fThe Department of Mechanical and Mechatronics Engineering
MTE 204 - Numerical Methods

204 Project: PID Ball Balancer

Bradley McAllister – bmcallis@uwaterloo.ca
Raj Mody - rmody@uwaterloo.ca
Kelvin Tezinde - ktezinde@uwaterloo.ca
Instructor: Charles Kwan
Group Number: 2

Date: November 10th 2017

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 PID Process and Motivation

This report will examine the PID controller; proportional integral derivative. This systematic controller works by constantly calculating the error value as the difference between a desired setpoint and a measured variable. A correction based on the proportional, integral and derivative of the error is then applied to actuate the system to the desired value. PID controller is a control feedback mechanism which is used in industry for many different applications , for instance a furnace temperature control [1]. This control method is very important for the field of engineering since it can be used for different dynamic systems as an accurate and optimized automatic control. Figure 1 is a diagram of a general PID controller system.

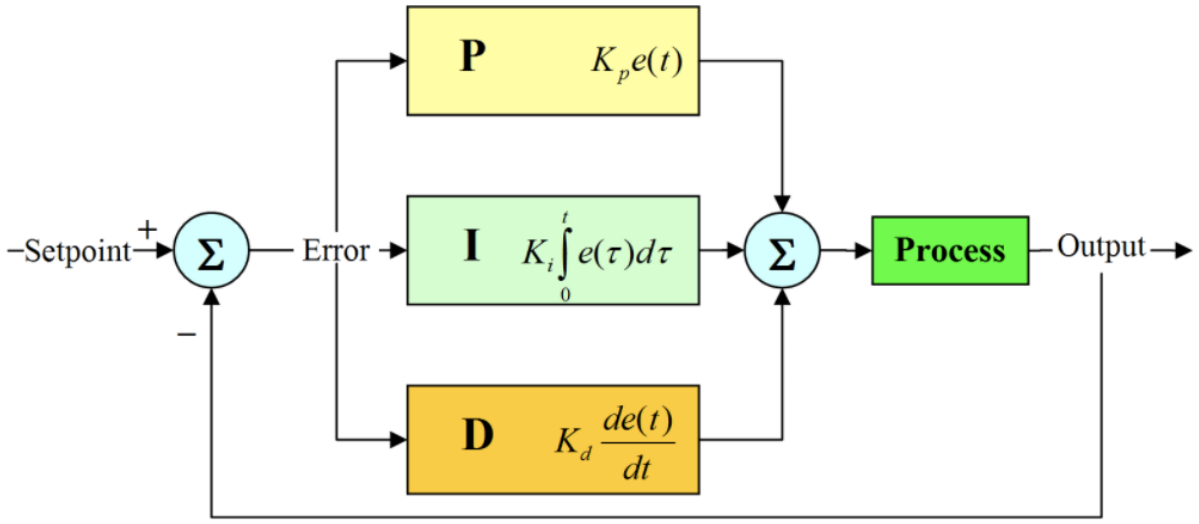

Figure 1: Simple PID Controller Diagram [2]

1.2 PID Validation Design and Process

To demonstrate the PID system, a “ball balancer” is constructed. This consists of a wooden apparatus, a sonar sensor, a microcontroller and a servo motor. The wooden apparatus CAD model shown in Figure 2 has a platform on which the ball is placed. Appendix D shows the experimental apparatus that was constructed. The platform is designed to constrain the ball to one-degree of freedom; 1-dimensional movement. This simplifies the experiment, as it would require only a single servo for a rotational movement along the servo rotary axis. The ultrasonic sensor is mounted to one side of the platform to measure the ball’s position relative to the platform’s axis. Referring to figure 3, the sensor data is fed into the PID algorithm as the input.

The output generated is the pulse width modulation (PWM) signal that is the input to the servo. Based on the response of the ball's movement, a correction output is generated. This process is continued until the ball reaches the desired location and the error is zero.

