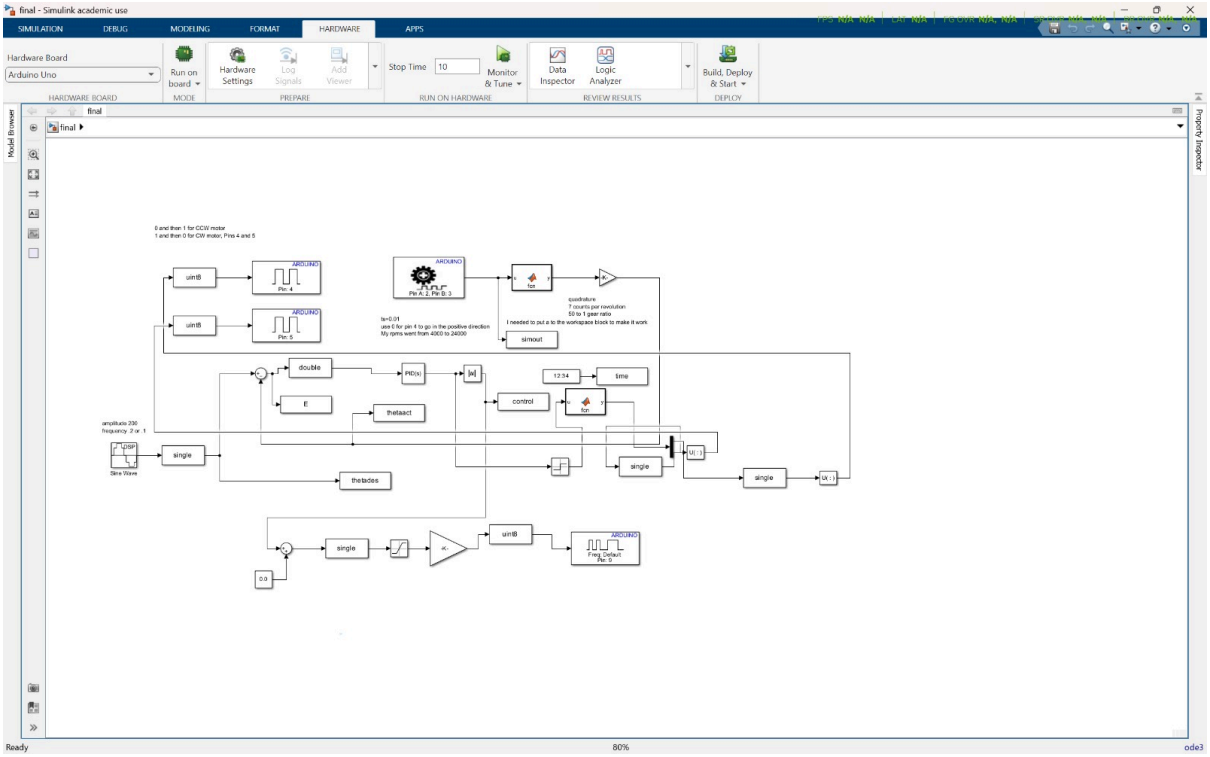
Closed-Loop Motor Control System

Anusha Chatterjee, ASUID: 1234397790

SCOPE: The goal of this project is to develop and validate a closed-loop motor position control system on an Arduino-based platform that can accurately track a desired reference signal. By designing Simulink models and implementing P, PD, and PID controllers for a DC motor with quadrature encoder feedback, the project aims to systematically tune and compare the three control schemes in terms of stability, steady-state error, and dynamic response. Ultimately, the objective is to show how adding derivative and integral actions improves tracking performance and reduces error, resulting in a more precise and robust motion control system.

Simulink Model:



Simulink Model Description

The overall Simulink model implements a closed-loop position control system for a DC motor using an Arduino board and a quadrature encoder. The main functional blocks and their sub-blocks are:

1. Direction Control Block (Digital Outputs)

This block sets the direction of rotation of the DC motor using two digital output pins on the Arduino:

- **Digital Output (Pin 4)**
- **Digital Output (Pin 5)**
- **Logic / Constant sub-blocks** generating `0` or `1` (in `uint8` format)

By driving complementary logic levels on pins 4 and 5 (e.g., `[1,0]` for clockwise and `[0,1]` for counter-clockwise), this block determines the rotation direction while the speed is controlled separately by PWM. The logic sub-blocks ensure that the signals are in a format compatible with the Arduino digital pins.

