Automatic Control Project

Balancing a Ball on a Beam using a PID Controller

Yusuf Borham-8676
Mohamed Salama-8649
Abdullrahman El Bakatooshy-8787

Mechatronics and Robotics Department
Alexandria University,Faculty of Engineering

January 2025

Contents

1 Introduction

1.1 Overview

The project aims to balance a ball on a beam by controlling the beam's angle through a servo motor. The system relies on a **PID controller** for precise position control, with real-time feedback provided by a distance sensor.

1.2 Features

1.2.1 Hardware Components

1. **Arduino Uno:** Serves as the main microcontroller.
2. **VL53L1X Distance Sensor:** Measures the ball's position with high accuracy.
3. **Servo Motor:** Adjusts the beam's angle to balance the ball.

1.3 Control System

Implemented a PID controller for stability and responsiveness. Tuned parameters to achieve minimal overshoot and steady-state error.

1.4 Simulation Tools

Developed a Simulink model to analyze system behavior and validate the control strategy. Wrote MATLAB code for simulation to ensure accuracy before hardware deployment.

2 Control System

2.1 Implementing the Physical Model

By using Newton's law for motion we derived the next equations given

F_{drag} : Drag force (N), C_d = Drag coefficient (dimensionless, typically 0.47 for a sphere in air) ,ρ = Air density (1.225 kg/m^3 at sea level, 20°C) , v = Velocity of the ball (m/s) ,$diameter = 0.04m$, $m = 0.0027kg$,$\mu_k = 0.2$, θ The angle of the beam , α the angle of the servo motor ,X the displacement of the ball

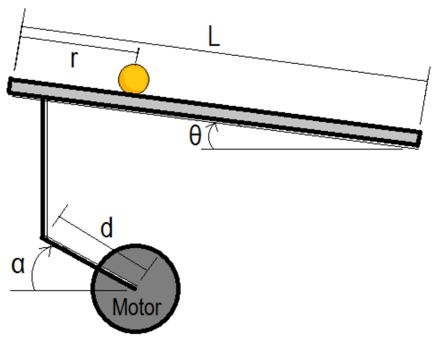

Figure 1: Physical Model

$$\sum Forces = ma \tag{1}$$

$$mg\sin\theta - F_{friction} - F_{drag} = ma \tag{2}$$

$$mg\sin\theta - \mu_k mg\cos\theta - \frac{1}{2}C_d\rho A v^2 = ma \tag{3}$$

$$mg\sin\theta - 0.0052 - 0.00036 = ma \tag{4}$$

Since the values of friction and drag force are negligible, they can be disregarded in the analysis. Including these forces would introduce nonlinearity into the system. Therefore, to ensure the system remains linear and suitable for implementing the PID controller, it was necessary to linearize the system beforehand.

$$mg\sin\theta = ma \tag{5}$$

$$g\sin\theta = a \tag{6}$$

$$g\sin\theta = \frac{d^2}{dt^2}x \tag{7}$$

as $\sin\theta$ will be very small value we could approximate it to θ

$$g\theta = \frac{d^2}{dt^2}x \tag{8}$$

where now θ is the input for the system and x is the output
by taking Laplace

$$g\Theta(S) = S^2 X(S) \tag{9}$$

$$\frac{X(S)}{\Theta(S)} = \frac{g}{S^2} \tag{10}$$

Now we have the transfer function for the system Lets build the Model
Now, we need to establish a relationship between the servo motor angle and the displacement of the ball. The servo motor angle serves as the actual input in this system. Through experimentation, we determined that the angle of the beam is approximately one-seventh of the servo motor angle then

$$\theta = \frac{1}{7}\alpha \tag{11}$$

$$\Theta(S) = \frac{1}{7}\alpha(S) \tag{12}$$

now the new transfer function for the system is

$$\frac{X(S)}{\alpha(S)} = \frac{1}{7}\frac{g}{S^2} \tag{13}$$

2.2 Analysis-for-The-System

2.2.1 Overview

First look at the system By taking a first look of the system it seams that the closed loop poles are always on the imaginary axis for any value of the gain and that means that the system is an oscillatory system and will never arrive a steady state and that will be clear from the system's root locus