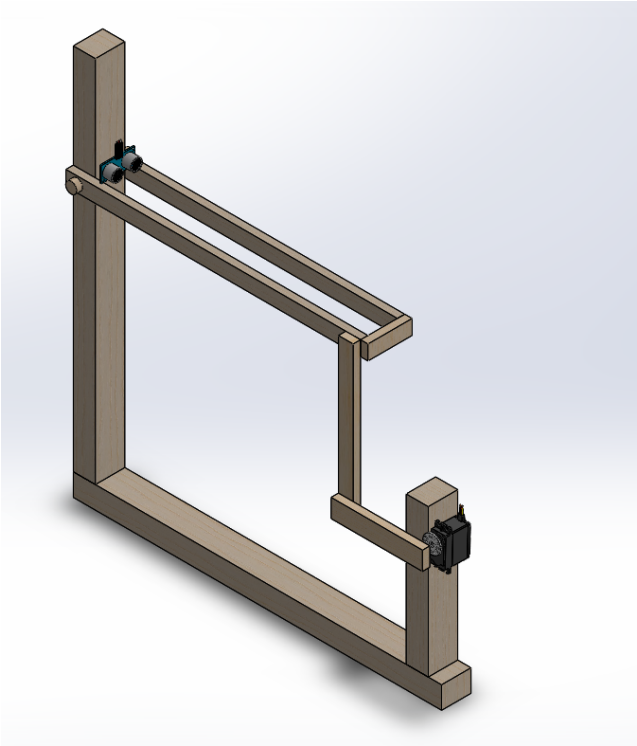

Figure 2: CAD Model of the Concept Apparatus

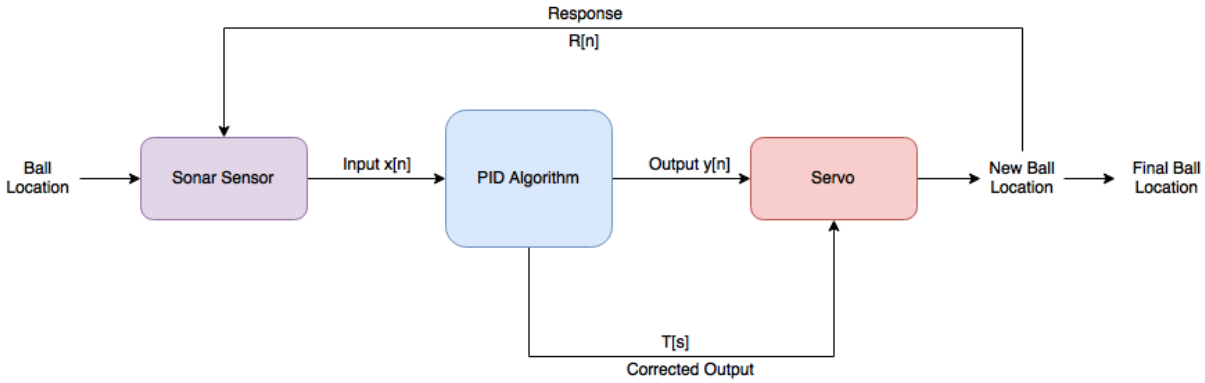

Figure 3: Operation of the Entire System

1.3 Introduction to PID Algorithm

The PID algorithm consists of three parts: the proportional, integral, and derivative. Each of these parts is determined from the error in the system. Since the group's system is a discrete time system, the continuous movement of the ball will be converted to discrete time signal by the

implementation of a polling cycle. Figure 2 demonstrates a high level view of the operation of the system.

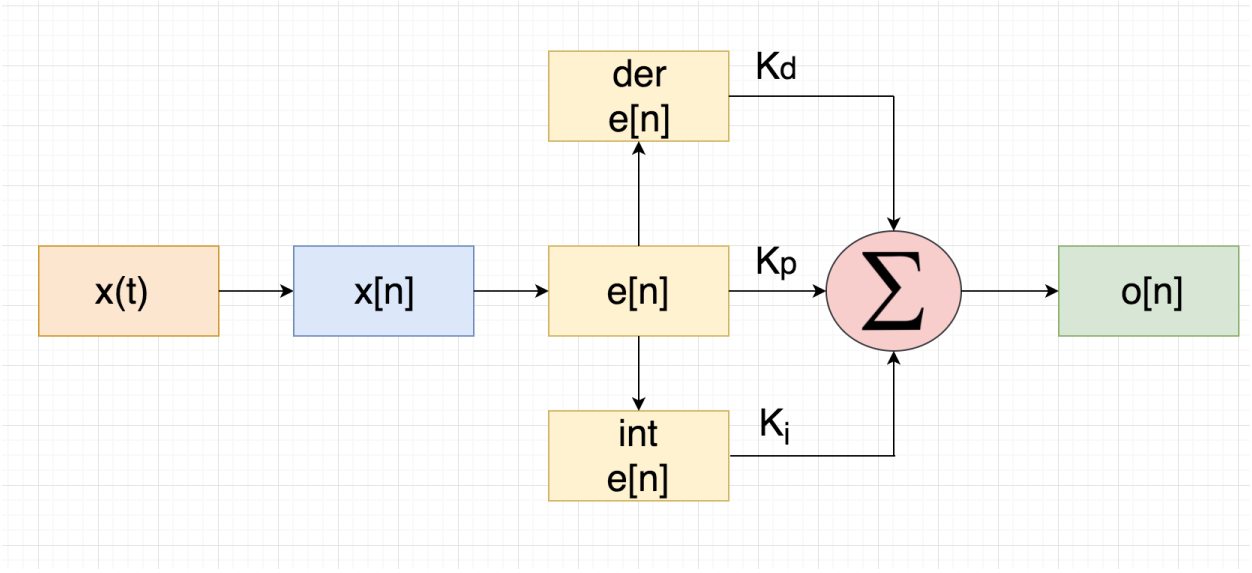

Figure 4: High Level Operation of System

The error is shown in equation 1, where SP is the setpoint and PV is the current location of the ball. This can also be seen in the code within Appendix A, and B at line 61.

$$e[n] = SP - PV[n] \quad (1)$$

Once the error is determined, the integral and derivative can also be determined.

Since the area under the error is the integral, it constantly grows with time. The relationship between the continuous-time and discrete-time signals are shown in equation 2. Furthermore, the integral removes the steady state error in the system; forcing the system to reach steady state at the setpoint. Since the system is a discrete-time system, the integral is calculated as a sum of the error multiplied by the polling time in seconds. This can also be seen in the code base in Appendix B at line 64. PT is the polling time of the system.

$$\int_0^t e(t) = \sum_{n=0}^{t} e[n] \times PT \quad (2)$$

The derivative of the error is calculated by determining the slope of the current and the past error values. Slope is defined as rise/run, where rise is the difference between the current error and the old error and the run is the time difference between them. PT (polling time) is the time difference between the two errors and as such, the differential on the rise is divided by PT. This portion of the PID algorithm ensures that the system does not overshoot the setpoint. Equation 3 shows how the derivative is calculated. Again, since the system is within the discrete time domain, this

equation becomes the best approximation for the derivative of the error. This can also be seen in the code base in Appendix A at line 65.

