معهد الأهرامات العالى للهندسة والتكنولوجيا
P.H.I For Engineering and Technology
Power and control department
5th level

Course project: Design and realization of ball and beam balancing mechanism

Course name: Digital control systems (ELP-412)

BY:

١- إسلام محمد حسن
٢- أيمن محمد عيد
٣- باسم محمد عبد الحميد
٤- عاطف إبراهيم محمود
٥- عمار بكرى محمد
٦- محمد أحمد عطا
٧- محمد عبد الله عبد الهادى
٨- محمد ياسر عبد المرضى
٩- محمود عادل السيد
١٠-محمود على حسين

Supervised by:

Prof: Gamal El-Sheikh

Abstract

This project is devoted design and realize a control system for ball and beam balancing. The controller design is carried out using classical control PID. The ball is left free on a bar where its position is detected or measured by an ultrasonic sensors (transducer).

Table of Contents

1 Introduction

the our project in digital control systems is **ball & beam** as shown in figure (1), A PID controller allows for precise control of the dynamics of a system during transitory and stationary states; This work describes the design of a Ball and Beam platform for testing control algorithms, and the implementation of a particular PID controller for it. The platform is made out of 2D-wooden parts, an Arduino UNO، a PING ultrasonic sensor, a standard servo motor and screws. The controller is implemented in Arduino language and includes proportional, integrative and derivative terms. The system is successfully stabilized; the ball can be positioned at any point within a range of distance. The platform is cheap, easy to replicate and allows for further testing of alternative control algorithms.

