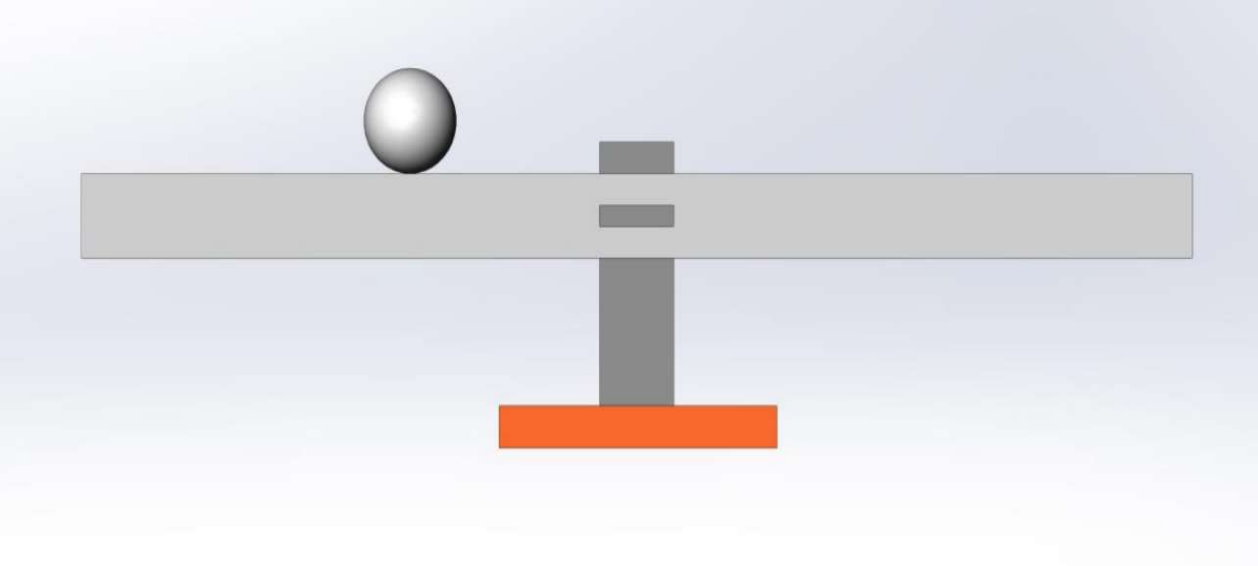
ME2400- MEASUREMENTS, INSTRUMENTATION AND CONTROL COURSE PROJECT REPORT

Balance a ball along a linear grooved track by incorporating closed loop P/PI/PID control



GROUP B110

ME17B075 YERRA JAI GANESH
ME17B076 ARNESH KUMAR BOSE
ME17B079 NANDAGOPAN G
ME17B104 BHUKYA JAYCHANDRA

Contents

1. ABSTRACT

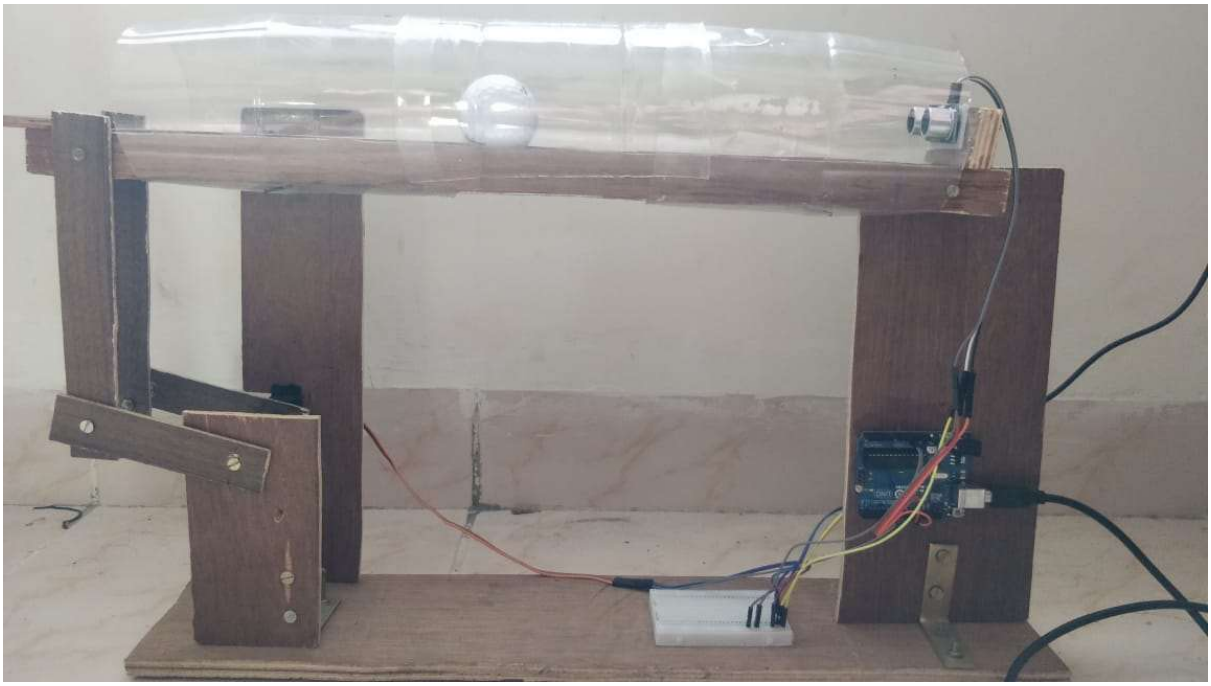


Figure 1. Final working model

This report aims at giving a brief summary of how we worked on project (Figure 1). The group met a few times, initially to discuss what basic logic to use, decided on a conceptual design for our mechanism, and later decided on what components to buy based on a tentative budget estimation; the lengths and heights were chosen following an approximate force analysis. Following this, the electrical components were purchased and their working was tested. After this, plywood was procured for making the frame and the links. The links and frame were fabricated in the Tech room of our hostel. We assembled the links and found that the mechanism was a bit unstable and showing undesirable movements. So, deviating from the original design, a redundant mechanism was constructed and introduced for support. Convinced that it was more stable than before, we started working on the Arduino code, and installing the components. The mechanism was tested with a golf ball and there appeared to be some faults with the proximity sensor. This was rectified using a shielding method which is discussed later. Further, the tuning was done and The Simulink model was made.

