Ball-Plate Balancing System with Stewart Platform

Naval Selvan S. (20084016), **Rohan Saha** (20085086), **Shalini Sihag** (20084023) under **Dr. Shyam Kamal**, Member, IEEE Department of Electrical Engineering, Indian Institute of Technology (BHU) Varanasi

***Abstract*—We are designing a Ball and Plate System (BPS) on a Stewart Platform. The system consists of an Arduino Uno which is programmed with a PID controller. The controller is tasked with restoring the ball to a particular position on the platform even when slightly disturbed.**

***Index Terms*— Ball-Plate System, Stewart Platform**

I. Introduction

Ball - Plate System

The Ball is balanced to always come back to a Specific Position/ coordinate [like (0,0)] or traverse any Specific Trajectory on a Position-Sensitive Resistive Touch Plate.

Stewart Platform

Stewart platform is a parallel manipulator used for position and motion control. It consists of two platforms with six adjustable legs in between them being actuated by linear actuators or servo motors. The top platform has six degrees of freedom: X, Y and Z translations, roll, pitch and yaw (rotations about the X, Y, and Z axes respectively).

:

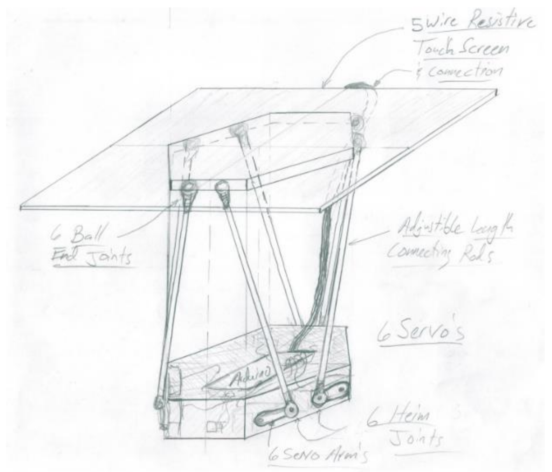

5 Wire Resistive Touch Screen & Connection
6 Ball End Joints
Adjustable Length Connecting Rods
6 Servo's
Arduino
6 Heim Joints
6 Servo Arm's

Fig 1.1 : Sketch of Ball-Plate Svstem with Stewart Platform

II. Mathematical modelling of Ball Plate system

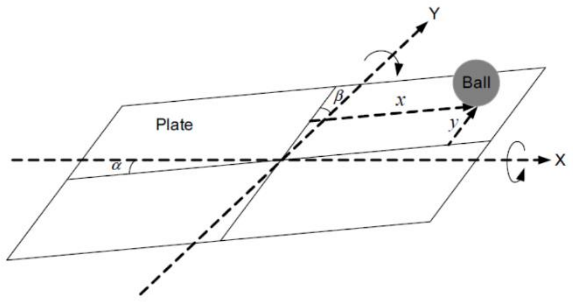


Fig 2.1: Resistive Plate Schematics

The above seen apparatus is Ball-on-Plate system. We will derive the non-linear differential equations of Ball-on-Plate apparatus using Lagrange- Euler equations. We use the linearised model to design compensator for the system.

In this section we are going to derive the motion equations of system. In this part we assume following simplifications:

- There is no slipping for ball.
- The ball is completely symmetric and homogeneous.
- All frictions are neglected.
- The ball and plate are in contact all the time.

Here we derive dynamical equations of ball-on-plate system by the help of Lagrangian. The Euler-Lagrange equation of ball-plate system is as followings:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_i$$

Where q_i stands for i-direction coordinate, T is kinetic energy of the system, V is potential energy of system and Q is composite force acting on them. The kinetic energy of ball consists of its both rotational with respect to its centre of mass and translational energy:

$$T_b = \frac{1}{2}m_b\left(\dot{x}_b^2 + \dot{y}_b^2\right) + \frac{1}{2}I_b\left({\omega_x}^2 + {\omega_y}^2\right)$$

Where m_b is mass of the ball and I_b is moment of inertia of the ball. $\dot{x}_b$ and $\dot{y}_b$ are ball's translational velocities along x-axis and y-axis. ω_x and ω_y are ball's rotational velocities along x axis and y-axis. The following relations between translational velocities and rotational velocities:

$$\dot{x}_b = r_b \omega_y \quad , \quad \dot{y}_b = r_b \omega_x$$

In which rb denotes ball's radius. So, by substituting the above equations, we get

$$T_b = \frac{1}{2}\left[m_b\left(\dot{x}_b^2 + \dot{y}_b^2\right) + \frac{I_b}{r_b^2}\left(\dot{x}_b^2 + \dot{y}_b^2\right)\right] = \frac{1}{2}\left(m_b + \frac{I_b}{{r_b}^2}\right)\left(\dot{x}_b^2 + \dot{y}_b^2\right)$$

The kinetic energy of the plate with respect to its centre of mass, by considering ball as a point mass which is placed in (x_b, y_b), is given as

$$\begin{aligned} T_p &= \tfrac{1}{2}\left(I_p + I_b\right)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \tfrac{1}{2}m_b\left(x_b\dot{\alpha} + y_b\dot{\beta}\right)^2 \\ &= \tfrac{1}{2}\left(I_p + I_b\right)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \tfrac{1}{2}m_b\left(x_b^2\dot{\alpha}^2 + 2x_b\dot{\alpha}y_b\dot{\beta} + y_b^2\dot{\beta}^2\right) \end{aligned}$$