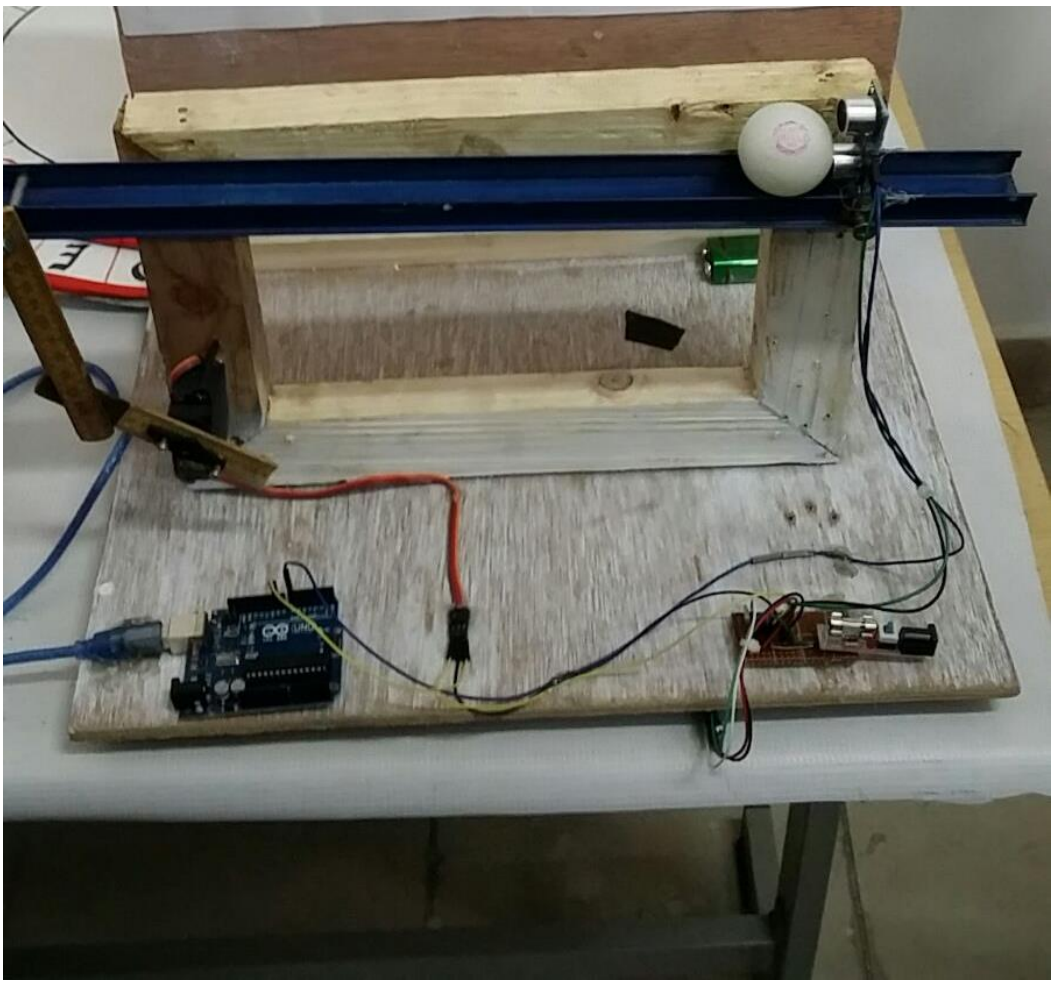

Figure 1 Ball and beam project

2 Physics setup

A ball is placed on a beam, see figure (2), where it is allowed to roll with 1 degree of freedom along the length of the beam. A lever arm is attached to the beam at one end and a servo gear at the other. As the servo gear turns by an angle, the lever changes the angle of the beam by. When the angle is changed from the horizontal position, gravity causes the ball to roll along the beam. A controller will be designed for this system so that the ball's position can be manipulated.

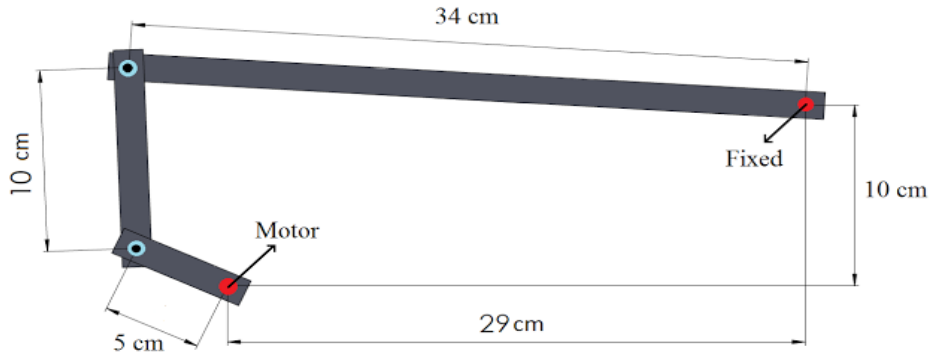

Figure 1 Physics setup

3 Modeling

3.1 system parameters

(m) Mass of the ball

(R) Radius of the ball

(d) Lever arm offset

(g) Gravitational acceleration 9.8 m/s^2

(L) Length of the beam

(J) Ball's moment of inertia

(r) Ball position coordinate

(Alpha) beam angle coordinate

(Theta) servo gear angle

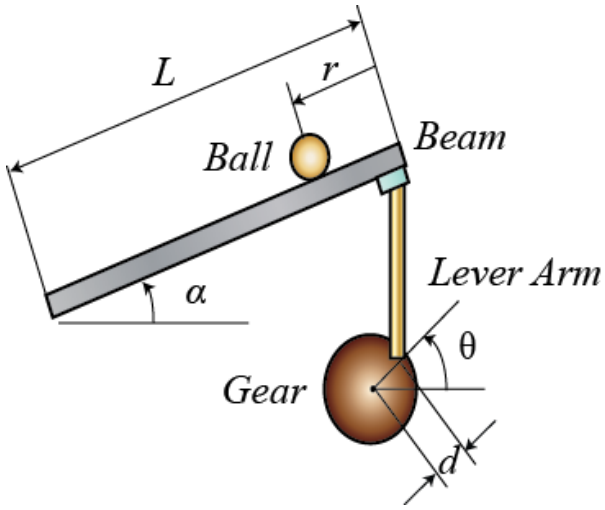

Figure 3 system mechanism

3.2 The design criteria

Settling time less than 3 seconds

Overshoot less than 5%

3.3 System equations

The second derivative of the input angle actually affects the second derivative of. However, we will ignore this contribution. The equation of motion for the ball is then given by the following:

$$0 = \left(\frac{J}{R^2} + m\right)\ddot{r} + mg\sin\alpha - mr\dot{\alpha}^2$$

Linearization of this equation about the beam angle =0, gives us the following linear approximation of the system:

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = -mg\alpha$$

The equation which relates the beam angle to the angle of the gear can be approximated as linear by the equation below:

$$\alpha = \frac{d}{L}\theta$$

Substituting this into the previous equation, we get:

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = -mg\frac{d}{L}\theta$$

3.4 Transfer Function

Taking the Laplace transform of the equation above, the following equation is found:

$$\left(\frac{J}{R^2} + m\right)R(s)s^2 = -mg\frac{d}{L}\Theta(s)$$

Rearranging we find the transfer function from the gear angle (theta) to the ball position R(s):

$$P(s) = \frac{R(s)}{\Theta(s)} = -\frac{mgd}{L\left(\frac{J}{R^2} + m\right)}\frac{1}{s^2} \qquad [\frac{m}{rad}]$$

3.5-Block diagram:

- An ultrasonic sensor detects the actual position of the ball (the "output ball position" .as shown in figure (4),
- The output ball position is compared to the "set point" (the position where we want the ball to be)
- A PID algorithm on the ARDUINO is used to compute the angle of the servo motor based on the difference between the output ball position and the set point (the error).

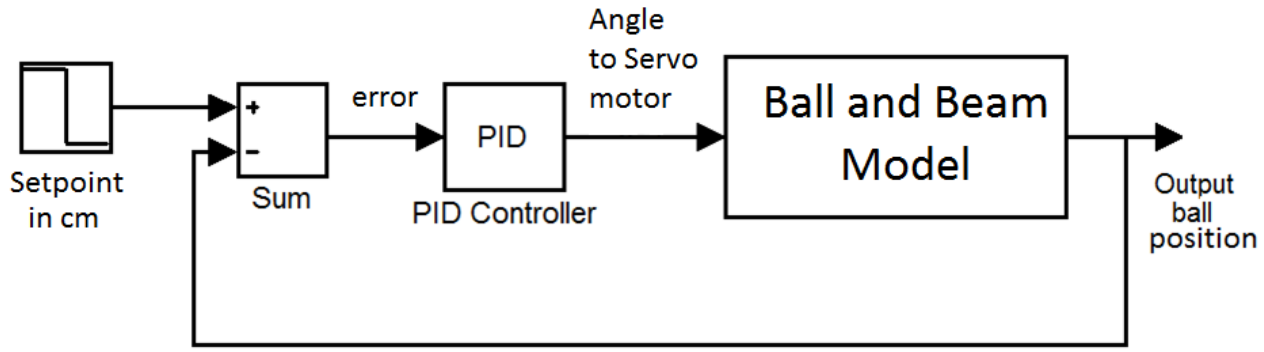

Figure 4 Block diagram

4 Digital PID controller

If you refer to any of the PID control problem for continuous systems, the PID transfer function was expressed as

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

As you noticed the above transfer function was written in terms of s. For the digital PID control, we use the following transfer function in terms of z.

$$C(z) = K_p + K_i \frac{z}{z-1} + K_d \frac{z-1}{z} = \frac{(K_p + K_i + K_d)z^2 - (K_p + 2K_d)z + K_d}{z^2 - z}$$

4.1 Discrete Transfer Function

The first thing to do here is to convert the above continuous system transfer function to an equivalent discrete transfer function. To do this, we will use the MATLAB function c2d. To use c2d, we need to specify three arguments: system, sampling time (Ts), and the 'method'. You should already be familiar with how to create a system from

numerator and denominator matrices. The sampling time should be smaller than 1/(30*BW) sec, where BW is the closed-loop bandwidth frequency. The method we will use is the zero-order hold ('zoh'). Assuming that the closed-loop bandwidth frequency is around 1 rad/sec, let the sampling time be 1/50 sec/sample. Now we are ready to use c2d. Enter the following commands to an m-file. Running this m-file in the MATLAB command window gives you the following matrices as shown in figure (5).