2.2.2 Root-Locus

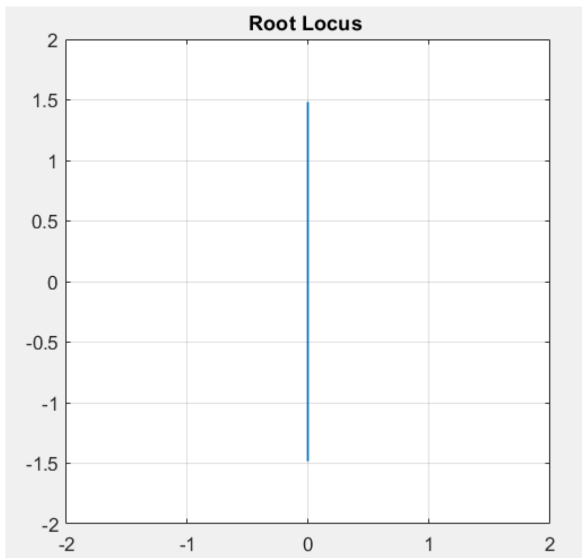

Figure 2: Root locus

Analyzing the root locus It is now obvious that the system is oscillatory But this system is not good as no steady state will be achieved so now we need a more complex controller such as compensator or a PID controller because we need to add some damping to the system and achieve an acceptable settling time we went with PID controller as it is more easy to implement and easy to analyze

2.2.3 Simulink-Response

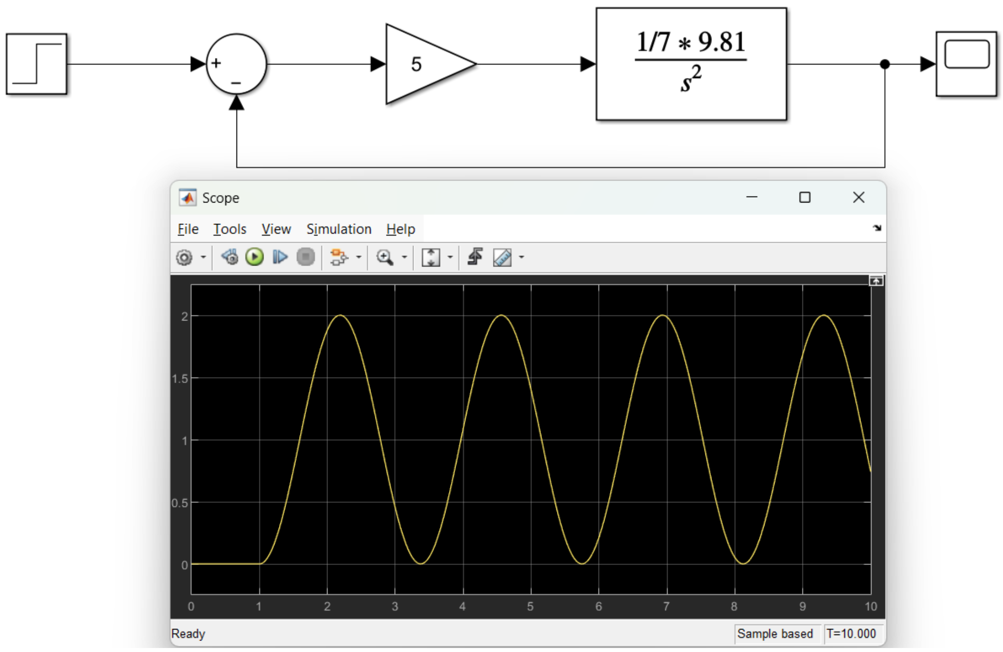

Figure 3: System's Response

2.2.4 Conclusion

Based on the root locus analysis of my system, I observed that it exhibits an oscillatory response for all values of gain. To address this issue and achieve a more stable and controlled response, I decided to implement a PID controller.

