BALL AND PLATE SYSTEM

Team members:

Amr El Gendy

Khaled Hefnawy

Mahmoud El Sayed

Youssef Sherif

Index:

Introduction:

In this paper we are going to study and derive a mathematical model for the ball and plate system, then this model is going to be implemented in Simulink with an animation that illustrates the model. A detailed description is then given about manufacturing a ball and plate device. Finally the results of the Simulink model and the animation will be reported accompanied with recommendations and future research.

Mathematical Model:

- Approximations and assumptions:

In this project, The model derived and used is a decoupled model, it assumes that the motion of the ball is uncoupled in the two perpendicular axes of the plate, which means that the motion of the ball due to the rotation of plate around one of its axes is independent on the motion due to the rotation of the plate around the other axis. Accordingly, a 2D-model will be derived for each degree of freedom of the plate, then the principle of superposition will be applied to obtain the exact position of the ball.

The assumptions used for this model are:

- No slipping conditions for the ball.
- The ball is completely symmetric and homogeneous.
- Rolling friction is neglected.
- The ball and plate are in contact at all times.
- The plate surface is perfectly smooth and rigid.

The following table contains the parameters used for deriving the model and their significance:

Parameter	Description
l	Length of the plate
r	Arm length of the servo
α	Plate angle in x-direction
β	Plate angle in y-direction
θ	Servo angle
m	Mass of the ball
R	Radius of the ball
I	Moment of inertia of the ball
g	Gravitational acceleration
V	Motor Voltage

- Derivation:

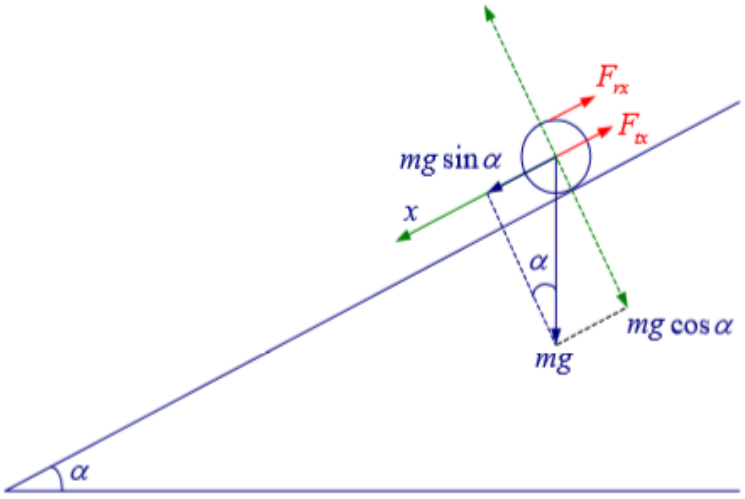


Fig. 1 free body diagram

Considering the free body diagram in fig. 1 where the direction of increasing x is upward along the inclined plane, According to Newton's second law the total force responsible for the translational motion is governed by the following equation

$$F_{tx} = m\frac{d^2x}{dt^2} = m\ddot{x} \qquad (1)$$

and the rotational motion (Torque) is given by

$$T_r = I\frac{\ddot{x}}{R}$$

thus, by dividing the last equation by R, and substituting $I = \frac{2}{5}mR^2$, we have

$$F_r = \frac{T_r}{R} = I\frac{\ddot{x}}{R^2} = \frac{2}{5}m\ddot{x} \qquad (2)$$

We can now say that according to fig.1, $F_{tx} = mgsin\alpha - F_r$. Thus by substituting (1) and (2) into the last equation:

$$m\ddot{x} + \frac{2}{5}m\ddot{x} = mgsin\alpha$$

by dividing by m and rearranging the equation, we obtain:

$$\ddot{x} = \frac{5}{7}gsin\alpha \qquad (3)$$

The same equation can be obtained for the y-direction:

$$\ddot{y} = \frac{5}{7}gsin\beta \qquad (4)$$

Equations (3) and (4) can be easily integrated to obtain the position of the ball at any instant. We now have the position of the ball in terms of the angles of the plate, but in order for this model to be integrated into the system, the position of the ball will have to be related to the voltage given to the servo motor in control of the rotation of the plate. Therefore we use the following model.

- Servo Model:

In an attempt to simplify calculations, A DC-Motor was used as an approximation for the servo. The standard transfer function for the DC-motor was used:

$$\frac{\theta(s)}{V(s)} = \frac{C_1}{s(s + C_2)}$$

Where C_1 and C_2 are both functions of the parameters of the motor, and since θ and α or β are related by the geometry of the setup, in our case $\alpha = \frac{r}{l}\,\theta$, therefore $\ddot{x}$ can now be related to the voltage of the motor.

Simulink Model:

The Simulink model for this system is formed of several blocks and subsystems:

- Ball Model subsystem:

As shown in fig.2, this subsystem models equation (3). It has one input, which is the angle α in case of x-direction and β in case of y-direction, and two outputs, x and $\dot{x}$.

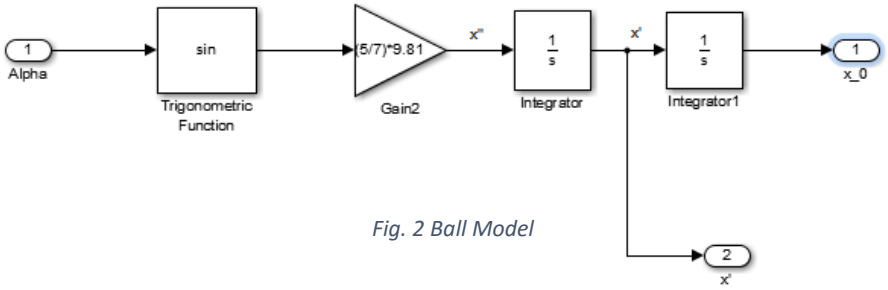


Fig. 2 Ball Model

- Feedback loop 1 (Servo PID):

Since the motor used in this model is only an "approximated" servo motor, The voltage corresponding to a specific θ may not be able to achieve that desired θ, which will affect the entire structure. To solve this, the feedback loop (PID controller) shown in

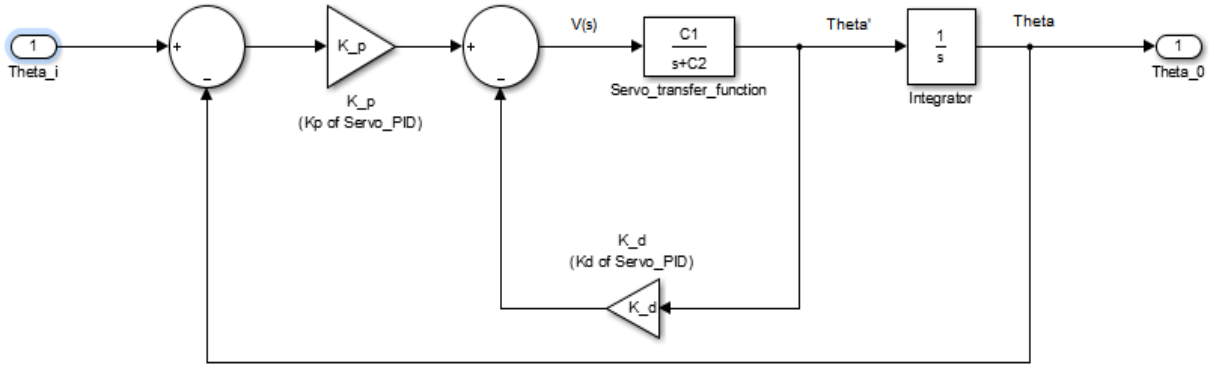


Fig. 3 Servo feedback loop

fig.3 was introduced to ensure θ reaches the desired set-point (Note that this problem only arises in the simulation because of the motor model used, thus this feedback loop will not be necessary when implementing the model on hardware with "real" servo). The input of the subsystem is the required theta and the output is the instantaneous theta. Since the original transfer function was

$$\frac{\theta(s)}{V(s)} = \frac{C_1}{s(s + C_2)}$$

By reducing the block diagram in fig.3, the transfer function for the subsystem with PID feedback loop becomes

$$\frac{\theta_0(s)}{\theta_i(s)} = \frac{K_p C_1}{s^2 + (K_D C_1 + C_2)s + K_p C_1}$$

by comparing the above equation to the general second order transfer function

$$\frac{\theta_0(s)}{\theta_i(s)} = \frac{\omega_n^2}{s^2 + 2\omega_n \zeta s + \omega_n^2}$$

The constants K_p and K_d can be obtained, where

$$K_d = \frac{2\omega_n \zeta - C_2}{C_1} \qquad K_p = \frac{\omega_n^2}{C_1}$$

Therefore, by choosing the parameters ω_n and ζ appropriate to the system, K_p and K_d can be determined. The parameters chosen for this system are attached in Appendix B and in the m-file attached to this report.