Where α and _ are plate's angle of inclination along x-axis and y-axis, respectively. Therefore $\dot{\alpha}$ and $\dot{\beta}$ are plate's rotational velocity. The total kinetic energy of the plate is given as

$$\begin{aligned} T &= T_b + T_p \\ &= \tfrac{1}{2}\left(m_b + \tfrac{I_b}{{r_b}^2}\right)\left(\dot{x}_b^2 + \dot{y}_b^2\right) + \tfrac{1}{2}\left(I_p + I_b\right)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) \\ &+ \tfrac{1}{2}m_b\left(x_b^2\dot{\alpha}^2 + 2x_b\dot{\alpha}y_b\dot{\beta} + y_b^2\dot{\beta}^2\right) \end{aligned}$$

The potential energy of the ball relative to horizontal plane in the centre of the inclined plate can be calculated as:

$$V_b = m_b g h = m_b g(x_b \sin\ \alpha + y_b \sin\ \beta)$$

Here we can derive the system's equation by Lagrangian and equations

$$L = T_b + T_p - V_b$$

So, by using Euler Lagrange equation, the non-linear differential equations for the ball-plate-system as followings:
We assume generalized toques as τ_x and τ_y which are exerted torques

$$\left(m_b + \frac{I_b}{r_b^2}\right)\ddot{x}_b - m_b\left(x_b\dot{\alpha}^2 + y_b\dot{\alpha}\dot{\beta}\right) + m_b g \sin\alpha = 0$$

$$\left(m_b + \frac{I_b}{r_b^2}\right)\ddot{y}_b - m_b\left(y_b\dot{\beta}^2 + x_b\dot{\alpha}\dot{\beta}\right) + m_b g \sin\beta = 0$$

$$\begin{aligned} \tau_x = &\ \left(I_p + I_b + m_b x_b^2\right)\ddot{\alpha} + 2m_b x_b\dot{x}_b\dot{\alpha} + m_b x_b y_b\ddot{\beta} \\ &+ m\dot{x}_b y_b\dot{\beta} + m_b x_b\dot{y}_b\dot{\beta} + m_b g x_b\cos\alpha \end{aligned}$$

$$\begin{aligned} \tau_y = &\ \left(I_p + I_b + m_b y_b^2\right)\ddot{}\beta + 2m_b y_b\dot{y}_b\dot{\beta} + m_b y_b x_b\ddot{\alpha} \\ &+ m_b\dot{x}_b y_b\dot{\alpha} + m_b x_b\dot{y}_b\dot{\alpha} + m_b g y_b\cos\beta \end{aligned}$$

on the plate.

The approximate value for a solid ball's moment of inertia is $I_{ball} = (2/5)m_b r_b^2$. So by substituting the value, we get

$$m_b\left[\frac{5}{7}\ddot{x}_b - \left(x_b\dot{\alpha}^2 + y_b\dot{\alpha}\dot{\beta}\right) + g\sin\alpha\right] = 0$$

$$m_b\left[\frac{5}{7}\ddot{y}_b - \left(y_b\dot{\beta}^2 + x_b\dot{\alpha}\dot{\beta}\right) + g\sin\beta\right] = 0$$

We can linearise above equation by assuming:

- Small angle of inclination for the plate(up to ±5º): α«1 and β « 1 so sinα and sinβ are taken as α and β.
- Slow rate of change for the plate: $\ddot{\alpha}$ and $\ddot{\beta}$ « 0 so $\ddot{\alpha}^2$, $\dot{\beta}$, $\dot{\alpha}\dot{\beta}$ are approximated as 0.

$$\frac{5}{7}\ddot{x}_b + g\alpha = 0$$

$$\frac{5}{7}\ddot{y}_b + g\beta = 0$$

By linearising the above equations, we find two separate differential equations for each of x and y axis. Note that we can use above linear differential equations to estimate the ball-on-plate system's states. By assuming α(s) and β(s) as inputs to ball-on-plate system, we find the transfer functions:

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

$$P_y(s) = \frac{Y_b(s)}{\beta(s)} = \frac{g}{\frac{5}{7}s^2}$$

III. Mathematics of Stewart platform

The Stewart Platform consists of 2 rigid frames connected by 6 variable length legs. The Base is considered to be the reference frame work, with orthogonal axes x, y, z. The Platform has its own orthogonal coordinates x', y', z'. The Platform has 6 degrees of freedom with respect to the Base. The origin of the Platform coordinates can be defined by 3 translational displacements with respect to the Base, one for each axis.

Three angular displacements then define the orientation of the platform with respect to the Base. A set of Euler angles are used in the following sequence:

1. Rotate an angle ψ (yaw) around the z-axis
2. Rotate an angle θ (pitch) around the y-axis
3. Rotate an angle φ(roll) around the x-axis

The full rotation matrix of the Platform relative to the Base is then given by:

$${}^{P}\mathbf{R}_{B} = \mathbf{R}_z(\psi).\,\mathbf{R}_y(\theta).\mathbf{R}_x(\varphi)$$

$$= \begin{pmatrix} \cos\psi\cos\theta & -\sin\psi\cos\varphi + \cos\psi\sin\theta\sin\varphi & \sin\psi\sin\varphi + \cos\psi\sin\theta\cos\varphi \\ \sin\psi\cos\theta & \cos\psi\cos\varphi + \sin\psi\sin\theta\sin\varphi & -\cos\psi\sin\varphi + \sin\psi\sin\theta\cos\varphi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{pmatrix}$$

Now, the diagram of the Stewart platform is given as:

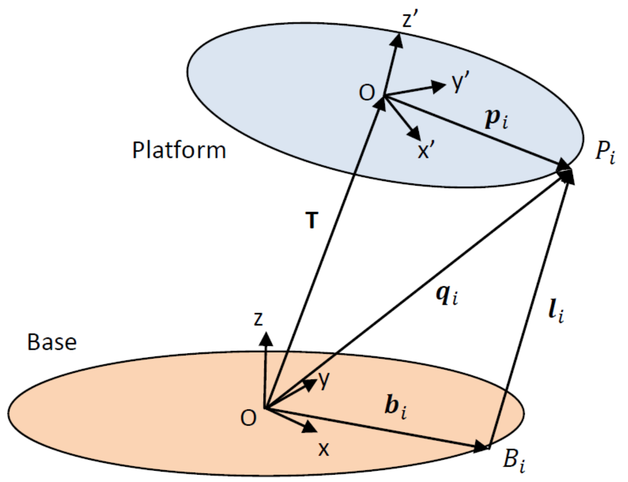


Fig 3.1 : Arm, Leg & Rod Vectors

The coordinates of q_i of the anchor point P_i with respect to the Base reference framework are given by the equation:

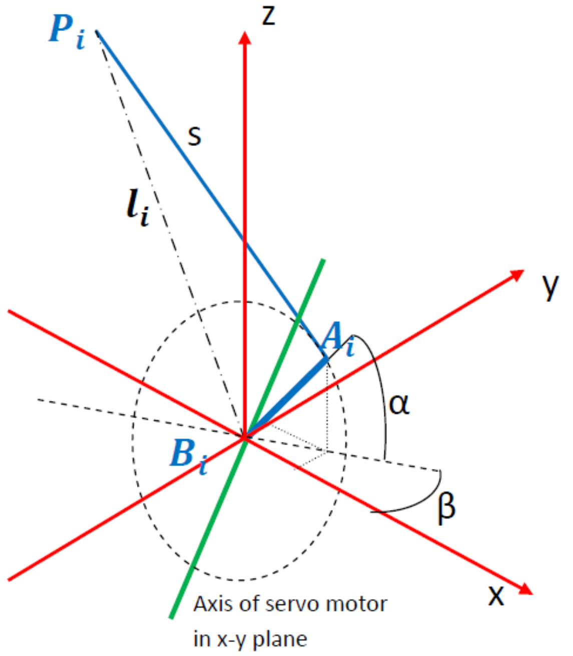
$$\boldsymbol{q_i} = \mathbf{T} + {}^{P}\mathbf{R}_{B} \cdot \boldsymbol{p_i}$$

Where **T** is the translation vector, giving the positional linear displacement of the origin of the Platform frame with respect to the Base reference framework, and p_i is the vector defining the coordinates of the anchor point P_i with respect to the Platform framework.

$$\boldsymbol{l_i} = \mathbf{T} + {}^{P}\mathbf{R}_{B} \cdot \boldsymbol{p_i} - \boldsymbol{b_i}$$

where b_i is the vector defining the coordinates of the lower anchor point B_i. These 6 equations give the lengths of the 6 legs to achieve the desired position and attitude of the platform.

If the leg lengths are achieved via rotational servos, rather than linear servos, a further calculation is required to determine the angle of rotation of the servo. Each servo / leg combination can be represented as follows:



Where: a = length of the servo operating arm

A_i are the points of the arm/leg joint on the i^{th} servo with coordinates

$\boldsymbol{a} = [x_a \quad y_a \quad z_a]^T$ in the base framework.

B_i are the points of rotation of the servo arms with the coordinates

$\boldsymbol{b} = [x_b \quad y_b \quad z_b]^T$ in the base framework.

P_i are the points the joints between the operating rods and the platform, with coordinates $\boldsymbol{p} = [x_p \quad y_p \quad z_p]^T$ in the platform framework

S = length of operating leg

$\boldsymbol{l_i}$ = length of the i^{th} leg as calculated from $\boldsymbol{l_i} = \mathbf{T} + {}^{P}\mathbf{R}_{B} \cdot \boldsymbol{p_i} - \boldsymbol{b_i}$

α = angle of servo operating arm from horizontal

β = angle of servo arm plane relative to the x-axis. Note that the shaft axis lies in the x-y plane where z = 0

Point A is considered to be on the servo arm, also we get the coordinates as:

$$\begin{aligned} x_a &= R\cos\alpha\cos\beta + x_b \\ y_a &= R\cos\alpha\sin\beta + y_b \\ z_a &= R\sin\alpha + z_b \end{aligned}$$

By using the distance formula, we get

$$a^2 = (x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2$$

$$l^2 = (x_p - x_b)^2 + (y_p - y_b)^2 + (z_p - z_b)^2$$

$$s^2 = (x_p - x_a)^2 + (y_p - y_a)^2 + (z_p - z_a)^2$$

So we get,

$$l^2-(s^2-a^2) = 2R\sin\alpha(z_p-z_b)+2R\cos\alpha[\cos\beta(x_p-x_b)+\sin\beta(y_p-y_b)]$$

Which is an equation of the form:-

$$L = M\ \sin\alpha + N\ \cos\alpha$$

Using the Trig identity for the sum of sine waves

$$a\sin x + b\cos x = c\sin(x+v)$$

$$\text{where}\quad c = \sqrt{a^2+b^2} \qquad \text{and} \qquad \tan v = \frac{b}{a}$$

We therefore have another sine function of α with a phase shift δ

$$L = \sqrt{M^2+N^2}\,\sin(\alpha+\delta) \qquad \text{where} \qquad \delta = \tan^{-1}\frac{N}{M}$$

Therefore

$$\sin(\alpha+\delta) = \frac{L}{\sqrt{M^2+N^2}}$$

And

$$\alpha = \sin^{-1}\frac{L}{\sqrt{M^2+N^2}} - \tan^{-1}\frac{N}{M}$$

$$\text{where}\quad L = l^2-(s^2-a^2)$$

$$M = 2a(z_p-z_b)$$

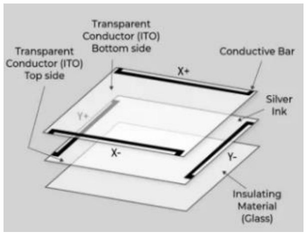
$$N = 2a[\cos\beta\,(x_p-x_b)+\sin\beta(y_p-y_b)]$$

IV. APPARATUS REQUIRED

Hardware

- Resistive Touch Plate
- Rotary Actuators : DS3218 6V 20Kg Digital Servo
- IMU (Inertial Measurement Unit)
- Arduino UNO

Resistive Touch Plate


Transparent Conductor (ITO) Bottom side
Transparent Conductor (ITO) Top side
Conductive Bar
X+
Y+
Silver Ink
X-
Y-
Insulating Material (Glass)

It is a Position-Sensitive Detector. When the Plate is touched at any point, there is a change in the Resistance of the Material of the surrounding area. The Circuit detects the change in the Resistance and identifies the Point of Contact.