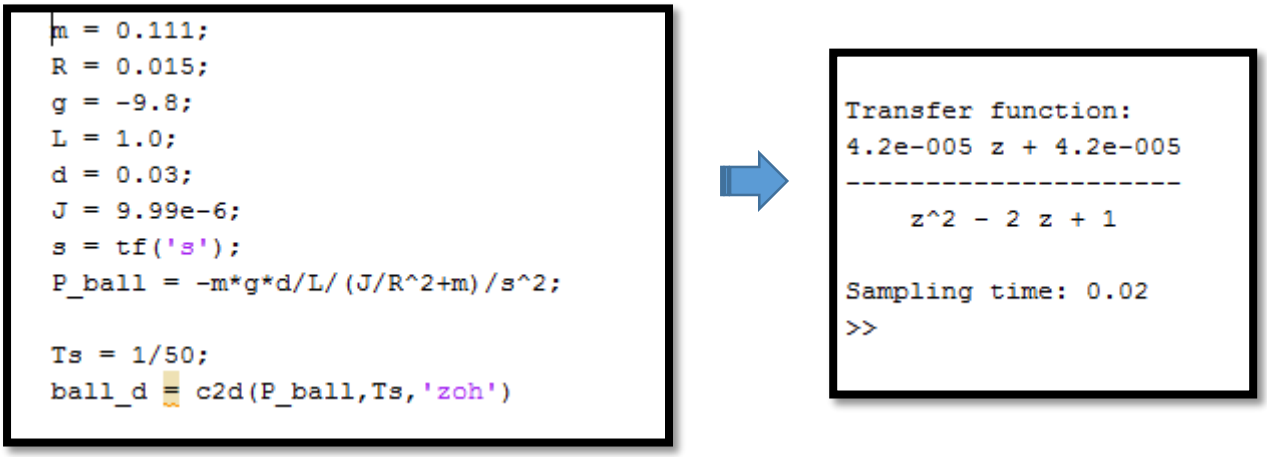

Figure 5 Discrete transfer function

Sample time: 0.02 seconds

Discrete-time transfer function.

4.2 Proportional control

Now we will add proportional control (Kp) to the system and obtain the closed-loop system response. For now let Kp equal 100 and see what happens to the response. Enter the following commands into a new m-file and run it in the command window as shown in figure (6).