- Feedback loop 2 (Distance PID):

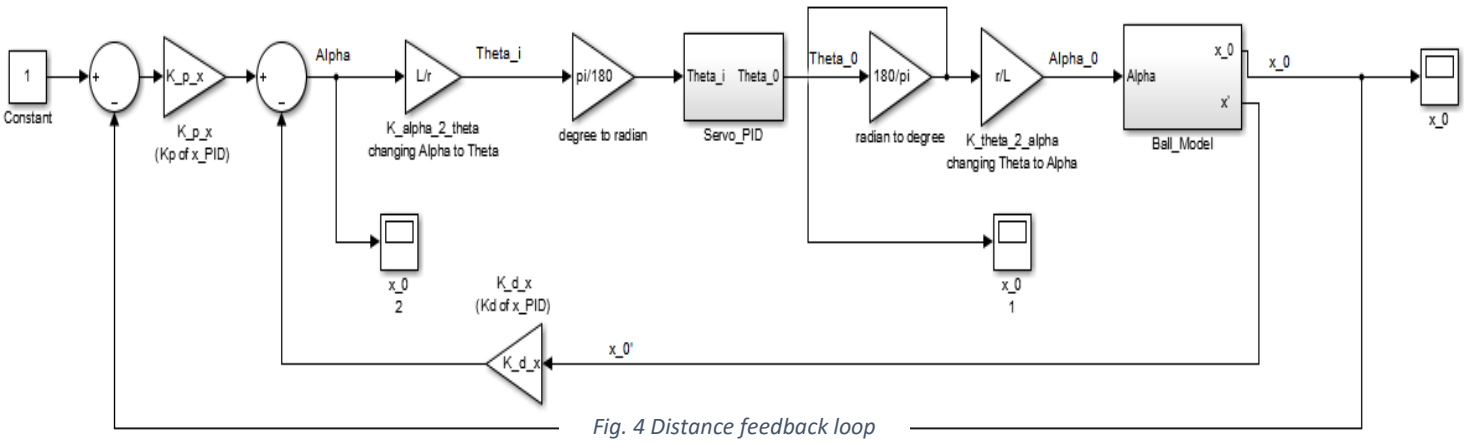


Fig. 4 Distance feedback loop

In fig.4, the main feedback loop is shown with the rest of the model, where a similar PID feedback control was used to ensure that the target position is reached. The input here (the constant) is the target x which is x_i and the output is the instantaneous x_0 This is done by applying the PID equation

$$\alpha = K_{px}(x_i - x_0) - K_{dx}\dot{x}$$

by transforming the previous equation into the frequency domain, it becomes

$$\alpha(s) = K_{px}\big(x_i(s) - x_0(s)\big) - K_{dx} s x(s)$$

and since for small α, equation (3) becomes $\ddot{x} = \frac{5}{7} g\alpha$, thus by transferring it into the frequency domain, the equation becomes

$$\alpha(s) = \frac{7s^2}{5g} x_0(s) = \frac{s^2}{7} x_0(s)$$

by substituting into the PID equation, it becomes

$$\frac{s^2}{7} x_0(s) = K_{px}\big(x_i(s) - x_0(s)\big) - K_{dx} s x_0(s)$$

by further simplifying, the transfer function for the feedback loop is obtained

$$\frac{x_0(s)}{x_i(s)} = \frac{7K_{px}}{s^2 + 7K_{dx}s + 7K_p}$$

Finally, by comparing with $G(s) = \frac{\omega_n^2}{s^2 + 2\omega_n \zeta s + \omega_n^2}$, we get

$$K_d = \frac{2\omega_n \zeta}{7} \qquad K_{px} = \frac{\omega_n^2}{7}$$

Animation: (details about VRML coordinate system found in appendix D)

We linked our model to animation using VR sink. The first step we made was to make the design file of the ball and plate. We entered the data which came from the PID x and y coordinates on a multiplexer and entered them on a VR expander to produce x, y, z coordinates of the ball as shown in figure 5. we faced a problem in rotation that when we enter the angle of rotation to the plate and enable it to rotate on x and z axes, it rotates only about the diagonal of the plate as shown in figure 6.we need to have to independent rotation axes. So we made a transform into a transform and defined two axis of rotation in each transform to have two independent axes of rotation. The transform is the container of geometry. For instance, if we want to make a sphere. We first make a transform and define its coordinates in space and then create inside it the shape we want. The transform is the parent node and the shape is the child node as illustrated in VRML method to represent a 3-D scene