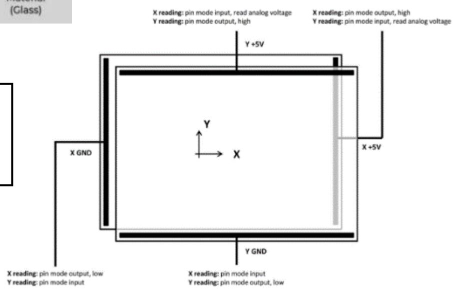

X reading: pin mode input, read analog voltage
Y reading: pin mode output, high
X reading: pin mode output, high
Y reading: pin mode input, read analog voltage
Y +5V
Y
X
X GND
X +5V
Y GND
X reading: pin mode output, low
Y reading: pin mode input
X reading: pin mode input
Y reading: pin mode output, low

Fig.4.1 : Resistive Plate Pin Voltage Representation

Rotary Actuators : DS3218 6V 20Kg Digital Servo

A rotary actuator is a type of motor that converts an electrical signal into rotational motion. The DS3218 6V 20Kg Digital Servo is a high-torque servo motor that can rotate up to 360 degrees.
Specifications of the DS3218 6V 20Kg Digital Servo:

Fig 4.2 : Servo Motor

- Operating voltage: 4.8V - 6.8V
- Stall torque: 20 kg-cm (6V)
- Operating speed: 0.16 sec/60° (6V)
- Rotation angle: 180 degrees
- Control interface: PWM (Pulse Width Modulation)
- Gear material: Metal

It consists of a DC motor, position sensing device, gear assembly and control circuit.
The position sensor gets the positional feedback. The control circuit sends analog or digital signal which tells the final command position for shaft.
Servo motors have three wires:
(Wire - color - pin connection on arduino board)
Power - Red - 5V pin
Ground - Black or Brown - ground pin
Signal - Yellow - PWM (~) pin

IMU (Inertial Measurement Unit)

IMU measures and reports the plate's orientation using a combination of accelerometers, gyroscopes and magnetometers.

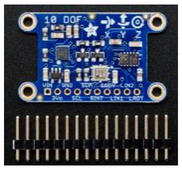


Fig 4.3 : IMU

- Accelerometer measures the acceleration on 3 axis also the angle to the earth surface can be calculated
- Gyroscope measures rotational speed on 3 axis
- Magnetometer measures the magnetic north pole of the earth
- Barometer measures atmospheric pressure and temperature.

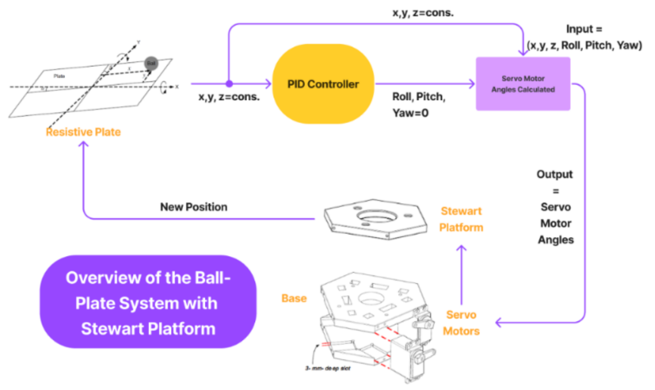
The data from these sensors is combined using algorithms to calculate the object's motion and orientation. Here IMU is used to verify the Roll (Rotation along x-axis), Pitch (Rotation along y-axis) and Yaw (Rotation along z-axis) which is zero here, of the plate after each iteration.

Arduino UNO

Fig 4.4 : Arduino UNO

- Micro-Controller Board comprising 14 Digital Input/Output Pins, 6 Analog Input Pins, a 16 MHz Quartz Crystal, a USB Connection, a Power Jack, and a Reset button.
- The Sensed Parameters are sent to Arduino and they are used in the Code for Evaluating Servo Angles.

V. PROCEDURE



STEP 1 - Resistive Plate Senses the (x,y) Coordinates, z = cons = height
STEP 2 -(x,y,z) are Fed into the PID Controller to give (Roll, Pitch, Yaw)
STEP 3 -Now all the (x,y,z, Roll, Pitch, Yaw) Values are Fed into the Servo Motor Angle Calculator to give Servo Motor Angles of all 6 Servo Motors
STEP 4 -These 6 Servo Motor Angles are then Actuated to give both the Stewart Platform & subsequently Resistive Plate
STEP 5 – The Ball traverses to a New Position
GO TO STEP – 1 Again

VI. SOFTWARE & CODES

We're primarily utilizing 2 Coding Platforms for this Project:-

1. Arduino IDE
2. MATLAB & Simulink

A. Plate Coordinates (x,y) from Resistive Touch Plate

This Arduino Code outputs Plate Coordinates (x, y) [z=constant, the height of Stewart Platform from Base Plate]. We're utilizing the "Touchscreen.h" In-Built Library from Adafruit, we obtain the Coordinates Linear Voltage Variation Technique.
Arduino Code-

```
// Touch screen library with X Y and Z (pressure)
readings as well as oversampling to avoid 'bouncing'
// This demo code returns raw readings, public domain
#include <stdint.h>
#include "TouchScreen.h"

/*#define YP A2  // must be an analog pin, use "An"
notation!
#define XM A5  // must be an analog pin, use "An"
notation!
#define YM A4  // can be a digital pin
#define XP A3// can be a digital pin*/
```