2.3 Simulik-Model-with-PID

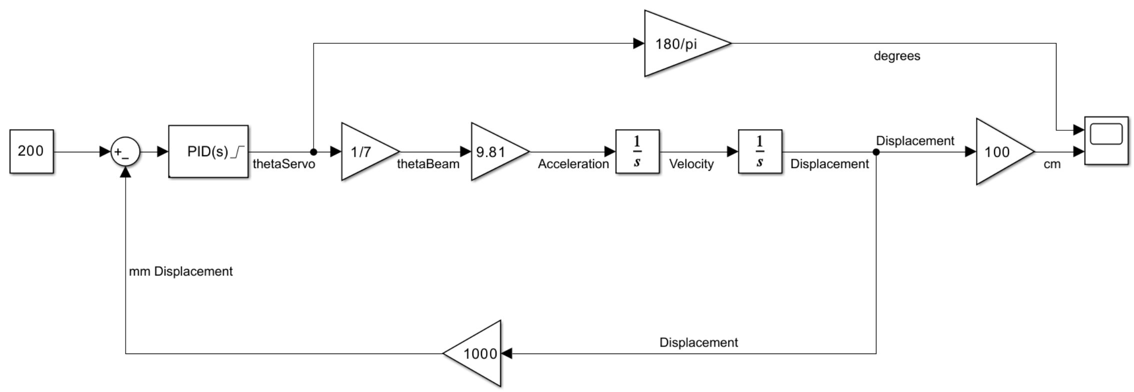

Figure 4: Controller Model

2.4 PID-System

Overview We implemented our own PID system on the Arduino we did not use any pre-made libraries . The input for our system was the desired position of the ball and we used a VL0531X Time of flight sensor to measure the distance and it was our feedback Then we implemented our algorithm using the Equation below where we use the error in order to compute all of the three terms

The PID control equation is given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Differentiating the Signal In order to differentiate we used Euler's differentiation where we save the last error and use it with the sample time to compute the error

Euler's method for numerical differentiation is given by:

$$\frac{dy}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

Integrating the signal For integrating the error we used Euler's Integration Formula (Numerical Integration) which also combining the last error value along side with new one and the sampling time we could numerically integrate the error

Euler's method for numerical integration is given by:

$$y(t + \Delta t) = y(t) + \Delta t \cdot f(t, y(t))$$

By applying the PID formula along with the calculated values above we arrive with output value which we feed to our plant

Plant We have not talk yet about the plant.The plant we are using is just a servo motor which takes an angle as an input so the output of the PID is just an angle which we feed to the servo in order to control the ball's position

Output Limits We encountered a limitation with the servo motor's angular range, which is restricted to values between -90° and 90°. If the output of the PID controller exceeds this range, it becomes necessary to constrain the controller's output to the same limits to protect both the servo motor and the overall system. However, implementing this constraint introduced a secondary issue.

Integral Saturation Problem When the controller's output is clamped at a specific limit (e.g., 90°), the integral term of the PID controller continues to accumulate error, even though the system is already saturated at its maximum value. This accumulation causes the effective output to increase beyond the servo's operational range, despite the physical system remaining at its upper limit (90°). Consequently, when the output eventually starts decreasing, the accumulated error in the integral term delays the system's response, preventing it from decreasing immediately.

Solving the Problem To address this issue, we modified the controller to halt the integration process whenever the output is constrained by the limits. This ensures that the integral term does not accumulate error unnecessarily, improving the controller's performance and response under these conditions.

Differentiating Problem A common issue in digital differentiation is the amplification of noise from the sensor, which can result in incorrect values being produced by the D-Controller. To address this, a digital low-pass filter, such as one designed using the Butterworth method, is typically applied. However, we have not implemented such a filter in this case to avoid adding complexity to the system. Nevertheless, it is important to acknowledge this potential problem.

2.5 Tuning-PID

Tuning Techniques In order to tune the PID constants PID tuner app was used first it did linearize the plant the it threw some values at first the did not work well but we just tuned them a bit in the code and we got very good results

Ziegler-Nichols Method Tuning may have done with other ways one way one way is to use Ziegler-Nichols method but it requires knowing the actual transfer function this may be done using the system identification tool on Matlab

PID Tuner App What we actually done is we used the values from the PID Tuner app and started trying them on our system and after further tuning we arrived to the perfect constants