$$\frac{de(t)}{dt} = \frac{e[n]-e[n-1]}{PT} \tag{3}$$

After all of these values are determined they can be used to calculate the final output of the system (which is the distance of the ball to the sensor), which will be used to change the servo angle, thus changing the incline of the platform. The final equation is shown in Equation 4. The K values are chosen gain values for the different error parameters, which will be expanded upon later in the text. This can also be seen in the code base in Appendix A at line 67. The function o[n] is the output of the system.

$$o[n] = K_p e[n] + K_i \sum_{n=0}^{t} e[n] \times PT + K_d \frac{e[n]-e[n-1]}{PT} \tag{4}$$

1.4 How The System Works

The system begins with a known desired setpoint; the location on the platform where the ball is to be moved to. The ball is then placed at any random position on the platform. The sonar sensor is the input to the system, as it calculates the relative position of the ball on the apparatus. This positional value allows the error for the location of the ball to be calculated, the integral, and derivative. As shown in Equation 4, these values will be multiplied by their respective gain values, $K_{p,i,d}$, then summed to give the output value of the system. This output correlates as a PWM signal to the servo motor which adjusts, moving the ball, thus decreasing the error. This iterative process continuously runs even when the ball has reached the setpoint. This is so that the system ensures that the ball remains at the setpoint, even when external forces move the ball from the setpoint.

2.0 Discussion

2.1 Numerical Analysis and Advantages

A ball balancer (or any type of balancer, whether it be temperature or water) requires some sort of feedback system that monitors the current state of the system and outputs a correction value. The output will either increase or decrease the state of said system in an attempt to reduce the error, and bring the system to the desired equilibrium state. There are several manners to do this, such as having several sensors that continuously monitor and control the system mechanisms. One could even use mathematical models that can solve the issue for certain setpoints and parameters. However, the methods above either require a large amount of resources or calculations and as such, they aren't as efficient as the implementation of a PID controller.

2.1.1 Advantages to PID

Using this specific numerical method is beneficial because:

- The equation used by the PID controller is simple and does not require mathematical manipulation at runtime; can be left to automatically correct for any error
- Does not need to re-compute a large and complex equation, thus saving time and resources
- Can be used for many different systems, only need to change the gain values

2.1.2 Fixed-Point Iteration Method

In order to maximize the use of the PID algorithm, the group employed the fixed point iteration method. This is the best method to go along with the algorithm because in order for the algorithm to function properly, it needs to constantly re-evaluated the most recent error value. Although the system is not using the last output of the equation, it is using a value which is a function from the past output value. This equation therefore converges on an error value of zero. The general equation can be seen in Equation 5.

$$o[n + 1] = f[e[o[n]]] \quad (5)$$

The PID method, even with issues such as integral windup, is the optimal method for this type of situation. This is because it is easy to implement and requires very little hardware components and computation time. It is also the most robust method when compared to the P, PD and PI controllers [3]. Other methods would require either complex mathematical analysis of the problem, lots of tools to analyze the system or a mixture of both. Overall using this numerical method to solve these type of problems will reduces the cost of analysis (not as many sensors

and parts required) whilst minimizing the mathematical complexity. Because of its simplicity and reliability, PID controllers tend to be used in complex systems, such as the autonomous vehicle as stated in [4].

2.2 Assumptions Made

There are some assumptions made during the creation of this PID algorithm.

- The setpoint will not change during the process of balancing
- The controller will not change from automatic to manual control
- The actuator will not be turned off during the balancing process
- The gain values are constant; no automatic tuning processes

Making these assumptions allows the system to maintain its simple algorithm and design. This allows the group to mainly focus on numerical methods instead of control theory.

2.3 Prototyping the Algorithm

To develop the algorithm and validate its numerical analysis, a matlab simulation was developed. However, to avoid modelling the servo's reaction and the ball's physical response to the output, a fixed path was defined. Referring to Appendix A, the path of the ball is stored in the PV vector. By fixing the movement of the ball, the error and output calculations can be validated.

2.3.1 Assumptions in Simulation

To simulate the error and the output response, a few assumptions were made to constrain the problem. Foremost, to isolate for output and simply the problem, the path (movement) of the ball about the platform was pre-determined. Secondly, the data was constrained to only 25 values, vastly simplifying the problem, as time is not a critical factor in the PID controller, it can be manipulated as needed. Furthermore, the polling time was assumed to be 0.01s, indicating that the servo response was also within 0.01s. Given the 25 data points for the ball's position, this indicates that the total system came to steady state after 0.25s. Although this is not the case in reality, the simulation is only a means to validate that the error and output calculations follow reality.

The predetermined input values are shown in Figure 5 as a means to visualize the moment of the ball. Once again, these input values would change in a modelled PID controller as the ball response is a function of the controller's output; which is removed in this simulation. Steady state or equilibrium is at 15cm for this model. This movement is designed to be a oscillatory to cover a wide range of cases.