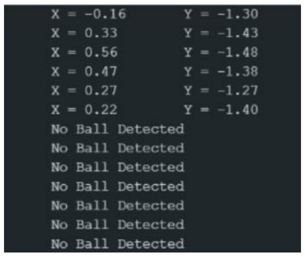
```
#define YP A2  // must be an analog pin, use "An"
notation!
#define XP A3  // must be an analog pin, use "An"
notation!
#define YM A4  // can be a digital pin
#define XM A5  // can be a digital pin*/

// For better pressure precision, we need to know the
resistance between X+ and X- Use any multimeter to read
it
// For the one we're using, its 300 ohms across the X
plate

TouchScreen ts = TouchScreen(XP, YP, XM, YM,1);

void setup() {
  Serial.begin(9600);
}

void loop() {
  // a point object holds x y and z coordinates
  TSPoint p = ts.getPoint();

  // we have some minimum pressure we consider 'valid'
  // pressure of 0 means no pressing!
  //if (p.z > ts.pressureThreshhold) {
     /*Serial.print("X = "); Serial.print(p.x);
     Serial.print("\tY = "); Serial.print(p.y);
     Serial.print("\tXCAL = ");
Serial.print(p.x/(1000/26));
     Serial.print("\tYCAL = ");
Serial.print(p.y/(900/20));
     Serial.print("\tPressure = ");
Serial.println(p.z);*/

    double XX = 11-(p.y/(900.000/20.000));
    double YY = 13-(p.x/(1000.000/26.000));

    if ((XX<-4.0) && (XX>-6.5) && (YY>12.5) && (YY<13.5))
{
      Serial.print("\tNo Ball Detected");
      Serial.print('\n');
    }
    else {
     Serial.print("\tX = "); Serial.print(XX);
     Serial.print("\tY = "); Serial.print(YY);
     Serial.print('\n');
    }
  //}

  delay(100);
}
```

For X-Coordinate Calculation-
0V is applied to X GND & +5V to X+ Pin. This creates a Linear Voltage Gradient from x=0 to x=26cm here. The Voltage at any Contact Point is sensed through either of the Idle Y+ or Y- Pins, say 4V. Then the x coordinate is calculated as -

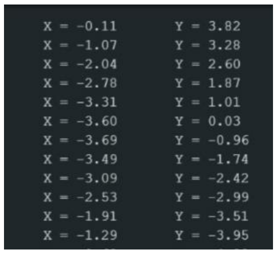
$$\mathbf{x = 0 + (4/5)*26cm}$$

The Same Process is repeated for **Y-Coordinate Calculation**.

Output Snippets-



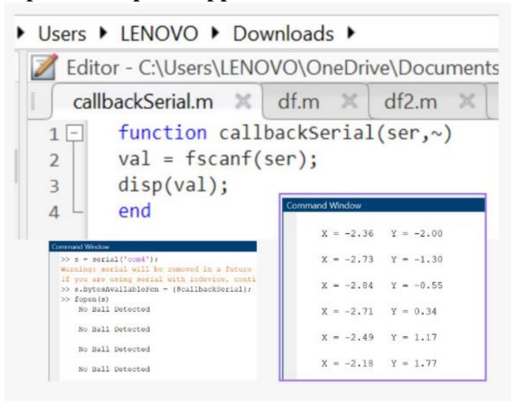
```
X = -0.16          Y = -1.30
X = 0.33           Y = -1.43
X = 0.56           Y = -1.48
X = 0.47           Y = -1.38
X = 0.27           Y = -1.27
X = 0.22           Y = -1.40
No Ball Detected
No Ball Detected
No Ball Detected
No Ball Detected
No Ball Detected
No Ball Detected
No Ball Detected
```



```
X = -0.11          Y = 3.82
X = -1.07          Y = 3.28
X = -2.04          Y = 2.60
X = -2.78          Y = 1.87
X = -3.31          Y = 1.01
X = -3.60          Y = 0.03
X = -3.69          Y = -0.96
X = -3.49          Y = -1.74
X = -3.09          Y = -2.42
X = -2.53          Y = -2.99
X = -1.91          Y = -3.51
X = -1.29          Y = -3.95
```

B. Arduino-MATLAB Serial Communication

This MATLAB Code basically feeds all the Arduino Output Data into MATLAB & subsequently Simulink for Modelling & Processing.
Input & Output Snippets-



Users ▸ LENOVO ▸ Downloads ▸
Editor - C:\Users\LENOVO\OneDrive\Documents
callbackSerial.m
df.m
df2.m
Command Window
Command Window

```
function callbackSerial(ser,~)
val = fscanf(ser);
disp(val);
end
```

```
>> s = serial('com4');
Warning: serial will be removed in a future
if you are using serial with icdevice, conti
>> s.BytesAvailableFcn = (@callbackSerial);
>> fopen(s)
   No Ball Detected

   No Ball Detected

   No Ball Detected

   No Ball Detected
```

```
X = -2.36   Y = -2.00

X = -2.73   Y = -1.30

X = -2.84   Y = -0.55

X = -2.71   Y = 0.34

X = -2.49   Y = 1.17

X = -2.18   Y = 1.77
```

C. PID Controller Design

PID controller is implemented for the tracking problem by considering the system transfer function as