2. FLOWCHART OF THE PROJECT

The flowchart of the entire process is shown in Figure 2

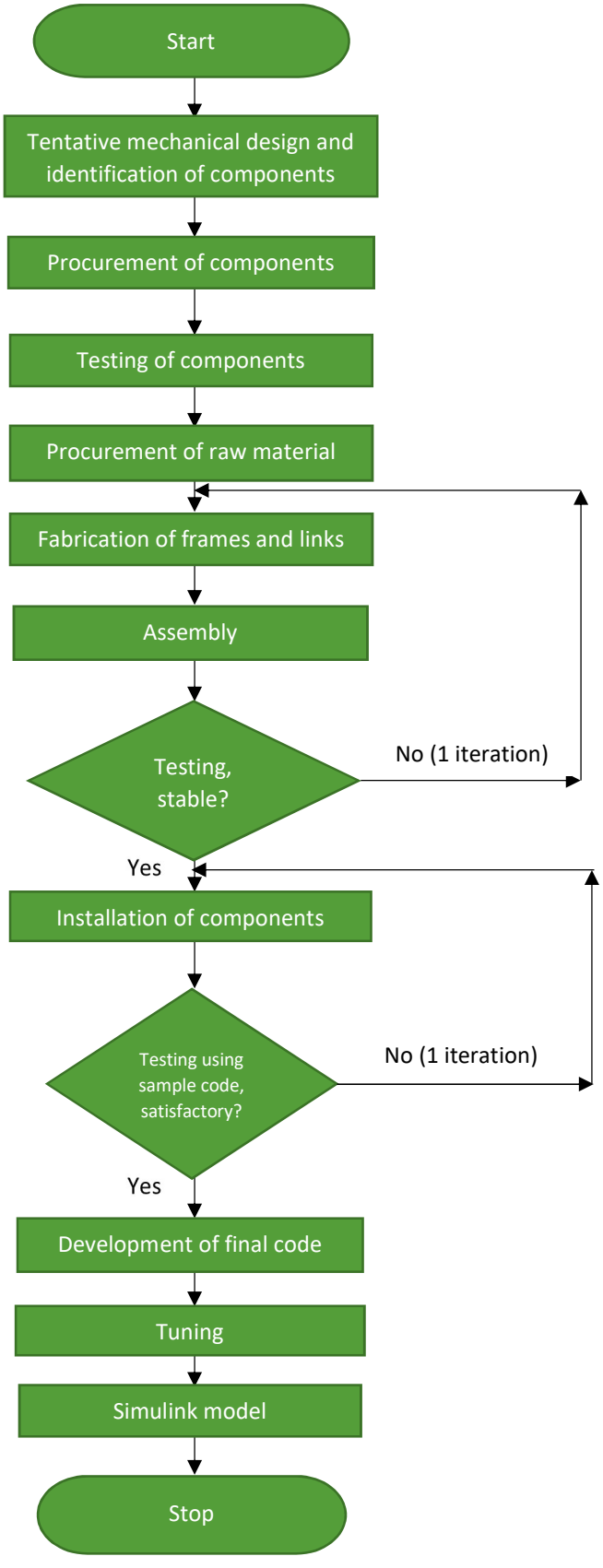


Figure 2. Flowchart of the project

3. CAD MODEL DESIGN

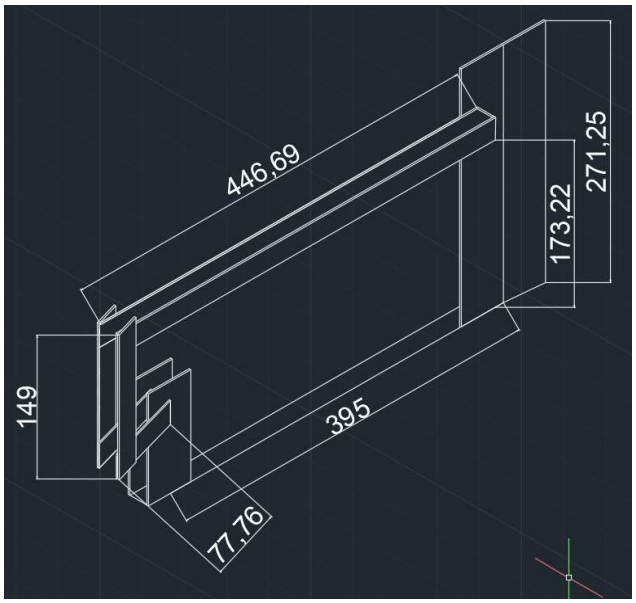


Figure 3. CAD drawing (all dimensions in mm)

Thin boards of wood were to be bought, large enough that all the links of the given dimensions (Figure 3) could be cut out of them. Thicker wood pieces were to be procured for the base and the frame.

We chose a design in which the track was pivoted at one end as this design gives us a particular type of advantage. Here, the motor does not have to bear the entire weight of the track (unlike in the centre-pivot case). Instead, part of the weight is borne by the end pivot, and the other end is moved by a crank and a coupler link. This reduces the magnitude of inertia forces that the motor has to overcome to rotate, compared to the centre pivot case. So, the net torque requirement of the motor is reduced and the life of the motor is increased as the load it has to bear is reduced. Also, fixing the sensor to the track at its fixed end, does away with the excessive movement of connection wires.

The track is constituted by two 50cm long thin wooden strips of 2.5cm width, with a fixed distance between them, and the ball would roll on top of the two strips. The gap between these two pieces was chosen as approximately 3cm after we decided we would use the golf ball of diameter 4.3cm. The golf ball was chosen as it weighed around 50g and was large enough to be detected by the sensor. The mechanism is a redundant pair of 4-bar linkages (the extra links were added for support).

4. ELECTRONIC CIRCUIT DESIGN

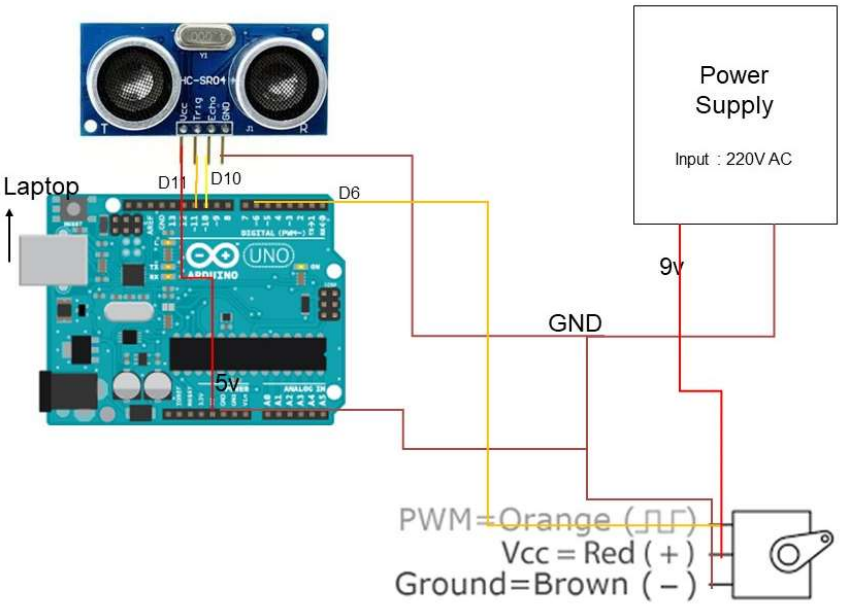


Figure 4. Circuit design

The circuit design is shown in Figure 4. We chose to use an 9V dc adapter instead of a battery, because then we wouldn't have to worry about recharging the battery. Jumper wires were chosen because they fit firmly and we wouldn't have to do any soldering.

