rectangle reference, it seems that the system is stable and both controllers for the inner-/outer loop work properly. Figure 4.20 shows that the actuator's position starts from zero and it will be settled in 0.322 [m] (which is the desired position based on design set-up) which means inner-loop controller is working well (but not properly, though).
Further, as it can be seen in figures 4.17 and 4.18, the designed model can follow the reference and a successive zero steady state error can be depicted from figure 4.19. Lastly, the final result is demonstrated in figure 4.21.
Although above figures show that the system is working well (beside the overshoots in final result), there are two major problems which are shown as following:

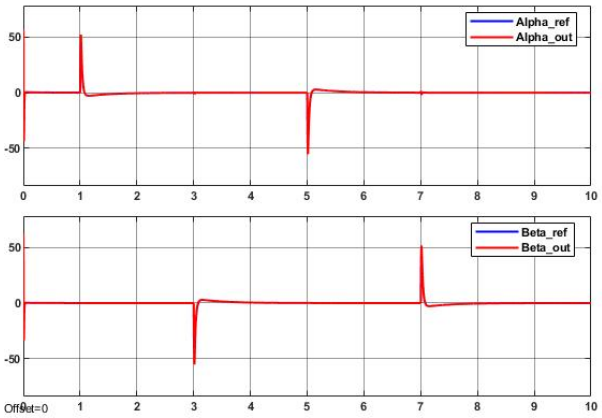


Figure 4.26: Reference angles and result

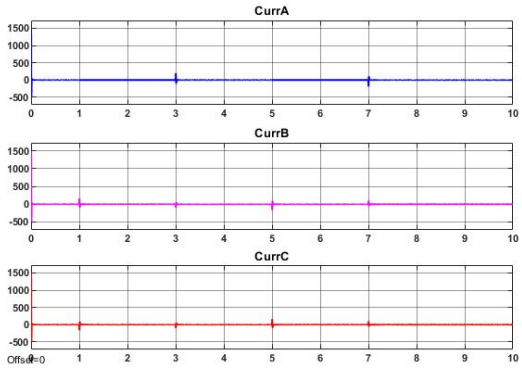


Figure 4.27: Actuators Current

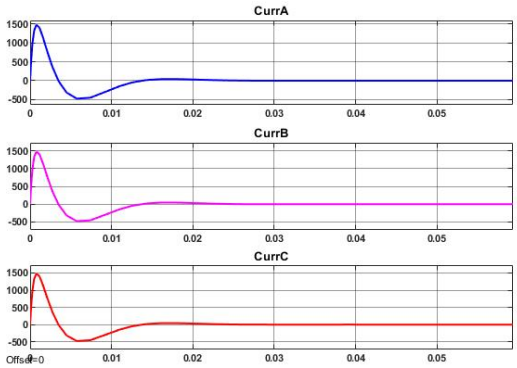


Figure 4.28: Actuators Current, Zoomed-in

As shown in figure 4.26, the magnitude of angles is 50 degrees, which is completely unacceptable. The figures 4.27 and 4.28 demonstrates violation of the current range. This happens five times: first at time [0 to 0.2 seconds] and then for each corners that rectangle makes. There are two solutions for these problems.

- Make the motors work slower
- Adjust the reference input

The first options is applied for a BW of 4 Hz for the motors, 0.15 Hz for the outer-loop controller and results are as following:

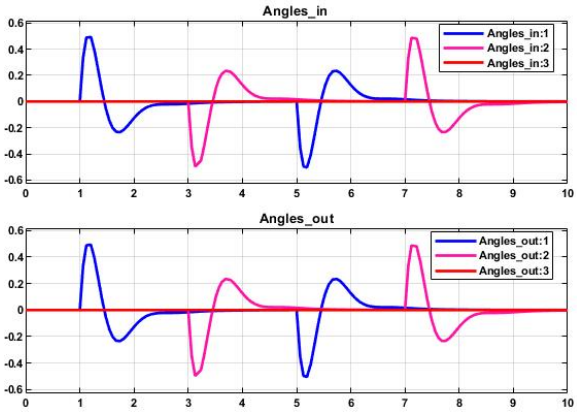


Figure 4.29: Angles-in-out-rect-slow motor

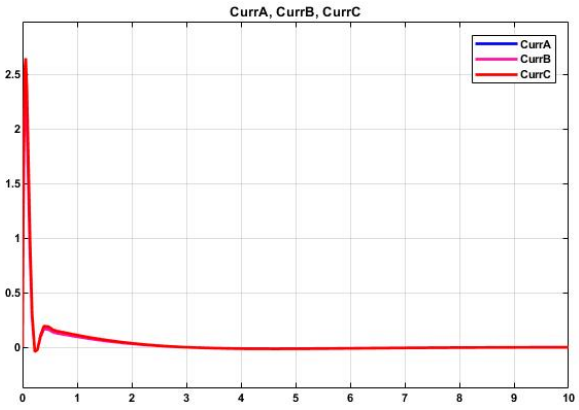


Figure 4.30: Actuators Current- slow motor

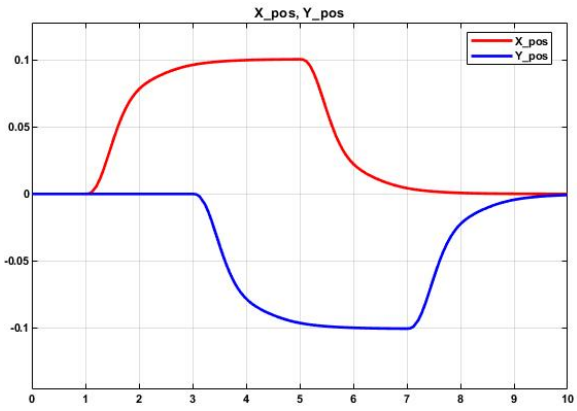


Figure 4.31: Output-position-rectangle-slow motor

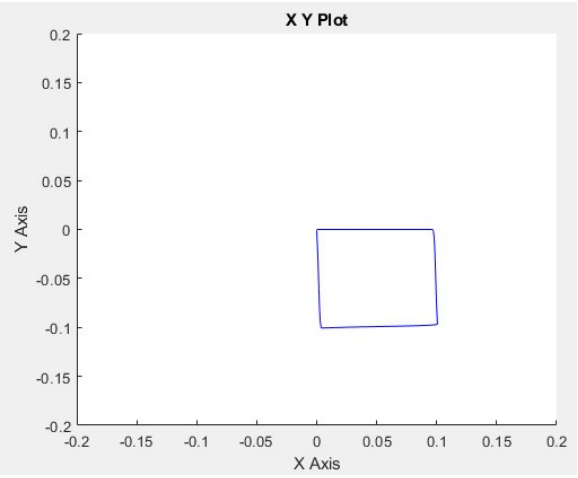


Figure 4.32: XY-plot-rectangle-slow motor

Although the first solution will resolve all the mentioned issues (based on figures 4.29,4.30,4.31 and 4.32), but it then violates the rule of cascade control loop which states that the inner-loop must be much faster than the outer-loop. Also it should be noted that the motor works very slow now and it is not logical to use such BW. To find a proper BW for both controllers, some combinations have been applied and the outcome is illustrated in tables 4.5 and 4.6.

Inner-Loop Controller			
BW (Hz)	**Initialization issue disappear: Yes/No**	**Angles issue disappear: Yes/No**	**Speed is logical: Yes/No**
0.63	Yes	Yes	No
15.91	Yes	No	Yes
47.74	Yes	No	Yes

Table 4.5: BW Comparison Inner-Loop Controller

Outer-Loop Controller			
BW (Hz)	**Initialization issue disappear: Yes/No**	**Angles issue disappear: Yes/No**	**Speed is logical: Yes/No**
0.15	Yes	Yes	No
0.5	Yes	No	Yes
5	Yes	Yes	No

Table 4.6: BW Comparison Outer-Loop Controller

***Note:** Initialization issue refers to figures 4.23 and 4.27 between [0-0.3 seconds], where the graph aggressively grows.

It is already discussed why 0.15Hz and 0.63Hz in tables above, are inappropriate. In table 4.5, to choose between the second or third option, based on what has been observed from how smoothly motors work, it is decided to choose 300 rad/s as its BW.
In table 4.6, the third alternative is not a good BW, since it violates the rule of cascade loop control. Hence, the second option is the best BW.
Back to the possible solutions mentioned above, it is observed and discussed that change of BW will not solve the problems and it is not a good solution. Therefore, the second option is going to be applied to the system, which is adjusting the reference input and that brings up the idea of designing a low pass filter.

Low Pass Filter

The reason for designing a LPF is that the used reference input is a combination of step functions. These functions include all frequencies and therefore, it is needed to be filtered. There are three factors which define the range of desired frequencies. Two of them were determined in previous chapters and there is one more limit for this system and that is the camera. It seems that the maximum BW a camera has, is 10 FPS. All constraints are lined up as following:

- Camera $\rightarrow$ 10 Hz
- Outer-Loop Controller $\rightarrow$ 0.5 Hz
- Inner-Loop Controller $\rightarrow$ 47.74 Hz
- Angles range for an easier mechanical design $\rightarrow$ 12.6 degrees

- Angels range for a more difficult and more expensive mechanical design → 25 degrees

***Note**: The last two items were simulated in Autocad by EPC team and it should be noted that the one with maximum 12.6 degrees has been chosen due to the fact that it costs lower than the second option.

So, designing a filter at which the cut-off frequency is higher than 10 Hz, is not meaningful since the camera works at 10 Hz. However, for the sake of theoretical analysis, five types of first order low pass filters are going to be designed and tested and the results are shown in table 4.7 and figures 4.33 up until 4.48.

Low Pass Filter BW (Hz)	**Initialization issues disappear: Yes/No**	**Current range(A)**	**Angles range (degrees)**	**Overshoot Percentage**
0.7	Yes	[-0.45 , 0.45]	[-2.7 , 2.7]	-
1	Yes	[-0.88 , 0.88]	[-4 , 4]	-
2	Yes	[-3.35 , 3.35]	[-8 , 8]	1.5
5	Yes	[-18.5 , 18.5]	[-21.5 , 21.5]	1.6
20	Yes	[-169 , 169]	[-65 , 65]	-