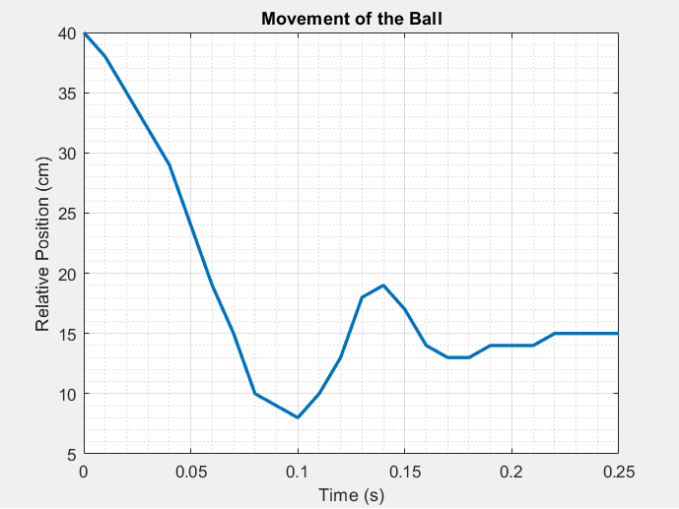

Figure 5: Relative Position of Ball vs Time

2.3.2 Validating the Algorithm

Running the code in matlab from Appendix A, the following Figures 6 and 7 are generated. Figure 6 indicates the error calculated from the input, which is a clear reflection of the input over the x-axis. This validates that the error calculations are practical.

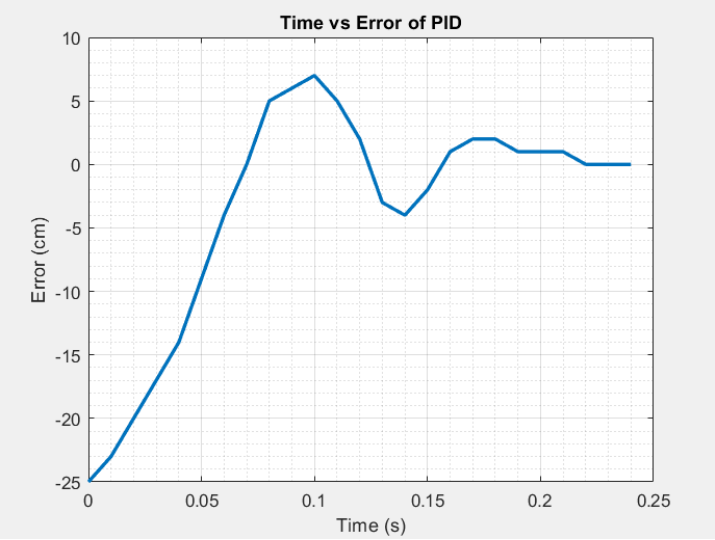

Figure 6: PID Error vs Time

Figure 7 represents the output of the PID controller. When the ball's location is greater than the setpoint, the output of the system should be positive, which correlates to an inclined platform. An inclined platform will move the ball closer to the sonar sensor, thus decreasing the relative position. When the ball's location over shoots the setpoint, the system output drops to a negative

output to correct for the error. This is expected as the apparatus would then be declined to decelerate the ball, then accelerate it in the opposite direction.

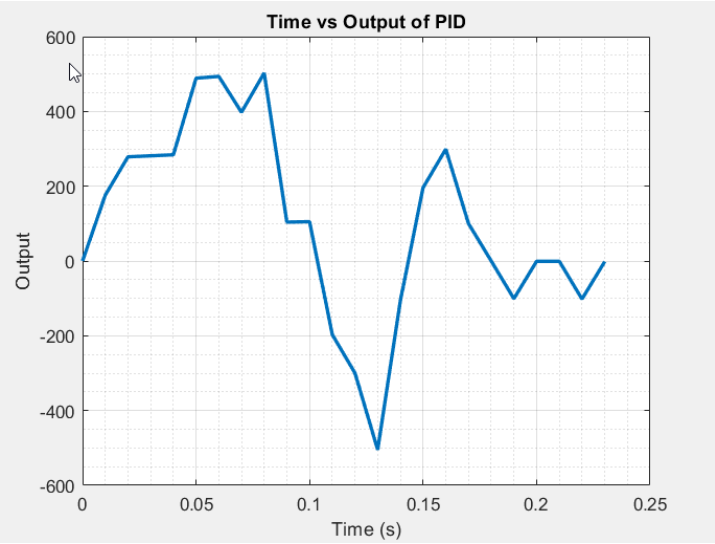

Figure 7: PID Output Value vs Time

2.4 Numerical Analysis in Achieving Goals (Explaining The Code)

As was mentioned in the introduction section, the PID Controller uses the proportional, derivative and integral of the error to balance a system to a chosen setpoint. A PID controller numerical model was implemented (with the group's specifications) using an arduino C coding language that ran on the Arduino Uno.

The code has two main functions and three helper functions. The two main functions are the *setup* and *loop*. The setup function is run once when the board has been loaded with the compiled code. Inside of this function are initializations and pin setup. The *loop* function is continuously called as long as the board has power. This function contains the PID controller logic. The logic can be seen in Figure 8. The logic uses the three helper functions which are *getSonarValue*, *getBallLocation* and *calculate*. The *getSonarValue* function simply sends a signal to different pins on the sonar sensor and calculates the duration that it takes for the sound wave to travel to the object and bounce back. This function returns the travel time in milliseconds. The *getBallLocation* uses the *getSonarValue* function and converts the time to a distance in centimeters and then ensures that the distance is within a reasonable range of values and returns the location of the ball in centimeters. The final function *calculate* takes in the current location of the ball in centimeters and computes the error, integral of the error, derivative of the error, which are all used to determine the output of the PID algorithm. The function then makes sure that the

output is within appropriate bounds and returns the output. The source code can be found in Appendix A.