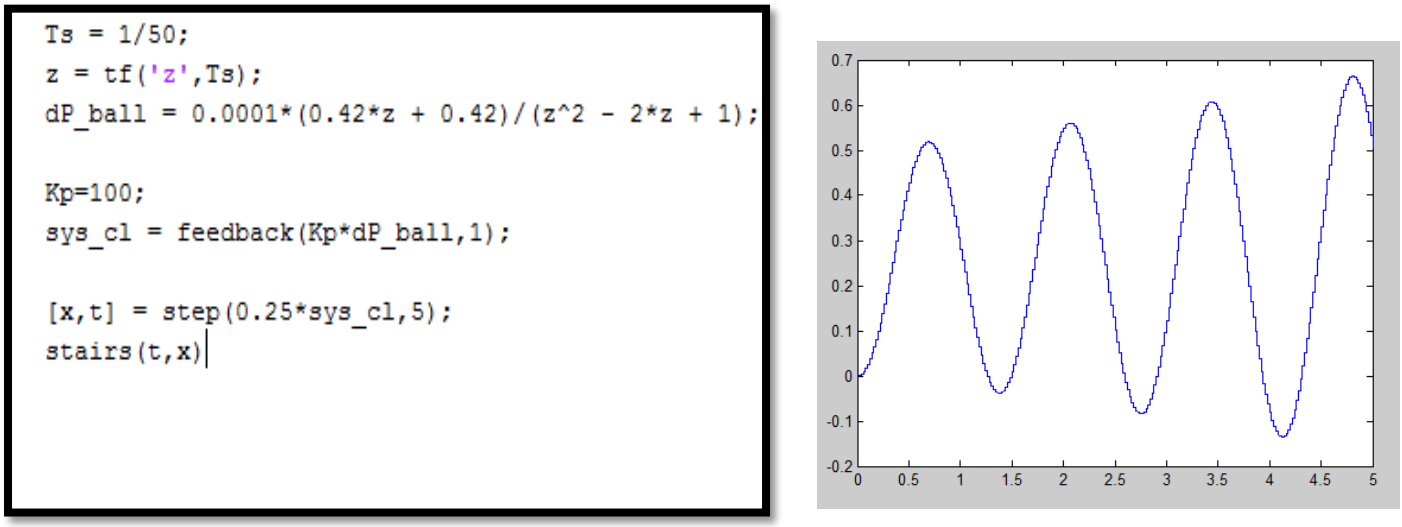

Figure 6 Proportional control

4.3 Proportional-derivative control

Now we will add a derivative term to the controller. Keep the proportional gain (Kp) equal to 100, and let the derivative gain (Kd) equal to 10. Copy the following code to a new m-file and run it to view the system response as shown in figure (7).

```
Ts = 1/50;
z = tf('z',Ts);
dP_ball = 0.0001*(0.42*z + 0.42)/(z^2 - 2*z + 1);

Kp=100;
Kd=10;

C = ((Kp+Kd)*z^2 - (Kp+2*Kd)*z + Kd)/(z^2 + z);

sys_cl = feedback(C*dP_ball,1);

[x,t] = step(0.25*sys_cl,5);
stairs(t,x)
```

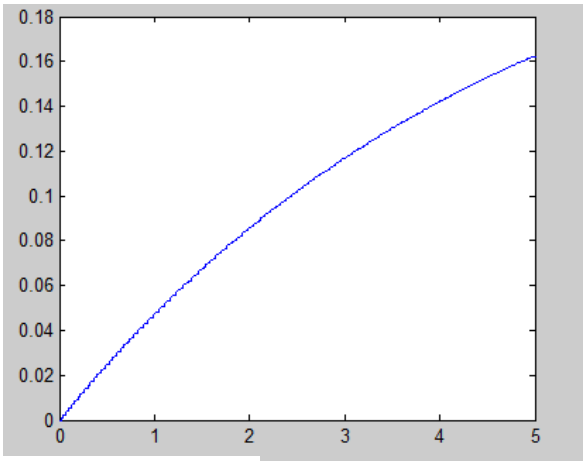

Figure 7 PD Controller

Now the system is stable, but the rise time is too long. From the PID Tutorial page, we see that the increasing the proportional gain (Kp) will decrease the rise time. Let's increase the proportional gain (Kp) to 1000 and see what happens. Change Kp in the above m-file from 100 to 1000 and rerun it in the command window. You should see the following step response.

```
Kp=1000;
Kd=10;

C = ((Kp+Kd)*z^2 - (Kp+2*Kd)*z + Kd)/(z^2 + z);

sys_cl = feedback(C*dP_ball,1);

[x,t] = step(0.25*sys_cl,5);
stairs(t,x)
```

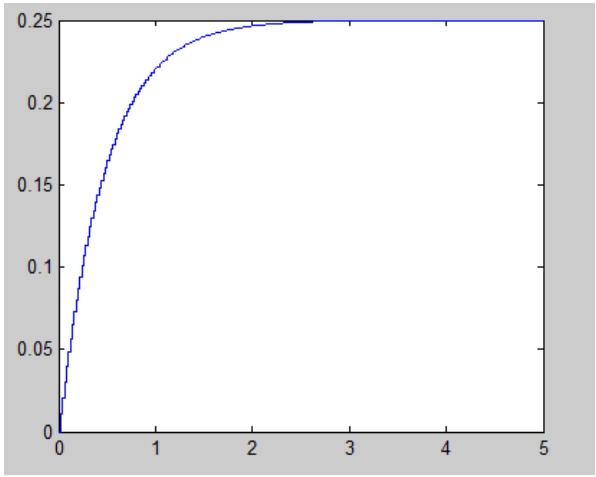

Figure 8 PD control after effect

As you can see figure (8), all of the design requirements are satisfied. For this particular problem, no implementation of an integral control was needed. But remember there is more than one solution for a control problem. For practice, you may try different P, I and D combinations to obtain a satisfactory response.