Table 4.7: Low Pass Filters specs

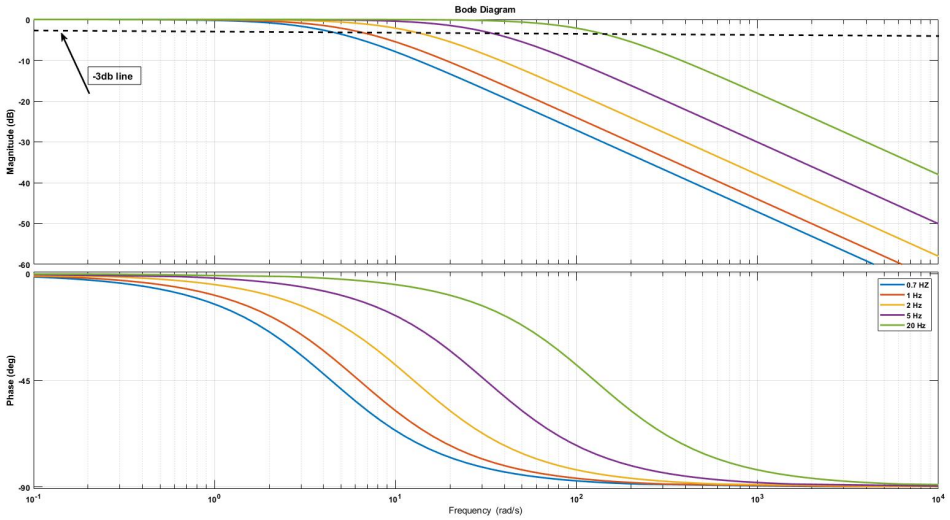


Figure 4.33: Bode plot of LPFs

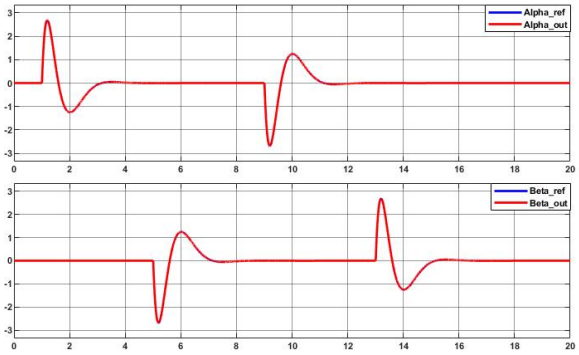


Figure 4.34: Angles-in-out-0.7Hz

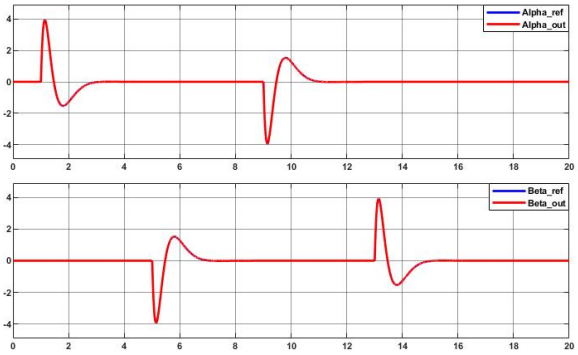


Figure 4.35: Angles-in-out-1Hz

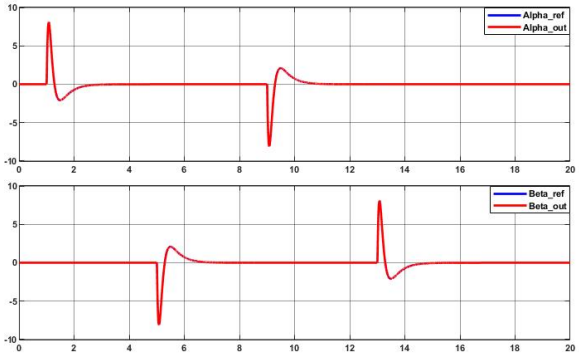


Figure 4.36: Angles-in-out-2Hz

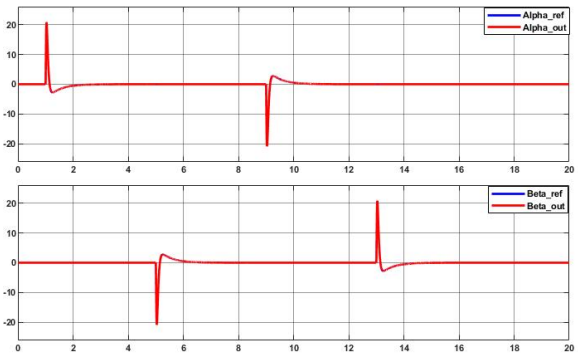


Figure 4.37: Angles-in-out-5Hz

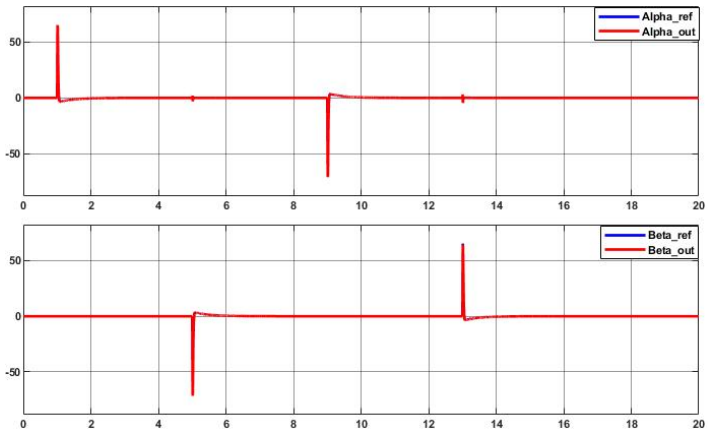


Figure 4.38: Angles-in-out-20Hz

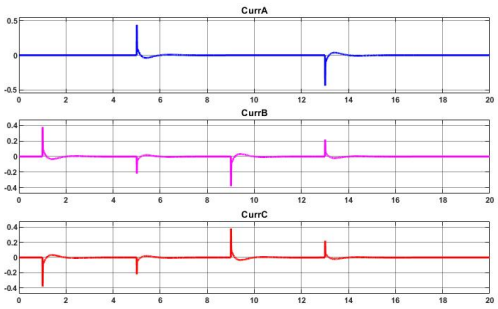


Figure 4.39: Actuator Currents-0.7Hz

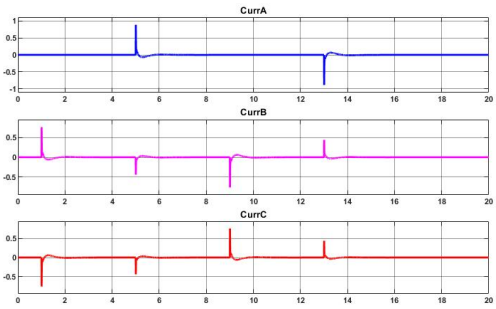


Figure 4.40: Actuator Currents-1Hz

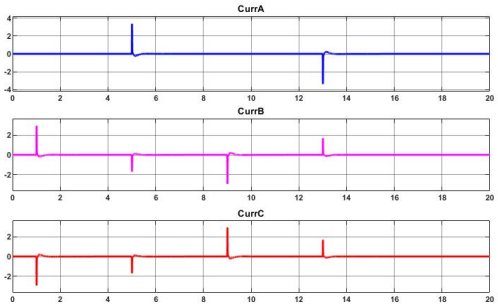


Figure 4.41: Actuator Currents-2Hz

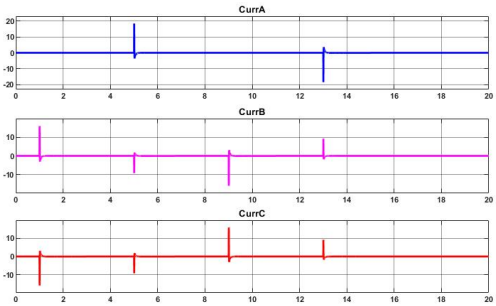


Figure 4.42: Actuator Currents-5Hz

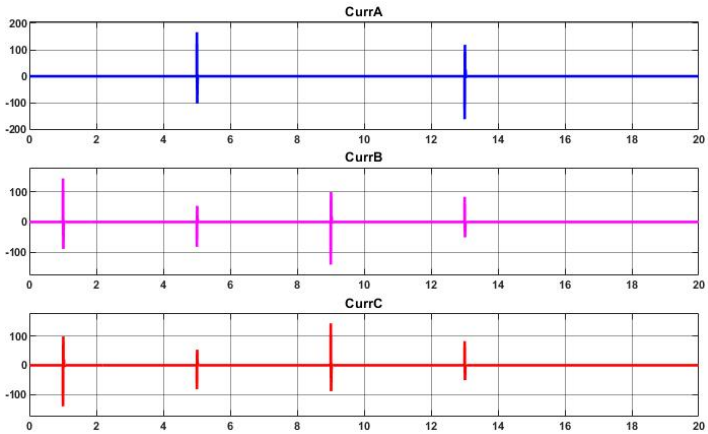


Figure 4.43: Actuator Currents-20Hz

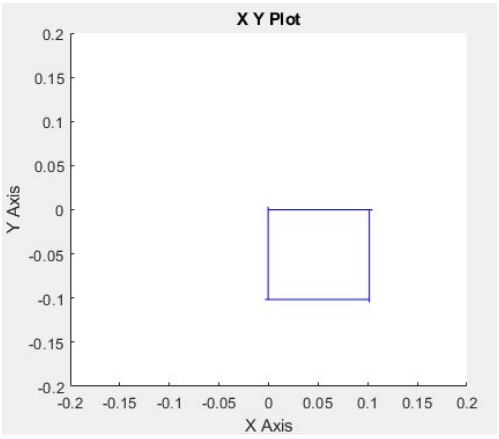


Figure 4.44: XY plot-0.7Hz

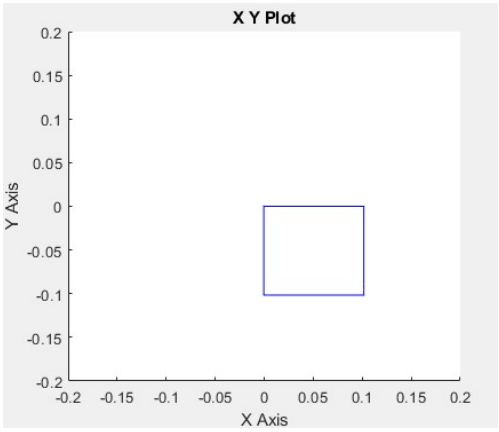


Figure 4.45: XY plot-1Hz

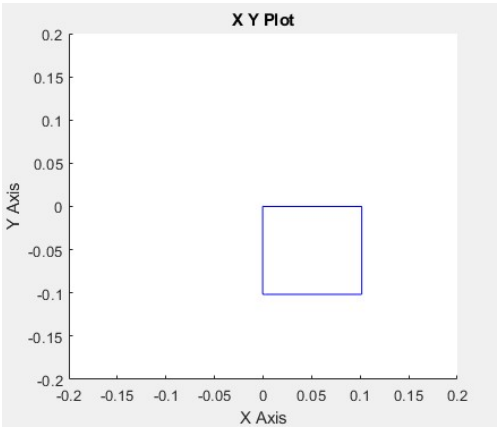


Figure 4.46: XY plot-2Hz

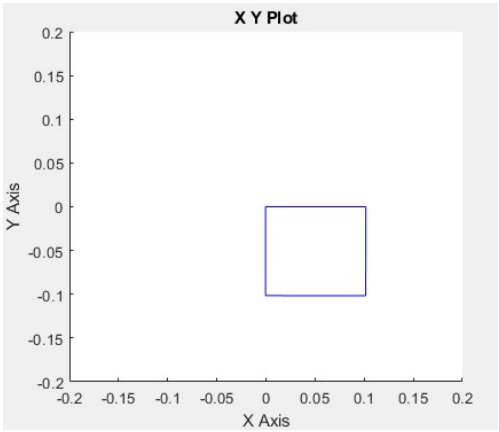


Figure 4.47: XY plot-5Hz

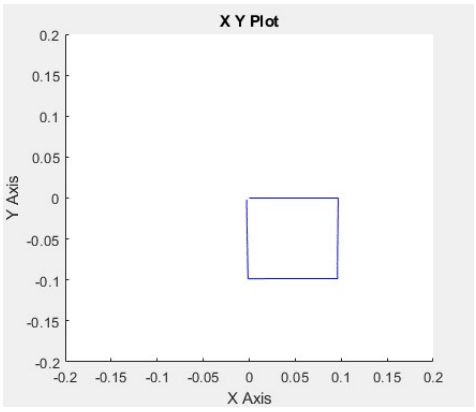


Figure 4.48: XY plot-20Hz

XY plots, Actuator's current and In/Out Angles figures and table 4.7 confirm that the filters greater than 2Hz have an unsatisfying results, such as violation in the range of Angles and in actuator's current range. However in XY-plots, it can be seen that using all these filters will not violate the tracking of the reference, but at which cost is more important. For instance, using a 5Hz filter will need 18.5 Amps, which is far greater than the range. Also the angles that plate is making to follow the reference is not realistic (21.5 degrees for each corner).
Based on above figures, it can be seen that the filters which are working at 1 and 2Hz, might satisfy the desired ranges. The second option draw too much current (even more than the range) and it has an overshoot (negligible, but still it is there), whereas the the filter with 1Hz fulfill all the requirements.
All in all, controllers are now working properly, but when it comes to implementing and integrating this system in the real world applications, designed system should be either converted into discrete domain or the whole system should be designed in discrete domain, which will be discussed in the next section.

4.4.4 Discrete system

To consider the role of frequency response of system, it is required to design the outer-loop controller in discrete domain. As it is stated earlier, camera is the component which has the dominant frequency (10Hz). Hence, this is the sampling time to design the controller in discrete domain. There are two methods to approach designing such controller, namely:

- Convert the current controller to discrete domain without converting ball and plate's plant and inner-loop block to discrete domain
- Design total system in discrete domain

Generally, first method would work and satisfy the requirements, but there are cases that might cause some violation in the ranges for angles and currents or overshoot. In the next sections, one method will be chosen, based on analyzing the behavior of the system.

Convert outer-loop controller from continuous domain to discrete domain

Discritization can be done in different methods [23]. This project uses Zero Order Hold (ZOH) which is a proper way for systems with staircase inputs. Further it should be noted that a sampling time of 0.1 second is used w.r.t camera. C2d command in MATLAB will lead to the following controller in discrete domain:

$$C2_{discrete} = \frac{0.012515(z - 0.9993)(z - 0.0005265)}{(z - 1)(z + 0.8841) * (z + 0.2987)} \tag{4.18}$$

After applying this controller to the system with regarding to Zero Order Hold block, it has been observed that all ranges were totally unacceptable. Following figures are illustrating the response of the system upon a rectangle input.

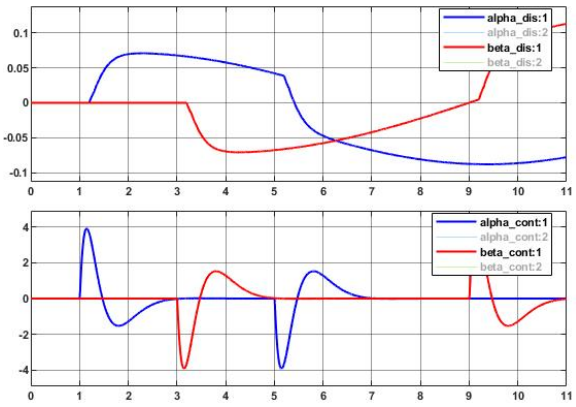


Figure 4.49: Angles-Discrete and Continuous

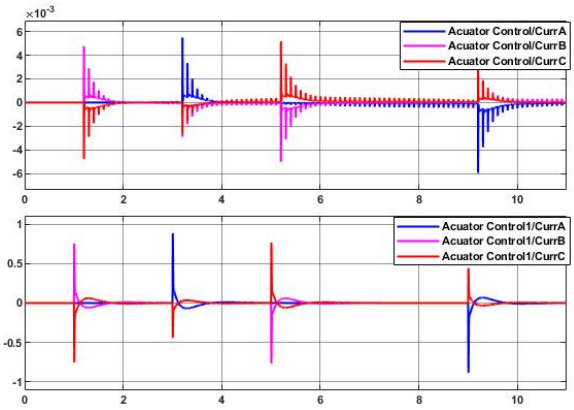


Figure 4.50: Currents-Discrete and Continuous

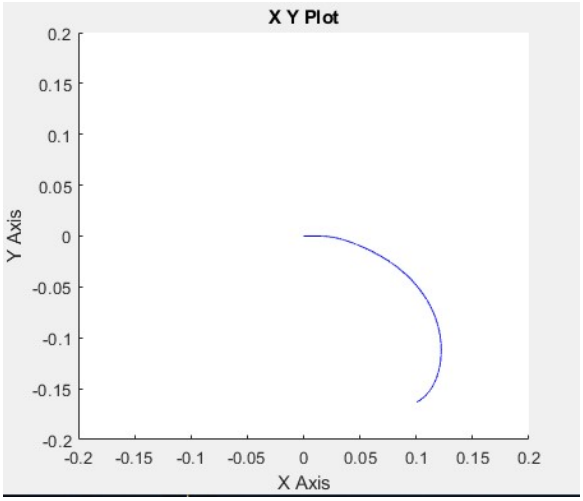


Figure 4.51: XY plot- Discrete

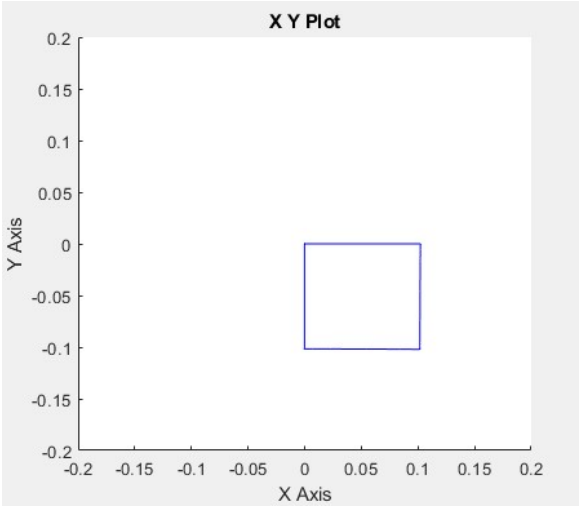


Figure 4.52: XY plot- Continuous

Based on above figures, it is needed to try the second approach, where the total system is going to be designed in discrete domain.

Design total system in discrete domain

To improve response frequency of system, it is needed to convert system's plant to discrete domain. Notice that based on figure 4.2, plant is as follows:

$$Plant = \frac{Km}{\tau s + 1} \cdot \frac{Kg}{s^2} \tag{4.19}$$

Where the first term is equation 4.5 and second term is equation 3.31. Mapping these equations considering values for constants and sampling time of 0.1 second in discrete domain, leads to :

$$Plant_{discrete} = \frac{0.22508(z + 1.207)(z + 0.004921)}{z(z - 1)^2} \tag{4.20}$$

Where following figure demonstrates the root-locus of the plant.

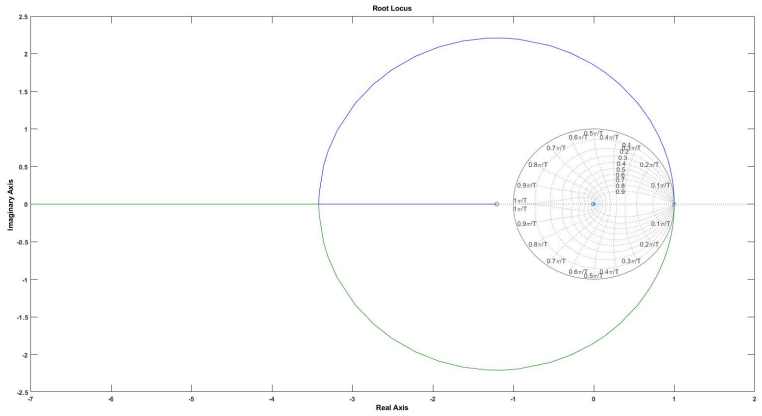


Figure 4.53: Root-Locus Plant

Instability of system is clear based on figure 4.53, since two poles are lying on the boundary. With help of sisotool feature in MATLAB, a controller has been designed, which equation 4.21 will describe the poles ans zeroes locations and figure 4.54 illustrates the Root-Locus of the system's closed-loop.

$$C2_{discrete} = \frac{1.9(z + 0.8546)(z^2 - 1.974z + 0.974)}{(z - 1)(z - 0.5504)(z + 0.9429)} \tag{4.21}$$

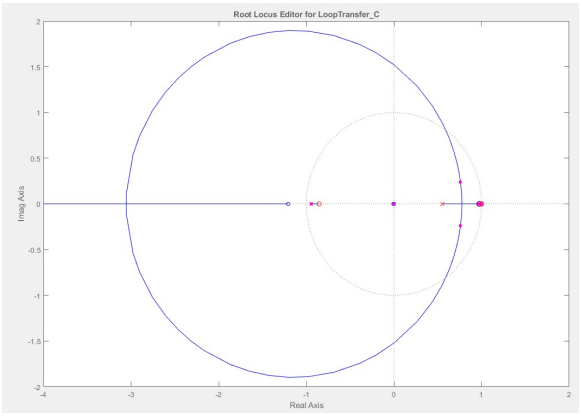


Figure 4.54: Root locus of the closed-loop

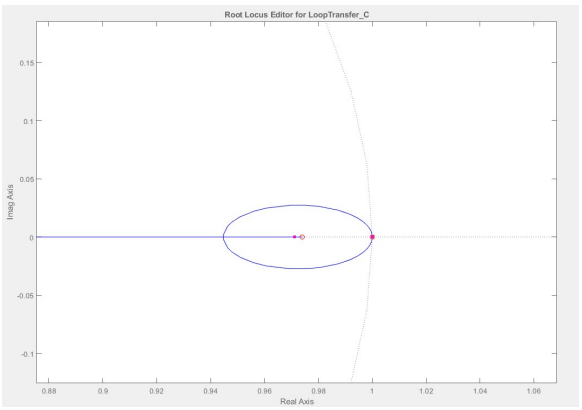


Figure 4.55: Zoomed in- Root locus of the closed-loop

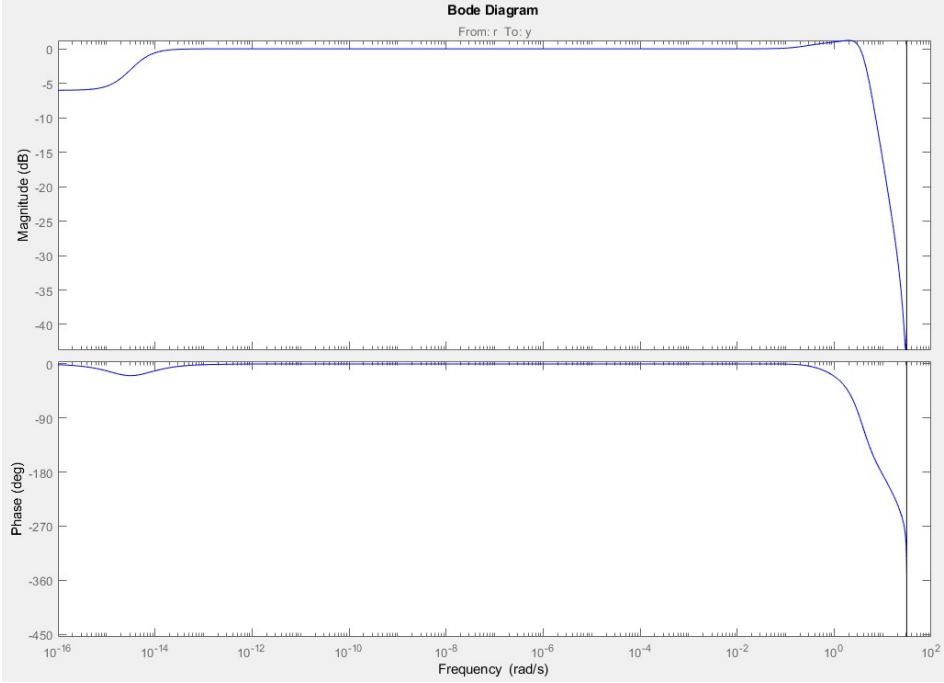


Figure 4.56: Bode diagram closed-loop

Now that a part of Root-Locus is within the stable region, by choosing a proper gain, overshoot issue which is stated in previous section will be solved.

***Note** : The same approach should be considered to simulate system, meaning that now the original inner-loop controller and plant should be placed in SIMULINK blocks and C2 block will be changed based on equation 4.21 in figure 4.1.

Following figures show how system reacts, considering the new designed controller.