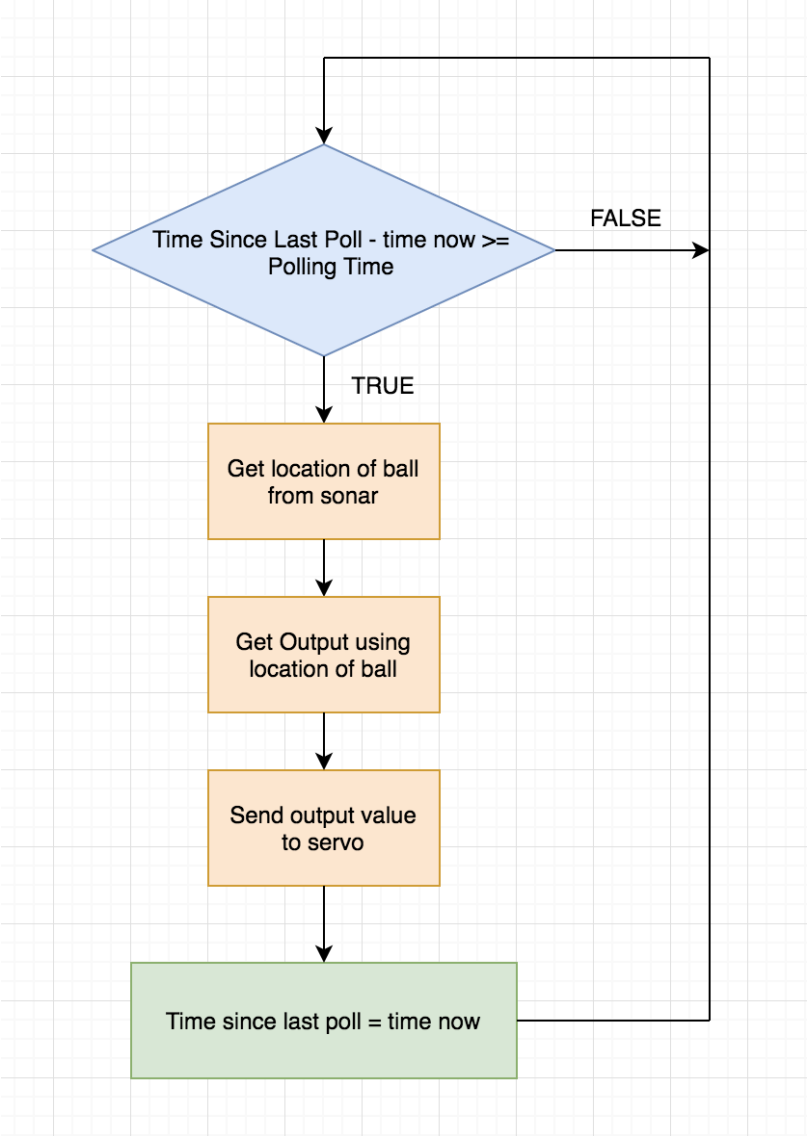

Figure 8: General Flow of Arduino Code

3.0 Testing & Results

3.1 Trial and Error

Trial and error is the primary method used to determine the three different gain values for the PID algorithm. However, the gain selection was not entirely random, since there are the different effects on the output depending on the values chosen for the proportion, integral and derivative gain values. Table 1 shows the different effects that each part of the algorithm will have on the output of the system. This table is used to tune the PID algorithm.

Table 1: PID Parameter Effects on Output [1]

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Kp	Decrease	Increase	Small Change	Decrease	Degrade
Ki	Decrease	Increase	Increase	Elimiate	Degrade
Kd	Minor change	Decrease	Decrease	No effect in theory	Improve if Kd small

3.2 Calibrating the System

During the calibrating process, many combinations of gain values yielded a desired outcome. However, minor variations in reaching the steady state were observed. The different set of gain values are compared against each other, and the most optimal set of gain values is chosen as the final values for the system. The optimal set would bring the ball to the setpoint with minimal servo movements. This would provide a smooth transition from the ball rolling to stabilizing at the setpoint. Table 2 displays the different gain sets that the group determined after testing. It was decided that set number 2 was the most optimal set.

Table 2: Tested PID Gain Values

Set Number	Kp	Ki	Kd
1	4.9	2.9	-2.15
2	5	3	-3.3
3	6	4.75	-3.95

3.3 Results

After the correct tuning, the desired result was achieved, which was the successful balancing of a ball. Section 3.3.2 contain several graphs from a test run of the system, showing that the algorithm works as intended (balancing the ball successfully).

From all the trials conducted, one result was chosen for the result analysis because it represents the general trend of group data. Since there are many variations and combinations of results, with the boundary conditions illustrated in 3.3.1, the best result was chosen for graphical representation.

3.3.1 Boundary Conditions for Testing

Table 3 outlines the testing conditions.

Table 3: **Testing Constraints**

Constrain	**Value**	**Explanation**
Output	0 - 33 duty cycle	Due to mechanical and physical limitations, the servo duty cycle is limited to range of 0 to 33.
Set point	6 cm	The set point chosen at random; can be any value between 5 to 15 cm to yield the best results.
Ball Location	0-20 cm	Since the sonar sensor cannot accurately detect the ball past 20 cm the maximum value is set to 20 cm.

3.3.2 Data

Figure 9 depicts the error of the system over time in a test run, where the error is defined as the difference between the setpoint and the current location of the ball (with respect to the sonar sensor). When the error reaches zero, it signifies that the ball has reached the setpoint, signifying that it has been balanced.