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

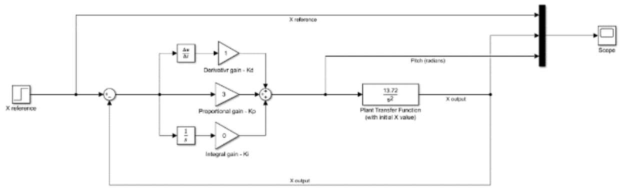


Fig 6.1 : PID Controller Implementation

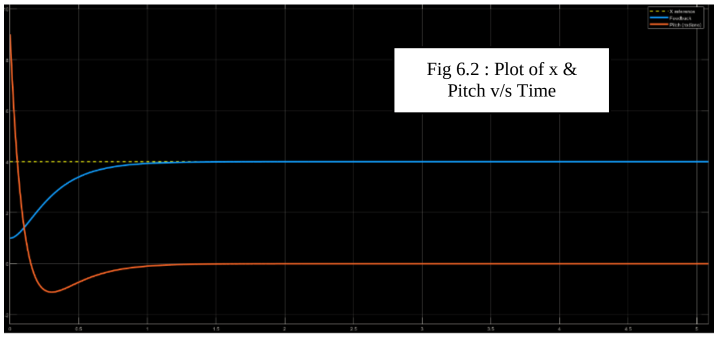


Fig 6.2 : Plot of x & Pitch v/s Time

D. Calculating Servo Motor Angles from (x,y,z,Roll,Pitch,Yaw) Input

This Arduino Code takes in the (x,y,z=cons) readings from Resistive Plate, Calculated (Roll, Pitch, Yaw=0) values from PID Controller, overall taking in a 6-Bit Array of (x, y, z, Roll, Pitch, Yaw) to **Output** another 6-Bit Array of **Servo Motor Angles.**
Arduino Code-

```
#include <math.h>
#include <Servo.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_L3GD20_U.h>
#include <Adafruit_10DOF.h>

/* Assign a unique ID to the sensors */
Adafruit_10DOF                dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accel =
Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   =
Adafruit_LSM303_Mag_Unified(30302);

Servo myservo[6];

#define pi 3.14159
#define rad_to_deg 180/pi
#define deg_to_rad pi/180
#define z_home  0.238

 float Trans_matrix[3];    //3*1 matrix
 float Rot_matrix[3][3];   //3*3 matrix
 float q[6][3];
 float l[6][3];
 float leg_length[6];
 float servo_angle[6];
```

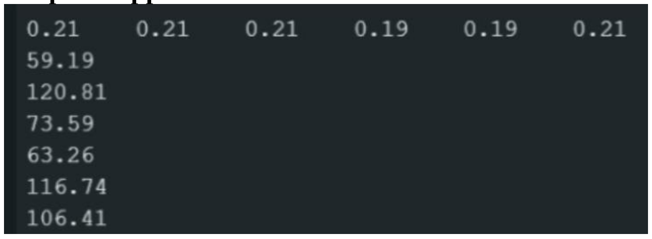
```
// ----------------------declaration of base points------
-----------------------------------//
 float  r_base = 0.0740;

 float base_point[6][3] = {
    {   -r_base * cos(28*deg_to_rad) ,   -r_base *
sin(28*deg_to_rad) , 0.03},
    {   -r_base * cos(28*deg_to_rad) ,    r_base *
sin(28*deg_to_rad) , 0.03},
    {    r_base * cos(88*deg_to_rad) ,    r_base *
sin(88*deg_to_rad) , 0.03},
    {    r_base * cos(32*deg_to_rad) ,    r_base *
sin(32*deg_to_rad) , 0.03},
    {    r_base * cos(32*deg_to_rad) ,   -r_base *
sin(32*deg_to_rad) , 0.03},
    {    r_base * cos(88*deg_to_rad) ,   -r_base *
sin(88*deg_to_rad) , 0.03}
};
//--------------------end of declaration of base points--
------------------------------------//

//----------------------declaration of platform points--
--------------------------------------//
 float r_top = 0.0790;

float top_point[6][3] = {
    {   -r_top * cos(pi/4)          ,   -r_top *
sin(pi/4)           ,   -0.005},
    {   -r_top * cos(pi/4)          ,    r_top *
sin(pi/4)           ,   -0.005},
    {   -r_top * sin(15*deg_to_rad) ,    r_top *
cos(15*deg_to_rad) ,   -0.005},
    {    r_top * cos(15*deg_to_rad) ,    r_top *
sin(15*deg_to_rad) ,   -0.005},
    {    r_top * cos(15*deg_to_rad) ,   -r_top *
sin(15*deg_to_rad) ,   -0.005},
    {   -r_top * sin(15*deg_to_rad) ,   -r_top *
cos(15*deg_to_rad) ,   -0.005}
};
//----------------------end of declaration of platform
points ----------------------------//

int zero_pos[6]={90,90,90,90,90,90};    // try to put
this in degrees


//initial position of platform centre from base centre
```

```
    float trans_init[3] =
{0    ,    0    ,  z_home  };       //translation info
x_home ,y_home ,z_home

    float rot_init[3]   =
{0    ,    0    ,    0      };         //rotation info roll,
pitch, yaw


 float beta[6]=  { pi/2 , -pi/2 , -pi/6 , 5*pi/6 , -
5*pi/6, pi/6 };               // orientation of servo wrt
x axis
 float Rm = 0.024;                // length of servo arm
 float  D = 0.205;                 // length of connecting
rod

  float req_pos[6] ={0   ,  0   ,  0   ,  0,  0,  0
};                        // x y z roll pitch yaw

void getTrans_matrix() {

   Trans_matrix[0] = req_pos[0]+trans_init[0];
   Trans_matrix[1] = req_pos[1]+trans_init[1];
   Trans_matrix[2] = req_pos[2]+trans_init[2];
}



void getRot_matrix() {

  float phi   = deg_to_rad * req_pos[3];    // roll wrt x
axis
  float theta = deg_to_rad * req_pos[4];    // pitch wrt
y axis
  float psi   = deg_to_rad * req_pos[5];    // yaw wrt z
axis

   Rot_matrix[0][0] = cos(psi)*cos(theta);
   Rot_matrix[0][1] = -
sin(psi)*cos(phi)+cos(psi)*sin(theta)*sin(phi);
   Rot_matrix[0][2] =
sin(psi)*sin(phi)+cos(psi)*cos(phi)*sin(theta);

   Rot_matrix[1][0] = sin(psi)*cos(theta);
   Rot_matrix[1][1] =
cos(psi)*cos(phi)+sin(psi)*sin(theta)*sin(phi);
   Rot_matrix[1][2] = -
cos(psi)*sin(phi)+sin(psi)*sin(theta)*cos(phi);
```