The electrical components used are listed in Table 1

Table 1. List of electrical components

S No.	Component name
1	Arduino UNO
2	HC-SR04 ultrasonic sensor
3	MG995 Servomotor (9 kg/cm)
4	9V=1A Adapter
5	Male-female jumper wires
6	Male-male jumper wires
7	Zener Diode (6V)
8	Bread board

4.1. CHOICE OF SENSOR

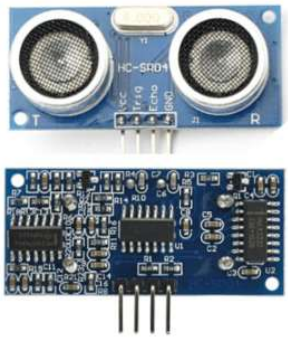


Figure 5. HC-SR04 ultrasonic sensor

We chose HC-SR04 ultrasonic (Figure 5) sensor as our proximity sensor, because it is appropriate for ranging of lengths of the order of a metre. In particular, the stated range of the sensor is 2cm-400cm, which is apt for our chosen track length of 45cm.

4.1.1. SPECIFICATIONS

- Power Supply: +5V DC
- Quiescent Current: <2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance: 2cm – 400 cm/1" – 13ft
- Resolution: 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10µS
- Dimension: 45mm x 20mm x 15mm

4.1.2. PINS

- VCC: +5VDC
- Trig: Trigger (INPUT)
- Echo: Echo (OUTPUT)
- GND: GND

4.2. CORRECTION ELEMENT

Figure 6 MG995 Servo motor

We chose a 4- bar mechanism powered by a MG995 Servo motor (Figure 6) as our correction element. MG995 is rated at a torque corresponding to 9kg-cm. This torque is sufficient to balance the weight of the wooden links and the ball at the extreme end. Also, the motor provides enough excess torque to take care of any frictional or damping forces that come into play. [Approximate calculation: A 50g ball at 45cm in addition to approximately 150g (weight of the track) at 22.5cm produces a torque of 5.625kg-cm, which is adequately balanced by a counter torque of 9kg-cm, including friction]

4.2.1. SPECIFICATIONS

- Weight: 55 gm
- Operating voltage: 4.8V-7.2V
- Servo Plug: JR
- Stall torque @4.8V: 10 kg-cm (stated)
- Stall torque @6.6V: 12 kg-cm (stated)

5. FABRICATION AND ASSEMBLY

5.1. MANUFACTURING PROCESS FLOW

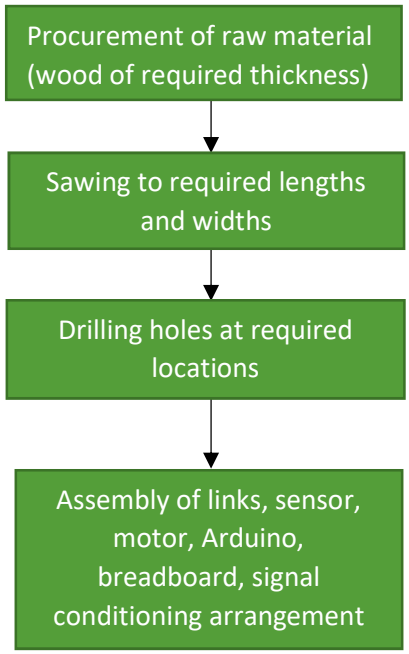


Figure 7. Manufacturing process flow

Wooden boards of required thickness were procured. These were reduced to the required lengths and widths using the jig-saw in the hostel tech-room. Holes were drilled at the joint locations on the links using the drilling machine in the tech-room.

5.2. ASSEMBLY PROCESS FLOW

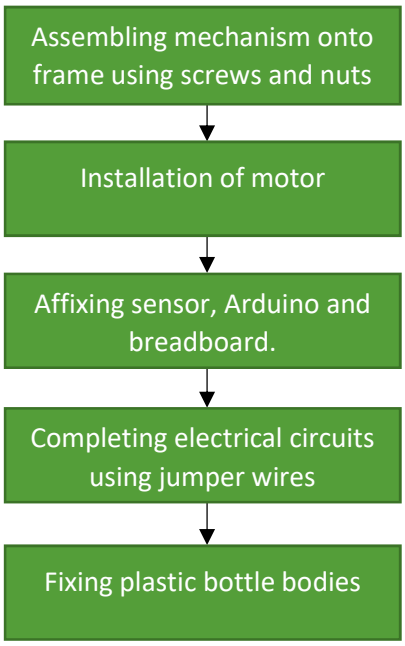


Figure 8. Assembly process flow

Screws, nuts and washers were bought and the mechanism was assembled onto a frame (made from thicker wood which was available in the tech room), with the motor installed (screwed in) at one of the two fixed joints of the 4-bar mechanism and fixed to the smallest link (crank). The Arduino was fixed onto the frame using screws. The bread-board was stuck onto the frame and the sensor onto the grooved track using double-sided tape. Due to erroneous outputs of the sensor, a signal conditioning arrangement was introduced. The bodies of three 2L plastic bottles were cut out and placed along the length such that they covered the track, so all the transmitted waves retuned to the sensor and were received. This resulted in the sensor giving accurate values. The final fully assembled model can be seen in Figure 19.

6. ARDUINO

6.1. CONNECTION DIAGRAM AND SIGNAL CONDITIONING

The complete connection diagram for the project is shown in Figure 4. No external electrical signal conditioning is required as the sensor is powered by the Arduino so it is already conditioned in the Arduino and the servo motor runs on 9V DC supply which also does not require any conditioning. The Arduino takes care of the signal processing.

For getting accurate readings from the ping sensor from the ball at large distances, we have used plastic bottle to cover the track so that the ultrasonic signal after striking the spherical surface of the ball do not go outside the range of the receiver.

6.2. CONTROLLER DESIGN, TUNING AND PARAMETER VALUES

We have used a PD controller for balancing the ball at a point. Although the algorithm allows for PID control and the Simulink model also uses PID but in reality we did not require the ki tuning so we have set the value for Ki to be 0.

For tuning,

1. we first kept Kd=0, Ki=0,
2. changed Kp such that at a particular Kp, we got a steady oscillation.
3. Then changed Kd to die out the oscillations.
4. Repeated steps 2 and 3 till the time when changing Kd was not required to reduce the oscillation (repetitions as in following steps)
5. Setting Kp=2, a response as seen in Figure 9 was obtained