2. Encoder Feedback Block

This block reads the motor shaft position from the quadrature encoder:

- **Encoder block (Pins 2 and 3)** to read channel A and channel B
- **Function (`Fcn`) sub-block** to convert encoder counts to angle or position
- **Gain sub-block (`-K`)** to scale and/or adjust sign

The encoder block outputs counts as an `int32` signal. The function sub-block applies the encoder resolution and gear ratio to convert counts into a meaningful physical quantity (e.g., degrees or radians). The gain sub-block may invert or scale the signal so that the measured position aligns with the chosen sign convention for the controller.

3. Error Computation Block

The error block compares the desired position to the actual position:

- **Desired position (`\theta_{des}`)** from the reference generator
- **Actual position (`\theta_{act}`)** from the encoder path
- **Subtract block** computing
  [e(t) = \theta_{des}(t) - \theta_{act}(t)]

This error signal is the primary input to the P, PD, or PID controller and represents how far the motor is from the target position at each instant.

4. PID Controller Block

This block generates the control effort based on the tracking error:

- **PID Controller block** (configured as P, PD, or full PID)
- **Absolute value sub-block `|u|`** to ensure a non-negative magnitude before PWM

By adjusting the proportional, integral, and derivative gains, this block shapes the transient and steady-state response. For the different experiments:

- **P control:** only the proportional gain is non-zero.
- **PD control:** proportional and derivative gains are non-zero.
- **PID control:** proportional, integral, and derivative gains are all active.

The absolute value sub-block outputs the magnitude of the control effort, while the direction information is handled separately by the direction control block.

5. Reference Generator Block

This block defines the target position for the motor:

- **Sine Wave sub-block** to generate a time-varying reference (e.g., sinusoidal position)
- **Gain sub-block(s)** to adjust amplitude, offset, and sign

The sine wave acts as the reference trajectory that the motor must follow. The gain sub-blocks scale the signal into a feasible range for the motor and align the units with the feedback signal (e.g., counts or degrees). The resulting signal is used as the desired position `\theta_{des}`.

6. PWM Output and Actuation Block

This block converts the control effort into a PWM signal for the motor driver:

- **Arduino PWM Output block (Pin 9)**

The control signal from the PID block is mapped to a PWM duty cycle (typically between 0 and 255 for Arduino). This PWM signal controls the effective voltage applied to the motor through the driver, thereby controlling the torque and speed. Together with the direction signals, this realizes the full actuation of the motor.

Operation of P, PD, and PID Controllers

In this project, the same closed-loop Simulink model is used to evaluate three different control strategies: P, PD, and PID control. The plant (DC motor + driver + encoder) and feedback structure remain identical; only the controller gains are changed between experiments. For each case, the motor is commanded to track a reference position signal (such as a sine wave), and the resulting response is recorded for comparison in terms of rise time, overshoot, settling time, and steady-state error.

1. Proportional (P) Control

In the P-control experiment, only the proportional gain (K_p) is enabled, while the integral and derivative gains are set to zero. The controller output is directly proportional to the instantaneous position error:

[u(t) = K_p,e(t)]

where (e(t)) is the difference between the desired and actual motor position. The proportional gain is increased gradually until a compromise between response speed and stability is reached. Typically, a higher (K_p) reduces the tracking error and speeds up the response, but too large a value can cause overshoot and oscillation. With P control alone, a non-zero steady-state error is generally expected, especially for constant or slowly varying reference signals.

2. Proportional–Derivative (PD) Control

In the PD-control experiment, both the proportional gain (K_p) and derivative gain (K_d) are used, while the integral gain is kept at zero. The controller now reacts to both the magnitude of the error and the rate at which the error is changing:

$$u(t) = K_p\, e(t) + K_d\, \frac{de(t)}{dt}$$

Starting from a stable P controller, a derivative term is added and tuned to improve damping. The derivative action predicts the future behavior of the error and counteracts rapid changes, which helps reduce overshoot and oscillations. As a result, PD control generally provides a faster and more stable transient response compared to pure P control, but it still may exhibit some steady-state error because there is no integral action.

3. Proportional–Integral–Derivative (PID) Control

In the PID-control experiment, all three gains—proportional (K_p), integral (K_i), and derivative (K_d)—are activated:

$$u(t) = K_p\, e(t) + K_i \int e(t)\, dt + K_d\, \frac{de(t)}{dt}$$

The integral term accumulates the error over time and forces it toward zero, eliminating the steady-state offset that remains in P and PD control. Tuning begins from the previously obtained PD gains and then gradually increases (K_i) until the steady-state error is acceptably small, while monitoring for excessive overshoot or slow settling time. The final PID controller aims to balance fast transient response, low overshoot, and minimal steady-state error, providing the best overall tracking performance among the three schemes.