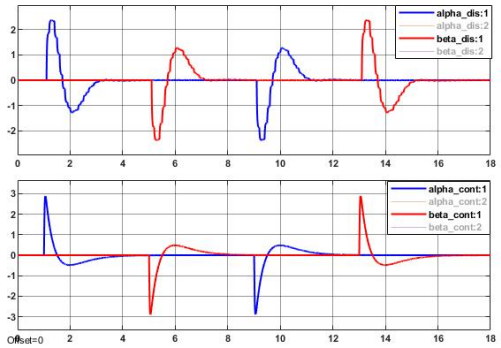


Figure 4.57: Angles-Discrete and Continuous

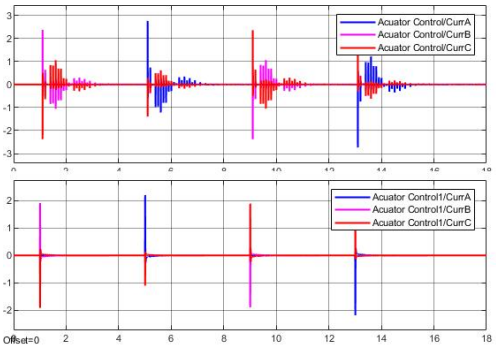


Figure 4.58: Currents-Discrete and Continuous

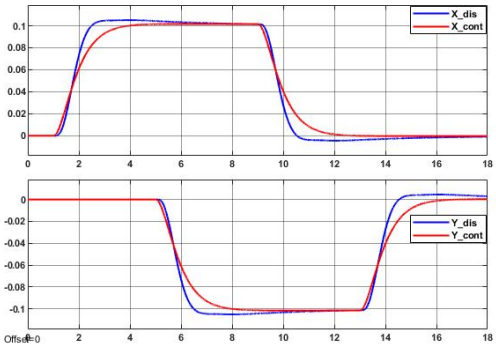


Figure 4.59: Position-Discrete and Continuous

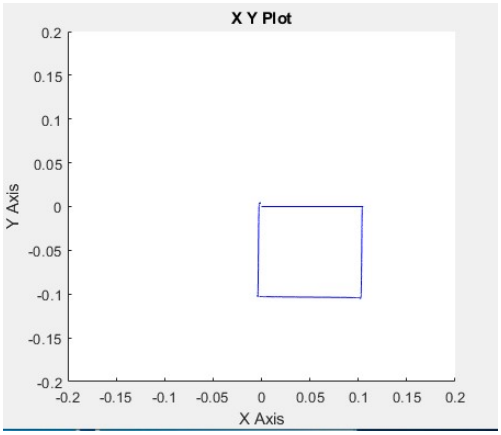


Figure 4.60: XY plot-Discrete

As it can be seen, requirements are satisfied by using this controller. Specifications that should be considered for such a reference are as follow:

- Filter after steps input has a frequency of 0.3 Hz
- Filter before actuator block has a frequency of 10 Hz
- Bandwidth of the inner-loop is 47.75 Hz
- Sample time is 0.1 s

It should be noticed that for having a rectangle as a reference, it is essential to add a low pass filter (beside the low pass filter as the input for actuators) to make the steps smoother and hence, smaller overshoot will be the result, which leads to draw less current for actuators.
Up until now, it is proved that the designed controller in discrete domain works properly and it should operate for all types of inputs. The only important point, when it comes to tuning the system, is considering which reference is going to be given to the system.
Following figures are the results of a circle as the reference:

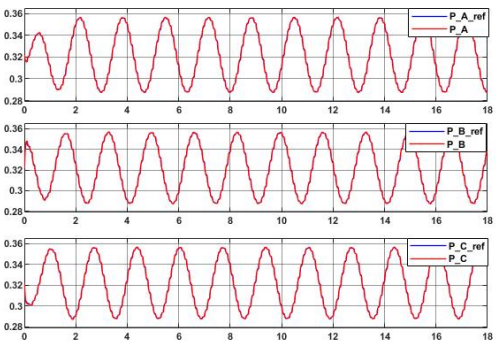


Figure 4.61: Actuators position

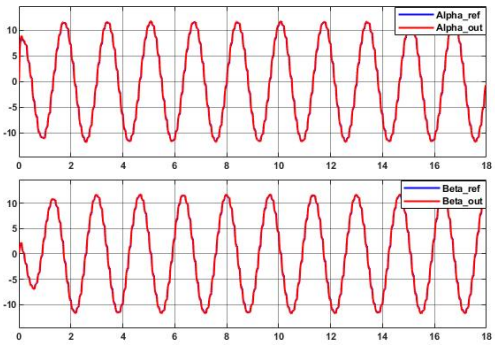


Figure 4.62: Angles

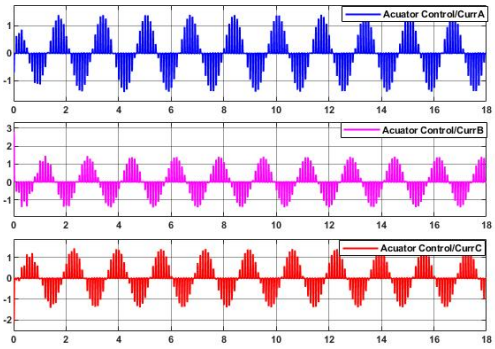


Figure 4.63: Actuators currents

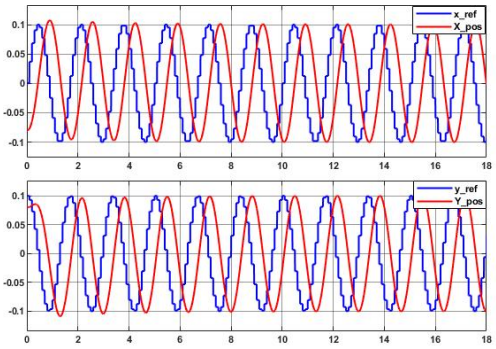


Figure 4.64: Ball position

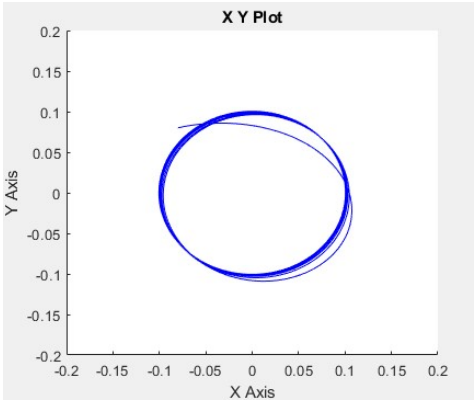


Figure 4.65: XY plot

To let the ball make a circle shape, there are some adjustments that should be considered to satisfy all requirements:

- Decrease the inner-loop BW to 15.9 Hz (it was 47.74 Hz)
- Reference has 0.6 Hz frequency
- Low pass filter before actuators has 6 Hz frequency
- Sampling time is 0.1 second

As it can be seen in figures 4.61 up to figure 4.65, this movement is succeeded. Angles range are [-11.5 , 11.5] degrees, actuators currents range are [-2.8 , 2.8] Amps and the amplitude of the circle is 10 cm. However, in figure 4.64, it can be observed that the output has some lags with respect to the reference (almost 0.4 second). Also, it is noticeable that in figure 4.61, the positions can not reach its final value and has some steady state error and this is caused by running actuators slower.

Chapter 5

Model Validation

In this chapter, a validation test for the open-loop of the system is going to be demonstrated.

5.1 Open-loop

The reason that is needed to perform this test is to check how close the system's plant in theory is in comparison to the actual system. To do this experiment, there are some consideration needed to be taken into account, such as:

- The angles are not in a very small range as mentioned in chapter 3. Therefore, linearization of ($\sin\alpha \approx \alpha$) is not valid.
- The length of incline is important for measuring the time. The plate for this project has 0.39 m length, but for having more precision for time, it is better to use a longer plate (this experiment uses a plate with length of 0.84 m).
- The acceleration of the ball 5.1, when it rolls off the plate, is going to be measured and compared between the actual system and the SIMULINK model.

Figure 5.1 illustrates a ball rolling on a plate with angle α.

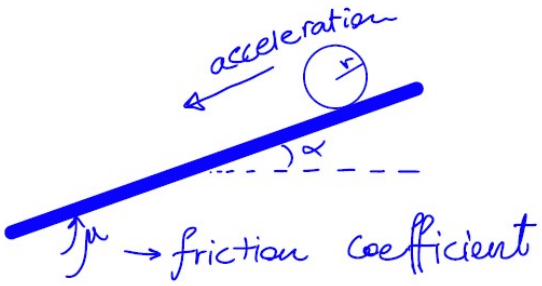


Figure 5.1: Rolling a ball on a plate

The acceleration of the ball for this set-up is as follow (proof in appendix):

$$a = \frac{5}{7} \cdot g \cdot \sin(\alpha) \tag{5.1}$$

5.1.1 SIMULINK model and set-up

Considering figure 3.2 and the fact that actuators provide the angle for the plate, following figure represents the model in SIMULINK.

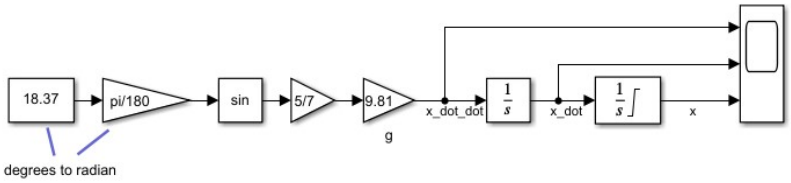


Figure 5.2: SIMULINK model

***Note**: It should be noticed that due to a symmetric system, this experiment is performed w.r.t one axis

Figure 5.3shows the set-up which is used for this test:

Figure 5.3: Set-up

5.1.2 Result

In this section the results of both models are demonstrated.

Set-up

Figures 5.4 and 5.5 show the measurements for calculating one angle. Other angles calculation follow the same principle.

Figure 5.4: Right angle triangle - Height

Figure 5.5: Right angle triangle - Hypotenuse

Based on figures 5.4 and 5.5, angle can be calculated as follow:

$$\alpha = \arcsin\left(\frac{Height}{Hypotenuse}\right) = \arcsin\left(\frac{29}{92}\right) = 18.37^{\circ} \tag{5.2}$$

The result of repeating the experiment for 6 times for each angles is shown in table 5.1:

Angles (deg)	**Time (s)**	**Time (s)**	**Time (s)**	**Time (s)**	**Time (s)**	**Time (s)**	**Average (s)**	**Acceleration** $m/(s)^2$
4.59	1.83	1.93	1.86	1.9	1.73	1.88	1.885	0.488
9.78	1.23	1.21	1.25	1.23	1.13	1.1	1.19	1.19
18.37	0.75	0.86	0.8	0.78	0.86	0.78	0.805	2.59

Table 5.1: Result of experiment

***Note**: Acceleration is calculated based on :

$$x = v(0) \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

Where : x is the length of the plate (0.84 m)
$v(0)$ represents the initial speed which is zero
t is the time
a stands for the acceleration

SIMULINK model

The result of simulated model in SIMULINK for mentioned angles in previous section is as follow:

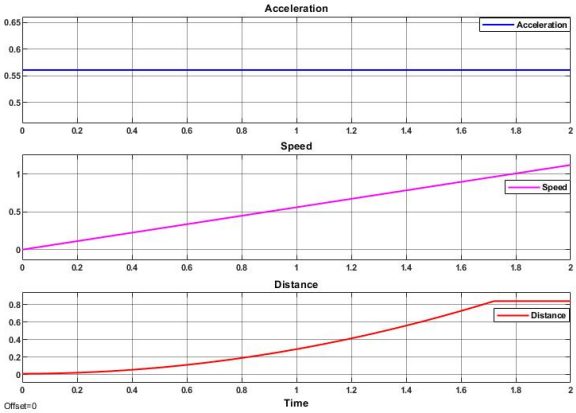


Figure 5.6: Angle 4.59 degree

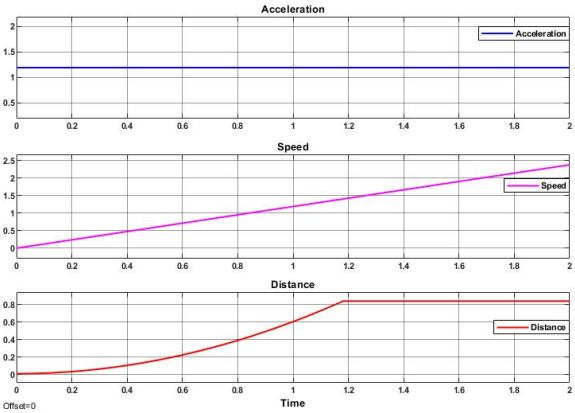


Figure 5.7: Angle 9.78 degree

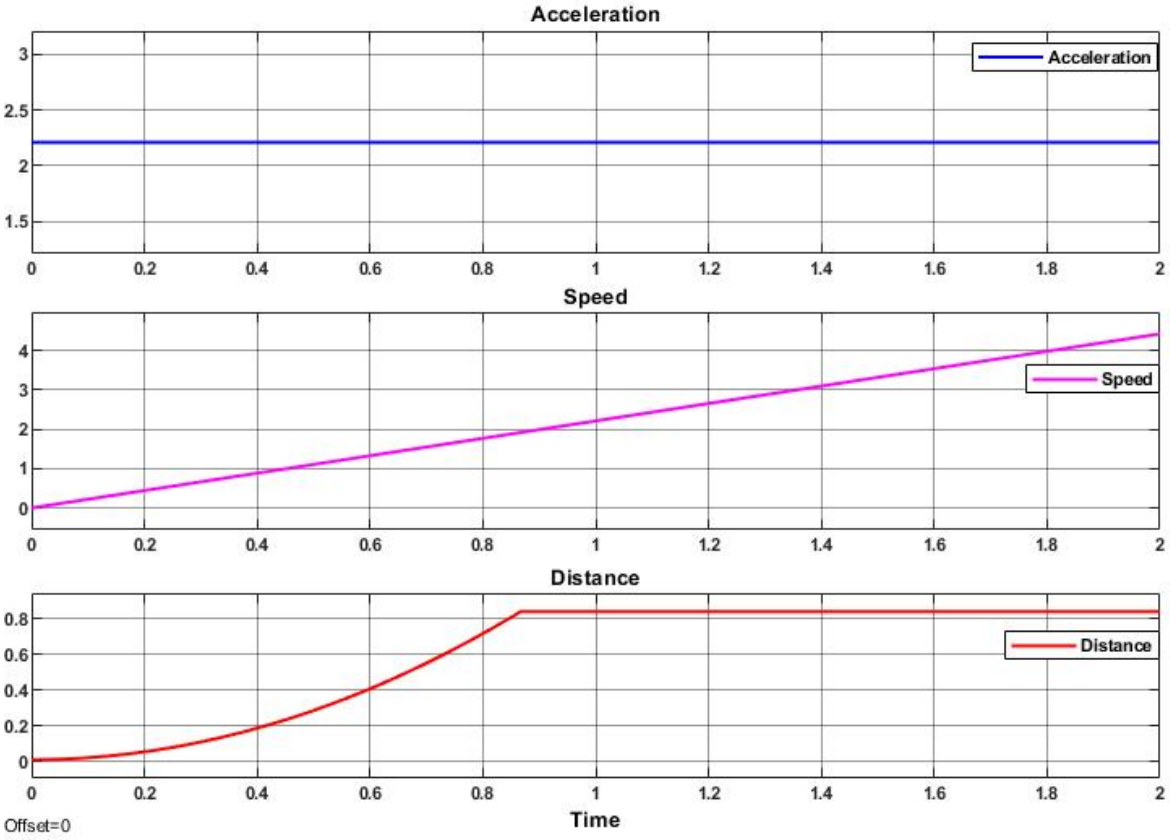


Figure 5.8: Angle 18.37 degree

***Note1**: The exact values for acceleration are 0.56, 1.19 and 2.2 (m/s^2), respectively.
***Note2**: There is a limit in distance figure (0.84 m), since after this limit, the ball is rolled off the plate and the values will not be important.

5.1.3 Conclusion

Based on results in previous section, it can be seen that there are some deviation between the values from the experiment and SIMULINK model. These deviations are caused by three factors, namely:

- SIMULINK analyzes the model ideally, meaning no error in measurements and no friction force
- Friction force (μ) between the ball and the plate
- Error in measurements

Despite these deviations, values for acceleration are still close to each other and therefore, the SIMULINK model is valid to use.

Chapter 6

Implementation and Testing

6.1 System Integration

In this chapter, firstly, the total system will be integrated for dSpace platform. Next, some considerations with respect to each part of the set-up will be explained. Afterwards, serial communication is going to be described. Then, how positions of the ball are calibrated, will be shown. Lastly, the results of experiments on the system will be discussed and compared to the simulation result in chapter four. Following figure illustrates the integrated model for the total system:

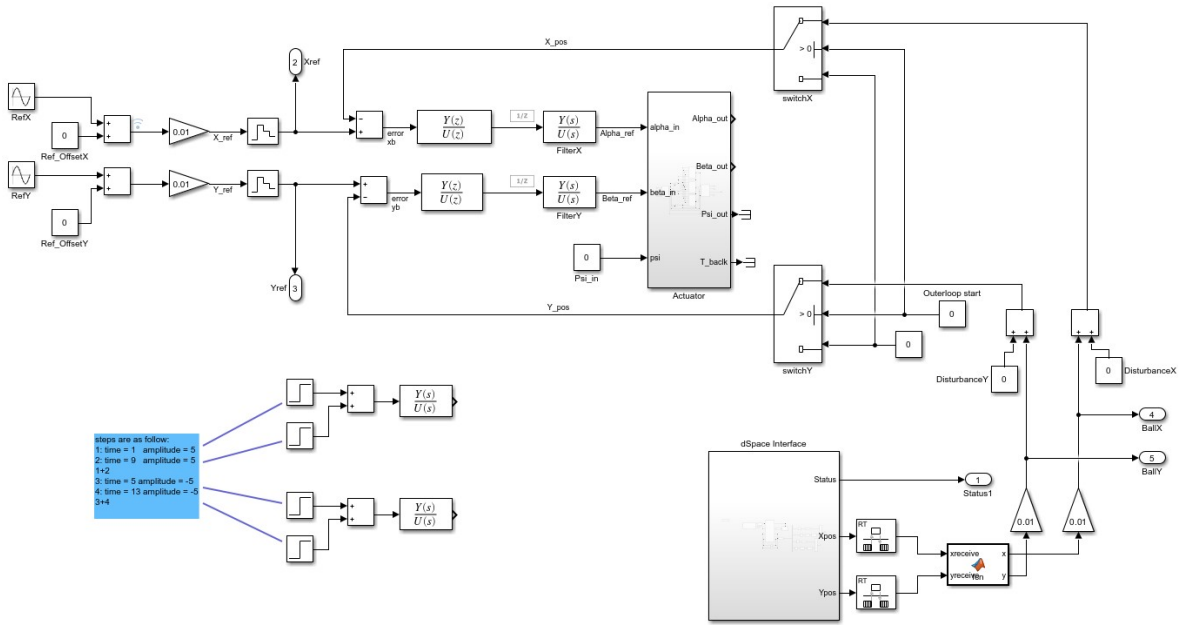


Figure 6.1: Integrated model for dSpace

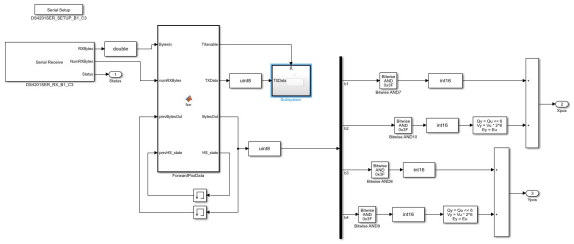


Figure 6.2: dSpace interface1

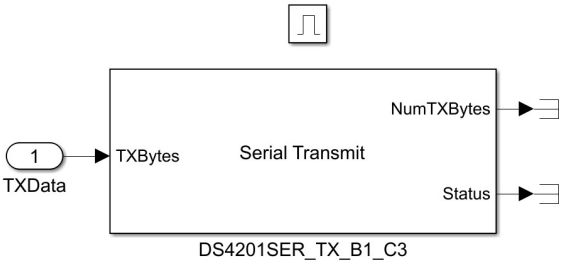


Figure 6.3: dSpace interface2

The way this system works is as following:
First, actuators need to be initialized to their operation point (0.322 m), while serial communication is not in the loop yet (hence, switches must be OFF). References should have also zero amplitude at this point, otherwise the ball will be detected and camera passes the position of the ball and this leads to an aggressive reaction of actuators and system will immediately go to an unstable status. When actuators are settled in their desired operation points, switches can be turned ON. It is advised to put the ball at the center of the plate (0,0) so that actuators do nothing and wait for a reference. Lastly, by applying different references, the position of the ball will be detected and passed to dSpace via rs485 serial communication and it will be led the desired points by the outer-loop controller.
It should be noticed that the motion plate will be steered by actuators and therefore, the plant in simulation model can be removed in practice model.