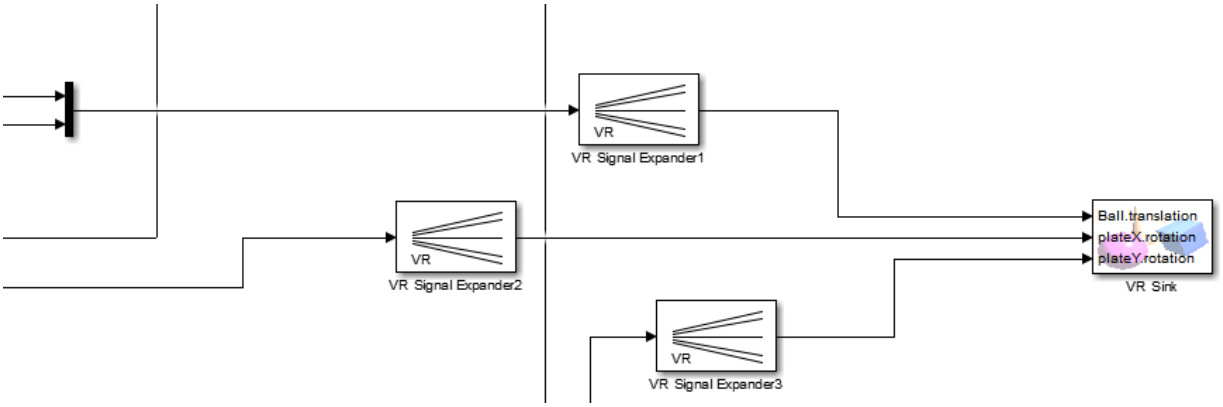


Figure 5

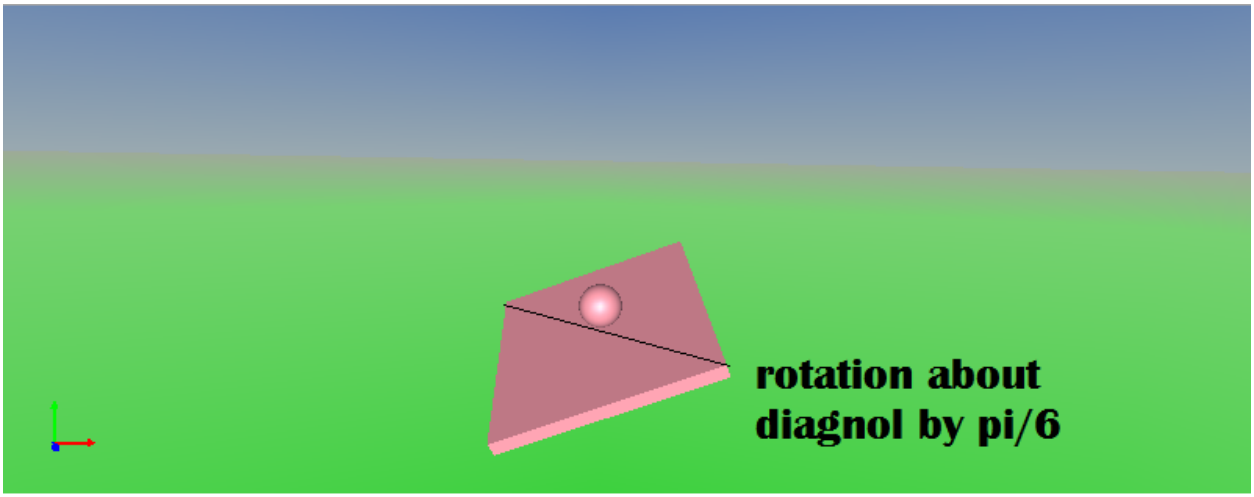


Figure 6

Figure 7 shows that in transform (plate X) we have defined rotation vector 0 0 1 0 which puts 0 in the x and y to disable the rotation in those directions and puts 1 in the z axis to enable the rotation in z axis. The last number in the rotation vector represents the value of the angle by which the plate will rotate. Inside transform X we made the rotation vector 1 0 0 0 to enable the x axis rotation. The last value in each vector is entered from our model to make the plate rotate about the specified axis with the specified value.

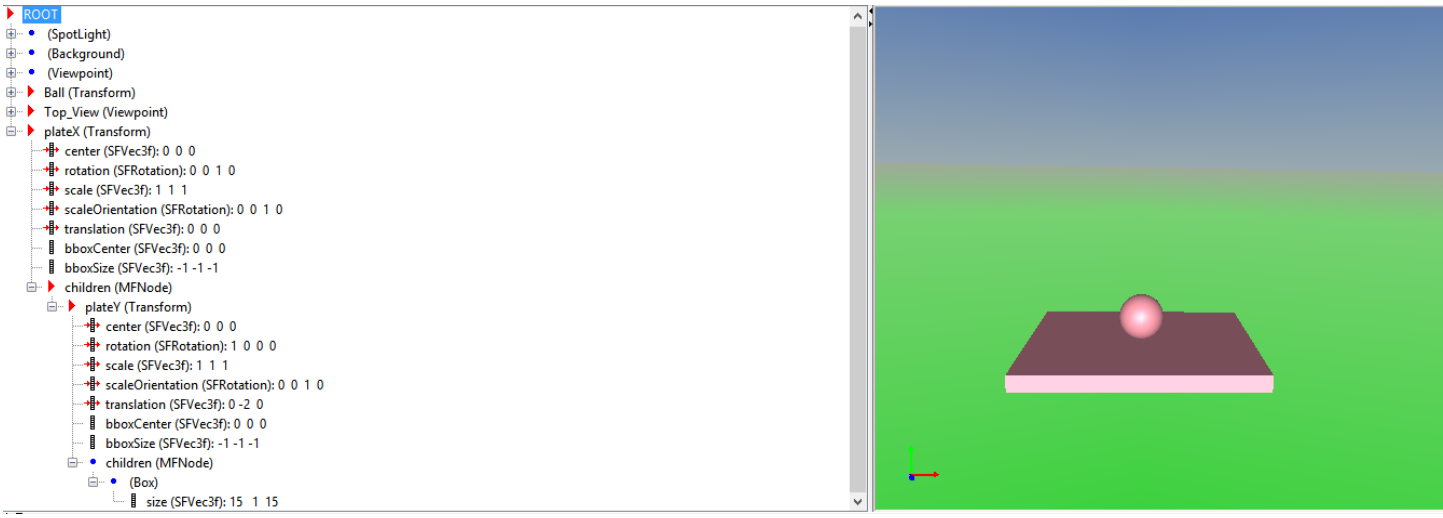


Figure 7

The results are shown in the attached video.

Mechanical implementation:

The model consists of two degrees of freedom, so in order to achieve such a model, we constructed the device based on two servo motors, one for each angle. The concept design was made using solid works, it consists mainly of three parts which are the plate with the arms, the base with the servos, and the camera holder.

First of all the plate, which is $35 \times 35\ cm$, is shown in figure 8. It had to be flat and strong. We decided to make it from wood in order to be light, but the problem with wood was that it bend pretty easily and it was hard to get a flat plate. So the chosen material was 1.5 mm thick stainless steel due to its strength and relatively less weight compared to aluminum with the same strength. To attach the arms to the plate we chose a special type of bearing called 'SKF GE8c', shown in figure 9, which acts like a ball and socket linkage to provide the degrees of freedom required. We fixed these bearings to the plates by making brass holders.

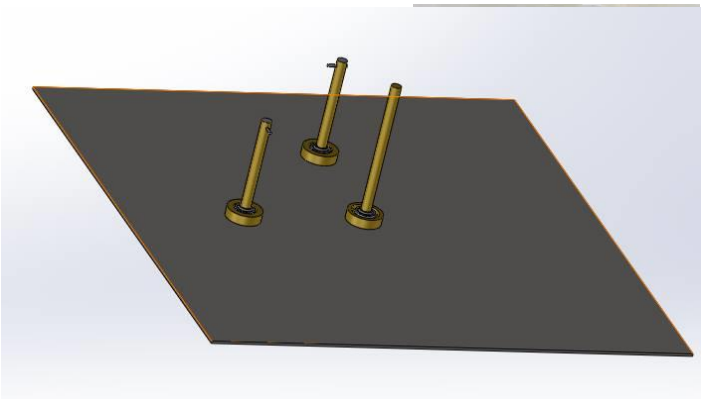


Figure 8

Figure 9

The base and the camera holder were made in the lab in order to get the appropriate lengths. We chose servos with torque equal to 10 kg.cm to avoid any unwanted plate movement due to its large moment of inertia. The camera holder was made by fixing a long piece of wood to the base, from its top we extended a screw rod which made the camera holder

adjustable. The whole device after manufacturing and assembly is shown in figure 10

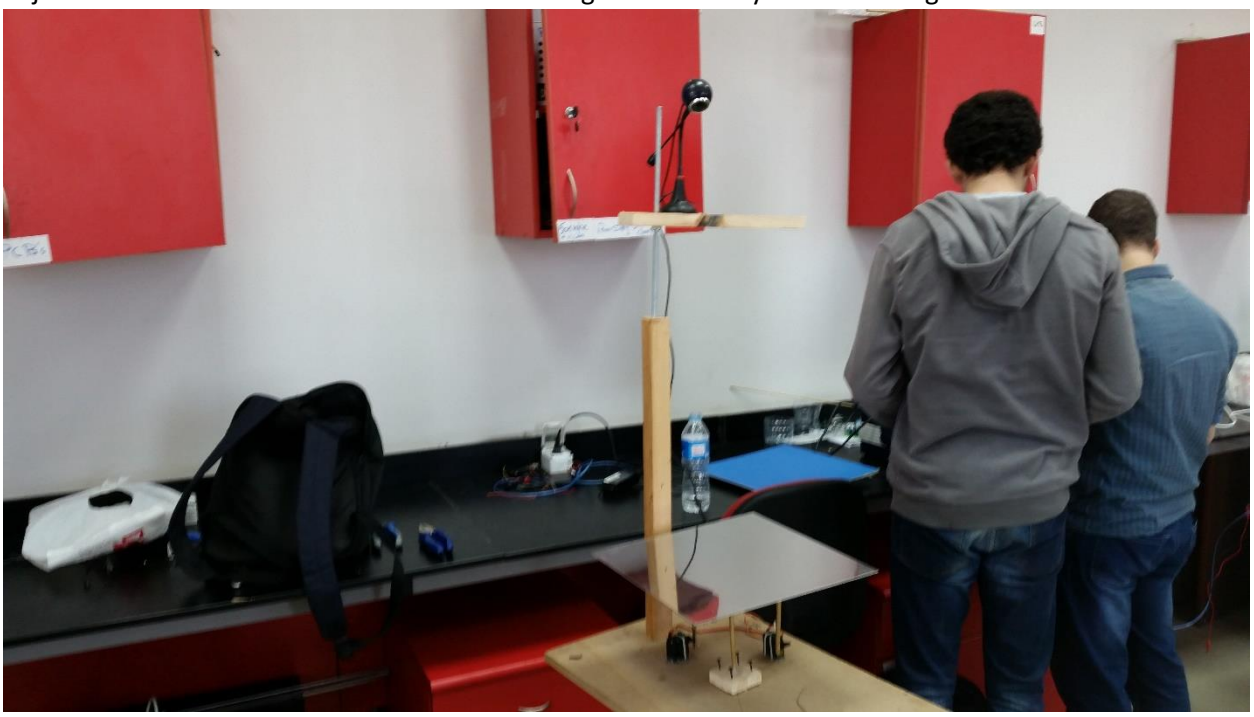


Figure 10

Detailed pictures are shown in Appendix A.