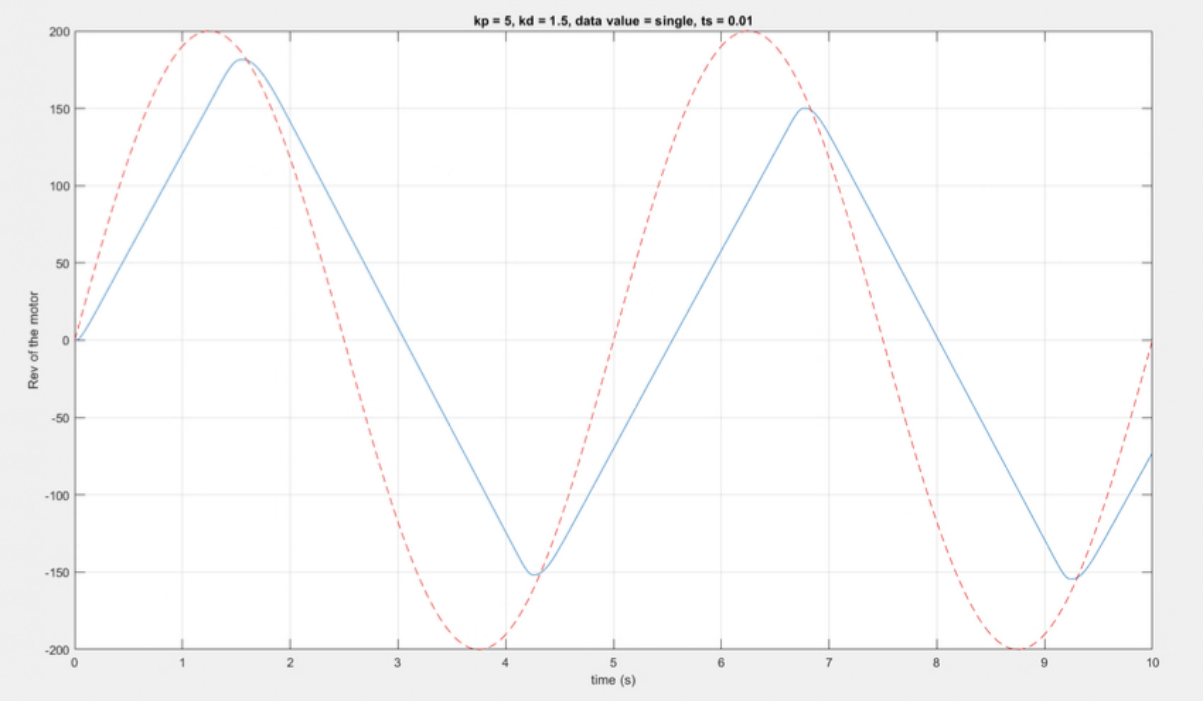
4. Comparison of Results

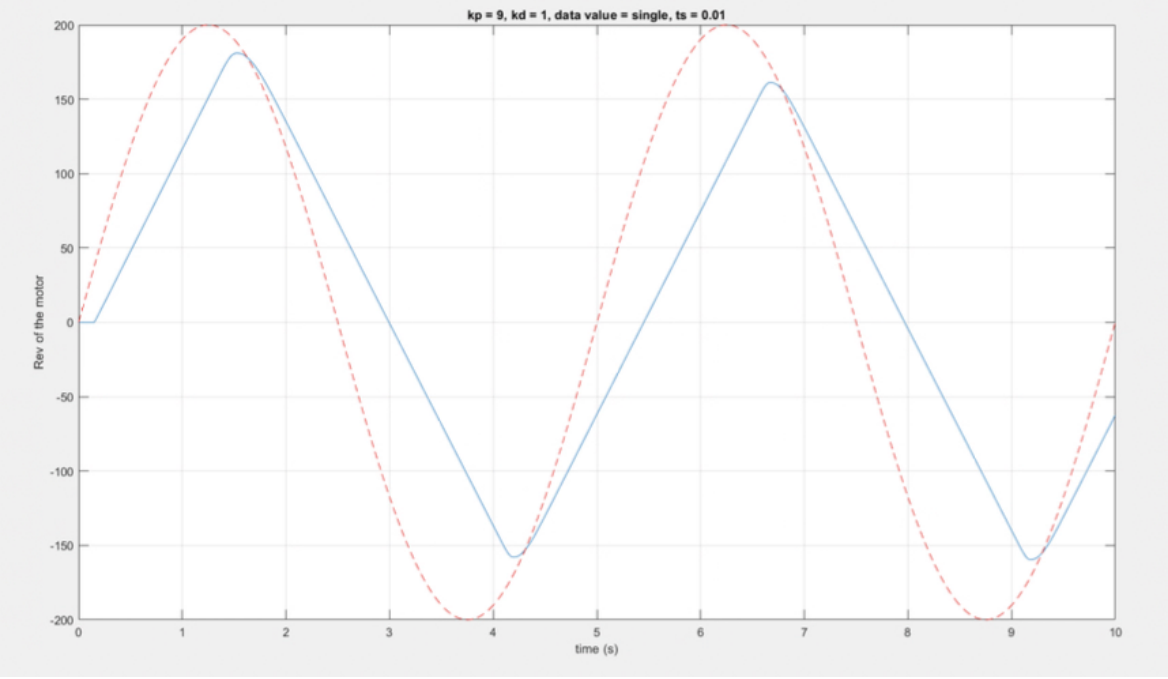
For each control mode (P, PD, PID), the motor's response to the same reference signal is plotted and analyzed. Key performance metrics such as steady-state error, rise time,