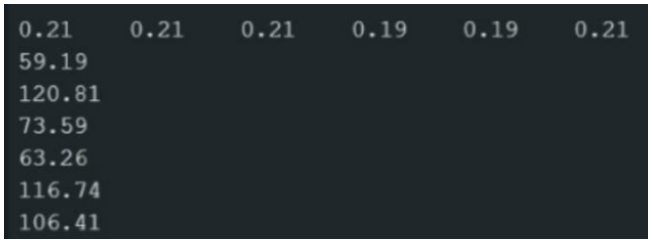
```
   Rot_matrix[2][0] = -sin(theta);
   Rot_matrix[2][1] = cos(theta)*sin(phi);
   Rot_matrix[2][2] = cos(theta)*cos(phi);
}

// qi is a vector representing the top frame points in
the base frame coordinate system
void getQ() {

  for(int i=0;i<6;i++){
  q[i][0]=Trans_matrix[0] +
Rot_matrix[0][0]*top_point[i][0] +
Rot_matrix[0][1]*top_point[i][1] +
Rot_matrix[0][2]*top_point[i][2];
  q[i][1]=Trans_matrix[1] +
Rot_matrix[1][0]*top_point[i][0] +
Rot_matrix[1][1]*top_point[i][1] +
Rot_matrix[1][2]*top_point[i][2];
  q[i][2]=Trans_matrix[2] +
Rot_matrix[2][0]*top_point[i][0] +
Rot_matrix[2][1]*top_point[i][1] +
Rot_matrix[2][2]*top_point[i][2];
 }
}

//************  leg length calculation ****************//
// the leg length is the length between base point and
the corresponding top points
void getLeg_length(){

    for(int i=0;i<6;i++){
      l[i][0]=q[i][0]-base_point[i][0];
      l[i][1]=q[i][1]-base_point[i][1];
      l[i][2]=q[i][2]-base_point[i][2];
    }

    for(int i=0;i<6;i++)
    leg_length[i] = sqrt (l[i][0]*l[i][0] +
l[i][1]*l[i][1] + l[i][2]*l[i][2]);

}
//**************************************//


//************* servo angle calculation
*****************//
void getservo_angle(){
```

```
 double a[6];
 double b[6];
 double c[6];
 double deno[6];

 for(int i= 0; i<6; i++)
 a[i] = 2*Rm*(q[i][2] - base_point[i][2]);

for(int i= 0; i<6; i++)
 b[i] =2*Rm*((q[i][0] - base_point[i][0])*cos(beta[i]) +
(q[i][1] - base_point[i][1])*sin(beta[i]));

for(int i= 0; i<6; i++)
 c[i] = leg_length[i]*leg_length[i] - (D*D) + Rm*Rm;

for(int i=0;i<6;i++)
  deno[i]=sqrt( a[i]*a[i] + b[i]*b[i]);

for(int i=0;i<6;i++)
  servo_angle[i] = rad_to_deg * (asin(c[i]/deno[i]) -
atan(b[i]/a[i]));


for(int i=0;i<6; i++){
  if(i==0 || i==2 || i==4)
   myservo[i].write((zero_pos[i]-servo_angle[i]));
  else
   myservo[i].write((zero_pos[i]+servo_angle[i]));
}
}
void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check
your connections */
    Serial.println(F("Ooops, no LSM303 detected ... Check
your wiring!"));
    while(1);
  }

  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check
your connections */
    Serial.println("Ooops, no LSM303 detected ... Check
your wiring!");
    while(1);
  }
}
//*************************************//

void setup() {
Serial.begin(115200);

myservo[0].attach(3);
myservo[1].attach(5);
myservo[2].attach(6);
myservo[3].attach(9);
myservo[4].attach(10);
myservo[5].attach(11);

for(int i=0 ; i<6 ;i++)
  myservo[i].write(90);

Serial.println(F("Adafruit 10 DOF Pitch/Roll/Heading
Example"));
Serial.println("");
/* Initialise the sensors */
initSensors();

delay(1000);
}

void loop() {

getTrans_matrix();
getRot_matrix();
getQ();
getLeg_length();
/*
for(int i=0;i<6;i++){
   Serial.print(leg_length[i]);
   Serial.print('\t');
}
Serial.print('\n');
*/
getservo_angle();
/*
 for(int i=0;i<6;i++){
   if(i==0 || i==2  || i==4){
   Serial.print(90-servo_angle[i]);
   Serial.print('\n');
   }
   else{
   Serial.print(90+servo_angle[i]);
   Serial.print('\n');
   }
}
*/
Serial.print('\n');

for(int i=0;i<6;i++){
   if(i==0 || i==2  || i==4)
   myservo[i].write(90-servo_angle[i]);
   else
  myservo[i].write(90+servo_angle[i]);
}
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_event_t bmp_event;
sensors_vec_t   orientation;

/* Calculate pitch and roll from the raw accelerometer
data */
accel.getEvent(&accel_event);
if (dof.accelGetOrientation(&accel_event, &orientation))
{
    /* 'orientation' should have valid .roll and .pitch
fields */
    Serial.print(F("Roll: "));
    Serial.print(orientation.roll-2);
    Serial.print(F("; "));
    Serial.print(F("Pitch: "));
    Serial.print(-(orientation.pitch+5));
    Serial.print(F("; "));
}
delay(100);
}
```