Position detection:

There are many ways to specify the location of the ball on the plate for the microcontroller such as the touch pad, ultrasonic sensor and image processing. Since the touch pad is a little expensive and not that available and the ultrasonic sensor may experience noise that make the mission harder, we went with the remaining solution, image detection. A cam is suspended perpendicularly above the plate that captures images of size 640×480. We chose the 480 to be the dimension of the Y-axis, hence the 640 is the X-axis. The plate is 35×35 cm square, the camera is adjusted so that the length of the plate is filling the height of the image, that's 35 cm is seen as 480 pixels (Height of the captured image). The camera is also adjusted to make the plate centered with respect to the X-axis.

Images are captured using the suspended cam, Visual Studio code converts it from RGB to HSV because the ball detection is easier that way, then the image is smoothed using Gaussian filter and some parameters are chosen to make the ball white and the plate black in the image to make it easier to detect. The code can now detect any circles in the image, a threshold is implemented to remove the small noise or errors. The next step is easy, is to determine the location of the center of the ball, which we consider the reference point on the ball, and the

radius of the ball. This location is in Pixels not cm, so we must map the values of the pixels to cm.

Since we adjusted the plate to fill the height of the image we know that $35\ cm = 480\ pixels$

$$or \quad 1cm = \frac{480}{35}\ pixels \qquad \rightarrow \ y_{cm} = y_{pixels} * \frac{35}{480}$$

For the X position, the plate is centered and we conclude that the plate is $\frac{640-480}{2}$ pixels away from the zero on the X-axis

$$x_{cm} = \left(x_{pixels} - 80\right) * \frac{35}{480}$$

By subtracting 80 from the x position, the reference is shifted 80 pixels to the right (The edge of the plate).

We have succeeded in knowing the instantaneous position of the ball on the plate (The Visual Studio Code is in Appendix C).

Results:

Here we discuss the results of the Simulink simulation and the animation. Also the results of the ball detection will be reported. But due to the shortage of time the hardware couldn't be tested, and thus the Arduino code might need some debugging.

- Simulink results:

  At first, the problem of obtaining a suitable response was approached by trying to achieve a critically damped response for the fastest non oscillatory time response. The response in figure 11 was the result of this trial. Unfortunately, the θ readings required to reach that response were not physically possible as shown in figure 12. As a result, another trial was executed, where the response was allowed to overshoot by a small percentage of about 10% to obtain reasonable practical θ values, figure 13 was the result of that trial, and the figure 14 shows the theta values.

The gains used in the first trial were: $K_{px} = 2.29, K_{dx} = 1.02857, K_p = 746.98\ and\ K_d = 6.078$. The gains for the second trial are those shown in the appendix and in the attached Parameters.m file

- The animation results are found in the video attached with the report.
- The ball detection code is working and it detects the ball.