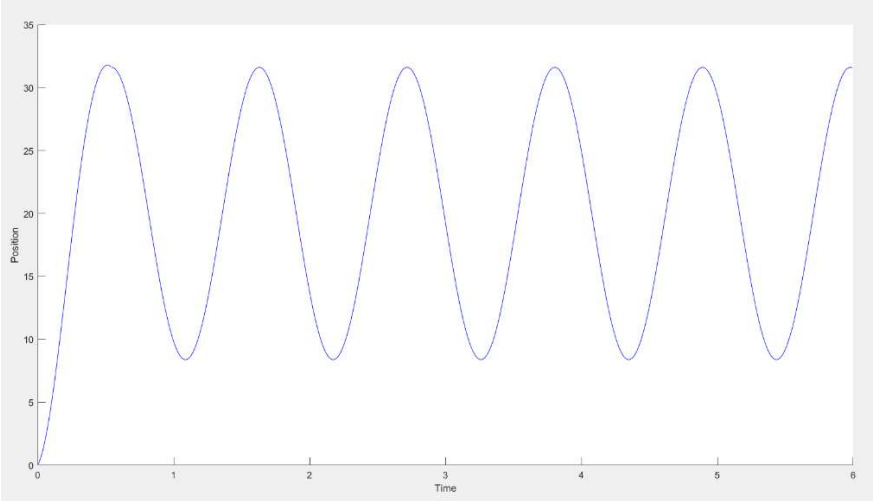


Figure 9. Kp=2

6. Using Kp=2, Kd=0.6 (Figure 10)

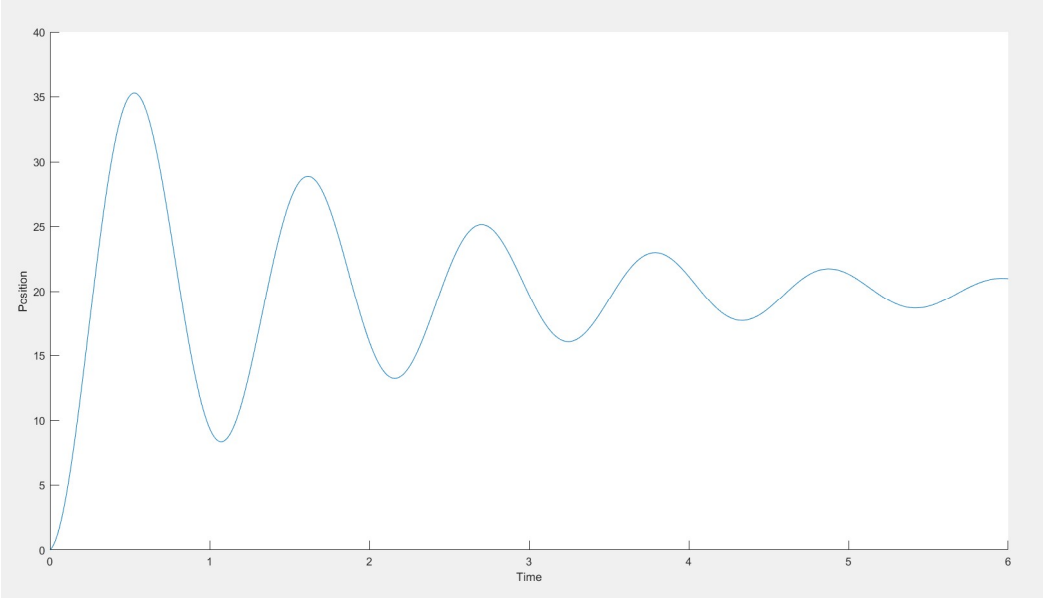


Figure 10. Kp=2, Kd=0.6

7. Using Kp=4, Kd=0.6 (Figure 11)

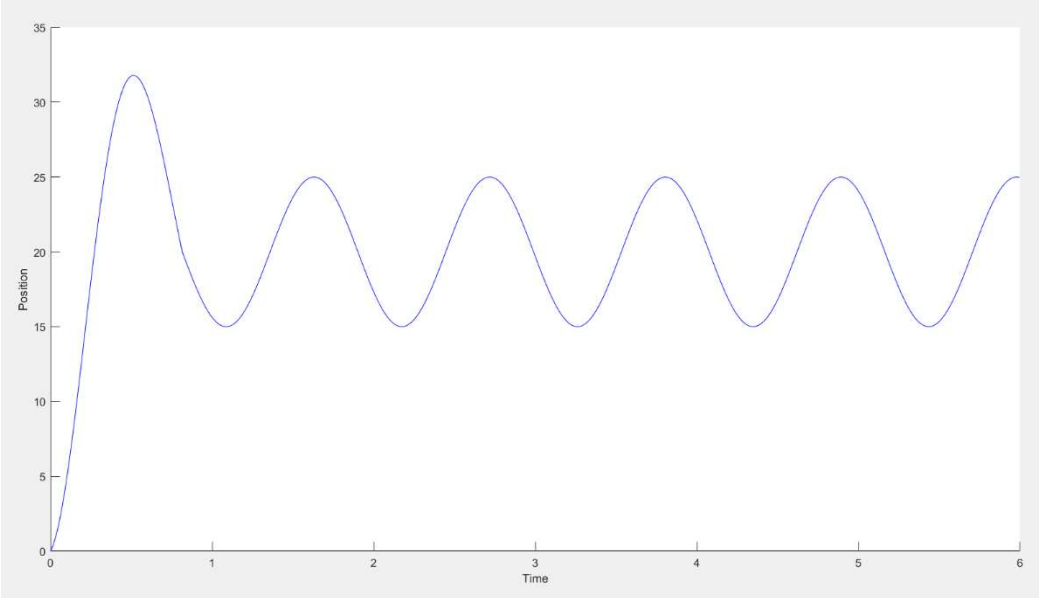


Figure 11. Kp=4, Kd=0.6

8. Then finally, using Kp=4, Kd=1.2, a desirable response was obtained (Figure 12)

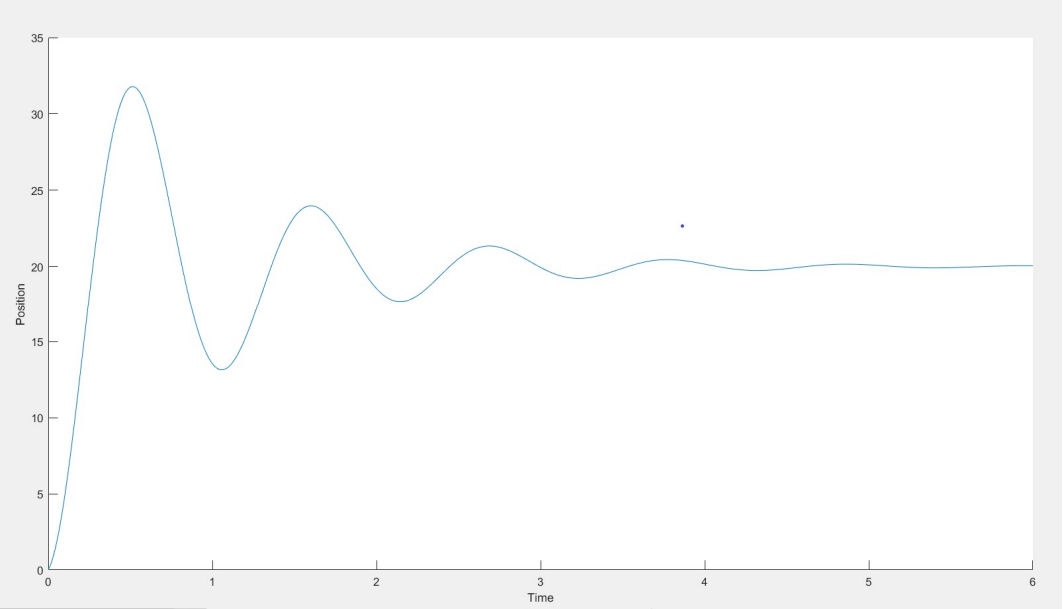


Figure 12. Kp=4, Kd=1.2

Final values of the constants are

- Kp=4
- Kd=1.2
- Ki=0

6.3. ARDUINO MODEL NUMBER AND SPECIFICATIONS

The model of the Arduino is Arduino UNO. The specifications are

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limit): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- PWM Digital I/O Pins: 6
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA
- DC current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- Clock Speed: 16 MHz