6.2 Validation of transformation of angle reference to actuator's position

As stated in chapter four and particularly equation 4.8, the height between the base frame and motion plate (T vector) is 322 mm. This distance is also added in practice and therefore it is needed to adjust the motor positions. Following figures illustrate the set-up and calculations of how initialization is done.

According to actuators datasheet, zero position at which the motor has a constant force, is 50 mm. Based on figure 6.5, zero position for this project can be defined by subtracting the varying distance and fixed part, which is 29 mm. To validate the motor's model in practice, it is needed to apply two types of inputs, namely:

- Case 1: Constant angles
- Case 2: Varying angles (two sine waves with different phases for rotation movement)

This test is done to check whether the motors are able to make such angles and also to observe what the currents ranges are. It should be reminded that this test is also done in simulation, in chapter four-section 3.1.
***Note :** Sensors for measuring the angles were not available for this project and instead, an application in phone is used to do the measurements. Also it should be noticed that first, zero degree should be defined by the app and then applying other angles.
Following figures are the outcome of case 1:

Figure 6.4: Set-up

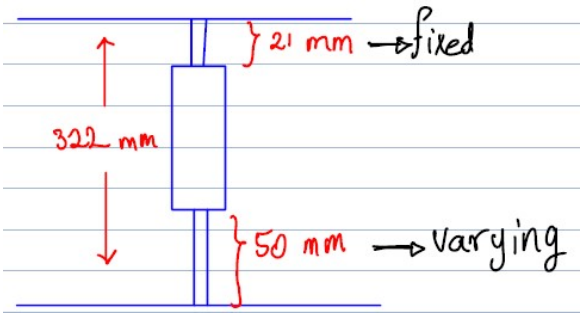


Figure 6.5: Height adjustments

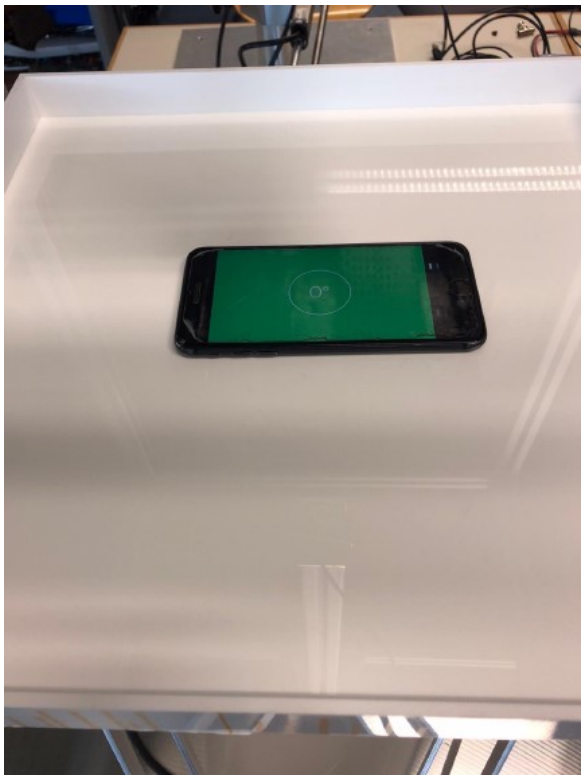


Figure 6.6: Zero degree

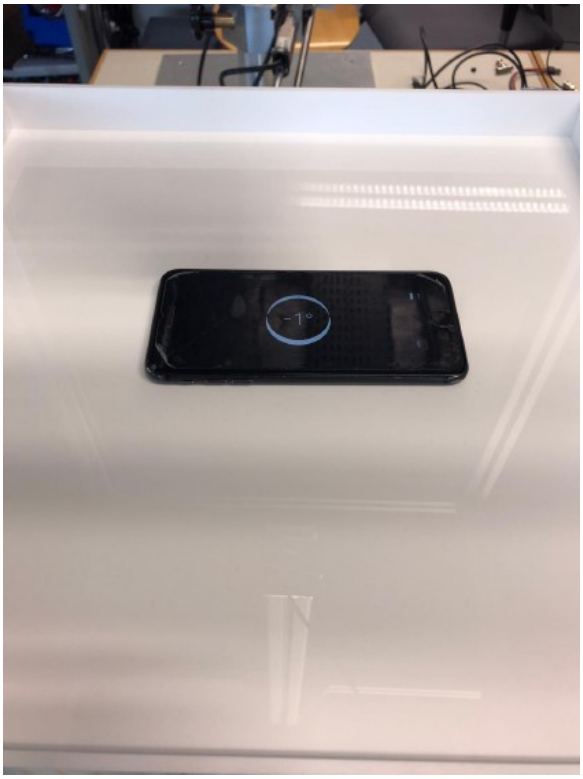


Figure 6.7: One degree

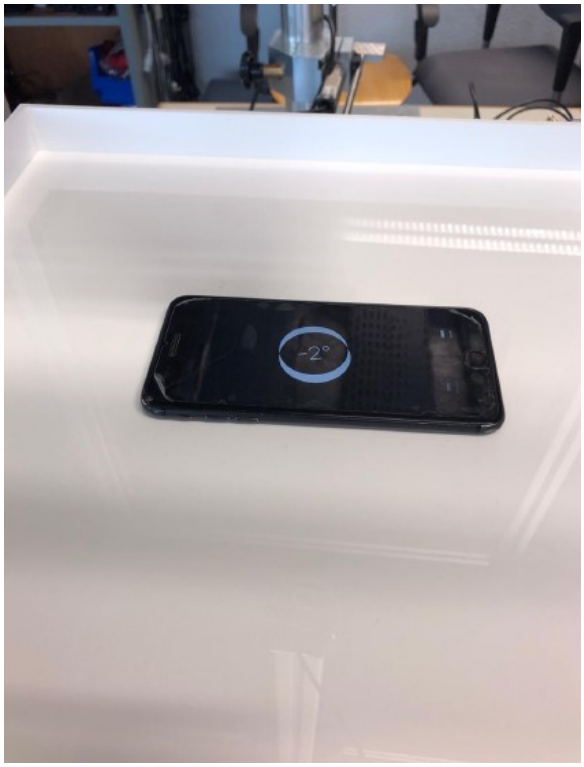


Figure 6.8: Two degree

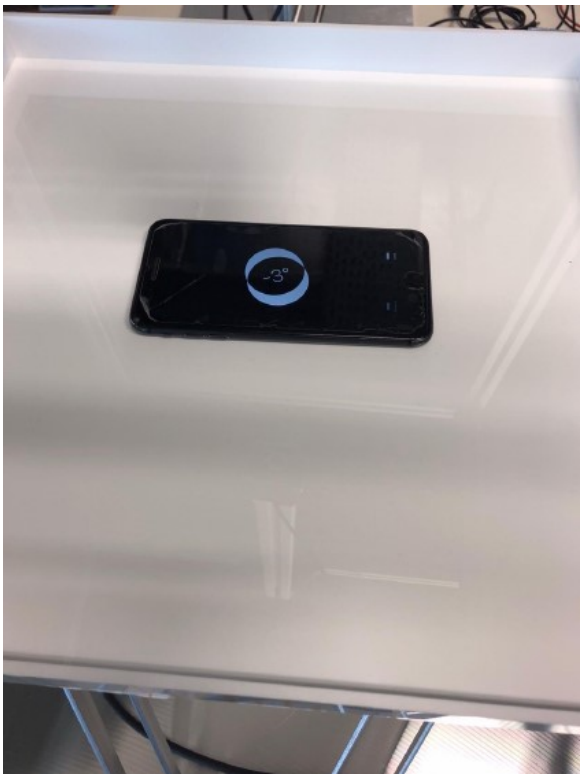


Figure 6.9: Three degree

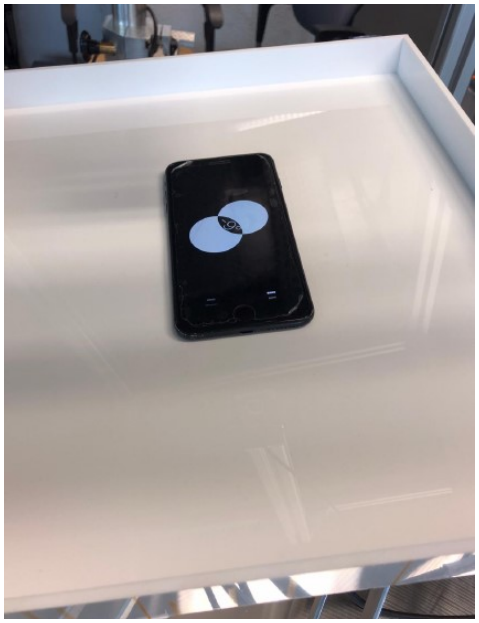


Figure 6.10: Nine degree

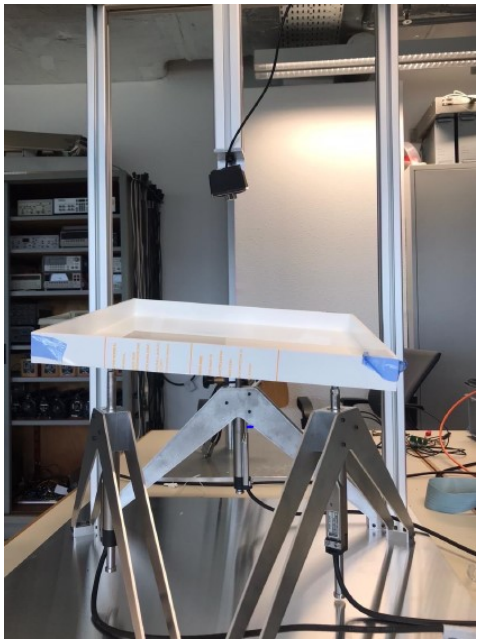


Figure 6.11: Plate with nine degree

For case 2, two sine waves are applied such that a circle with a radius of four cm, can be made by actuators (frequency = 6 rad/s, inner-loop bandwidth = 47.74 HZ).

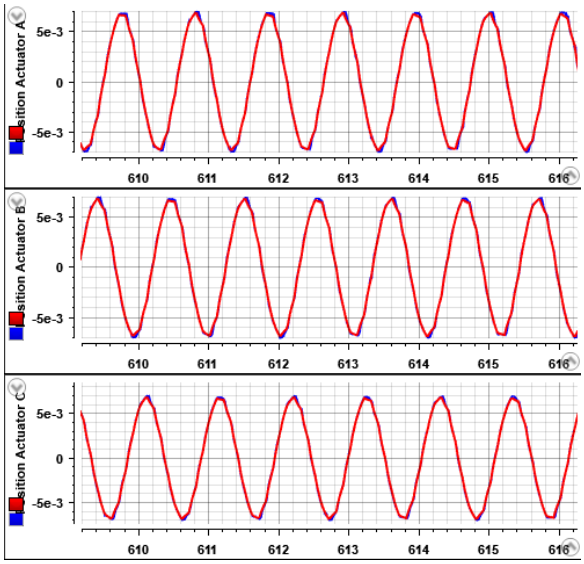


Figure 6.12: Actuator positions$_sinewave$

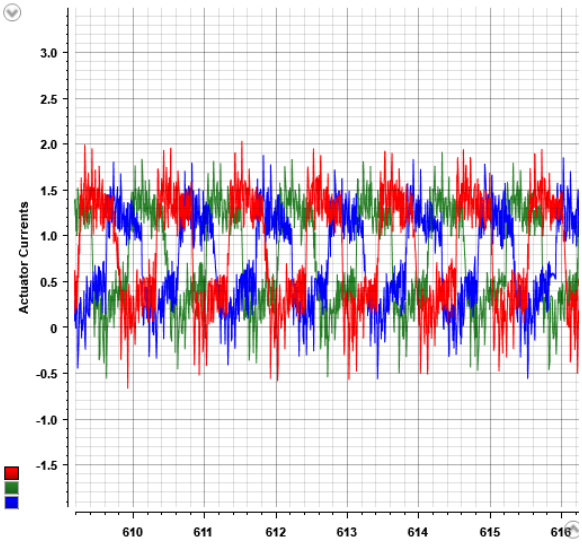


Figure 6.13: Actuator currents$_sinewave$

Based on figure 6.13, it can be seen that currents are in the range ([-3 , 3]Amps and there is no sign of overshoot. Figure 6.12 illustrates that the actuators are able to follow the reference at almost every points, except two cases (when the wave reaches its highest/lowest amplitude), which it might caused by the slider's friction.
In conclusion, it is safe to say that actuators are operating properly and there is not a remarkable issue. So, integration and implementing of the total system can be started.

6.2.1 Actuators operation

There are several considerations for actuators that should be taken into account while testing, otherwise, motors will move choppy or a violation in the current range will happen. The solution to avoid such movement, is Lubricating some grease, since the friction force in motors is relatively high.
To keep the current in the range, as it is mentioned in previous chapters, a low pass filter should be used.

6.2.2 Ball

During some experiments, it is realized that the required angles for the plate to let the ball move depend on the status of the ball. When the ball stands still, plate's angle in the range of [-1.5 , 1.5] will not make the ball rolling and this might happen due to friction forces between the ball and the plate. However, when the ball is rolling, this issue will not happen.

6.2.3 Serial Communication

dSpace platform, generally provides all types of serial communications, such as: RS-232, RS-485 and etc, but in current module, RS-485 chip is the only option. Therefore, this type of serial is chosen. On the other hand, Raspberry Pi has RS-232 as its default option for serial communication. Using an USB RS-232 to RS-485 converter will solve this problem. After some experiment, it is realized that

dSpace has 5 volts on its output ports whereas Pi has 3.3 volts on all its ports. A voltage level shifter is the solution to fix this issue.

Protocol

Both devices need to have the same protocol for serial communication. The structure is as following:

Raspberry Pi

```
import time
import serial
import serial.rs485

ser = serial.rs485.RS485(
        port='/dev/ttyUSB0',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)
```

dSpace

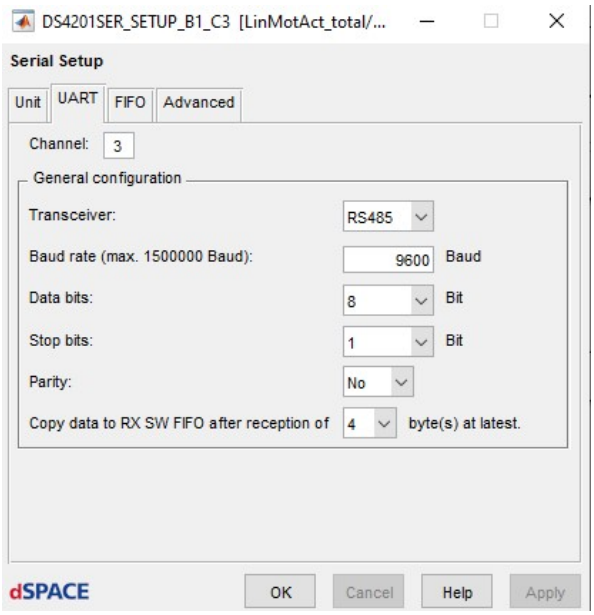


Figure 6.14: Serial protocol dSpace

Byte and Char

There are two possibilities to transmit/receive data and since has always a positive value, it is decided to use two possible ranges, in particular:

- Byte → [0,254]

- Char → [0,127]

With using Byte structure, it is possible to transmit/receive data in 8 bits while Char provides 7 bits transfer rate. Based on some experiments and also considering the values that should be transmitted/received (which are pixels with a range of [0 - 400] for both x and y), the second option (Char) has been chosen with following structure:
*Note: Positions are in pixels and considering their ranges, it is decided to divide them into two bytes.
Raspberry Pi

```
            #Cx = 400    binary: 1 1001 0000
            #Cy = 400     binary: 1 1001 0000

      #byte 1
      #passing the 6 lsb bits with MSB bits sign : 0x3f = 0011 1111
      byte1 = (Cx) & (0x3f)
      byte1_char = chr(byte1)   #convert to char

      #byte2
      #the first two MSB are the sign for x + shift right 6 bits (0x3f= 0011
    1111, 0x40= 0100 0000)
      byte2 = ((Cx>>6) & 0x3f)|0x40
      byte2_char = chr(byte2)

      #byte 3
      #the first two MSB are the sign for y + shift right 6 bits (0x3f= 0011
    1111, 0x80= 1000 0000)
      byte3 = ((Cy) & (0x3f))|0x80
      byte3_char = chr(byte3)   #convert to char


      #byte4
      #the first two MSB are the sign for y + shift right 6 bits (0x3f= 0011
    1111, 0xC0= 1100 0000)
      byte4 = ((Cy>>6) & 0x3f)|0xc0
      byte4_char = chr(byte4)

      ser.write(byte1_char)
      ser.write(byte2_char)
      ser.write(byte3_char)
      ser.write(byte4_char)
```

dSpace
In dSpace, the opposite process should happen.
*Note: It should be noticed that this serial communication is a FIFO (First In First Out) type.
Although a sign bit is given to all four bytes, but experiments has shown that this is not a solid communication, since most of the time, data was missing due to shifting or having a different timing in devices (since serial communication is an asynchronous process). Therefore, a handshake must be made.