5 Implementation

5.1 Circuit diagram

Parts used:

- Arduino Uno
- Ping ultrasonic sensor
- 10 Kg.cm Digital Servo Motor
- 5 V DC power supply

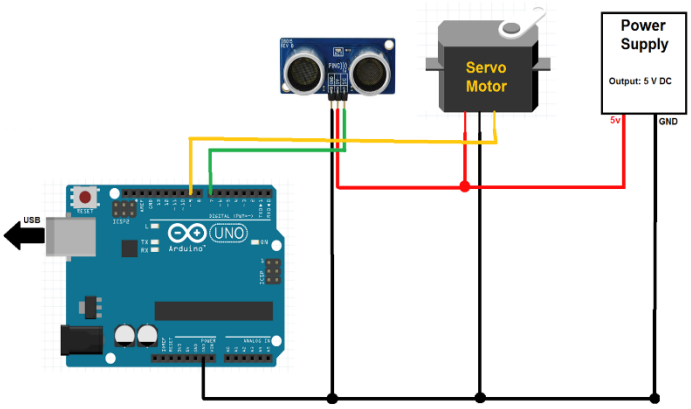

Figure 9 Circuit diagram

5.1.1 ARDUINO UNO

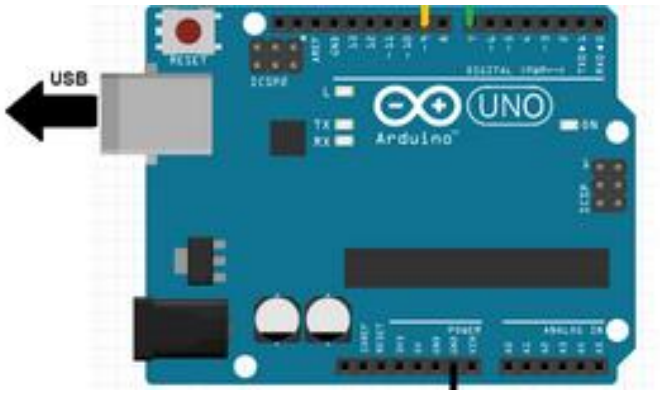

Figure 10 Arduino UNO

ARDUINO is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, though a new model has been designed around a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller

5.1.2 Ultrasonic Distance Sensor

The Parallax PING)))™ ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to microcontrollers such as the BASIC Stamp Propeller chip, or Arduino, requiring only one I/O pin .The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be calculated.