6.4. ARDUINO PROGRAM AND FLOWCHART

6.4.1. PROGRAM

Given below is the Arduino program used

```
#include<Servo.h> // servo library
#include<PID_v1.h> // PID library
const int servoPin = 6;                          //servo pin
const int trigPin = 10;                          //trig pin
const int echoPin = 11;                          //echo pin
float Kp = 4;                                    //Initial Proportional Gain
float Ki = 0;                                    //Initial Integral Gain
float Kd = 1.2;                                  //Intitial Derivative Gain
double Setpoint, Input, Output, ServoOutput;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);        //Initialize PID object, which is in
the class PID.
Servo myServo;                                   //Initialize Servo.


void setup() {
  Serial.begin(9600);                            //Begin Serial
  myServo.attach(servoPin);                            //Attach Servo
  Input = readPosition();                        //Calls function readPosition() and sets the balls
position as the input to the PID algorithm
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  myPID.SetMode(AUTOMATIC);                            //Set PID object myPID to AUTOMATIC
  myPID.SetOutputLimits(-45,45);                       //Set Output limits to -45 and 45 degrees.
}

void loop(){
  Setpoint = 20;     // setting the point where it is to be balanced
  Input = readPosition();
  myPID.Compute();                               //computes Output in range of -45 to 45 degrees
  ServoOutput=87+Output;                               // 87 degrees is my horizontal
  myServo.write(ServoOutput);                          //Writes value of Output to servo
}


float readPosition() {
  delay(40);
  double duration, cm;
  unsigned long now = millis();
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
```

```
 delayMicroseconds(5);
 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);
 cm = duration/(29*2);
 if (cm>2000) cm=0;  // ball is very close to the sensor
 if(cm > 45)  cm=45;   // 45 cm is the maximum position for the ball
 Serial.println(cm);
 return cm;                              //Returns distance value.
}
```

The PID library used by us is as follows

```
#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif

#include <PID_v1.h>

PID::PID(double* Input, double* Output, double* Setpoint,
      double Kp, double Ki, double Kd, int ControllerDirection)
{

   myOutput = Output;
   myInput = Input;
   mySetpoint = Setpoint;
          inAuto = false;

          PID::SetOutputLimits(-45, 45);                          //default output limit corresponds
to

          //the arduino pwm limits

   SampleTime = 100;                                                    //default Controller Sample
Time is 0.1 seconds

   PID::SetControllerDirection(ControllerDirection);
   PID::SetTunings(Kp, Ki, Kd);

   lastTime = millis()-SampleTime;
}
bool PID::Compute()
{
  if(!inAuto) return false;
  unsigned long now = millis();
  unsigned long timeChange = (now - lastTime);
```

```
  if(timeChange>=SampleTime)
  {
    /*Compute all the working error variables*/
          double input = *myInput;
          double error = *mySetpoint - input;
          if(error>=-1.0 && error<=1.0) error=0;
    ITerm+= (ki * error);
    if(ITerm > outMax) ITerm= outMax;
    else if(ITerm < outMin) ITerm= outMin;
    double dInput = (input - lastInput);
    if (dInput>=-1 && dInput<=1) dInput=0;

    /*Compute PID Output*/
    double output = kp * error + ITerm- kd * dInput;

          if(output > outMax) output = outMax;
    else if(output < outMin) output = outMin;
          *myOutput = output;

    /*Remember some variables for next time*/
    lastInput = input;
    lastTime = now;
          return true;
  }
  else return false;
}
void PID::SetTunings(double Kp, double Ki, double Kd)
{
  if (Kp<0 || Ki<0 || Kd<0) return;

  dispKp = Kp; dispKi = Ki; dispKd = Kd;

  double SampleTimeInSec = ((double)SampleTime)/1000;
  kp = Kp;
  ki = Ki * SampleTimeInSec;
  kd = Kd / SampleTimeInSec;

 if(controllerDirection ==REVERSE)
  {
    kp = (0 - kp);
    ki = (0 - ki);
    kd = (0 - kd);
  }
}
void PID::SetSampleTime(int NewSampleTime)
{
  if (NewSampleTime > 0)
```

```
  {
    double ratio  = (double)NewSampleTime
                    / (double)SampleTime;
    ki *= ratio;
    kd /= ratio;
    SampleTime = (unsigned long)NewSampleTime;
  }
}
void PID::SetOutputLimits(double Min, double Max)
{
  if(Min >= Max) return;
  outMin = Min;
  outMax = Max;

  if(inAuto)
  {
          if(*myOutput > outMax) *myOutput = outMax;
          else if(*myOutput < outMin) *myOutput = outMin;

          if(ITerm > outMax) ITerm= outMax;
          else if(ITerm < outMin) ITerm= outMin;
  }
}
void PID::SetMode(int Mode)
{
   bool newAuto = (Mode == AUTOMATIC);
   if(newAuto == !inAuto)
   {  /*we just went from manual to auto*/
      PID::Initialize();
   }
   inAuto = newAuto;
}
void PID::Initialize()
{
  ITerm = *myOutput;
  lastInput = *myInput;
  if(ITerm > outMax) ITerm = outMax;
  else if(ITerm < outMin) ITerm = outMin;
}
void PID::SetControllerDirection(int Direction)
{
  if(inAuto && Direction !=controllerDirection)
  {
          kp = (0 - kp);
    ki = (0 - ki);
    kd = (0 - kd);
  }
```

```
   controllerDirection = Direction;
}
double PID::GetKp(){ return  dispKp; }
double PID::GetKi(){ return  dispKi;}
double PID::GetKd(){ return  dispKd;}
int PID::GetMode(){ return  inAuto ? AUTOMATIC : MANUAL;}
int PID::GetDirection(){ return controllerDirection;}
```

6.4.2. FLOWCHART

The flow chart for the arduino program is shown in Figure 13 below.