Handshake

To make sure that communication happens properly, it is needed to have a handshake structure, which a flowchart represents how this is defined:
Raspberry Pi
Following code in Python is implemented in Pi's side:

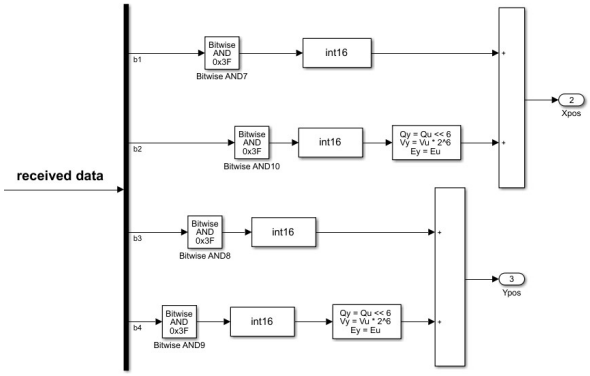


Figure 6.15: splitting Received Bytes in dSpace

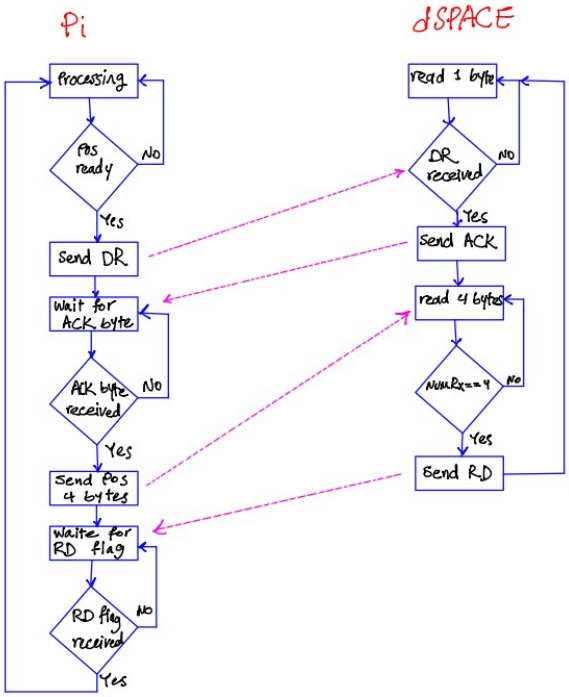


Figure 6.16: Handshake flowchart

```
    ser.write(chr(0xff))        #to confirm that data is ready
    ser.write(chr(0xff))
    ser.write(chr(0xff))
    ser.write(chr(0xff))
    ack_byte = 0
    received_byte = 0
    while (ack_byte != chr(239)):
      ack_byte = ser.read()    #to confirm that dSpace is ready
    ser.write(byte1_char)
    ser.write(byte2_char)
    ser.write(byte3_char)
    ser.write(byte4_char)
```

```
13      while(received_byte != chr(223)):
14        received_byte = ser.read()
```

MATLAB

Figures 6.2 (receive data), 6.3 (Transmitting data) and MATLAB function code below, illustrates how handshake process in MATLAB is implemented.

```
1  %Handshake in MATLAB
2  function [TXenable,TXData, BytesOut, HS_state] = fcn(BytesIn, numRXBytes,
       prevBytesOut, prevHS_state)
3
4  ACK=239; %0xEF;
5  RXRD=223; %0xDF;
6  DR=255; %0xFF;
7
8  TXenable=0;
9  TXData=0;
10
11 HS_state=prevHS_state;
12 BytesOut=prevBytesOut;
13
14 switch HS_state
15     case 0                          %wait for Data Ready signal from PI
16         if BytesIn(1)== DR || BytesIn(2)== DR || BytesIn(3)== DR || BytesIn(4)==
       DR
17             HS_state=1;
18         else
19             HS_state=0;
20         end
21     case 1                          %make ACK high
22         TXData=ACK;
23         TXenable=1;
24         HS_state=2;
25     case 2                          %make ACK low
26         TXenable=0;
27         HS_state=3;
28     case 3                          % forward 4 bytes of position data
29         if  numRXBytes==4
30             BytesOut=BytesIn;
31             HS_state=4;
32         else
33             HS_state=3;
34         end
35     case 4                          % reading done, notify PI
36         TXData=RXRD;
37         TXenable=1;
38         HS_state=5;
39     case 5
40         TXenable=0;
41         HS_state=0;
42 end
```

Figure 6.17 shows that the handshake is working properly and it also demonstrates the value for FPS at which the camera handles data per frame.
In the next sections, it can be seen that because of a working communication, the ball is controlled.

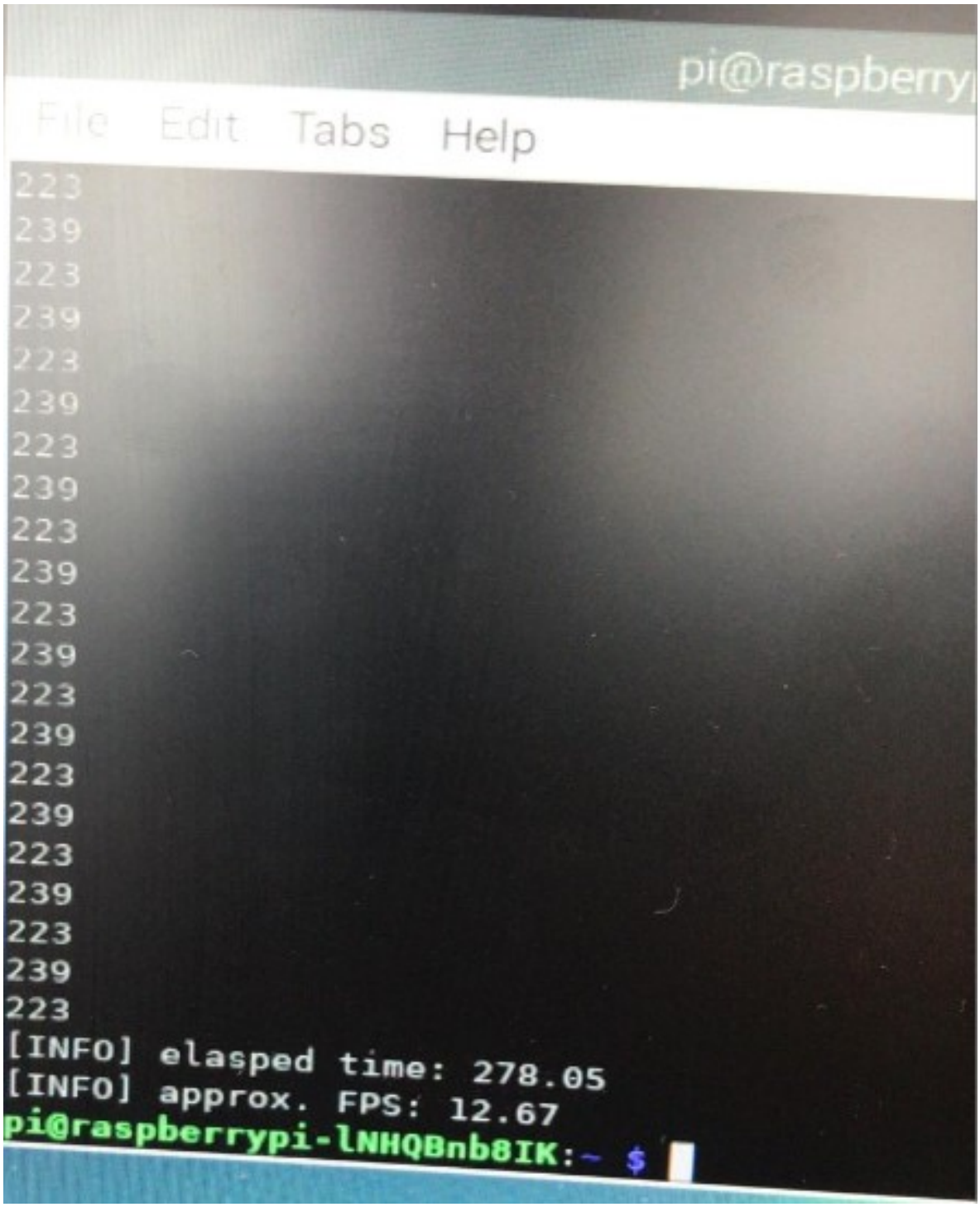


Figure 6.17: FPS camera - Handshake flags

6.2.4 calibration

Detection of the ball is designed within a 400x400 frame. Incoming bytes are the pixels of the position of the center of the ball. However the camera does not cover only the plate. Figure **??** shows the frame

and it can be seen where the plate is. Therefore, it is needed to define where the origin is and also the limits where the ball hits the edge of the plate and rolls back. Following figure illustrates which points should be detected and afterwards the MATLAB function codes shows how this part is implemented.

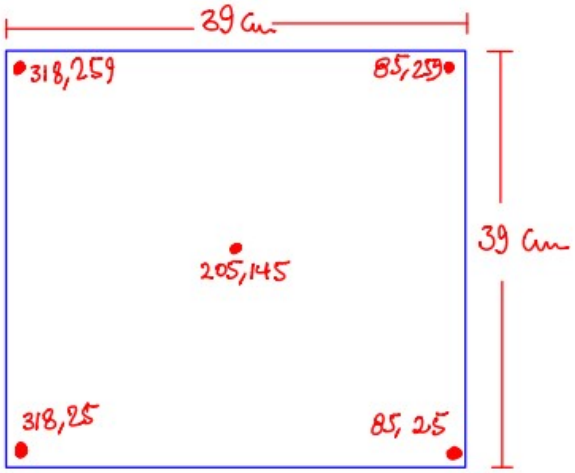


Figure 6.18: Detected pixels

```
function [x,y]  = fcn(xreceive,yreceive)

%camera:    xrange = [85,318], yrange = [25,259]
%origin = 205,145
%pixel convert to cm w.r.t x-axis:max - min / 39
%pixel convert to cm w.r.t y-axis:max - min / 39

if (((xreceive<=205)&&(xreceive>=85))&&((yreceive>=145)&&(yreceive<=259)))%1st
    quad
    x = (1/6.15)*(205-xreceive);              % x = 180 >> (205-180)/6.15 = 4.06
    y = (1/5.86)*(yreceive-145);               % y = 200 >> (200-145)/5.86 = 9.38
    if (xreceive<85)    % if it goes beyond the range from the right side(1st
    quad)
        x = int16(19.5);                       % x becmes the maximum
    end
    if (yreceive>259)   % if it goes beyond the range from the right side(1st
    quad)
        y = int16(19.5);                      % y becmes the maximum
    end
elseif (((xreceive>205)&&(xreceive<=318))&&((yreceive>145)&&(yreceive<259)))%2nd
    quad
    x = (-1/6.15)*(xreceive-205);             % x = 250 >> 250-205/(-6.15) = -7.31
    y = (1/5.86)*(yreceive-145);              % y = 200 >> (200-145)/5.86 = 9.38
    if (xreceive>318)
        x = int16(-19.5);
    end
    if (yreceive>259)
        y = int16(19.5);
    end

elseif (((xreceive>205)&&(xreceive<318))&&((yreceive<145)&&(yreceive>=25)))%3rd
    quad
```

```
28      x = (-1/6.15)*(xreceive-205);
29      y = (-1/5.86)*(145-yreceive);
30      if (xreceive>318)
31          x = int16(-19.5);
32      end
33      if (yreceive<25)
34          y = int16(-19.5);
35      end
36
37  else                                                    %4th quad
38      x = (1/6.15)*(205-xreceive);
39      y = (-1/5.86)*(145-yreceive);
40      if (xreceive<85)
41          x = int16(19.5);
42      end
43      if (yreceive<25)
44          y = int16(-19.5);
45      end
46  end
```

The only challenge in this part is that when the plate is moving, detected points (pixels) will change and pixels have another values and this leads to passing a false value to the system and hence, system will go to instability. There are many options to solve this issue, but due to the fact this project is not designed to analyze image processing, the simplest and fastest answer is chosen. So, the selected solution is calculating the average of three points in different status of the plate, namely:

- When the plate stands still (1 point)
- When the plate reaches its maximum angle for both sides of the plate (2 points)

After running some tests, it is observed that this solution works properly and errors are negligible. So, now that everything operates accordingly, the total system can be tested.

6.3 Testing

In this section, results in practice will be represented. The first thing that should be pointed out, is that due to some numeric errors, output angles can not be shown. These numeric errors happen based on equations 4.14, 4.16 and 4.17, when denominator goes to zero and this leads to having a fraction of $(\frac{1}{0})$ which is infinity. Following figure, shows this case clearly.

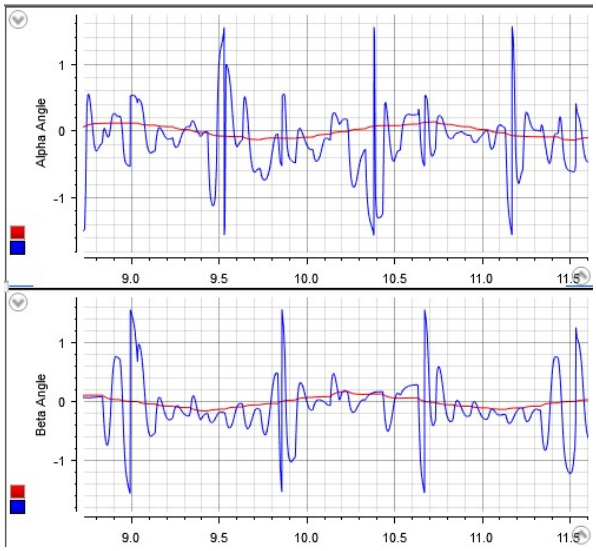


Figure 6.19: Output angles and numeric errors

As it can be seen in figure 6.19, the values for the output angles are about 1.5 radians, when the denominator reaches zero, which is 50-60 degrees. On the other hand, reference angles are correct and they are within the range. It should be noted that the output angles will not be used in this system, since the input angles are controlled by the positions of the motors and therefore, output angles have no use.
There are some points that should be considered before performing the tests, namely:

- To let the ball make a circle shape, it is needed to do this in some steps instead of letting the ball make a circle with 10 cm radius in one step
- Following the start-procedure which is at the beginning of this chapter introduced, is a pre
- Inner-loop test will be done in total system, meaning the behaviour of the actuator's position, Bw and currents range will be analyzed within the following sections

6.3.1 Test 1

According to what has been discussed in chapter four, after the figure 4.21, mentioned controller is implemented in dSpace with the stated pre-conditions for the reference and following figures are the results for a circle with 6 cm radius.

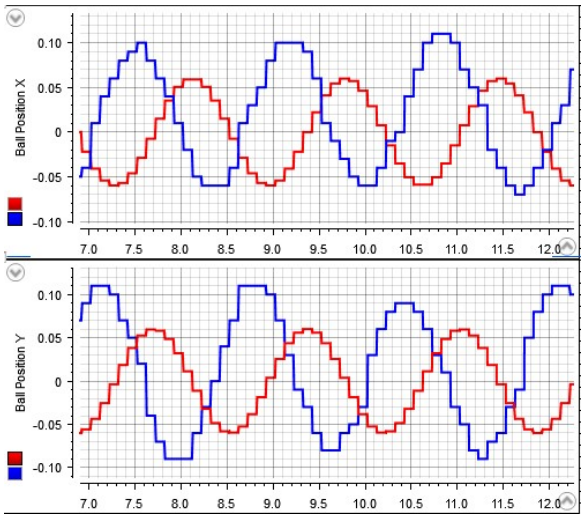


Figure 6.20: Ball Positions

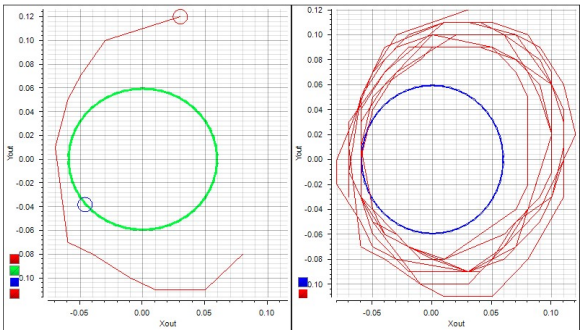


Figure 6.21: XY plot

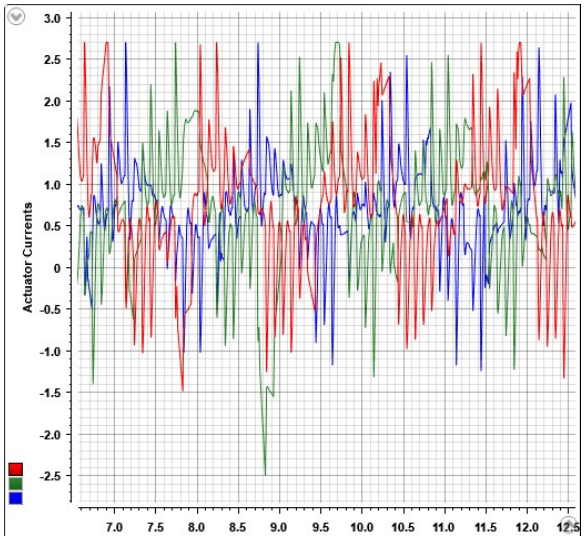


Figure 6.22: Actuator currents

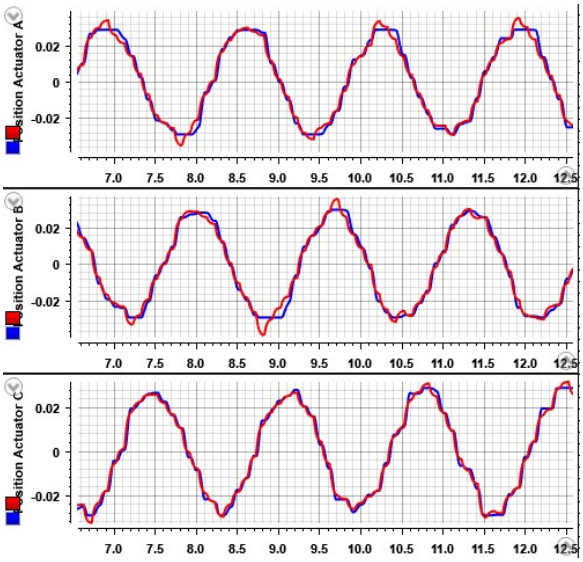


Figure 6.23: Actuator positions

As it can be seen in figures 6.34 and 6.35, the output positions are having a large amount of deviation. it can not follow the reference completely. There is also almost 180°lag between the reference and the real output position, which both behaviour caused by the range of frequency from the designed controller (based on figure 4.56 and having 0.6 Hz $\sim$ 3.77 rad/s as the reference frequency leads to the mentioned lagging and large amplitude). Figure 6.36 illustrates how close the actuators currents are to their range and lastly, according to figure 6.37, it is clear that a band-width of 15.9 Hz can not follow its reference properly and it has some overshoots. These overshoot are the reason of such a high range of currents for the actuators. This leads to using a low pass filter with lower frequency and increasing the inner-loop band-width to 47.74 Hz. It should be also noted that these results do not match to the results from simulation in chapter four, in terms of amplitude (radius of the circle), range of currents and the final shape. All in all, this controller does not operate fast enough to follow the

reference. Therefore, it is needed to make the range of frequency higher and thus, following controller is designed.