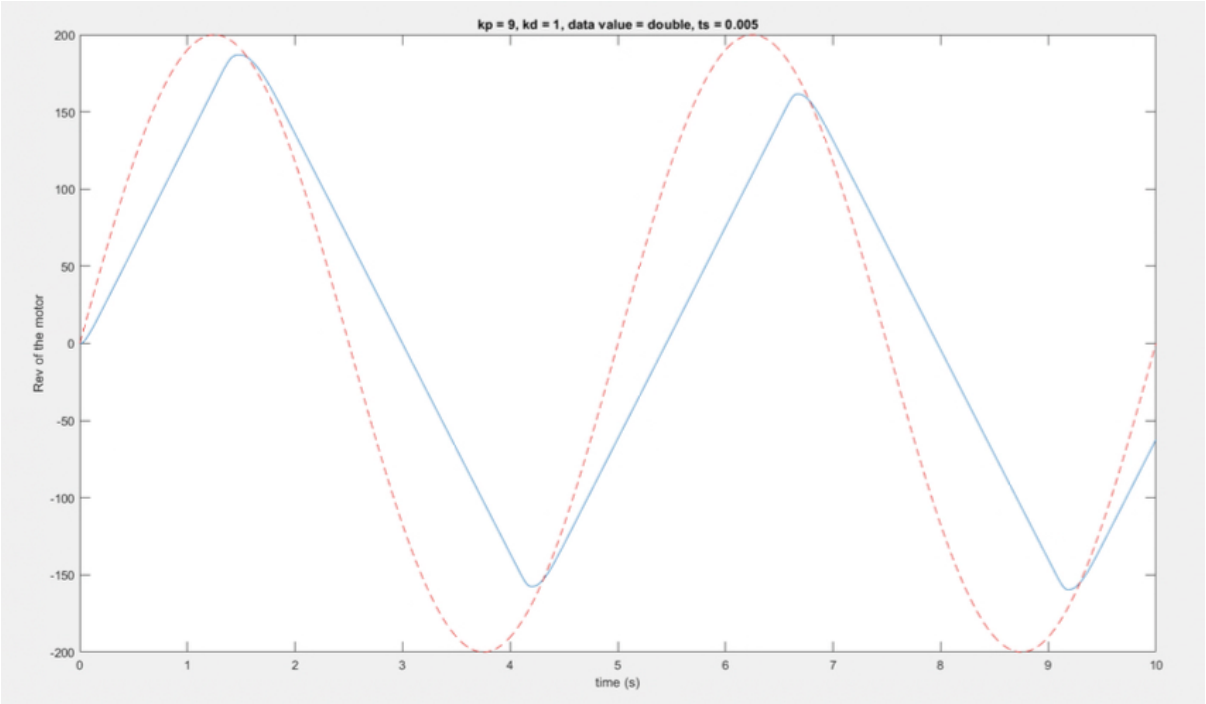
overshoot, and settling time are measured. By comparing these results, the project demonstrates how adding derivative and integral actions to a basic proportional controller progressively improves stability and accuracy, and highlights the trade-offs involved in tuning real-world motor control systems.