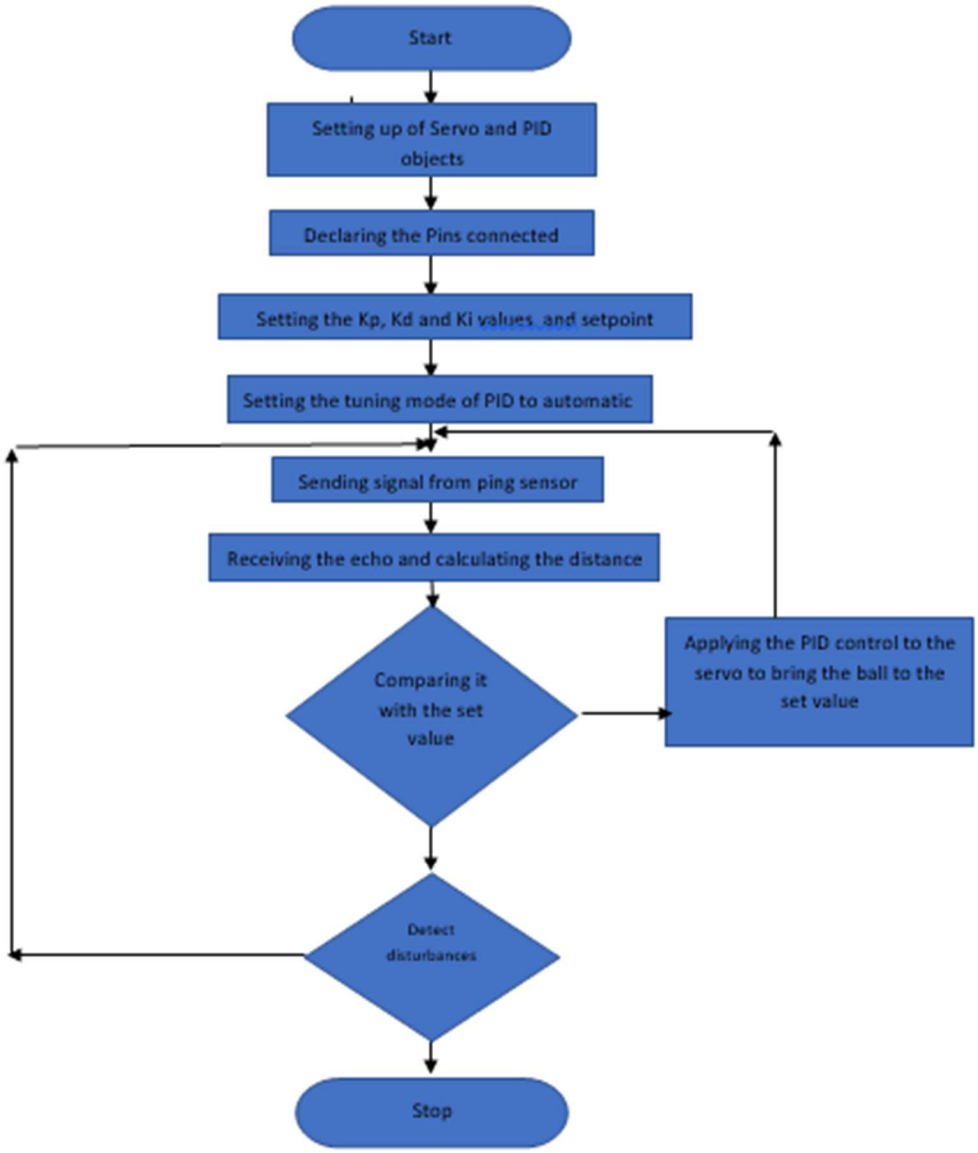


Figure 13. Arduino flow chart

7. SIMULINK MODEL AND INTEGRATION WITH ORIGINAL MODEL

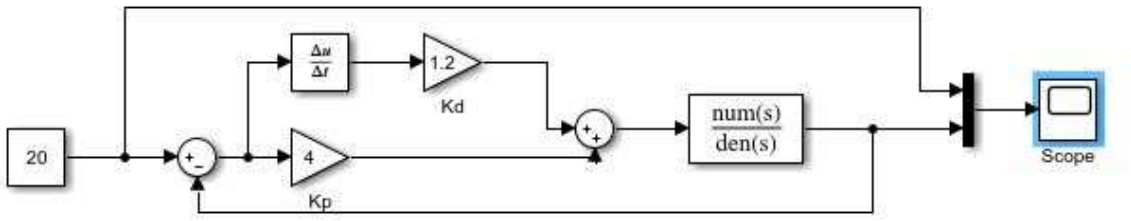


Figure 14. Control system block diagram

Figure 14 shows the Simulink model of the control system block diagram with a step input corresponding to an initial position of the ball of 20cm from the sensor. The transfer function is obtained as

$$G= -\frac{mgd}{L(\frac{J}{R^2}+m)}.\frac{1}{s^2}$$

m = 0.05

g = -9.8

d = 0.042

L = .45

J = 0.00000999

R = d/2

Putting in the values (all in SI units), and using the values of Kp and Kd (2 and 1.2 respectively) a scope graph was obtained as shown in Figure 15.

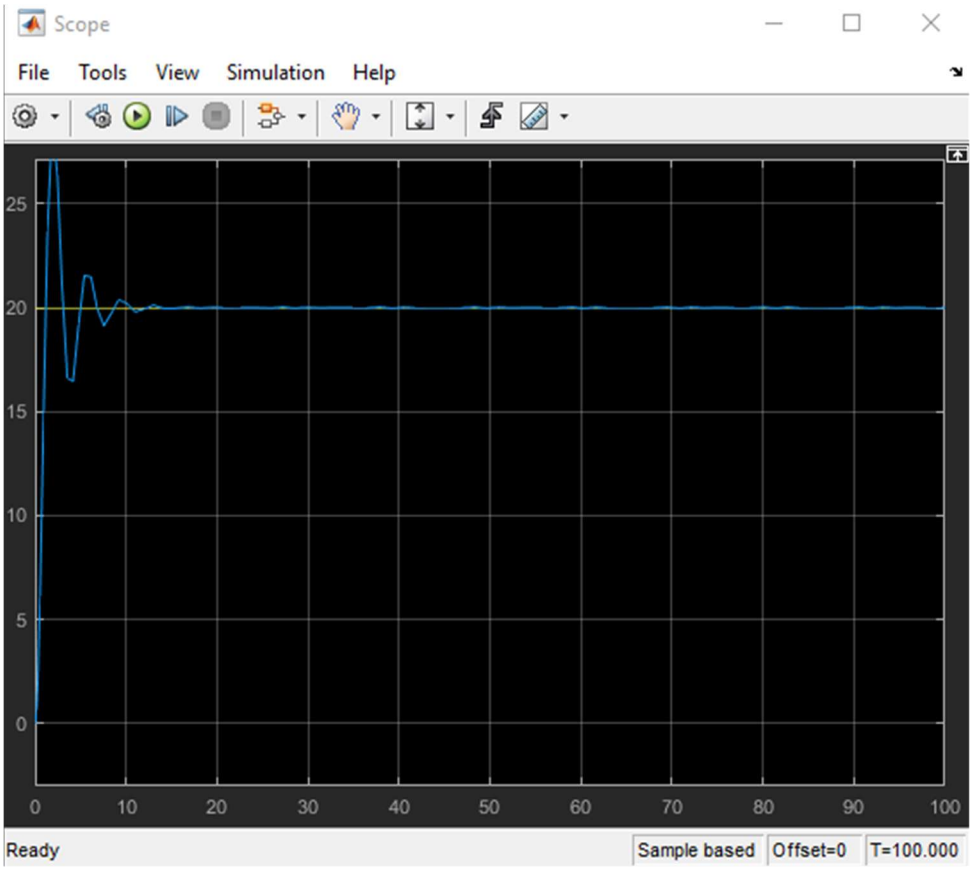


Figure 15. Plot of ball position vs time using the given values

The time taken for the ball to finally attain a stable position varies from 5s to 20s depending on the initial position of the ball.