Output Snippets-



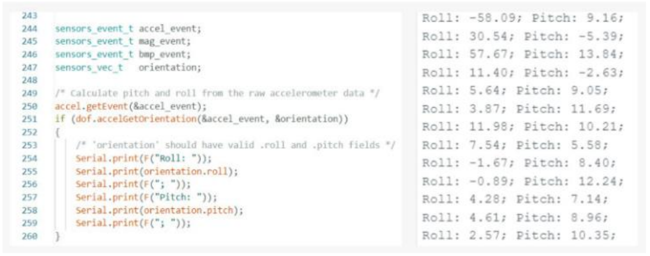
```
0.21     0.21     0.21     0.19     0.19     0.21
59.19
120.81
73.59
63.26
116.74
106.41
```



```
0.21     0.21     0.21     0.19     0.19     0.21
59.19
120.81
73.59
63.26
116.74
106.41
```

1st Line 6 Values Represent **Leg-Lengths**
2nd-7th Line 6 Values Represent **Servo Motor Angles**

E. IMU Readings (Roll, Pitch) Cross-Check



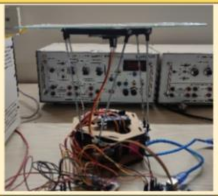
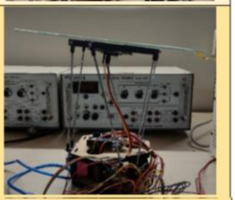
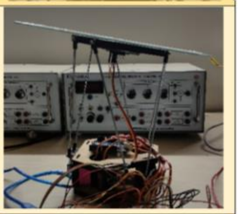
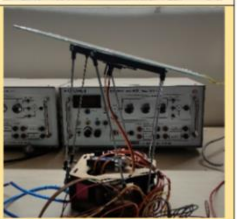
```

sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_event_t bmp_event;
sensors_vec_t   orientation;

/* Calculate pitch and roll from the raw accelerometer data */
accel.getEvent(&accel_event);
if (dof.accelGetOrientation(&accel_event, &orientation))
{
  /* 'orientation' should have valid .roll and .pitch fields */
  Serial.print(F("Roll: "));
  Serial.print(orientation.roll);
  Serial.print(F("; "));
  Serial.print(F("Pitch: "));
  Serial.print(orientation.pitch);
  Serial.print(F("; "));
}
```

```
Roll: -58.09; Pitch: 9.16;
Roll: 30.54; Pitch: -5.39;
Roll: 57.67; Pitch: 13.84;
Roll: 11.40; Pitch: -2.63;
Roll: 5.64; Pitch: 9.05;
Roll: 3.87; Pitch: 11.69;
Roll: 11.98; Pitch: 10.21;
Roll: 7.54; Pitch: 5.58;
Roll: -1.67; Pitch: 8.40;
Roll: -0.89; Pitch: 12.24;
Roll: 4.28; Pitch: 7.14;
Roll: 4.61; Pitch: 8.96;
Roll: 2.57; Pitch: 10.35;
```

This Code Snippet is Added to the Above Code of D. Calculating Servo Motor Angles from (x,y,z,Roll,Pitch,Yaw) Input just to Check the Final Roll, Pitch Values of the Stewart Platform.

VII. RESULTS

- We've successfully obtained (x,y) Coordinates upto an Error of 1cm
- We've successfully fed (x,y,z) into our PID Controller & obtained very Accurate Values of (Roll, Pitch, Yaw)
- We've successfully fed (x,y,z, Roll, Pitch, Yaw) Values into the Servo Motor Angle Calculator to give very Accurate Servo Motor Angles of all 6 Servo Motors upto an Error of 1.5 Degrees
- We've achieved Smooth Actuation of Servo Motors to bring the Stewart Platform to our Desired Position

Theoretical (Roll, Pitch)	Actual (Roll, Pitch)	Pictorial Depiction
(0, 0)	(0.27, -0.69)	
(0, 5)	(0.26, 5.22)	
(0, 10)	(-0.69, 10.71)	
(0, 15)	(0.27, 15.67)	

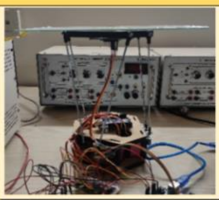
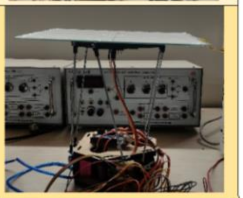
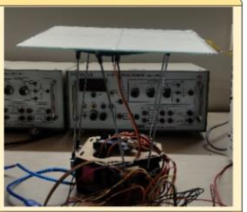
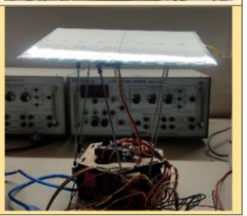
Theoretical (Roll, Pitch)	Actual (Roll, Pitch)	Pictorial Depiction
(0, 0)	(0.27, 0.69)	
(5, 0)	(6.73, 0.22)	
(10, 0)	(11.72, -0.51)	
(15, 0)	(16.83, -0.54)	

Table 7.1 : Theoretical v/s Actual (Roll, Pitch)

VIII. FUTURE PROSPECTS

- Position and trajectory control of ball will be achieved by combining the Arduino code and values with MATLAB environment.
- Testing of Stewart platform on ground vehicles moving in an inclined plane.

IX. REFRENCES

Szufnarowski, Filip. "Stewart platform with fixed rotary actuators: a low cost design study." *Advances in medical Robotics* 4 (2013).

Bang, Heeseung, and Young Sam Lee. "Implementation of a ball and plate control system using sliding mode control." *IEEE access* 6 (2018): 32401-32408.