2.6 PID-OUTPUT

RED -angle , Blue - position

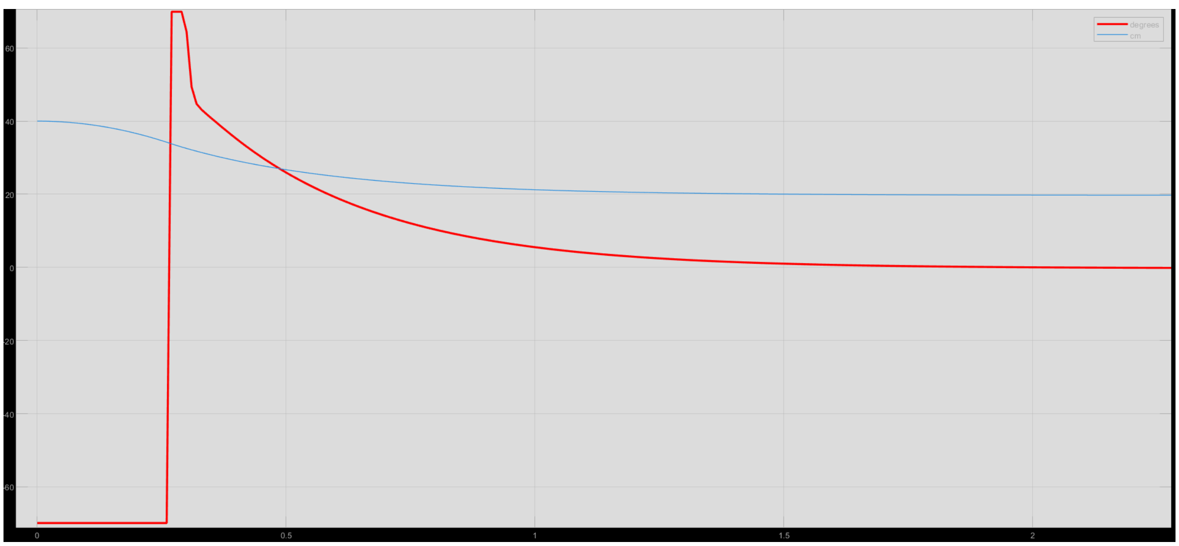

Figure 5: PID OUTPUT

2.7 System-Response

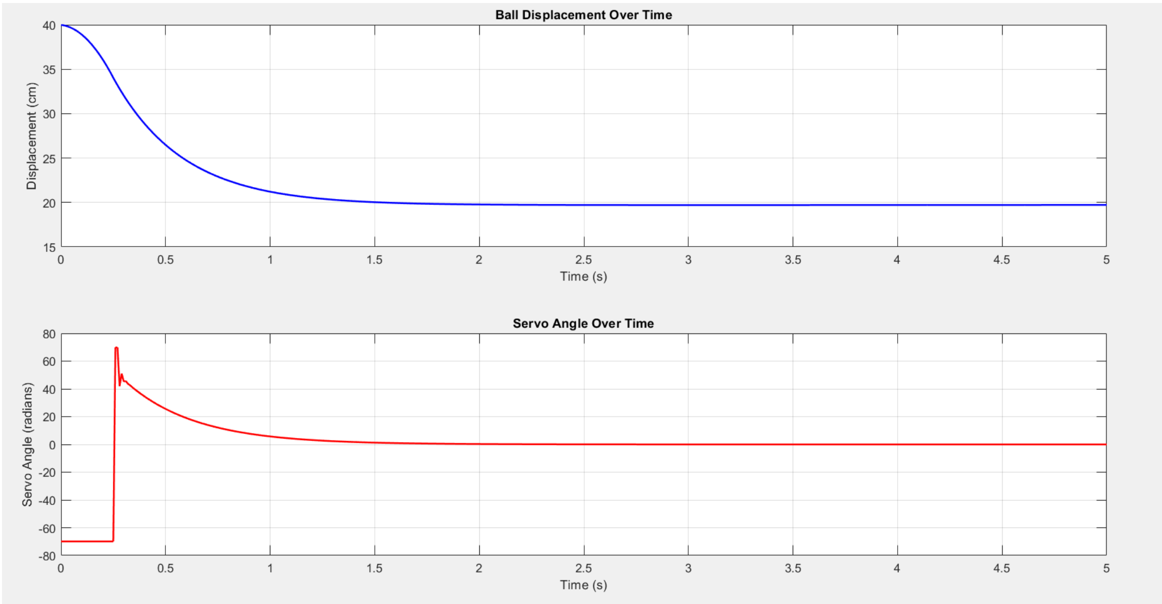

Figure 6: Schematic

3 Hardware-System

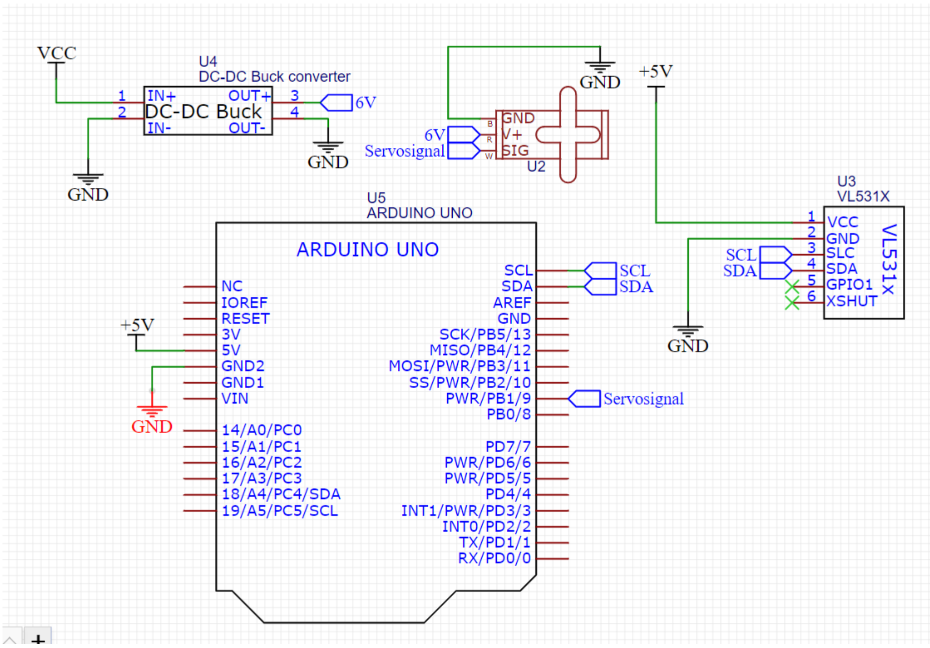

Figure 7: Schematic

The hardware system is very simple it's just the arduino the code implemented on along side with the servo motor and the VLX sensor also we used a buck converter to convert the 24 volts from the adapter to the six volts the servo needs and here is the schematic

4 Software

4.1 Matlab-Code

```
% Define Parameters
setpoint = 200; % Target displacement in mm
kp = 0.3; % Proportional gain
ki = 0.01; % Integral gain
kd = 0.1; % Derivative gain

thetaBeamFactor = 1/7; % Conversion from servo angle to beam angle
g = 9.81; % Acceleration due to gravity in m/s^2
dt = 0.01; % Time step for simulation (s)
simulationTime =5; % Total simulation time (s)

servoMin = -1.22; % Minimum servo angle in radians
servoMax = 1.22;  % Maximum servo angle in radians

% Initialize Variables
thetaServo = 0; % Servo angle in radians
thetaBeam = 0; % Beam angle in radians
acceleration = 0; % Ball acceleration (m/s^2)
velocity = 0; % Ball velocity (m/s)
displacement = 0.4; % Ball displacement (m)

integralError = 0; % For PID integral term
previousError = 0; % For PID derivative term

% Time vector for simulation
time = 0:dt:simulationTime;
displacementHistory = zeros(size(time)); % Store displacement for plotting
servoHistory = zeros(size(time)) ;

% Simulation Loop
for i = 1:length(time)
    % Calculate error
    error = setpoint - displacement * 1000; % Convert displacement to mm

    % PID Controller
    derivativeError = (error - previousError) / dt; % Derivative term