Below is the Simulink model of the beam along with the ball (Figure 16 and Figure 17). We have hinged it at the centre in this model since the angle of inclination for a particular velocity and position of the ball on the track will not change whether it is hinged at the end or centre. Only the linear displacement of the end will be twice. The tuned Kp, Kd and Ki values had been used by us to tune the mechanism. But we have not used the Ki values as the ball is coming back to the set point in a stable manner even without the ki value. The reasons might be the fact that here we are deviating from the ideal conditions.

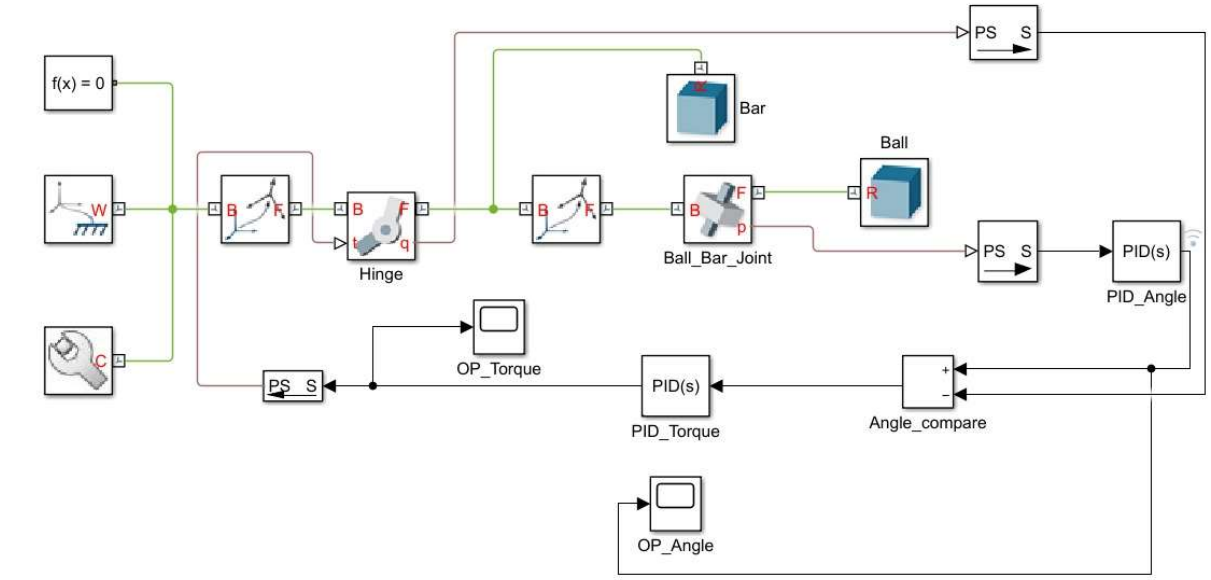


Figure 16. Simulink model block diagram

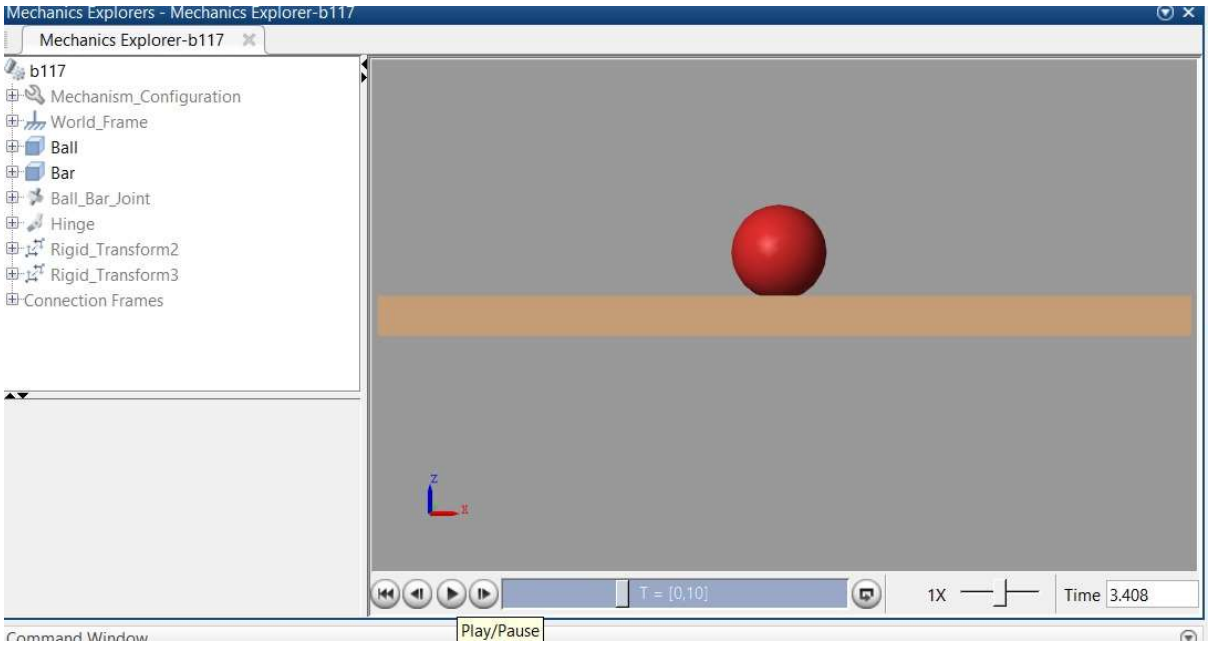


Figure 17. Screenshot of simulation

8. ADDITIONAL WORK FOR BONUS MARKS

- The track has been made with a total length of 50cm with 45cm of it being used as the effective length on which the ball rolls.

- Control of the track using MATLAB and a connected phone (making use of the phone's orientation sensor)

  MATLAB code for setting up mobile:

```
ad=arduino('COM5', 'Uno','Libraries','Servo');
s=servo(ad,'D6');
m=mobiledev;
m.Logging=1;
```

  MATLAB code for control:

```
while 1
   angle=m.orientation(2);
   angle=angle-0;
   angle=angle+87;
   if(angle>135) angle= 135;
   end
   if(angle<45) angle=45;
   end
   angle=angle/180;
   writePosition(s,angle);
end
```

- The Arduino code has been integrated with a feature to balance the ball at an input location

9. CHALLENGES FACED AND HOW WE OVERCAME THEM

9.1. TIME SPENT

We started working on the project a few days after the project statement was released, and until the day before the final deadline. During this time-span of three months, we communicated a lot through a WhatsApp group. We met around 7 times; each meeting approximated to 5 hours give 35 hours. In addition to this, we all worked individually too in some parts. Assuming each of us worked for 7 hours outside the team meetings, the total time approximates to 63 hours. It was the Almost all the sections took an equal amount of time, but it was the fabrication that took at least a little more time than the other sections.