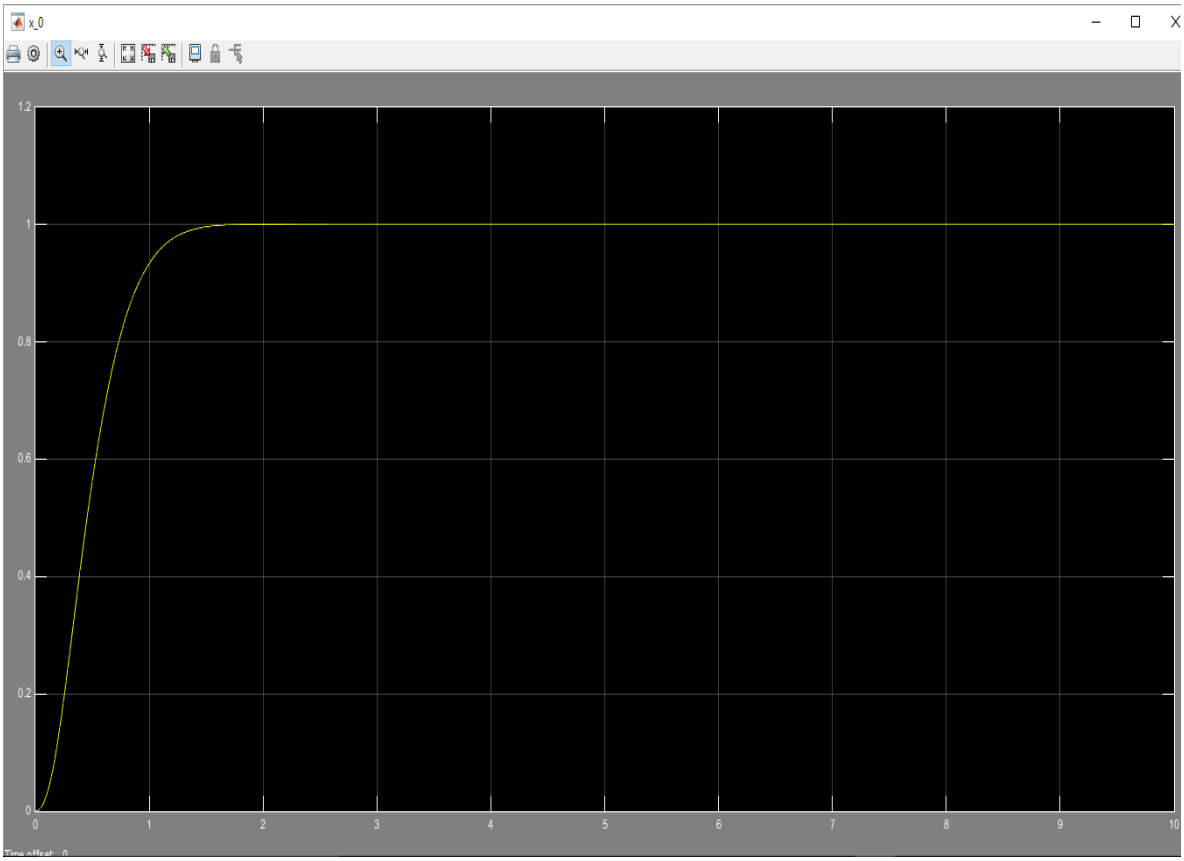


Figure 11

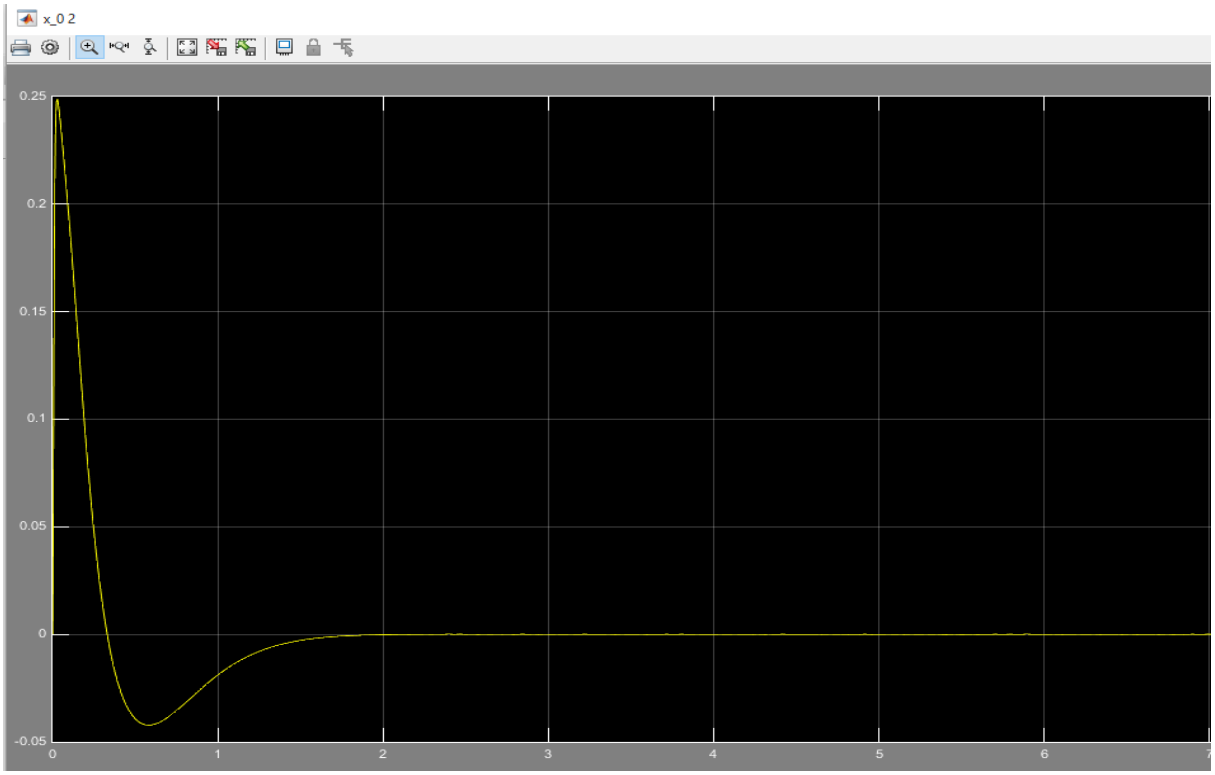


Figure 12

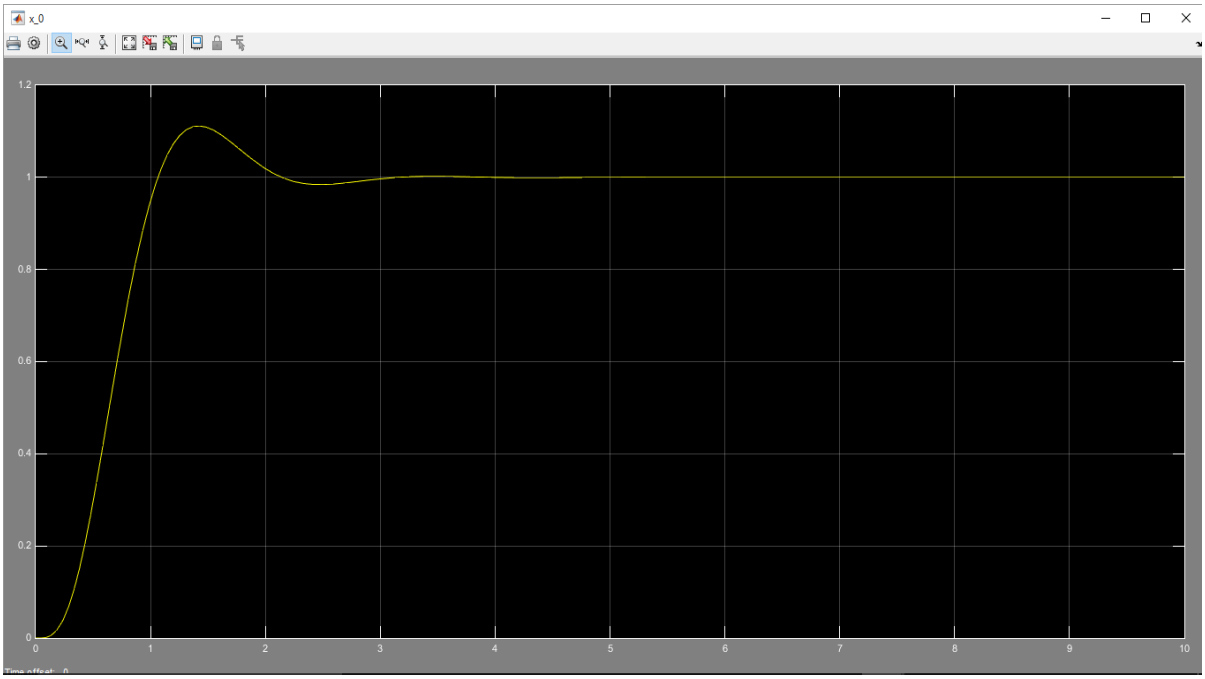


Figure 13

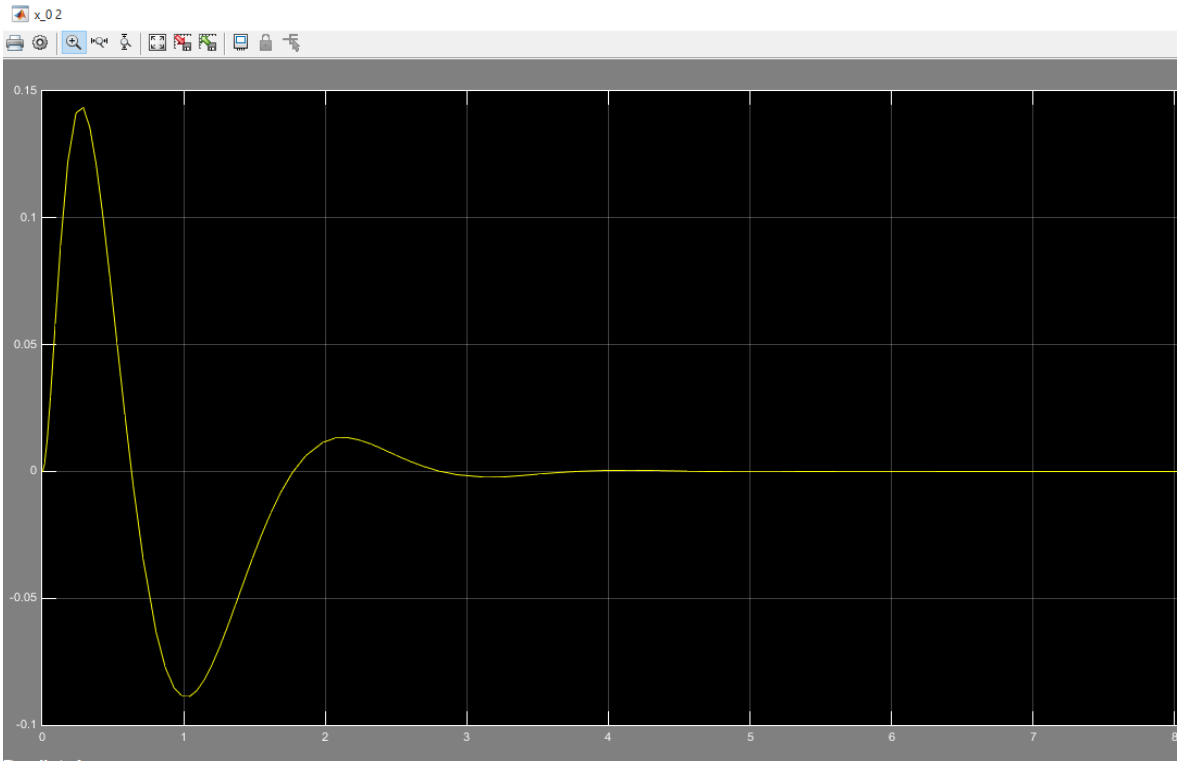


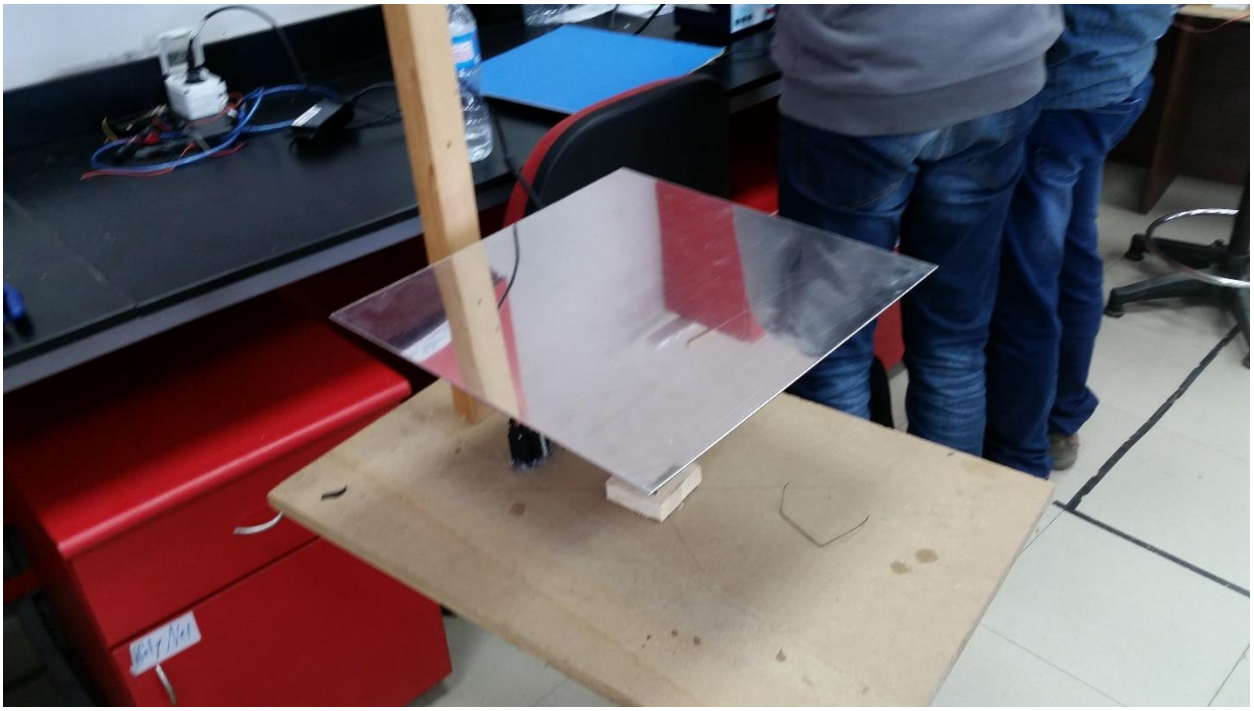
Figure 14

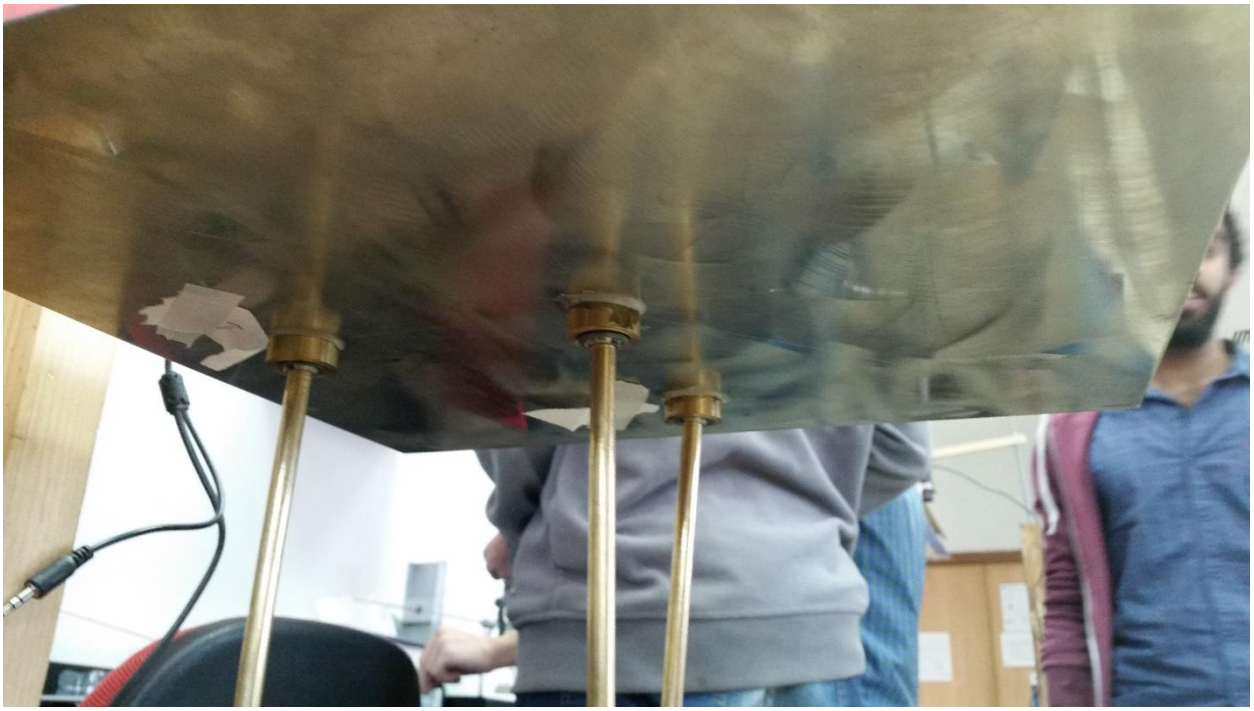
Future research:

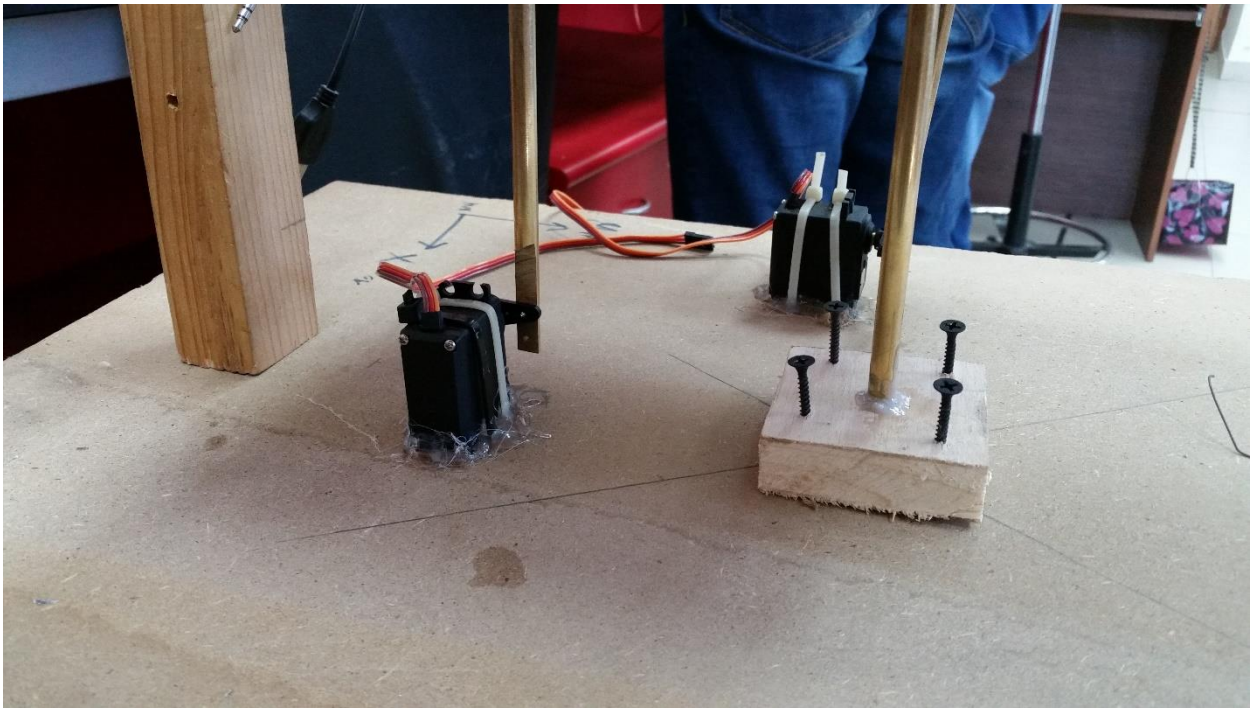
The hardware device must be tested in order to verify the Arduino code. Further improvements can then be done to the Simulink model in order to make it as realistic as possible, one of the improvements is to perform a parameter estimation on the gain of the angle by taking real data from the hardware device and performing parameter estimation in Matlab. The Simulink model can be further improved by adding nonlinear terms such as $\sin(\boldsymbol{\theta_1})$ where $\boldsymbol{\theta_1}$ is the corresponding servo angle, also the coupling between the two degrees of freedom due to Coriolis Effect can be added by putting in each degree of freedom equation another term equals to $\boldsymbol{\theta_2}$ where $\boldsymbol{\theta_2}$ the angle of the opposite servo. All these added terms are multiplied by unknown coefficients which can be acquired using the great parameter estimation tool in Matlab.