$$C2_{New-discrete} = \frac{2.5037 * (z - 0.06981) * (z^2 - 1.92 * z + 0.9213)}{(z - 1) * (z - 0.3774) * (z^2 + 0.3764 * z + 0.03567)} \tag{6.1}$$

Root-locus of the closed-loop and bode diagram are illustrated as follow: Root-locus of this controller

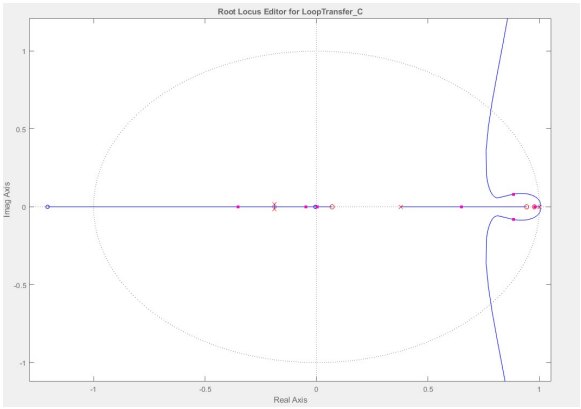


Figure 6.24: Root-locus of the closed-loop

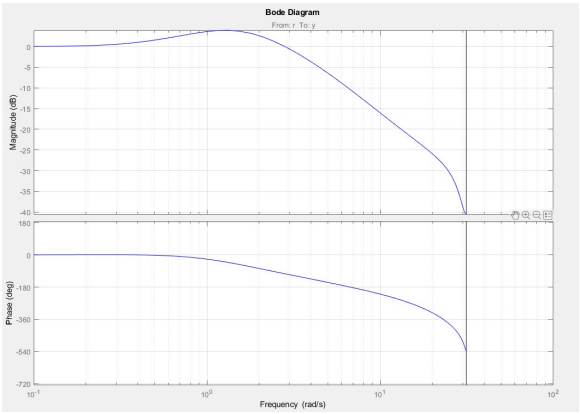


Figure 6.25: Bode diagram

confirms that by using a proper gain, system will be stable. Equation 6.1 approves that there is no zero steady state error, since an integrator is used. Further, the bode diagram of this controller illustrates two points. First, the amplitude of the output increases for the lower frequencies and after $\sim$1 rad/s, it will decrease with -20 db. To use this controller in an optimal way, the required frequency for the references, would be $\sim$0.5 to $\sim$0.8 rad/s, which is again slow but not as slower as the previous controller. Second, based on the number of zeros and poles, it is expected that this controller would have a better performance in tracking the references. Following figures are the results of using this controller.

***Note1 :** It should be noted that the band-width of actuators is changed to 47.74 Hz, since motors and system can handle operating with this frequency for the inner-loop. To do so, it is needed to use a 1 Hz low pass filter before the actuators block.

6.3.2 Test 2

In this section, two tests are going to be demonstrated. First test is done with 2.82 rad/s for the reference and it is tried to make a circle shape with a radius of 6cm. Second test shows the behaviour of the system for even lower frequencies with a larger amplitude for the reference (0.8 rad/s and radius of 10cm).

6.3.3 Test2.0

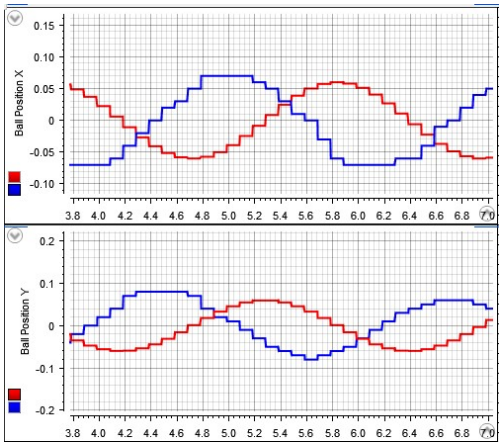


Figure 6.26: Ball Positions

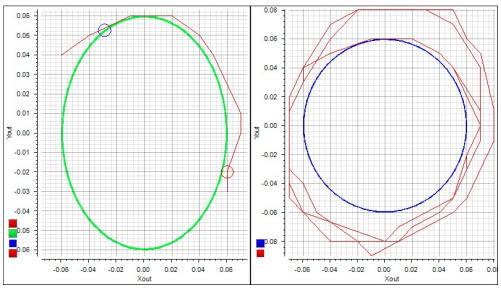


Figure 6.27: XY plot

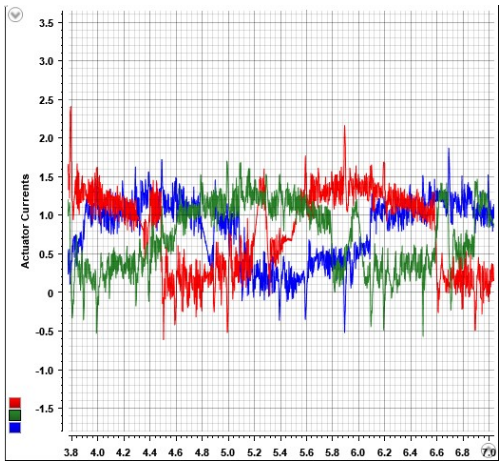


Figure 6.28: Actuator currents

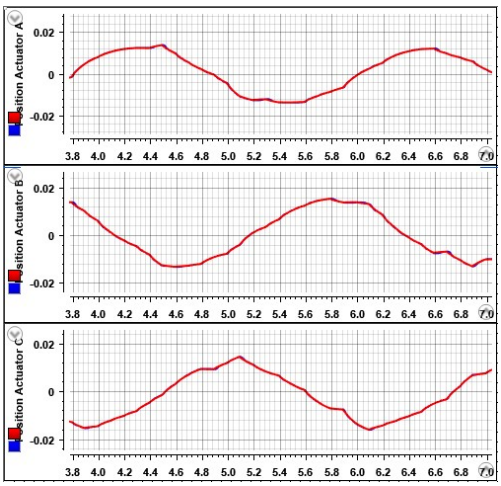


Figure 6.29: Actuator positions

XY plot illustrates that the ball is more concentrated and has lower deviation with respect to the reference. Figure 6.26 shows that there is still a lag between the output position and reference and this is no surprise, since according to the bode plot of this controller, for this frequency, phase is $\sim 180\,^{\circ}$. On the other hand, it can be seen that the higher frequency for the actuators is the better solution and motors are tracking the reference completely. Also, it should be noted that the current range is lower than test 1. Hence, motors are operating properly with the new frequency.

6.3.4 Test2.1

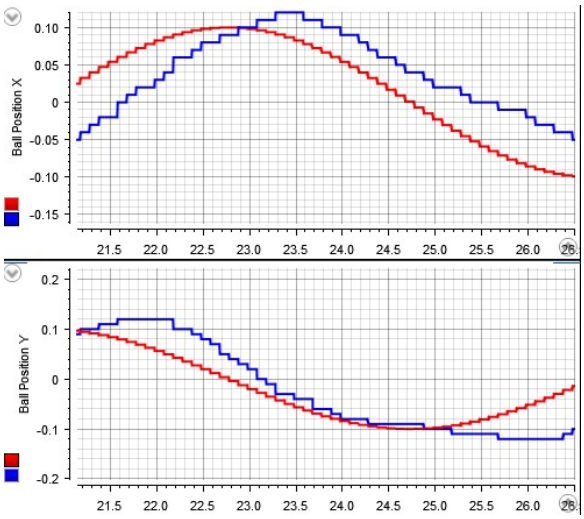


Figure 6.30: Ball Positions

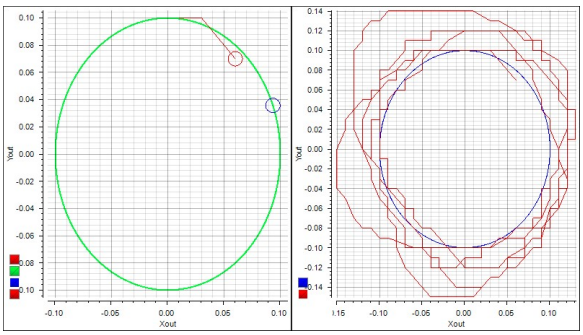


Figure 6.31: XY plot

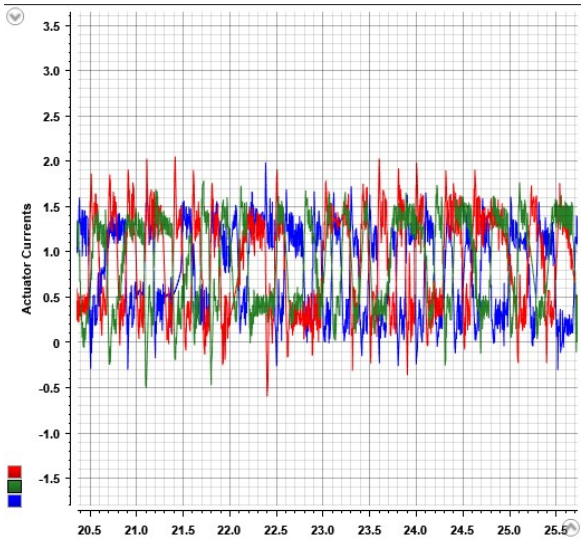


Figure 6.32: Actuator currents

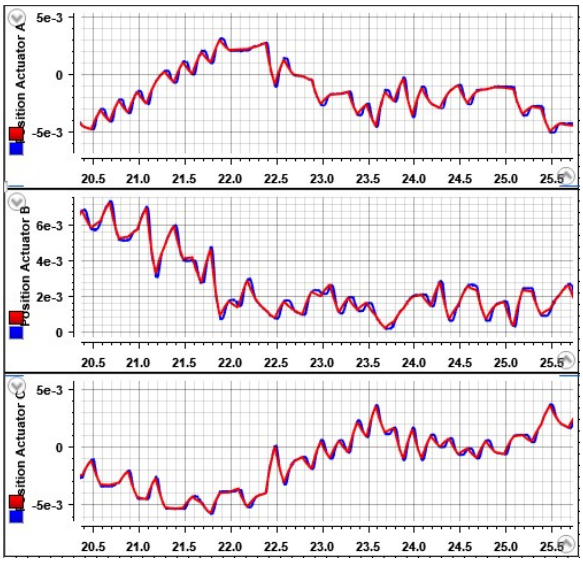


Figure 6.33: Actuator positions

To improve the lagging issue between the output position of the ball and the reference, based on figure 6.25, it is needed to set the reference frequency lower than 1 rad/s.). 0.8 rad/s is chosen, since the controller operates better in the low frequency ranges. As it can be seen, actuators are still having a good tracking for the reference and the current range is still satisfying the requirements. XY plot demonstrates that the circle has less deviation and based on figure 6.30, lagging problem is decreased. However, system operates very slow.

6.3.5 Test 3

Following test is done to see whether the system can follow other types of references. Since, this controller operates for the lower frequencies, an infinity shape is applied by having 0.8 rad/s for one direction and 1.2 rad/s for the other direction. Results are reasonably good and the ball could follow the reference, not precisely though. It is observed that when the ball is going to make a turn, it stands still for a fraction of a second and then it started to move. This is led to having a larger amplitude and not tracking the reference completely. This is caused by two factors, namely: the friction force between the ball and the plate and the frequency. As stated earlier, when the ball stands still, it needs more effort to make it move and it takes more time to get the ball under the control and hence, the output position will not be the same as the reference. Moreover, from Actuators figures, it can be seen, that motors are operating good. In conclusion, it is safe to say that this controller operates much better than the previous version.

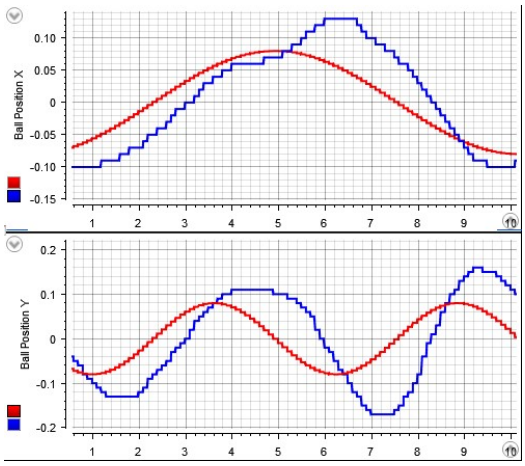


Figure 6.34: Ball Positions

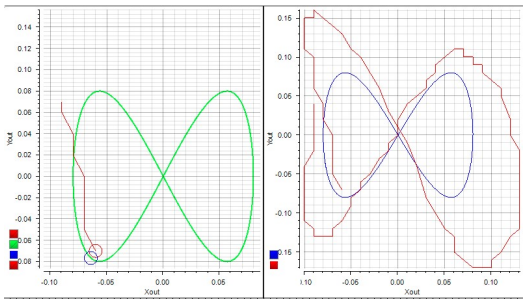


Figure 6.35: XY plot

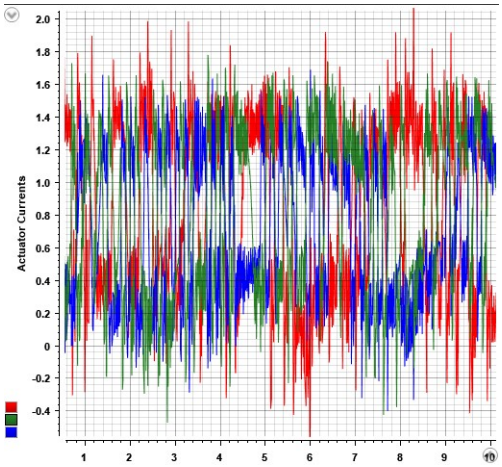


Figure 6.36: Actuator currents

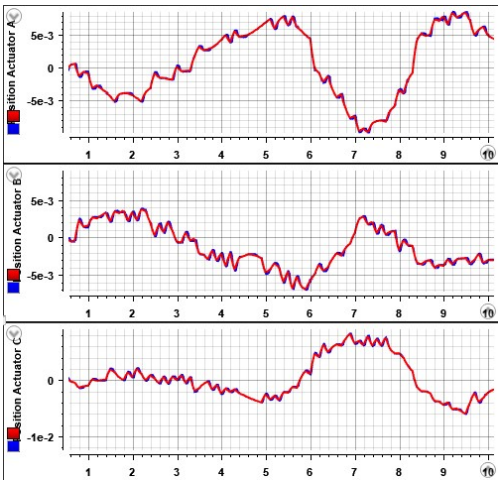


Figure 6.37: Actuator positions

Chapter 7

Conclusions and Recommendation

7.1 Conclusion

Following conclusions can be drawn from designing a model for this system throughout simulation, up to testing and implementations:

- Model is validated and results of both simulation and experiments are in one line
- Inner-loop controller operates properly. outcomes demonstrate how simulation and experiments are almost the same
- Image processing works reasonably and the ball is detected continuously
- Output angles are not available in experiments due to some numeric errors
- Serial communication works properly without losing data
- Ball can follow the reference (circle shape)
- Ball can follow the reference (Infinity shape)

7.2 Recommendation

Below, a list of recommendation can be found for this project:

- A method can be found to solve the numeric errors to observe output angles of the plate. This project could not achieve this goal and it led to following recommendation
- Although the ball is detected and the position is passed to the system, these positions are not accurate enough, since the position is an average of three positions. The position of the ball can be calculated by the angles that the plate make.
- Type of ball can be something different such that it does not suffer from static friction with the plate
- Ball detection can be much better such that when the ball is detected, a region in its neighbourhood is chosen to not having an interference from other objects with the same color

- Detection can take much less time (higher FPS) by using a better camera. This property adds a critical constraint to the system for designing the controller
- Controller can be designed much faster with more accuracy

Bibliography

[1] Mohammed Babiker Abdullah Hussien, Mohamed Gamal Eldeen Mohamed Yousif, Abdullah Omer Haj Ali Altoum, Mohammed Faisal Ali Abdullraheem, et al. *DESIGN AND IMPLEMENTATION OF BALL AND PLATE CONTROL SYSTEM USING PID CONTROLLER*. PhD thesis, Sudan University of Science and Technology, 2017. 4

[2] Dongfeng Yuan and Zhenhao Zhang. Modelling and control scheme of the ball–plate trajectory-tracking pneumatic system with a touch screen and a rotary cylinder. *IET control theory & applications*, 4(4):573–589, 2010. 4

[3] Miad Moarref, Mohsen Saadat, and Gholamreza Vossoughi. Mechatronic design and position control of a novel ball and plate system. In *2008 16th Mediterranean Conference on Control and Automation*, pages 1071–1076. IEEE, 2008. 4, 15

[4] Jong Hyeon Park and Young Jong Lee. Robust visual servoing for motion control of the ball on a plate. *Mechatronics*, 13(7):723–738, 2003. 4

[5] Marco A Moreno-Armendáriz, César A Pérez-Olvera, Floriberto Ortiz Rodríguez, and Elsa Rubio. Indirect hierarchical fcmac control for the ball and plate system. *Neurocomputing*, 73(13-15):2454–2463, 2010. 4