```

```
    % Update integral term only if within servo limits
    if servoMin <= thetaServo && thetaServo <= servoMax
         integralError = integralError + error * dt; % Integral
            term
    end

    % PID output (Servo angle)
    thetaServo = kp * error + ki * integralError + kd *
       derivativeError;

    % Apply servo angle limits
    thetaServo = max(servoMin, min(servoMax, thetaServo)); %
       Clamp to limits

    servoHistory(i) = thetaServo * (180/pi) ; % convert it to
       degrees
    % Update previous error
    previousError = error;

    % System Dynamics
    thetaBeam = thetaServo * thetaBeamFactor; % Beam angle
    acceleration = g * sin(thetaBeam); % Acceleration of the
       ball
    velocity = velocity + acceleration * dt; % Update velocity
    displacement = displacement + velocity * dt; % Update
       displacement

    % Store displacement for plotting
    displacementHistory(i) = displacement * 100; % Convert to
       cm for output
end

% Plot Results
subplot(2, 1, 1);
plot(time, displacementHistory, 'b', 'LineWidth', 1.5);
title('Ball Displacement Over Time');
xlabel('Time (s)');
ylabel('Displacement (cm)');
grid on;

subplot(2, 1, 2);
plot(time, servoHistory, 'r', 'LineWidth', 1.5);
title('Servo Angle Over Time');
xlabel('Time (s)');
ylabel('Servo Angle (radians)');
```

```
grid on;
```

4.2 Arduino-Code

```
#include <Arduino.h>
#include <Servo.h>
#include "Adafruit_VL53L1X.h"

#define TRIG_PIN 2
#define ECHO_PIN 3
#define IRQ_PIN 2
#define XSHUT_PIN 3

// create servo object to control a servo
Servo servo;
// create a VL53L1X object
Adafruit_VL53L1X vl53 = Adafruit_VL53L1X(XSHUT_PIN, IRQ_PIN);

// declare variables
float input = 0, error = 0, kp = 0.3, ki = 0.01, kd = 0.15,
    setpoint = 195, output = 0;
float maxOutput = 70, minOutput = -60;
float distance = 0;

// PID controller
void PID() {
  // declare variables
  double derror;
  static float ierror = 0;
  static float prvError;
  double dt = 0;
  static unsigned long prvMillis = 0;

  // wait till the data is ready
  if (vl53.dataReady()) {
    // new measurement for the taking!
    distance = vl53.distance();
    // check if the distance is valid
    if (distance >= 0 && distance < 500) {
      input = distance;
    }
    // data is read out, time for another reading!
    vl53.clearInterrupt();
    // Calculate the delta time
```

```
    dt = (millis() - prvMillis) / 1000.0;
    prvMillis = millis();
    // Calculate the error
    error = setpoint - input;
    // Calculate the derivative of the error
    derror = (error - prvError) / dt;
    // Update the previous error
    prvError = error;
    // Calculate the integral of the error
    // Check if the output is within the limits
    if (output < maxOutput && output > minOutput) {
      ierror += error * dt;
    }

    // Calculate the output

    output = kp * error + ki * ierror + kd * derror;

    // constarin the output
    if (output > maxOutput) output = maxOutput;
    if (output < minOutput) output = minOutput;

    Serial.print("Distance is   =  ");
    Serial.print(input);
    Serial.print(" Output of the PID is   =  ");
    Serial.println(output);
  }
}


void setup() {
  // initialize serial communication
  Serial.begin(115200);
  // initialize servo
  servo.attach(9);
  // initialize the VL53L1X sensor
  while (!Serial) delay(10);

  Serial.println(F("Adafruit VL53L1X sensor demo"));

  Wire.begin();
  if (!vl53.begin(0x29, &Wire)) {
    Serial.print(F("Error on init of VL sensor: "));
    Serial.println(vl53.vl_status);
    while (1) delay(10);
```

```
  }
  Serial.println(F("VL53L1X sensor OK!"));

  Serial.print(F("Sensor ID: 0x"));
  Serial.println(vl53.sensorID(), HEX);

  if (!vl53.startRanging()) {
    Serial.print(F("Couldn't start ranging: "));
    Serial.println(vl53.vl_status);
    while (1) delay(10);
  }
  Serial.println(F("Ranging started"));

  // Valid timing budgets: 15, 20, 33, 50, 100, 200 and 500ms!
  vl53.setTimingBudget(15);
  Serial.print(F("Timing budget (ms): "));
  Serial.println(vl53.getTimingBudget());
  // wait for serial port to open on native usb devices
}

void loop() {
  PID();
  servo.write(110 + output);
}
```

References

References

[1] GitHub Repo, *A ball on a beam*, Available at: `https://github.com/yusufborham/Balancing-a-ball-on-a-beam`, Accessed: January 1, 2025.