After acquiring an almost complete model of the hardware in Simulink, optimal control techniques can be applied in order to get the PID constants which will provide the best response according to the cost function which will mainly depend on the setting time and the steady state error.

Appendix A: Detailed pictures of the hardware







Appendix B: Parameters and Arduino code

- System parameters:

```
% motor Parameters
 R_a=2.6;
 K_m=0.00767;
 K_t=K_m;
 J_m=3.87*(10)^-7;
 J_t=0.7*(10)^-7;
 K_g=14*5;
 B_eq=4*(10)^-3;
 eff=0.7395;
 J_eq=0.0023;
 C1=(eff*K_m*K_g)/(R_a*J_eq);
 C2=(B_eq/J_eq)+((eff*K_m*K_m*K_g*K_g)/(R_a*J_eq));

 % x_PID
 t_p_x=1.8;
 damping_ratio_x= 0.65;
 w_n_x=3.14/(t_p_x*(1-damping_ratio_x^2)^(1/2));
 K_p_x=w_n_x^2/7;
 K_d_x=(2*damping_ratio_x*w_n_x)/7;

 % Servo_PID
 t_p=0.5;
 damping_ratio= 0.85;
 w_n=3.14/(t_p*sqrt(1-damping_ratio^2));
 K_p=w_n^2/C1;
 K_d=((2*damping_ratio*w_n)-C2)/C1;

 % Plate
 L=0.35;
 r=0.0254;
```

- Arduino code:

```
#include <Servo.h>
Servo servo1;   // creating a servo object
Servo servo2;   // creating a servo object

/// Variables ///
float Kp=2.1405;
float Kd=-0.25785;
float Kp_x=0.75277;
float Kd_x=0.42631;
float Setpoint_x;  // this is set manually
float Setpoint_y;  // this is set manually
float L;  // half the length of the plate
float r;  // the arm length of the servo
int pos_x=0;
```

```
int pos_y=0;
float error_x, error_d_x, error_x_old, Alpha_x, Theta_setpoint_x;
float error_y, error_d_y, error_y_old, Alpha_y, Theta_setpoint_y;
float error_theta_x, error_theta_old_x, Theta_x, error_d_theta_x, V_x;
float error_theta_y, error_theta_old_y, Theta_y, error_d_theta_y, V_y;

void setup() {
 servo1.attach(9);       // attaches the servo object to pin 9 on the arduino board
 servo2.attach(10);      // attaches the servo object to pin 9 on the arduino board
}

void loop() {
  ////////////////// PID_x ////////////////////////
  // put here instead of this line the function that reads the position and assume it gives pos_x and
pos_y relative to the a fixed origin (center of the plate) ///
  error_x=pos_x-Setpoint_x;
  error_d_x=error_x-error_x_old;
  error_x_old=error_x;
  Alpha_x=Kp_x*error_x+Kd_x*error_d_x;      // alpha is the angle of plate relative to the zero
position
  Theta_setpoint_x=Alpha_x*(L/r)*(3.14/180);  // Theta is the angle of the servo
  ////////////////// PID_y ////////////////////////
  error_y=pos_y-Setpoint_y;
  error_d_y=error_y-error_y_old;
  error_y_old=error_y;
  Alpha_y=Kp_x*error_y+Kd_x*error_d_y;      // alpha is the angle of plate relative to the zero
position
  Theta_setpoint_y=Alpha_y*(L/r)*(3.14/180);  // Theta is the angle of the servo
  ////////////////// Servo_PID_x ////////////////////
  Theta_x=servo1.read();
  error_theta_x= Theta_x-Theta_setpoint_x;
  error_d_theta_x=error_theta_x-error_theta_old_x;
  error_theta_old_x=error_theta_x;
  V_x=Kp*error_theta_x+Kd*error_d_theta_x;
  ////////////////// Servo_PID_y ////////////////////
  Theta_y=servo2.read();
  error_theta_y= Theta_y-Theta_setpoint_y;
  error_d_theta_y=error_theta_y-error_theta_old_y;
  error_theta_old_y=error_theta_y;
  V_y=Kp*error_theta_y+Kd*error_d_theta_y;

  /// an equation depending on the parameters of the servo will change the voltage into a
corresponding angle for the servo to be able to use servo.write() ///
```

```
// Theta_x=some_equation(V)*(180/3.14)
servo1.write(Theta_x);

 servo2.write(Theta_y);
```

Appendix C: ball detection ‘visual studio’ code

```
#include <iostream>
#include<opencv/cvaux.h>
#include<opencv/highgui.h>
#include<opencv/cxcore.h>
#include <sstream>
#include <string>
#include <opencv\cv.h>
#include<stdio.h>
#include<stdlib.h>
// Need to include this for serial port communication
#include <Windows.h>
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
using namespace cv;
const string trackbarWindowName = "Trackbars";
void on_trackbar( int, void* )
void createTrackbars(){
//create window for trackbars
namedWindow(trackbarWindowName,0);
//create memory to store trackbar name on window
char TrackbarName[50];
sprintf( TrackbarName, "H_MIN", H_MIN);
sprintf( TrackbarName, "H_MAX", H_MAX);
sprintf( TrackbarName, "S_MIN", S_MIN);
sprintf( TrackbarName, "S_MAX", S_MAX);
```

```
sprintf( TrackbarName, "V_MIN", V_MIN);
sprintf( TrackbarName, "V_MAX", V_MAX);

createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );



}
int main(int argc, char* argv[])
{
FILE *fp;
fp=fopen("file.txt","w");
HANDLE hSerial = CreateFile(L"COM7", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0);

if (hSerial !=INVALID_HANDLE_VALUE)
{printf("Port opened! \n");
DCB dcbSerialParams;
GetCommState(hSerial,&dcbSerialParams);
dcbSerialParams.BaudRate = CBR_9600;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.Parity = NOPARITY;
dcbSerialParams.StopBits = ONESTOPBIT;
SetCommState(hSerial, &dcbSerialParams);
}
else
```

```
{
if (GetLastError() == ERROR_FILE_NOT_FOUND)
{printf("Serial port doesn't exist! \n");}

printf("Error while setting up serial port! \n");
}

char outputChars[] = "c";   //
DWORD btsIO;

CvSize size640x480 = cvSize(640, 480);
CvCapture* p_capWebcam;                                                    / . .

IplImage* p_imgOriginal;
IplImage* p_imgProcessed;
IplImage* p_imgHSV;
CvMemStorage* p_strStorage;
CvSeq* p_seqCircles;
float* p_fltXYRadius;

int i;
char charCheckForEscKey;
p_capWebcam = cvCaptureFromCAM(1);
if(p_capWebcam == NULL) {
printf("error: capture is NULL \n");
getchar();
return(-1);
}
cvNamedWindow("Original", CV_WINDOW_AUTOSIZE);
cvNamedWindow("Processed", CV_WINDOW_AUTOSIZE);
```