[6] Robert J. Teather Benjamin F. Janzen. Is 60 fps better than 30? the impact of frame rate and latency on moving target selection. pages 1–6, 2014. 6

[7] Brenton Bailey and Alexander Wolf. Real time 3d motion tracking for interactive computer simulations. *Imperial College: London, UK*, 3, 2007. 6

[8] Bc L'uboš Spaček. Digital control of ce 151 ball & plate model. *Zlín, República Checa*, 2016. 10, 11, 23

[9] Yongkun Wang, Mingwei Sun, Zenghui Wang, Zhongxin Liu, and Zengqiang Chen. A novel disturbance-observer based friction compensation scheme for ball and plate system. *ISA transactions*, 53(2):671–678, 2014. 10, 11, 15

[10] Derek K Brecht. A 3-dof stewart platform for trenchless pipeline rehabilitation. 2015. 11, 32

[11] 11

[12] Marie-Neige Chapel and Thierry Bouwmans. Moving objects detection with a moving camera: A comprehensive review. *arXiv preprint arXiv:2001.05238*, 2020. 14

[13] dSPACE. Ds 1104 rd controller board. 2020. 14

[14] Mohammad Nokhbeh, Daniel Khashabi, and HA Talebi. Modelling and control of ball-plate system. *Teheran: Amirkabir University of Technology*, 2011. 15

[15] Kwang-Kyu Lee, Georg Batz, and Dirk Wollherr. Basketball robot: Ball-on-plate with pure haptic information. In *2008 IEEE International Conference on Robotics and Automation*, pages 2410–2415. IEEE, 2008. 15

[16] TU/e. Bw500 - laboratory setup ball and beam. 1999. 18

[17] Ming-Tzu Ho, Yusie Rizal, and Li-Ming Chu. Visual servoing tracking control of a ball and plate system: design, implementation and experimental validation. *International Journal of Advanced Robotic Systems*, 10(7):287, 2013. 23

[18] Doug Stewart. A platform with six degrees of freedom. *Proceedings of the institution of mechanical engineers*, 180(1):371–386, 1965. 32

[19] Clément M Gosselin and J-F Hamel. The agile eye: a high-performance three-degree-of-freedom camera-orienting device. In *Proceedings of the 1994 IEEE international conference on robotics and automation*, pages 781–786. IEEE, 1994. 32

[20] TP Jones and GR Dunlop. Analysis of rigid-body dynamics for closed-loop mechanisms—its application to a novel satellite tracking device. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 217(4):285–298, 2003. 32

[21] 35

[22] Wikipedia contributors. Euler angles — Wikipedia, the free encyclopedia, 2020. [Online; accessed 4-May-2020]. 36

[23] MATLAB. *version 7.10.0 (R2020a)*. The MathWorks Inc., https://www.mathworks.com/help/control/ug/continuous-discrete-conversion-methods.html, 2020. 52

.1 Modelling

In this section, acceleration and velocity of the ball on a plate with an angle will be analyzed in detail. There are five important questions about the motion of the ball on the plate which will be answered, namely:

- What is the acceleration of the ball?
- Will the ball roll or slip?
- What is the maximum angle at which the ball starts slipping instead of rolling?
- What is the velocity of the ball?
- What is the acceleration of the plate?

.1.1 Acceleration of the ball

One of the profound concept for this project is finding whether the acceleration of the ball is greater than gravity force or not. Because if this is the case, the model will not be valid anymore. In this section, the maximum acceleration of the ball will be calculated and compared to the gravity force(9.81 m/s^2). The maximum value for the acceleration of the ball happens when the plate has an angle and the ball reaches the edge of the plate. Based on figure 5.1, plate has a friction (μ) which provides the catalyst to do the angle acceleration because it provides the torque in order to let the ball roll instead of slide out the incline.
There are two equations to deal with, namely:

$$F = m \cdot a \tag{1}$$

Where: F is net force
m stand for the mass
a represents the acceleration
An equivalent equation for a rotational motion is as follow:

$$\tau = I \cdot \alpha \tag{2}$$

Where: τ stands for the torque
I is inertia
α represents the angle acceleration
Involved forces on the ball can be depicted from figure 1.

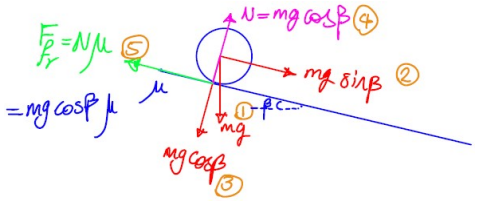


Figure 1: Involved forces

Where: 1 is the gravity force which is acting straight down
2 is parallel component of weight to surface which pulls the ball down the incline
3 is perpendicular component of weight to surface which in turn causes the normal force
4 is the normal force which provides the friction
5 is the friction force which is the force between the rolling ball and the plate

From equation 2, inertia is known since the ball is a spherical object and its value can be found from following equation:

$$I = \frac{2}{5} \cdot m_{ball} \cdot r_{ball} \tag{3}$$

Where: m_{ball} and r_{ball} are the mass and the radius of the ball, respectively.
α can be calculated as:

$$\alpha = \frac{a}{r_{ball}} \tag{4}$$

Where a is the acceleration of the ball.
τ can be defined as:

$$\tau = F_{friction}.r_{ball} \tag{5}$$

Where $F_{friction}$ is calculated as following:

$$F_{friction} = m_{ball} \cdot g \cdot \cos\beta \cdot \mu \tag{6}$$

Where: g is the gravity acceleration (9.81 m/s^2)
β is the angle between the plate and the x-axis
μ is the friction coefficient
By substituting equations 3,4,5 and 6 in equation 2, following expression will be the outcome:

$$F_{friction} \cdot r_{ball} = \frac{2}{5} \cdot \frac{a}{r_{ball}} \cdot m_{ball} \cdot r^2 \rightarrow g \cdot \cos\beta \cdot \mu = \frac{2}{5} \cdot a \rightarrow \mu = \frac{2 \cdot a}{5 \cdot g \cdot \cos\beta} \tag{7}$$

Equation 1 can be used here because after all, all the forces aiding the acceleration minus all the forces opposing the acceleration equals the mass times acceleration. It should be noticed that in figure 1, normal force and perpendicular component of weight to surface will cancel each other out and therefore, remained forces will be the friction force and the parallel component of weight to the surface. Equation 8 will represent these forces.

$$m \cdot g \cdot \sin\beta - m \cdot g \cdot \cos\beta \cdot \mu = m \cdot a \xrightarrow{Equation7} a = \frac{5}{7} \cdot g \cdot \sin\beta \tag{8}$$

This equation answers the first question which stated in the beginning of this section and also it proves that the acceleration will always be smaller than the gravity value since the gravity force is multiplied by two terms which both are smaller than one.

.1.2 Rolling or slipping

Based on figure 1, the question is whether the torque caused by the friction between the ball and inclined plate is sufficient to keep up with the acceleration down the incline. To answer this question, forces should be analyzed. Considering equations 1 and 2, If the torque is sufficient to make the rotate fast enough so that it can keep up the translational motion, the ball will not slip.
It is needed to find the maximum acceleration and to compare it then the required forces to push the ball downward. From equations 1 to 5, maximum friction force is as following:

$$F_{fr_{max}} = \frac{2}{5} \cdot m \cdot a_{max} \xrightarrow{\text{equation 6}} a_{max} = \frac{5}{2} \cdot g \cdot \cos\beta \cdot \mu \tag{9}$$

Using the net force equation 1, maximum force can be calculated as following:

$$F_{max} - F_{fr_{max}} = m \cdot a_{max} \rightarrow F_{max} = m \cdot a_{max} + \frac{2}{5} \cdot m \cdot a_{max} \rightarrow F_{max} = \frac{7}{5} \cdot m \cdot a_{max}$$

Considering equation 9:

$$F_{max} = \frac{7}{2} \cdot m \cdot g \cdot \cos\beta \cdot \mu \tag{10}$$

Friction constant for a plate made of perspex is 0.8. Hence, equation 10 becomes:

$$F_{max} = 2.8 \cdot m \cdot g \cdot \cos\beta$$

By comparing it to the force component which pulls the ball down:

$$m \cdot g \cdot \sin\beta$$

It is clear that F_{max} is greater and therefore, the ball will roll.

.1.3 Maximum angle at which the ball still rolls

Based on equation 8 and solving it for μ, the maximum angle at which the ball starts slipping, is as follow:

$$\mu = \frac{g \cdot \sin\beta - a}{g \cdot \cos\beta}$$

$$a = \frac{5}{7} \cdot g \cdot \sin\beta$$

$$\mu = \frac{2}{7} \cdot \frac{\sin\beta}{\cos\beta}$$

Hence,

$$\tan(\beta) = \frac{7}{2} \cdot \mu \rightarrow \beta = \arctan(\frac{7}{2} \cdot \mu) \tag{11}$$

Therefore, the maximum angle by considering $\mu = 0.8$for perspex material is:

$$\beta_{max} = \arctan(\frac{7}{2} \cdot 0.8) = 70.35^\circ$$

.1.4 Velocity of the ball

The velocity of the ball at the bottom of the incline can be calculated based on figure 2 and following equation which describes the net work done and energy conversions:

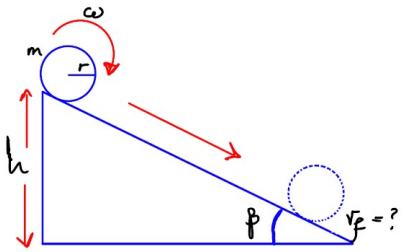


Figure 2: Velocity of the ball

$$PE = KE + KE_{rot} \xrightarrow{\text{figure 1}} m \cdot g \cdot h = \frac{1}{2} \cdot m \cdot v^2 + \frac{1}{2} \cdot I \cdot \omega^2 \tag{12}$$

Where: PE : Gravitational potential energy
KE : kinetic energy
$(KE)_{rot}$: Rotational kinetic energy
I : moment of inertia of the ball ($I = \frac{2}{5} \cdot m \cdot r^2$)
ω : Rotational speed ($\omega = \frac{V}{r}$)
m : mass of the ball
r : radius of the ball

By substituting the moment of inertia and rotational speed in equation 12:

$$g \cdot h = \frac{7}{10} \cdot V^2 \rightarrow V = \sqrt{\frac{10}{7} \cdot g \cdot h} \tag{13}$$

.1.5 Acceleration of the plate

There is also another case that should be considered, namely, when the plate has a higher speed than the ball, then the mass of the ball will not affect the net force (refer to figure 1 and 3) and it is needed to check whether at that moment, the acceleration of the plate is greater than the gravity force or not.

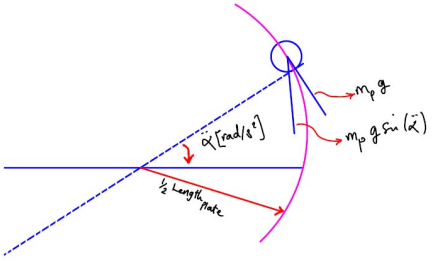


Figure 3: Acceleration of the plate

This experiment is done in SIMULINK. Considering equation 4, the result for a motion in one axis is as follow:
***Note**: It should be noticed that the initial values of all variables are zero and trajectory is a line along the x-axis.

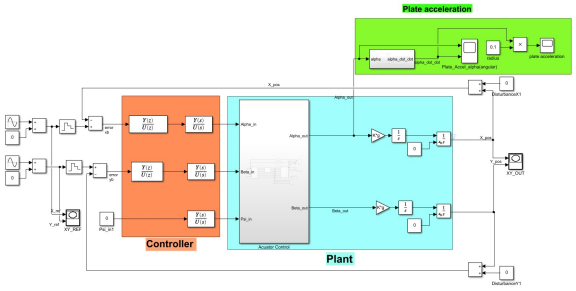


Figure 4: Simulink Block for Plate's acceleration

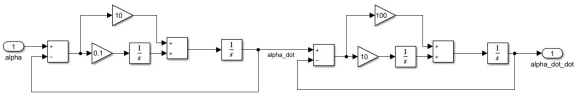


Figure 5: Angular acceleration conversion block

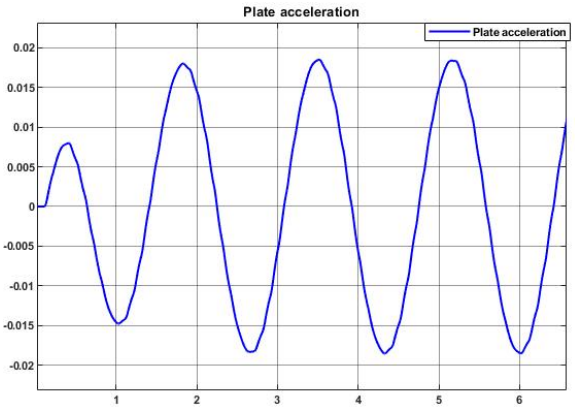


Figure 6: Plate acceleration

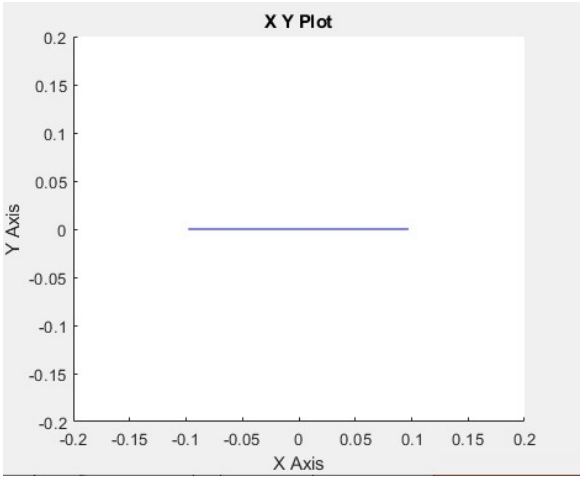


Figure 7: XY position of the plate

As it can be seen, the range of the plate acceleration in figure 6 is way lower than the gravity force ($9.81 \frac{m}{s^2}$. Hence, the ball will not jump on the plate and this case will not be an issue in this project.

.2 Matlab Code

.2.1 Model and Controller

```
%% Ball and Plate Constants
%Ball specs
r_b = 19.05*(10^-3);    %radius of the rubber ball(m)> diameter = 38.1 mm
Vb = (4*pi*(r_b^3))/3;                  %volume of the ball
density_ball = 1522;                    %density of a rubber ball
m_b = Vb*density_ball;                  %mass of the ball
Ib = (2/5)*m_b*r_b^2;                           % inertia of the ball (Kg.m^2)

K =  (m_b/(m_b+(Ib/r_b^2)));                    %constant (m/m+(Ib/r^2)) = 5/7
```

```
g = 9.81;

%Plate specs
W  = 0.34;                                    %width plate(m)
L = 0.34;                                     %Length plate(m)
H = 0.005;                                   %height plate(m)
density = 1180;                            %Perspex_density (kg/m^3)
m_p = W*L*H*density ;                     %mass of the plate = volume*density
Ip = (1/12)*m_p*(L^2+W^2);                  %inertia of the plate (Kg.m^2)

%TF's:
G_px = tf(K*g,[1 0 0]);
%% **** linear plant model*****
%
%  X(s)       Km/m
%  ---- = ---------------
%  I(s)     s(s+Cv/m)
%
% with x in [m] and i in [A]

%m=0.139;
%Km=11;
%Cv=2.5;
m=0.118+(1/3)*m_p;   %slider mass + 1/3 * Plate mass
Km=11;
Cv=16.5;

Ts=1/2000;

% **** non-linear aspects ********
% Commutation

% distance in [m] between center of two adjacent magnets, pole pitch
TAU=0.01;

CurAngOffset=-0.00137*pi/TAU;



% system bandwidth
BW=Cv/m;

Hp=zpk([],[0 -Cv/m],Km/m);
[nump,denp]=tfdata(Hp,'v');
Hp_mi = zpk([],[-Cv/m],Km/m);
[nump_mi,denp_mi] = tfdata(Hp_mi,'v');
Kp=Km/m;
p=-Cv/m;
Ts_Outer= 0.1;

% controller BW 100 rad/s
% Hc=zpk([-100/3 -100/6],[-3*100 -6*100 0],330000);
% [numc,denc]=tfdata(Hc,'v');

% controller BW 200 rad/s (32 Hz)

B=300;
```

```
% choose the controller gain such that the open loop magnitude is 0dB at w=B
% the plant magnitude at w=B is:
P_magn=Kp/(B*sqrt(p^2+B^2));
% the controller magnitude at w=B is:
C_magn=1/(B*18);
%The required controller gain to make the open loop magnitude 0dB at w=B must
    then be:
Kc=1/(P_magn*C_magn);


Hc=zpk([-B/3 -B/6],[-3*B -6*B 0],Kc);
[numc,denc]=tfdata(Hc,'v');

% same controller minus the integrator for dSPACE implementation.
% integrator is added as separate block, where the saturation limits can be
% set to avoid integrator wind-up.

% Hc_mi=zpk([-100/3 -100/6],[-3*100 -6*100],330000);
% [numc_mi,denc_mi]=tfdata(Hc_mi,'v');

Hc_mi=zpk([-B/3 -B/6],[-3*B -6*B],Kc);
[numc_mi,denc_mi]=tfdata(Hc_mi,'v');

% m=0.367;
% Km=11;
% Cv=16.5;
% Hpreal=zpk([],[0 -Cv/m],Km/m);
% s = tf('s');
% Cont = 54*((s/(2*pi*10/3)+1)*(s+(2*pi*10/5)))/(s*(s/(2*pi*10*3)+1)*(s/(2*pi
    *1000)+1));
% Cont = 19*((s/(2*pi*5/3)+1)*(s+2*pi*5/5))/(s*(s/(2*pi*5*3)+1)*(s/(2*pi*1000)+1)
    );
% Cont_mi = 50*((s/(2*pi*10/3)+1))/((s/(2*pi*10*3)+1));
% [numc_b,denc_b]=tfdata(Cont_mi,'v');

% Hc=zpk([-100/3],[-3*100 -6*100],330*7/6);
% [numc,denc]=tfdata(Hc,'v');

% same controller minus the integrator for dSPACE implementation.
% integrator is added as separate block, where the saturation limits can be
% set to avoid integrator wind-up.

% Hc_mi=zpk([-100/3],[-3*100 -6*100],330*7/6);
% [numc_mi,denc_mi]=tfdata(Hc_mi,'v');

%rltool(Hp,Hc)

%%% Based on above calculations, below the tf of the plant and the controller

Km = 1;
Tau_m = 0.005175;
G_m1=tf(Km,[Tau_m 1]);   % a similar TF of the motor eith same settling time
Plant = G_m1*G_px;   %Total Plant
Ts_Outer = 0.1;
z=tf('z',Ts_Outer);
% Case 2: design total system in discrete:
Plant_dis = c2d(Plant,Ts_Outer,'ZOH'); %convert plant to discrete
```

```
[num_plant,den_plant] = tfdata(Plant_dis,'v');
zpk(Plant_dis);                          %gives the function in order for poles
    and zeroes


C12_dis_designed = 1.9*(z+0.8546)*(z^2-1.974*z+0.974)/((z-1)*(z-0.5504)*(z
    +0.9429)); % chosen one
[numco12_dis_des,denco12_dis_des,Ts_Outer] = tfdata(C12_dis_designed,'v');

C13_dis_designed = 2.5037*(z-0.06981)*(z^2-1.92*z+0.9213)/((z-1)*(z-0.3774)*(z
    ^2+0.3764*z+0.03567));
[numco13_dis_des,denco13_dis_des,Ts_Outer] = tfdata(C13_dis_designed,'v');

%% filter before actuators circle
F = tf(1,[(1/(2*pi*1)) 1]);
[numF,denF] = tfdata(F,'v');
zpk(F);
```