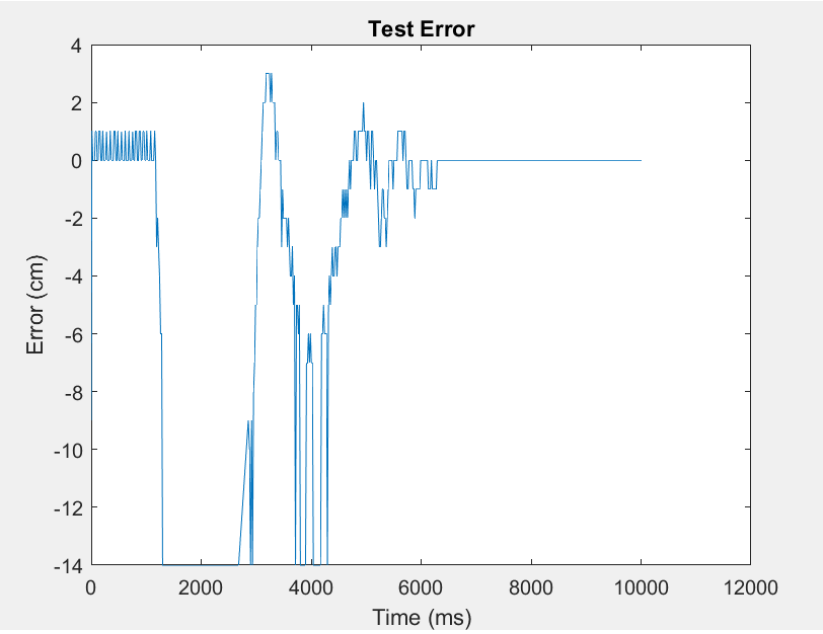

Figure 9: Error vs Time for Test Run

The sonar sensor readings vary from 3 to 20 centimeters. The group decided to make the maximum distance from the sonar sensor 20 centimeters due to the unreliability of the sonar sensor for values exceeding 20 centimeters. A graph of the sonar sensor readings during a test run can be seen in Figure 10.

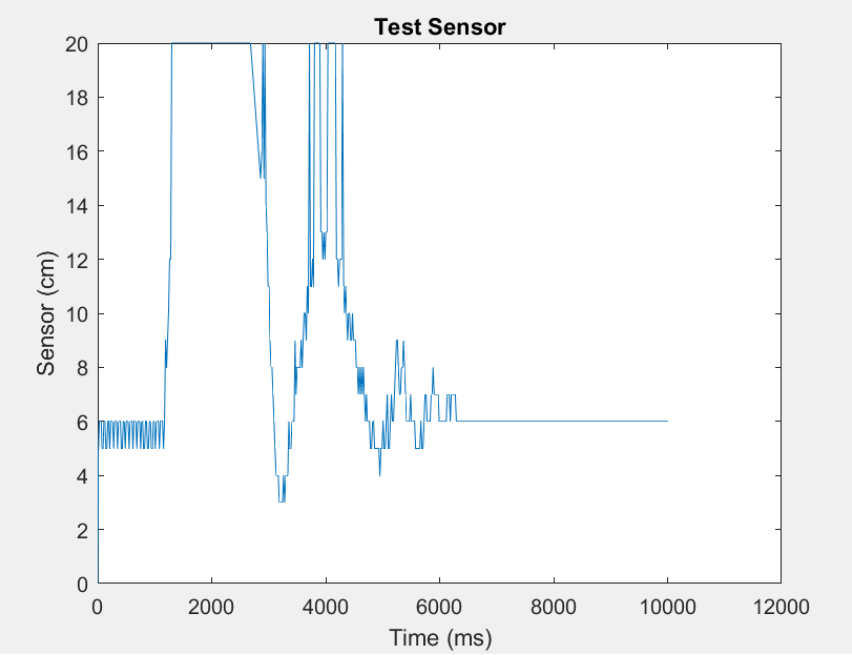

Figure 10: Sonar Reading vs Time for Test Run

The output values are the results from the PID logic and are the PWM signals that are sent to the servo motor. The output values have a set range of 0 to 33, which was decided by the group. This range of values allows for good control over the ball's movements. A graph of the output values during a test run can be seen in Figure 11.

Figure 11: Output Values vs Time for Test Run

3.3.3 Data Analysis

The data shown in Figures 9 to 11 show different variables from the same test run. As can be seen for the beginning of the test the ball was close to the setpoint this is why there are small fluctuations in the error and sonar values at the beginning. The ball was then moved away from the sonar sensor and the balancing process began. From the output values it can be seen that the smaller the value the more elevation that the platform will have. A higher elevation moves the ball closer to the sonar sensor. The opposite goes as well that the higher the output value the lower the elevation, which results in the ball moving away from the sonar sensor. The error and sonar values show that the ball moves in a damped sinusoidal fashion which converges on zero error; the setpoint. Once the ball has stabilized on the setpoint the output value remains at a constant value to keep the platform flat so that the ball does not move.

3.3.4 Comparative Analysis between simulated and experimental data

By using Figure 10 as the observed movement of the ball, a comparative analysis between the simulated and experimental values can be made. Comparing Figures 6 and 9, the error value is a reflection of the observed movement of the ball over time. This validates the relationship between the expected and experimental data, as the ball becomes further away from the setpoint, the error becomes larger in the negative direction if moving away from the sensor, or positive if moving closer to the sensor. Comparing Figures 7 and 11, the output follows the error value. The output is greater for the same scenarios indicated above, hence, the expected and experimental data match.

4.0 System Limitations

In theory, a PID controller can be a perfect solution to control system, however, practical limitations in sensors, computation and actuators vastly constrain the efficiency. The test apparatus built was greatly limited by these factors, however, the system was still useful in proving the functionality of the system.

4.1 Actuator Saturation

Actuator saturation is when the actuator, in the groups case the servo motor, is not large enough to cause a large control effect as request by the controller. This causes the integrated error to continually rise since the error is not being properly corrected for [5]. The setpoint can, in response, be lowered to a value that is achievable by the actuator, but since the integrated error is now a large value, the controller will overshoot the new setpoint.