9.2. LESSONS LEARNT

This project had a lot to offer. Firstly, and indispensably, the hands-on experience gave us a deeper understanding of the concepts taught in the course. Secondly, we understood the value of working as a team. It was clear how fast the work progressed when we were working altogether compared to when each of us was working alone. It also helped in creating the best possible final result as all our many diverse (sometimes crazy) opinions went into it and the best were chosen.

9.3. CHALLENGES FACED

We faced two major challenges during the development of our project:

9.3.1. THE REDUNDANT LINKS

Figure 18. Picture taken initially, without redundant links

In our initial design, we did not have the redundant links (see **Error! Reference source not found.**). When we were done fabricating the original set of links assembled it to test its movement, we found that the mechanism was quite unstable and moving out of its plane due to the weight of the track. Only one of the two wood pieces comprising the track was initially involved in the mechanism and the other was attached to the first piece by screws. Thus, the centre of gravity shifted out of the plane causing the undesirable motions. To overcome this defect, a redundant pair of links (cam and follower) attached to a different frame was introduced (see Figure 18). Thus, the entire mechanism was now stabilised.

9.3.2. SIGNAL CONDITIONING-PLASTIC BOTTLES

Figure 19. Picture taken with plastic bottle bodies placed along the length

Once the mechanism was ready, we tested the output of the sensor with the ball and found that the output of the sensor was not accurate for distances beyond 15cm. At first, we thought this was because the golf ball has pits and crests which deflected the ultrasonic waves undesirably. But on experimenting with different balls, we found that for all balls, the sensor was giving faulty values. So, we concluded that it was possibly because the received signals were weak. Then, we came up with the idea of plastic bottles. We cut out the cylindrical bodies of three 2L plastic bottles and placed them along the length as seen in Figure 19 and tested it again. Now the sensor was giving accurate values for all the balls. So, the faulty values were indeed due to the weak strength of the received waves, and the plastic casing now ensured that most of the transmitted waves are contained within the casing always and received at the sensor end.

9.4. EXPENSES INCURRED

The expenses for the project are tabulated in

Table 2, Figure 20 and Figure 21 show the soft copies of the bills.

Table 2. Expenses

Label No.	Component name	Qty	Rate (₹)	Amount (₹)
1	Arduino UNO	1	400	400
2	HC-SR04 ultrasonic sensor	1	100	100
3	MG995 Servomotor (9 kg-cm)	1	300	300
4	9V=1A Adapter	1	160	160
5	Male-female jumper wires	10	2	20
6	Male-male jumper wires	10	2	20
7	Zener Diode (6V)	1	5	5
8	3.9Ω resistors	5	1	5
9	Bread board	1	50	50
10	Golf ball	1	90	90
11	L-clamps	4	10	40
12	Bolts-nuts	6	3	18
13	Screws	12	1	12
14	Double sided tape	1	8	8
15	Flex Kwik super glue	1	40	40

Total expenses, inclusive of taxes: 1268+191= ₹1459.00

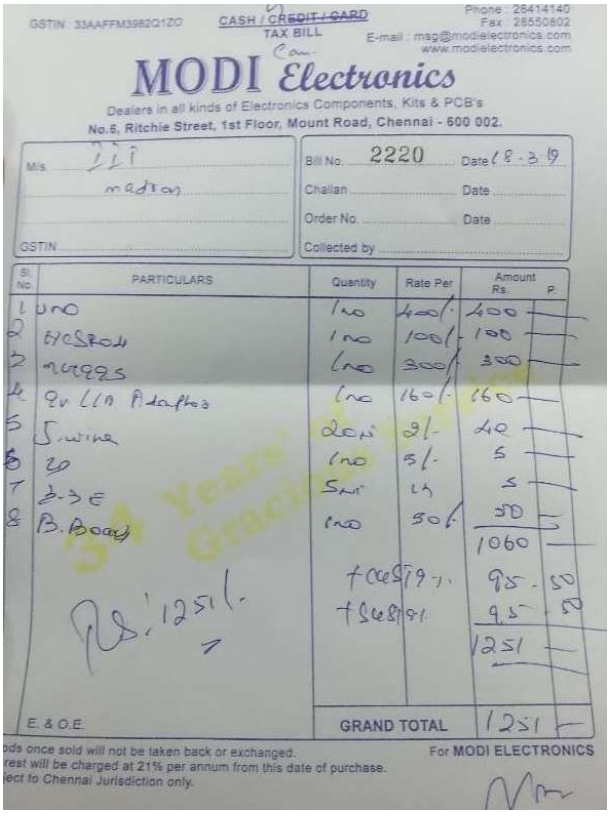
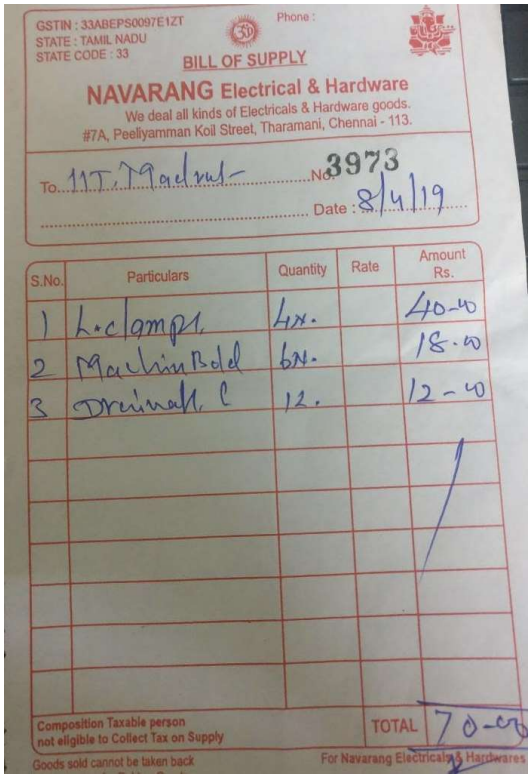


Figure 20. Bill for electrical components



GSTIN : 33ABEPS0097E1ZT
STATE : TAMIL NADU
STATE CODE : 33
Phone :

BILL OF SUPPLY

NAVARANG Electrical & Hardware
We deal all kinds of Electricals & Hardware goods.
#7A, Peeliyamman Koil Street, Tharamani, Chennai - 113.

To. IIT Madras No. 3973
Date : 8/4/19

S.No.	Particulars	Quantity	Rate	Amount Rs.
1	L.C lamps	4x.		40-00
2	Machine Bolt	6N.		18-00
3	Drainage C	12.		12-00

Composition Taxable person not eligible to Collect Tax on Supply
TOTAL 70-00
Goods sold cannot be taken back
For Navarang Electricals & Hardwares

Figure 21. Bill for a few mechanical components

9.5. PICTURES

Figure 22, Figure 23 and Figure 24 show the team working on the project.

Figure 22. Setting the track horizontal using a fluid level indicator

Figure 23. Checking the movement of the mechanism

Figure 24. Tuning