.2.2 Matlab Functions$_{a}nglestopositiontransformation$

```
function [P_A_ref,P_B_ref,P_C_ref,L_in1,L_in2,L_in3] = myfunc(alpha,beta,psi)

%% Rotation matrices

%    Z rotation with angle  psi
RotZ  = [cos(psi) -sin(psi) 0;
         sin(psi)  cos(psi) 0;
          0          0      1]

%    Y rotation with angle beta+
RotY  = [cos(beta)       0         sin(beta);
             0           1             0    ;
         -sin(beta)      0        cos(beta)]

%    X rotation with angle alpha
RotX  = [1             0                0    ;
         0        cos(alpha)      -sin(alpha);
         0        sin(alpha)       cos(alpha)]
Rot_BtoP = RotZ*RotY*RotX

%% Specification of the base plate and plate

%Location of motors on the base plate
r_base = 0.17;               %radii of the base plate (m)
B1     = [r_base; 0; 0]      %position of motor wrt base plate origin
B2     = [-0.5*r_base; (sqrt(3)/2)*r_base; 0]  %120degrees wrt the position of
    first motor and base plate origin
B3     = [-0.5*r_base; -(sqrt(3)/2)*r_base; 0] %240 degrees wrt first motor and
    base plate origin

%Location of joints on the plate
r_plate = 0.17;               %radii of the plate (m)
P1     = [r_plate; 0; 0]      %position of motor wrt base plate origin
P2     = [-0.5*r_plate; (sqrt(3)/2)*r_plate; 0]  %120degrees wrt the position of
    first motor and base plate origin
```

```
P3      = [-0.5*r_plate; -(sqrt(3)/2)*r_plate; 0] %240 degrees wrt first motor and base plate origin

% distance from base plate origin(OXYZ) to plate origin(oxyz)
T = [0; 0; 0.322]                    %30 cm height from (OXYZ) to (oxyz)

%%
L_in1 = T + (Rot_BtoP*P1)-B1        %change of position of lin motor 1
P_A_ref = sqrt(L_in1(1,1)^2+L_in1(2,1)^2+L_in1(3,1)^2)   %length of L_in1 for passing to motor tf

L_in2 = T + (Rot_BtoP*P2)-B2
P_B_ref = sqrt(L_in2(1,1)^2+L_in2(2,1)^2+L_in2(3,1)^2)

L_in3 = T + (Rot_BtoP*P3)-B3
P_C_ref = sqrt(L_in3(1,1)^2+L_in3(2,1)^2+L_in3(3,1)^2)
end
```

.2.3 MATLAB function$_{positiontoangles}$

```
function [Alpha_angle,Beta_angle,Psi_angle,T_back]       = myfunc(Delta_P_A,Delta_P_B,Delta_P_C,L_in1,L_in2,L_in3)



%% Specification of the base plate and plate

%Location of motors on the base plate
r_base = 0.17;                %radii of the base plate (m)
B1      = [r_base; 0; 0]      %position of motor wrt base plate origin
B2      = [-0.5*r_base; (sqrt(3)/2)*r_base; 0]  %120degrees wrt the position of first motor and base plate origin
B3      = [-0.5*r_base; -(sqrt(3)/2)*r_base; 0] %240 degrees wrt first motor and base plate origin

%Location of joints on the plate
r_plate = 0.17;               %radii of the plate (m)
P1      = [r_plate; 0; 0];      %position of motor wrt base plate origin
P2      = [-0.5*r_plate; (sqrt(3)/2)*r_plate; 0];  %120degrees wrt the position of first motor and base plate origin
P3      = [-0.5*r_plate; -(sqrt(3)/2)*r_plate; 0]; %240 degrees wrt first motor and base plate origin

% distance from base plate origin(OXYZ) to plate origin(oxyz)
T = [0; 0; 0.322]                    %30 cm height from (OXYZ) to (oxyz)

%actuator plate connections in base frame
% P1b= B1+(L_in1)-T;
% P2b= B2+(L_in2)-T;
% P3b= B3+(L_in3)-T;

P1b= B1+(L_in1*(Delta_P_A))-T;
P2b= B2+(L_in2*(Delta_P_B))-T;
P3b= B3+(L_in3*(Delta_P_C))-T;

%base frame unit vectors
```

```
UnitX=[1;0;0];
UnitY=[0;1;0];
UnitZ=[0;0;1];
UnitXp=P1b/sqrt(dot(P1b,P1b));
UnitZp=cross(UnitXp,P3b);
UnitZp=UnitZp/sqrt(dot(UnitZp,UnitZp)); %now length 1
UnitYp=cross(UnitXp,UnitZp);

%this is according to https://en.wikipedia.org/wiki/Euler_angles, Tait-Bryn
%angles

Beta_angle=asin(-UnitXp(3))
Psi_angle=asin(UnitXp(2)/sqrt(1-UnitXp(3)^2))
Alpha_angle=asin(UnitYp(3)/sqrt(1-UnitXp(3)^2))

%Calculating rotation matrix with above angles
RotZ1  = [cos(Psi_angle) -sin(Psi_angle) 0;
          sin(Psi_angle)  cos(Psi_angle) 0;
           0           0       1];

%    Y rotation with angle beta
RotY1  = [cos(Beta_angle)        0          sin(Beta_angle);
              0                            1              0     ;
          -sin(Beta_angle)       0         cos(Beta_angle)];

%    X rotation with angle alpha
RotX1  = [1              0                                  0     ;
          0         cos(Alpha_angle)       -sin(Alpha_angle);
          0         sin(Alpha_angle)        cos(Alpha_angle)];

% Calculating back the T matrix which was the height between the base plate
% and plate. They must be the same.
Rm_times_P = RotZ1*RotY1*RotX1
T_back = L_in1+B1-(Rm_times_P*P1)

end
```

.3 Python codes

.3.1 Serial Communication handshake test

```
import time
import serial
import serial.rs485
import numpy as np


ser = serial.rs485.RS485(
        port='/dev/ttyUSB0', #Replace ttyS0 with ttyAM0 for Pi1,Pi2,Pi0
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)
```

```
15
16
17 ser.rs485_mode = serial.rs485.RS485Settings(False,True)
18
19
20 ######################################################################
21 # data must be char type to send out to dSpace, range: 0-127
22 # two bytes for x-position and two bytes for y-position
23
24 #two bytes: first byte that is sent is the right byte
25 #example : input: 500 >>> 111110100
26 #first byte is 1110100
27 #second byte is 11
28
29 #in dSpace should be in opposite, since this is a serial communication
30 #first byte is 11
31 # second byte is 1110100
32
33 ######################################################################
34
35 cx = 8
36 cx_bin = bin(cx)
37 cy = 63
38 cy_bin = bin(cy)
39
40
41 #byte 1
42 byte1 = (cx) & (0x3f)      #passing the 6 lsb bits
43 byte1_char = chr(byte1)    #convert to char
44 byte1_bin = bin (byte1)
45
46
47 #byte2
48 byte2 = ((cx>>6) & 0x3f)|0x40   #the first two MSB are the sign for x
49         # + shift right 6 times (0x3f = 0011 1111)
50 byte2_char = chr(byte2)
51 byte2_bin = bin (byte2)
52
53
54 #byte 3
55 byte3 = ((cy) & (0x3f))|0x80      #passing the 6 lsb bits
56 byte3_char = chr(byte3)    #convert to char
57 byte3_bin = bin (byte3)
58
59 #byte4
60 byte4 = ((cy>>6) & 0x3f)|0xc0   #the first two MSB are the sign for y
61         # + shift right 6 times (0x3f = 0011 1111)
62 byte4_char = chr(byte4)
63 byte4_bin = bin (byte4)
64
65 print(cx)
66 print('cx',cx_bin)
67 print('byte1',byte1_bin)
68 print('byte2',byte2_bin)
69
70 print(cy)
71 print('cy',cy_bin)
```

```
print('byte3',byte3_bin)
print('byte4',byte4_bin)


#start handshake

ser.write(chr(0xff))    #to confirm that data is ready
print(1)
ser.write(chr(0xff))
print(2)
ser.write(chr(0xff))
print(3)
ser.write(chr(0xff))
print(4)
ack_byte = 0
received_byte = 0
while (ack_byte!= chr(239)):
  ack_byte = ser.read()   #to confirm that dSpace is ready
print(ord(ack_byte))
print(5)
ser.write(byte1_char)
print(6)
ser.write(byte2_char)
print(7)
ser.write(byte3_char)
print(8)
ser.write(byte4_char)
print(9)
while(received_byte != chr(223)):
  received_byte = ser.read()
print(ord(received_byte))
```

.4 Color detection

```
import cv2
import numpy as np

def nothing(x):
  pass

cap = cv2.VideoCapture(0);

cv2.namedWindow("Tracking")

cv2.createTrackbar("LH", "Tracking", 0, 255, nothing)
cv2.createTrackbar("LS", "Tracking", 0, 255, nothing)
cv2.createTrackbar("LV", "Tracking", 0, 255, nothing)
cv2.createTrackbar("UH", "Tracking", 255, 255, nothing)
cv2.createTrackbar("US", "Tracking", 255, 255, nothing)
cv2.createTrackbar("UV", "Tracking", 255, 255, nothing)
while True:
  _, frame = cap.read()

  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

```

```
  l_h = cv2.getTrackbarPos("LH", "Tracking")
  l_s = cv2.getTrackbarPos("LS", "Tracking")
  l_v = cv2.getTrackbarPos("LV", "Tracking")

  u_h = cv2.getTrackbarPos("UH", "Tracking")
  u_s = cv2.getTrackbarPos("US", "Tracking")
  u_v = cv2.getTrackbarPos("UV", "Tracking")

  l_b = np.array([l_h, l_s,l_v])
  u_b = np.array([u_h, u_s,u_v])

  mask = cv2.inRange(hsv, l_b,u_b)

  res = cv2.bitwise_and(frame, frame, mask=mask)

  cv2.imshow("frame", frame)
  cv2.imshow("mask",mask)
  cv2.imshow("res",res)

  key = cv2.waitKey(1) & 0xFF
  # if the 'q' key is pressed, stop the loop
  if key == ord("q"):
    break

cap.release()
cv2.destryAllWindows()
```

.5 Tracking Ball - Python Code

```
# import the necessary packages
from collections import deque #maintain a list of past location of the ball
from imutils.video import VideoStream #this package makes some few basic function
    easier such as resizing
from imutils.video import WebcamVideoStream
from imutils.video import FPS
import numpy as np
import argparse
import cv2
import imutils
import time
import serial
import serial.rs485

ser = serial.rs485.RS485(
        port='/dev/ttyUSB0', #Replace ttyS0 with ttyAM0 for Pi1,Pi2,Pi0
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)


ser.rs485_mode = serial.rs485.RS485Settings(False,True)

```

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()

#optional argument 1(--video) : if this path is supplied opencv grab a
#pointer to the video file and read frames from it. if not,
#opencv will try to access the webcam
ap.add_argument("-v", "--video",
  help="path to the (optional) video file")

#optional argument 2 (--buffer): is the maximum size of our deque
#which maintains a list of previous (x,y)-coordinates
#the greater the buffer the longer tail for the ball
ap.add_argument("-b", "--buffer", type=int, default=0,
  help="max buffer size")
args = vars(ap.parse_args())

ap.add_argument("-n", "--num-frames", type=int, default=100,
  help="# of frames to loop over for FPS test")

ap.add_argument("-d", "--display", type=int, default=-1,
  help="Whether or not frames should be displayed")
args = vars(ap.parse_args())
print("[INFO] sampling frames from webcam...")


# define the lower and upper boundaries of the "green"
# ball in the HSV color space, then initialize the
# list of tracked points
Lower_Bound_color = (81,91,0)
Upper_Bound_color = (255,255,39)
pts = deque(maxlen=args["buffer"])
# if a video path was not supplied, grab the reference
# to the webcam
if not args.get("video", False):
  vs = WebcamVideoStream(src=0).start()

  fps = FPS().start()
# otherwise, grab a reference to the video file
else:
  vs = WebcamVideoStream(src=0).start()
  fps = FPS().start()

# allow the camera or video file to warm up
time.sleep(2.0)

# keep looping
while True:
  # grab the current frame
  frame = vs.read()

  # handle the frame from VideoCapture or VideoStream
  frame = frame[1] if args.get("video", False) else frame
  # if we are viewing a video and we did not grab a frame,
  # then we have reached the end of the video
  if frame is None:
    break
  # resize the frame, blur it, and convert it to the HSV
```

```
83    # color space
84
85    frame = imutils.resize(frame, width=400)
86
87    blurred = cv2.GaussianBlur(frame, (11, 11), 0)
88
89    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
90
91    # construct a mask for the color "green", then perform
92    # a series of dilations and erosions to remove any small
93    # blobs left in the mask
94    mask = cv2.inRange(hsv, Lower_Bound_color, Upper_Bound_color)
95
96    #mask = cv2.erode(mask, None, iterations=2)
97    #mask = cv2.dilate(mask, None, iterations=2)
98
99      # find contours in the mask and initialize the current
100   # (x, y) center of the ball
101   cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
102     cv2.CHAIN_APPROX_SIMPLE)
103   cnts = imutils.grab_contours(cnts)
104   center = None
105   # only proceed if at least one contour was found
106   if len(cnts) > 0:
107     # find the largest contour in the mask, then use
108     # it to compute the minimum enclosing circle and
109     # centroid
110
111     c = max(cnts, key=cv2.contourArea)
112     ((x, y), radius) = cv2.minEnclosingCircle(c)
113     M = cv2.moments(c)
114
115     try:
116       center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
117       Cx = int(M["m10"] / M["m00"])
118       Cy = int(M["m01"] / M["m00"])
119     except ZeroDivisionError:
120       Cx =0
121       Cy =0
122
123
124     tempStr = str(Cx)+","+str(Cy)
125
126     # only proceed if the radius meets a minimum size
127     if radius > 5:
128       # draw the circle and centroid on the frame,
129       # then update the list of tracked points
130       cv2.circle(frame, (int(x), int(y)), int(radius),
131         (0, 0, 255), 2)
132       cv2.circle(frame, center, 5, (0, 0, 0), -1)
133       cv2.putText(frame, tempStr,(Cx, Cy),cv2.FONT_HERSHEY_TRIPLEX
      ,0.4,(255,255,255),1)
134
135       #byte 1
136       byte1 = (Cx) & (0x3f)      #passing the 6 lsb bits
137       byte1_char = chr(byte1)    #convert to char
138       byte1_bin = bin (byte1)
```

```
139
140
141         #byte2
142         byte2 = ((Cx>>6) & 0x3f)|0x40   #the first two MSB are the sign for x
143                 # + shift right 6 times (0x3f = 0011 1111)
144         byte2_char = chr(byte2)
145         byte2_bin = bin (byte2)
146
147
148         #byte 3
149         byte3 = ((Cy) & (0x3f))|0x80      #passing the 6 lsb bits
150         byte3_char = chr(byte3)   #convert to char
151         byte3_bin = bin (byte3)
152
153         #byte4
154         byte4 = ((Cy>>6) & 0x3f)|0xc0   #the first two MSB are the sign for y
155                 # + shift right 6 times (0x3f = 0011 1111)
156         byte4_char = chr(byte4)
157         byte4_bin = bin (byte4)
158
159         ser.write(chr(0xff))    #to confirm that data is ready
160         ser.write(chr(0xff))
161         ser.write(chr(0xff))
162         ser.write(chr(0xff))
163
164         ack_byte = 0
165         received_byte = 0
166         while (ack_byte != chr(239)):
167           ack_byte = ser.read()   #to confirm that dSpace is ready
168         print(ord(ack_byte))
169
170         ser.write(byte1_char)
171
172         ser.write(byte2_char)
173
174         ser.write(byte3_char)
175
176         ser.write(byte4_char)
177
178         while(received_byte != chr(223)):
179           received_byte = ser.read()
180
181         print(ord(received_byte))
182
183
184
185     # update the points queue
186     pts.appendleft(center)
187       # loop over the set of tracked points
188     for i in range(1, len(pts)):
189       # if either of the tracked points are None, ignore
190       # them
191       if pts[i - 1] is None or pts[i] is None:
192         continue
193       # otherwise, compute the thickness of the line and
194       # draw the connecting lines
195       thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
```

```
196     cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)
197   # show the frame to our screen
198   cv2.imshow("Frame", frame)
199   #cv2.imshow("mask",mask)
200   key = cv2.waitKey(1) & 0xFF
201   fps.update()
202   # if the 'q' key is pressed, stop the loop
203   if key == ord("q"):
204     break
205 fps.stop()
206 print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
207 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
208 # if we are not using a video file, stop the camera video stream
209 if not args.get("video", False):
210   vs.stop()
211 # otherwise, release the camera
212 else:
213   vs.release()
214 # close all windows
215 cv2.destroyAllWindows()
```