Results:

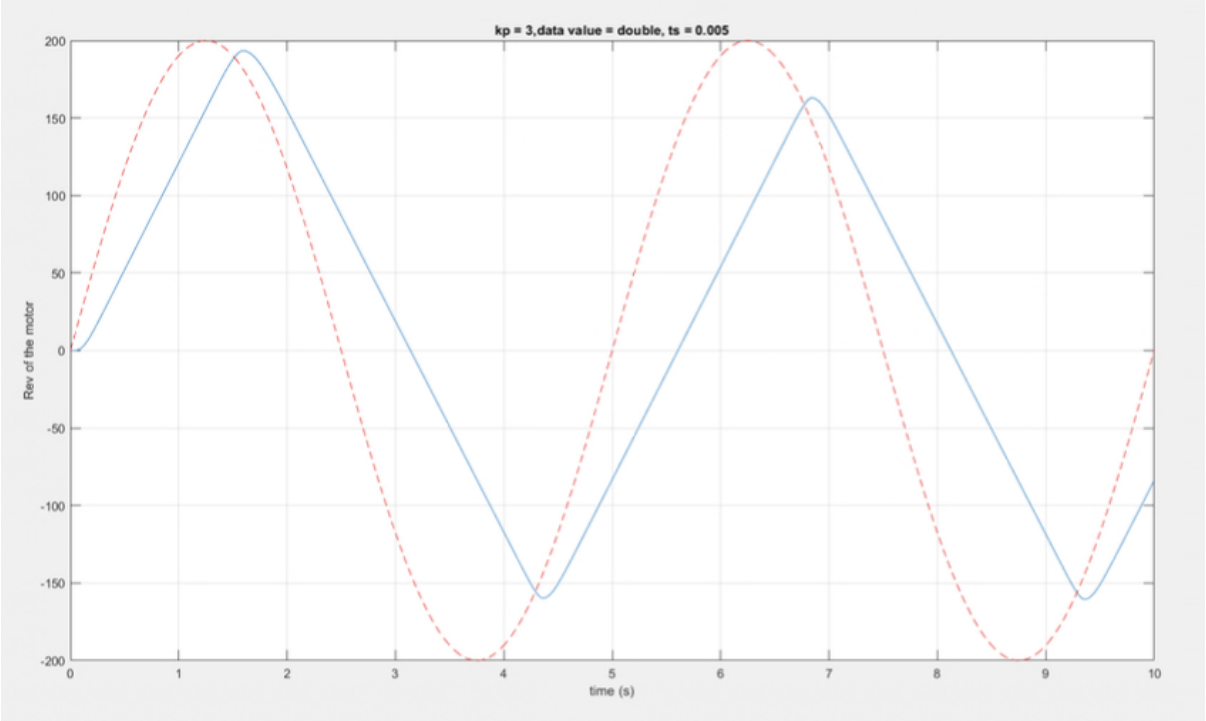
Diagrams with pd control:

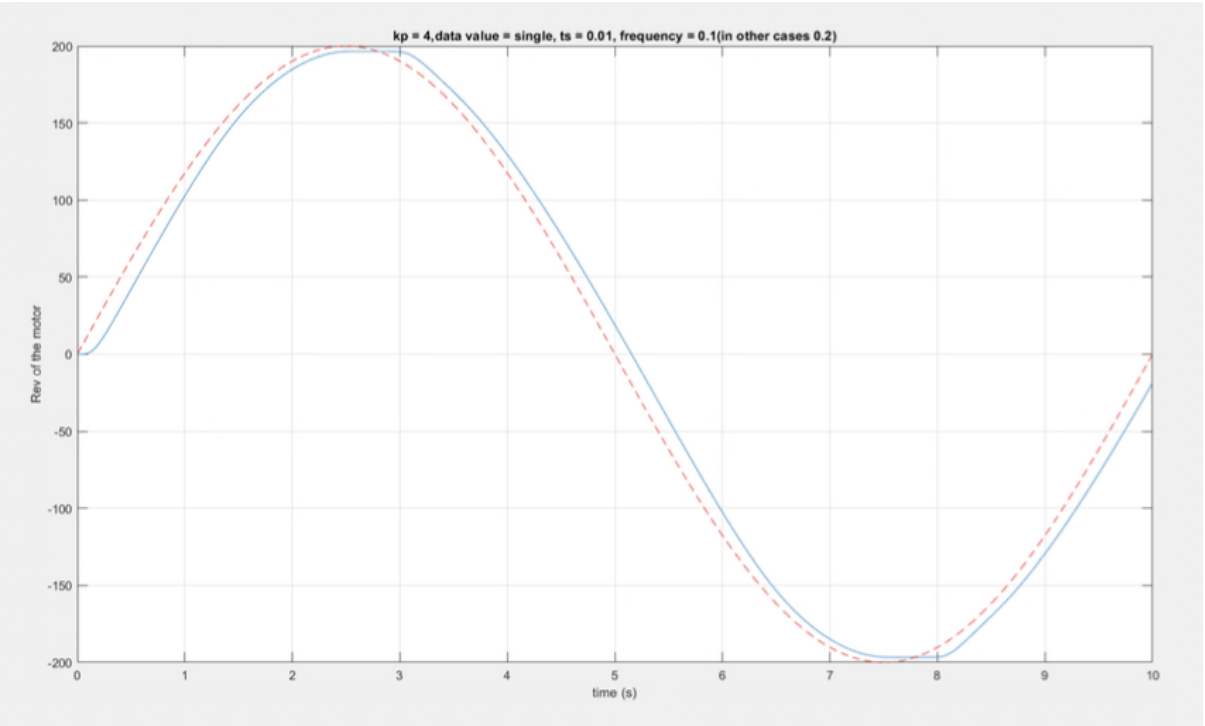


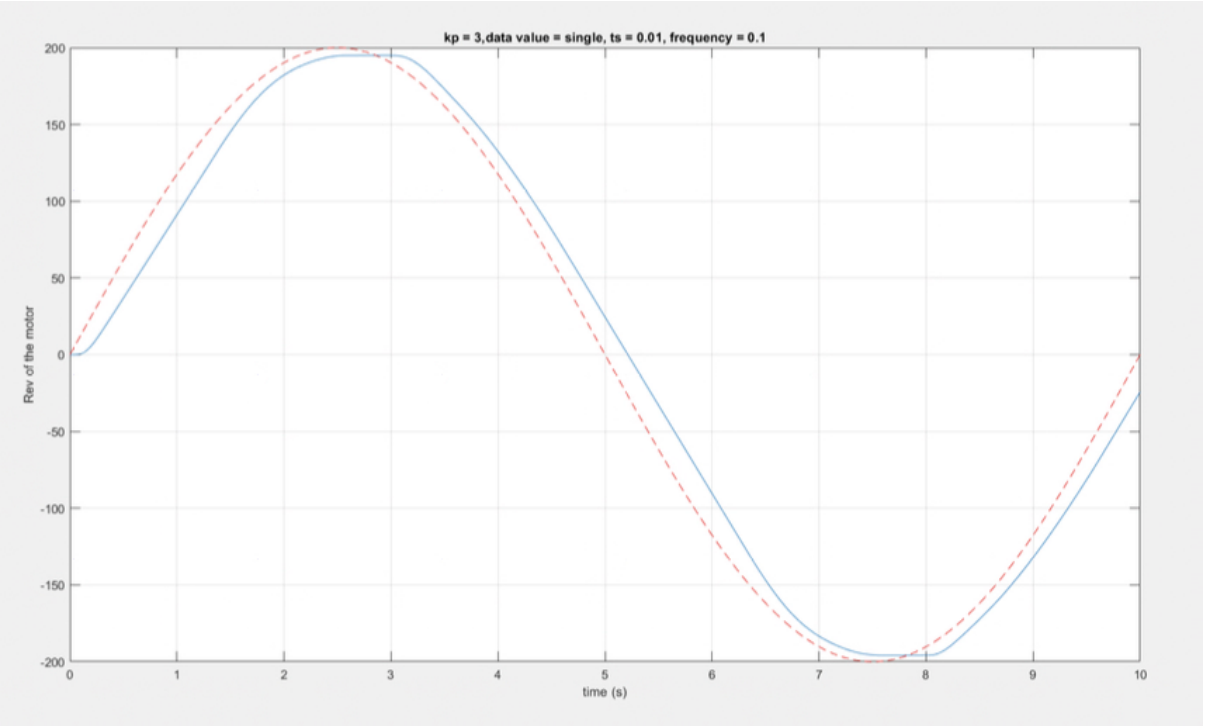




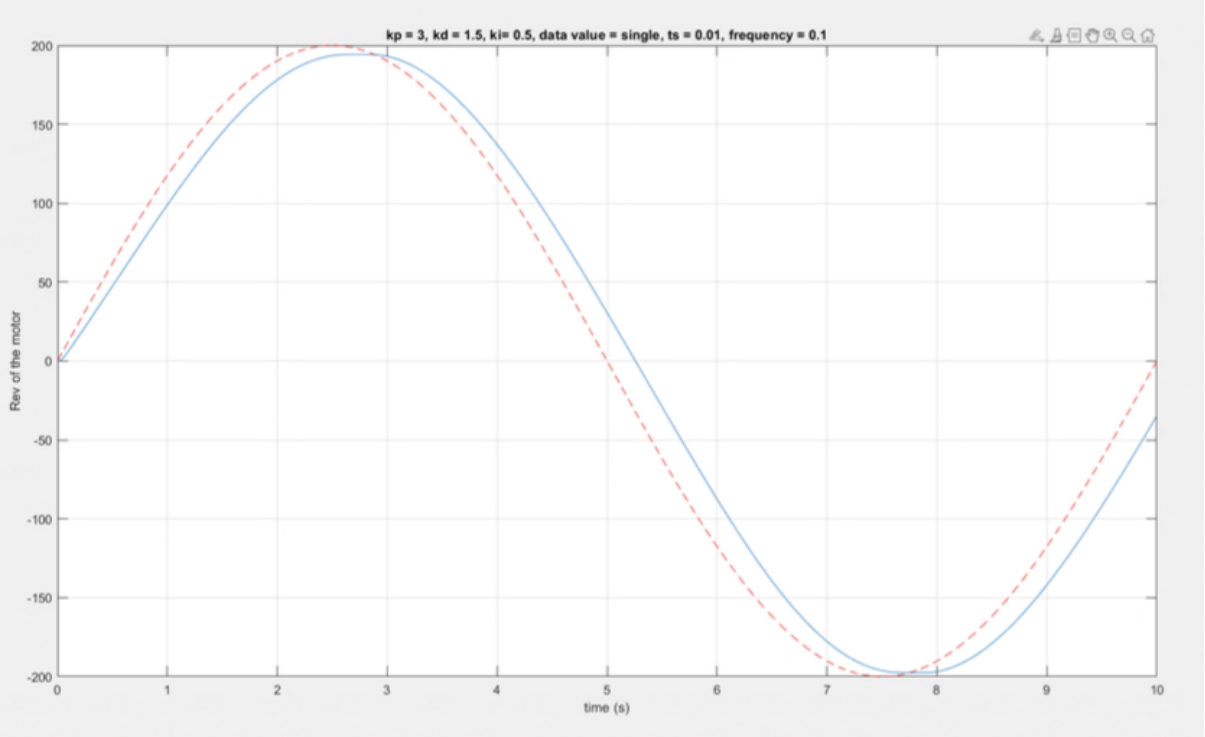
With P control

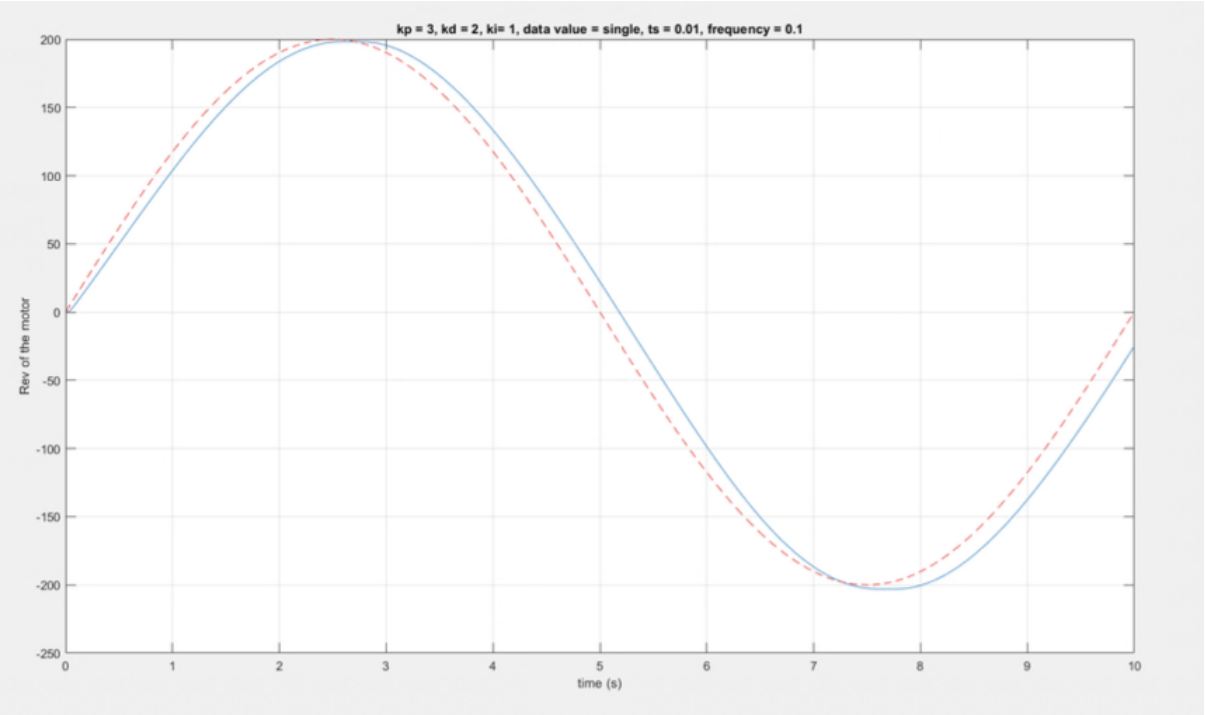



kp = 4,data value = single, ts = 0.01, frequency = 0.1(in other cases 0.2)
Rev of the motor
time (s)
200
150
100
50
0
-50
-100
-150
-200
0
1
2
3
4
5
6
7
8
9
10