For the groups implementation, this could occur if the ball was rolling too fast and the servo could not elevate the platform high or fast enough to slow down or change the direction of the ball. In order to ensure that this does not happen, the group had to determine the maximum and minimum elevation of the platform needed to almost instantaneously change the direction of the ball. These max and min values are then used during the system calibration process to ensure that the output values from the PID algorithm would range from the max and min values.

4.2 Ultrasonic Sensor Limitation

Since the ultrasonic sensor depends on the reflection of the transmitted sound waves, it posses both physical and electrical limitations. Physically, a sound wave's amplitude and energy diminish over time and distance. The further the sound wave travels, the more the energy diffuses in the air. This means that to detect an object further away, the sound wave needs to be stronger and the object of detection needs to have a greater and flatter surface area. The HC-SR05 used in the apparatus is a low-end ultrasonic sensor, highly dependant on the distance and shape of the object being measured. Flatter objects will reflect the sound wave in the correct direction; back to transmitter and receiver, whereas, spherical objects will reflect the soundwaves in different locations.

These physical properties of the sound wave made detecting the ball limited to a range of 0 to 20cm. To compensate for the limited range, the PID controller's algorithm had to take into considering boundary conditions. This limited the PID system to only be capable of balancing the ball between 5 to 15 cm. Furthermore, the reliability of the ultrasonic sensor was noticeably

decreasing with range. This resulted in inconsistency in actuation when the ball was from 15 to 20 cm.

4.3 Apparatus Limitation

The apparatus is subject to manufacturing defects, this affects the response time, stability and precision of the system. Wood was chosen as primary manufacturing material due to its relative simplicity to be cut, shaped, and used for prototyping. However, it was notable that the apparatus was limited in the movement; up or down, therefore restricting the pwm output from 0 to 33, correlating to a slower response.

If the ball was heading toward the sonar sensor with a velocity greater than the pwm 33 drop de-acceleration could slow it down, it would collide with the sensor barrier. Some of the energy in the system is then dissipated due to the collision, simplifying the PID controller's task to balance the ball.

Conclusion and Recommendations

The project investigates a PID algorithm using the fixed point iteration method to balance a ball. Using an Arduino Uno, sonar sensor, servo motor, and a mechanical apparatus, a ball balancer is built. This apparatus encompasses the closed loop system with feedback via the numerically implemented PID controller. The simulation and experimental data validate that the error and output are reflections of the input over the time axis. This indicates that the system calculates the error and correctly maps to a correctional output. Although the PID system is successful in ball balancing, there are a few aspects of it that could be improved for faster and more accurate results.

Because the sensor used in the groups model lacks accuracy when the ball is further than 20cm, a better and more accurate manner of reading distance is highly recommended. This could be achieved by using a camera and computer vision or a better distance sensor (such as the Parallax Ping). This is important because the accuracy of the ball balancer, as well as the speed of convergence, relies heavily on the robustness of the distance measuring device.

A more responsive servo motor would be another recommendation that could improve the effectiveness of the system. With a more responsive servo, the system could do faster and more precise angle changes which could potentially increase the rate at which the system converges.

A faster microcontroller that runs at a 32-bit architecture will allow for higher numerical accuracy and calculation time. This would allow for faster calculations of data, resulting in a faster convergence time. For example, a beaglebone operating a 1GHz clock speed would be an optimal solution to run any PID algorithm with 2 input/outputs or more.

Modeling the system using software such as MATLAB and Simulink, would allow for more precise gain values for each error parameter, ‘k’. This would allow for dynamic tuning functionality, this would allow for a more accurate and faster method than the trial and error method implemented for the experiment. Furthermore, modelling the system can allow a greater test case of parameters as the system is not mechanically limited. For example, the model can be used for finding the optimal servo armature length, for the fastest response time and maximum platform deflection, given a certain servo.

Incorporating such changes for a future design would allow for faster convergence, more reliable results and a larger range for the setpoint.

References

- [1] J. Zhong, "PID Controller Tuning: A Short Tutorial", *Saba.kntu.ac.ir*, 2006. [Online]. Available: http://saba.kntu.ac.ir/eecd/pcl/download/PIDtutorial.pdf.
- [2] Image, [Online]. https://raw.githubusercontent.com/ivmech/ivPID/master/docs/images/pid_control.png
- [3] C. Schmid, "Course on Dynamics of multidisciplinary and controlled Systems", *Atp.ruhr-uni-bochum.de*, 2005. [Online]. Available: http://www.atp.ruhr-uni-bochum.de/rt1/syscontrol/main.html
- [4] P. Zhao, J. Chen, Y. Song, X. Tao, T. Xu and T. Mei, "Design of a Control System for an Autonomous Vehicle Based on Adaptive-PID", *Journals.sagepub.com*, 2012. [Online]. Available: http://journals.sagepub.com/doi/full/10.5772/51314. .
- [5] V. VanDoren, "Fixing PID | Control Engineering", *Controleng.com*, 2012. [Online]. Available: https://www.controleng.com/single-article/fixing-pid/3975cad3f121d8df3fc0fd67660822b1.html.