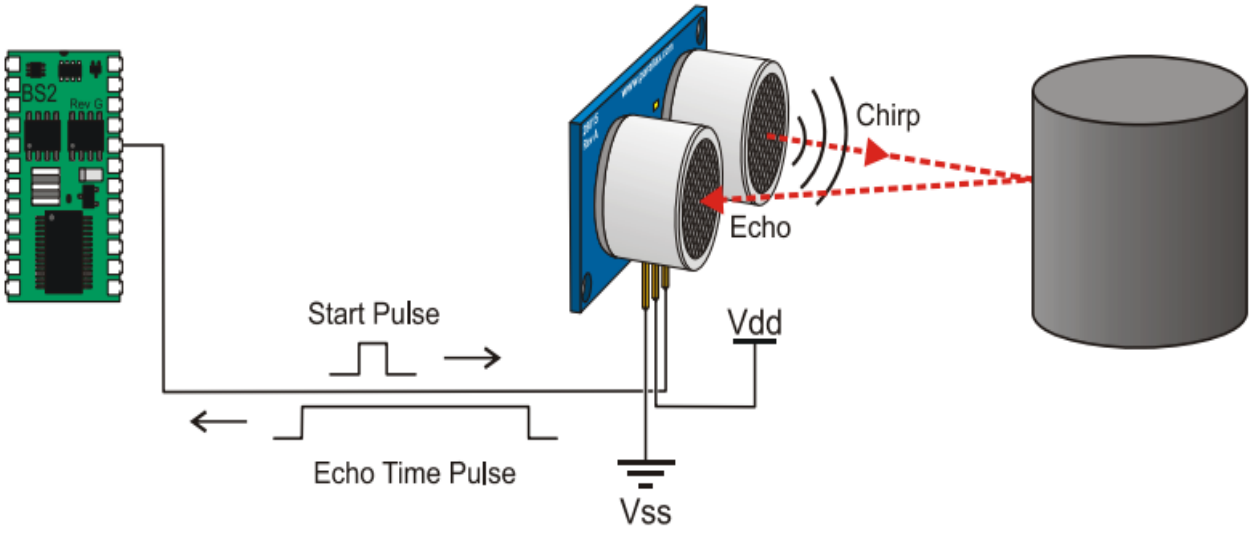

Figure 11 Ultra sonic

5.1.3 Servo motor

Specifications

- •Weight: 55 g
- •Dimension: 40.7 x 19.7 x 42.9 mm approx .
- •Stall torque: 8.5 kgf*cm (4.8 V), 10 kgf*cm (6 V)
- •Operating speed: 0.2 s/60º (4.8 V), 0.16 s/60º (6 V)
- •Operating voltage: 4.8 V a 7.2 V
- •Dead band width: 5 µs
- •Stable and shock proof double ball bearing design

•Temperature range: 0 ºC – 55 ºC

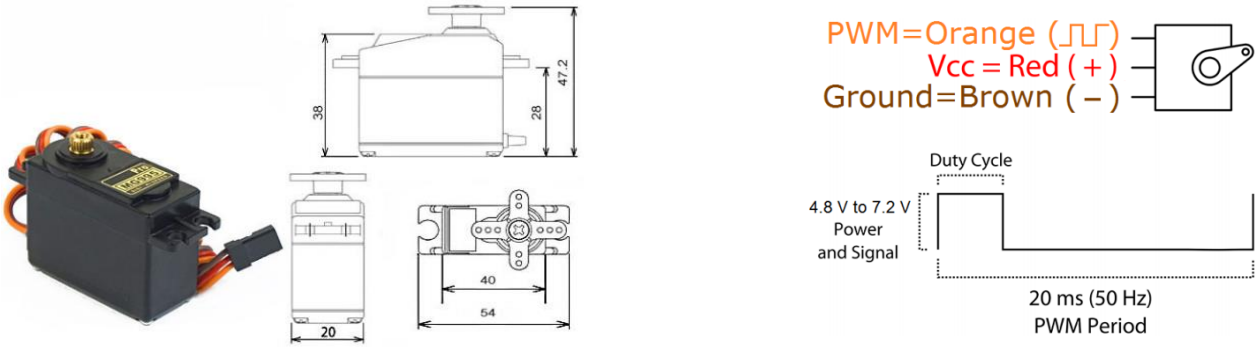

Figure 12 Servo motor

5.2 Programming

Our programing by C language to control by ARDUINO:

```
#include<Servo.h<
#include<PID_v1.h<
const int servoPin = 9;                                    //Servo Pin
float Kp = 14.4;                                           //Initial Proportional Gain
float Ki = 3.6;                                            //Initial Integral Gain
float Kd = 9;                                              //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
  myPID.SetMode(AUTOMATIC);                                //Set PID object myPID
to AUTOMATIC
  myPID.SetOutputLimits(-45,45);                           //Set Output limits to -80 and
80 degrees .
{
void loop()
}
  Setpoint = 17؛
  Input = readPosition                                     ؛()
```

```
  myPID.Compute();                                    //computes Output in range of -80
to 80 degrees
  ServoOutput=90+Output;                               // 102 degrees is my horizontal
  myServo.write(ServoOutput);                          //Writes value of Output to
servo
{
float readPosition} ()
  delay(40);                                          //Don't set too low or echos will run
into eachother      .
const int pingPin = 7؛
long duration, cm؛
unsigned long now = millis؛()
  pinMode(pingPin, OUTPUT)؛
  digitalWrite(pingPin, HIGH)؛
  delayMicroseconds)2؛(
  digitalWrite(pingPin, LOW)؛
  delayMicroseconds)5؛(
  digitalWrite(pingPin, HIGH)؛
  pinMode(pingPin, INPUT)؛
  duration = pulseIn(pingPin, LOW)؛
  cm = duration)/2*30؛(
  if(cm > 31)     // 30 cm is the maximum position for the ball
}  cm=31{؛
    Serial.println(cm)؛
  return cm;                                  //Returns distance value.
{
```