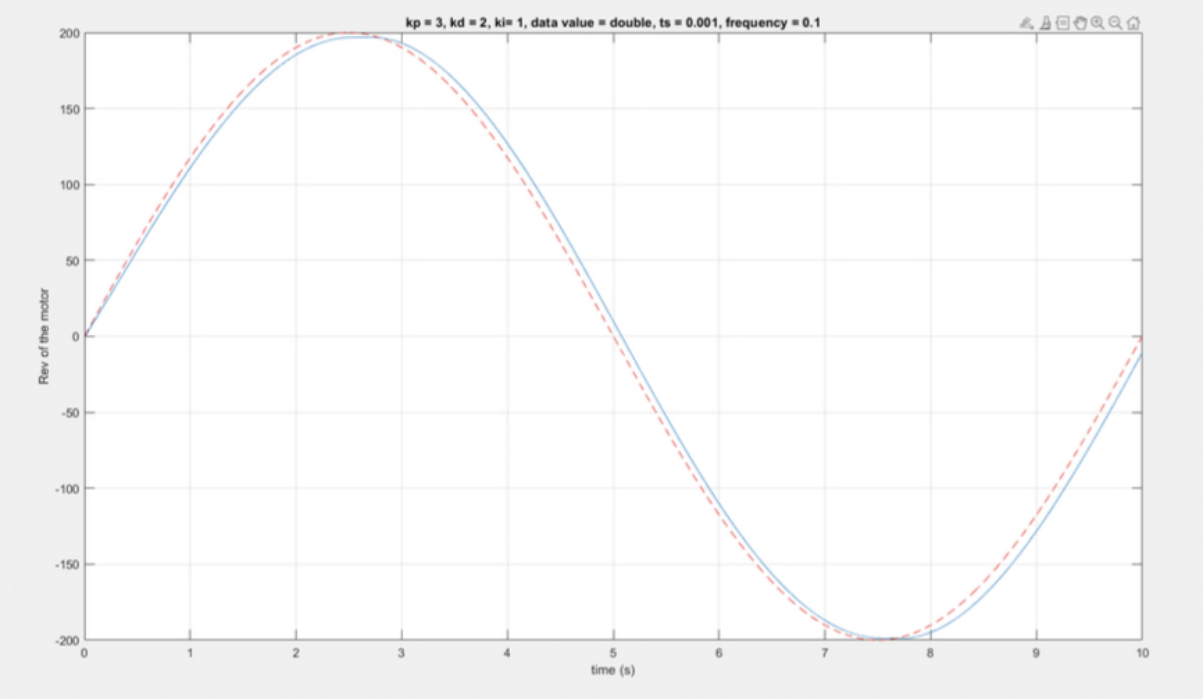

kp = 3,data value = single, ts = 0.01, frequency = 0.1
Rev of the motor
time (s)
200
150
100
50
0
-50
-100
-150
-200
0
1
2
3
4
5
6
7
8
9
10

With PID Control





There was a slight overshoot into the negative region.



Conclusion

In this project, a closed-loop motor position control system was designed and implemented using Simulink and an Arduino microcontroller to drive a DC motor with quadrature encoder feedback. The system was tested under three control configurations: P, PD, and PID. For each case, the motor was commanded to track a reference position signal, and the resulting responses were recorded and analyzed.

The experimental results show that **P control** is simple to implement and can achieve basic tracking, but it typically leaves a noticeable steady-state error and may cause oscillations when the proportional gain is increased too much. **PD control** significantly improves the transient response by reducing overshoot and oscillations through derivative damping, but still cannot completely remove steady-state error. **PID control** provides the best overall performance: by adding integral action, the steady-state error is driven close to zero while maintaining acceptable rise time and stability when properly tuned. Overall, the project successfully demonstrates how the addition of derivative and integral terms to a proportional controller can enhance tracking accuracy and robustness in a practical motor control system.

Google Drive link of the video:

https://drive.google.com/file/d/1rqqiYZKDZaguEpFDiF7OSoR5UjCef8WX/view?usp=sharing