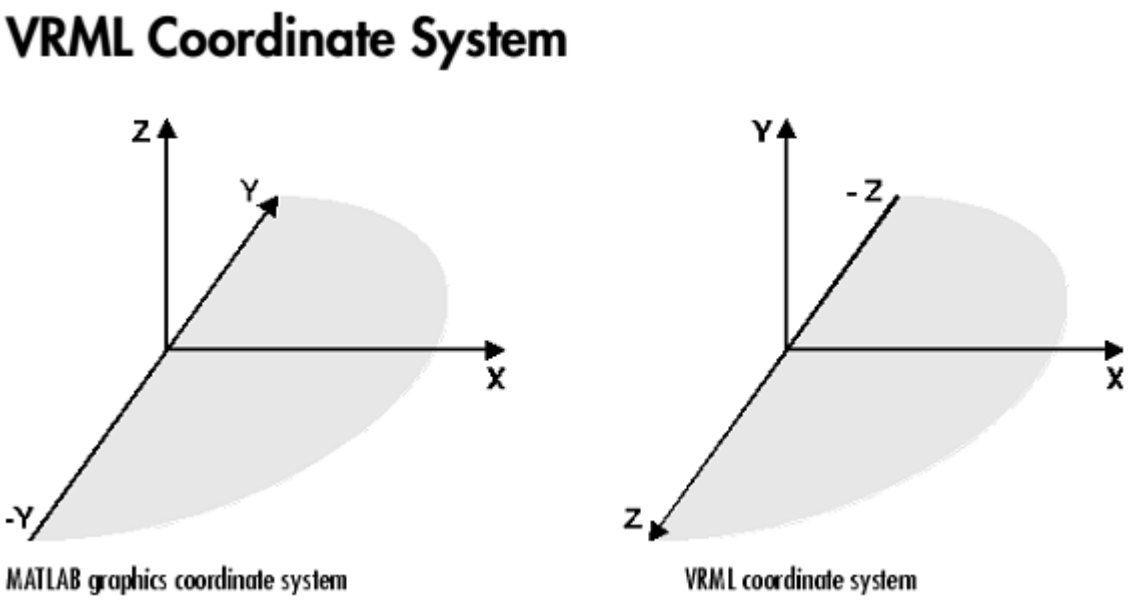
```
createTrackbars();
p_imgProcessed = cvCreateImage(size640x480,
IPL_DEPTH_8U,
1);
p_imgHSV = cvCreateImage(size640x480, IPL_DEPTH_8U, 3);
int x_pos;
int y_pos;
int servoPosition = 90;
int servoOrientation = 0;
int servoPosition1=90;
int servoOrientation1=0;
char si[2];
int g=0;
while(1) {
g++;
p_imgOriginal = cvQueryFrame(p_capWebcam);

if(p_imgOriginal == NULL) {
getchar();
break;
}
cvCvtColor(p_imgOriginal, p_imgHSV, CV_BGR2HSV);
cvInRangeS(p_imgHSV,
cvScalar(H_MIN,  S_MIN,  V_MIN),
cvScalar(H_MAX, S_MAX, V_MAX),
p_imgProcessed);
p_strStorage = cvCreateMemStorage(0);



p_imgProcessed,
```

```
CV_GAUSSIAN,
9,
9);



p_seqCircles = cvHoughCircles(p_imgProcessed,
p_strStorage,
CV_HOUGH_GRADIENT,
2,
p_imgProcessed->height / 4,
100,
50,
10,        //10
400);
if (p_seqCircles->total == 1)
{
p_fltXYRadius = (float*)cvGetSeqElem(p_seqCircles, 1);
x_pos=(p_fltXYRadius[0]-80)*40/(480);
y_pos=p_fltXYRadius[1]*40/480;
si[0]=x_pos+40;
si[1]=y_pos;
WriteFile(hSerial, si, strlen(si), &btsIO, NULL);
printf("%d   ball position x = %d, y = %d, r = %f \n",g,x_pos,
y_pos,
p_fltXYRadius[2]);
fprintf(fp,"%d  %d  %d\n",g,x_pos,y_pos);
servoOrientation = 0;
cvCircle(p_imgOriginal,
cvPoint(cvRound(p_fltXYRadius[0]), cvRound(p_fltXYRadius[1])),
3,
```

```
CV_RGB(0,255,0),
CV_FILLED);



cvCircle(p_imgOriginal,
cvPoint(cvRound(p_fltXYRadius[0]), cvRound(p_fltXYRadius[1])),
cvRound(p_fltXYRadius[2]),
CV_RGB(255,0,0),
3);
}

cvShowImage("Original", p_imgOriginal);
cvShowImage("Processed", p_imgProcessed);

cvReleaseMemStorage(&p_strStorage);
charCheckForEscKey = cvWaitKey(10);
if(charCheckForEscKey == 27) break;
}
cvReleaseCapture(&p_capWebcam);
cvDestroyWindow("Original");
cvDestroyWindow("Processed");
// This closes the Serial Port
CloseHandle(hSerial);
return(0);
}
```

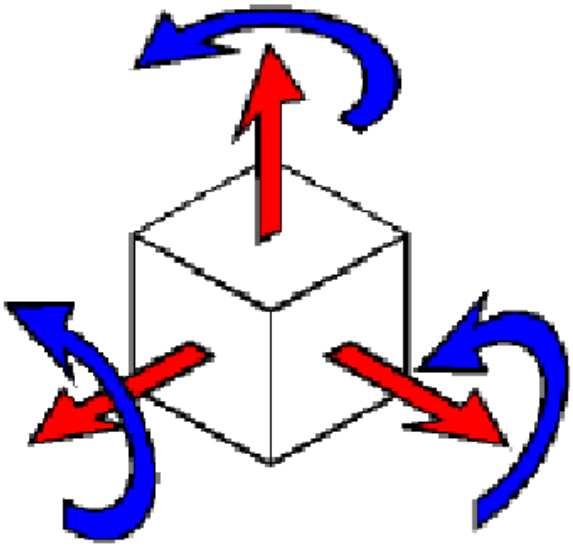
Appendix D:

Brief description of VRML (Virtual Reality Modeling Language)



The VRML coordinate system is different from the MATLAB and Aerospace Blockset™ coordinate systems. VRML uses the *world coordinate system*: the *y*-axis points upward and the *z*-axis places objects nearer or farther from the front of the screen. Understanding the coordinate system is important when you interact with different coordinate systems. SimMechanics uses the same coordinate system as VRML.

Rotation angles — In VRML, rotation angles are defined using the *right-hand rule*. Imagine your right hand holding an axis while your thumb points in the direction of the axis toward its positive end. Your four remaining fingers point in a counterclockwise direction. This counterclockwise direction is the positive rotation angle of an object moving around that axis.



VRML method to represent a 3-D scene

VRML uses a hierarchical tree structure of objects (nodes) to describe a 3-D scene. Every node in the tree represents some functionality of the scene. There are many different types of nodes. Some of them are *shape nodes* (representing real 3-D objects), and some of them are *grouping nodes* used for holding child nodes. Here are some example nodes:

• Box — Represents a box in a scene.

• Transform — Defines position, scale, scale orientation, rotation, translation, and children of its subtree (grouping node).

• Material — Corresponds to material in a scene.

• DirectionalLight— Represents lighting in a scene.

• Fog — Allows you to modify the environment optical properties.

• ProximitySensor — Brings interactivity to VRML97. This node generates events when you enter, exit, and move within the defined region in space.

Child objects — In the hierarchical structure of a VRML file, specify the position and orientation of child objects relative to the parent object. The parent object has its local coordinate space defined by its own position and orientation. Moving the parent object

also moves the child objects relative to the parent object.

Measurement units — All lengths and distances are measured in *meters*, and all angles are measured in *radians*.

References

(n.d.). Retrieved from saadat.us/download/ee371lab_manual/9_Ball_and_Beam.pdf

Nise, N. S. (2011). *Control systems engineering.*

Nokhbeh, M., & Khashabi, D. (2011). Modelling and Control of Ball-Plate System.