Appendix A

(Math Lab Simulation Code)

```
% PID Constants
kp = 1;
ki = 2;
kd = 1;

% Set Points in CM
SP = 15;
PV = [40 38 35 32 29 24 19 15 10 9 8 10 13 18 19 17 14 13 13 14 14 14 15 15 15 15];
% Polling time in milliseconds
PT = 0.01;

% Error and Output Holders
e = zeros(1,25);
output = zeros(1,1);

% Helper Variables
newOutput = 0;
n = 2;
servoRange = 0:33;

% Initial Error on Ball Detection (no System reponse at n = 1)
e(1) = SP - PV(1);

% run for 25 measurements
while n < 25
   e(n) = SP-PV(n);
   newOutput = kp*e(n) + ki*sum(e)*PT + kd*((e(n)-e(n-1))/PT);
   output = [ output newOutput];
   display(output(n));
   n = n + 1;
end

% Generate the tables
time = 0:0.01:0.01*25;

figure;
plot(time,PV,'LineWidth', 2);
title('Movement of the Ball');
```

```
xlabel('Time (s)');
ylabel('Relative Position (cm)');

grid on
grid minor

time = 0:0.01:0.01*24;

figure;
plot(time,e,'LineWidth', 2);
title('Time vs Error of PID');
xlabel('Time (s)');
ylabel('Error (cm)');

grid on
grid minor

time = 0:0.01:0.01*23;

figure;
plot(time,output, 'LineWidth', 2);
title('Time vs Output of PID');
xlabel('Time (s)');
ylabel('Output');

grid on
grid minor
```

Appendix B

(Arduino Code)

```
#include <Servo.h>

Servo servo;

// Pin locations
const int servoPin = 5;
const int trigPin = 11;
const int echoPin = 10;

// Constants for PID algorithm
const float Kp = 5;
const float Ki = 3;
const float Kd = -3.3;

// Global variables
float setPoint = 6;
float intError = 0;
const unsigned long pollingTime = 10;
unsigned long lastTimeCalculated = 0;
uint32_t lastOutput = 0;
float lastError = 0;

void setup(){
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    Serial.begin(9600);
    servo.attach(servoPin);
    lastTimeCalculated = millis();
}

void loop(){
    uint32_t now = millis();
    if (now - lastTimeCalculated >= pollingTime){
        int ballLocation = getBallLocation();
        int output = calculate(ballLocation);
        lastTimeCalculated = now;
        servo.write(output);
```

```
    }
}

int getBallLocation(){
    int duration = getSonarValue();
    // Calculating the distance
    int distance = duration*0.034/1;
    if (distance < 0) distance = 20;
    if (distance > 20) distance = 20;
   return distance;
}

int getSonarValue(){
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time
    return pulseIn(echoPin, HIGH);
}

int calculate(int input){
    float error = setPoint - input;
    Error = error;

    intError += error * pollingSecs;
    float derError = (error - lastError) / pollingSecs;

    float output = error * Kp + intError * Ki + derError * Kd;
    if (output > 33) output = 33;
    else if (output < 0) output = 0;

    if (abs(output - lastOutput) < 5) output = lastOutput;
    lastOutput = output;
    Serial.println(error);
    return output;
}
```

Appendix C

(Test Data Plotting Over UART in Matlab)

```
pause on;

% Do in command prompt
% s = serial('COM9');
% fopen(s);

error = zeros(1,1);
sensor = zeros(1,1);
PWM = zeros(1,1);
Time = zeros(1,1);

counter = 1;

End = "0";


while End == "0"
      data = fscanf(s);
      strSplit = split(data, ',');
      error = [error str2double(strSplit(1))];
      sensor = [sensor str2double(strSplit(2))];
      PWM = [PWM str2double(strSplit(3))];
      Time = [Time str2double(strSplit(4))];
      End = strSplit(5);
      disp(data);
end

% Plotting the Data
figure;
plot(Time, error);
title('Test Error');
xlabel('Time (ms)');
ylabel('Error (cm)');

figure;
plot(Time, sensor);
title('Test Sensor');
```

```
xlabel('Time (ms)');
ylabel('Sensor (cm)');

figure;
plot(Time, PWM);
title('Test Output');
xlabel('Time (ms)');
ylabel('Output');
% fclose(s);
```

Appendix D

(Images of Ball Balancer)

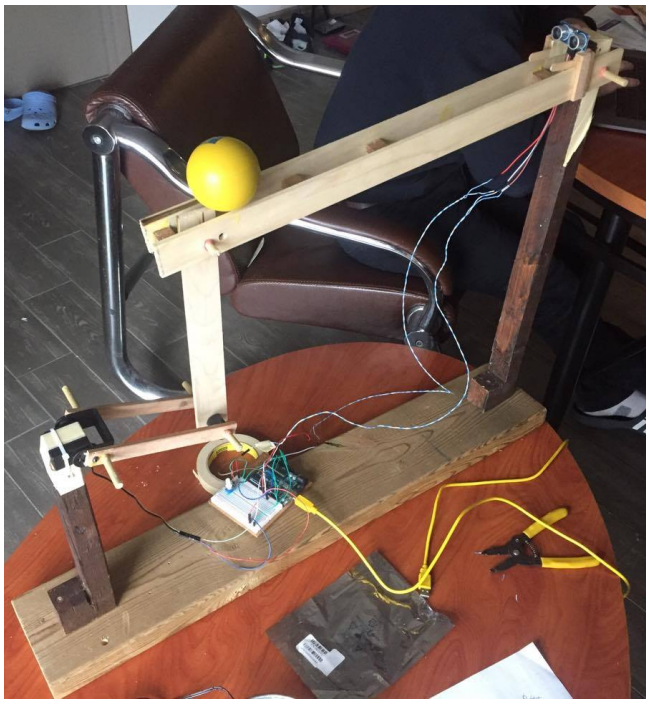

Figure D1: Image of Ball Balancer #1

Figure D2: Image of Ball Balancer #2