Northeastern University

ME 5245 Mechatronics

Grant Maziarz and Kriish Hate

DC Motor Control System Part C

Spring 2025

Project Overview:

This section of the project covers the use of a closed loop system to meet two control objectives separately. Control objective 1 is for position control in Simulink using a Real Time Target which was undertaken by Grant Maziarz and control objective 2 is for speed control in Simulink using a real time target undertaken by Kriish Hate. This is an expansion of the prior work where an open loop DC motor modeling was undertaken with the same values being used for the purposes of modeling the motors.

Motor Modeling Theory:

The transfer function for a DC motor relating input voltage (V) to output angular velocity (ω) is given by:

$$\frac{w(s)}{V(s)} = \frac{k_t}{(Ls+R)(Js+b)+k_b k_t}$$

With our assumptions of L = 0 and $k_t = k_b$ the transfer function simplifies to:

$$TF = \frac{k_b}{RJs+Rb+k_b^2}$$

The system then simplifies to a standard first-order system of the form:

$$TF = \frac{K}{\tau s+1} = \frac{\frac{k_b}{Rb+k_b^2}}{(\frac{RJ}{Rb+k_b^2})+1}$$

Where:

- K is the DC gain (rad/s per V)
- τ is the time constant (s)

The b value was calculated with the following formula:

$$b = (\frac{1}{R})(\frac{k_b}{K} - k_b^2)$$

Where:

- K is the DC gain (rad/s per V)
- R is inductance resistance Ω
- k_b is back EMF constant V*sec
- b viscous friction coefficient

Given:
J = 0.00976 kg*m²
R = 5 Ohm
W no load = 35 rpm
Vin = 6V

Calculated:
Ts = 1.342388 Nm
Kt = 1.118657 Nm * A
b = 0.0000246
Kb = 0.009529205843 V * s

Closed Loop Control Objective 1:

Introduction:

The goal of this section of the project is to make improvements on the prior modeling and hardware control of the DC Motor specifically to allow for position control of the motor. This change will be done by making use of a closed loop system with the reference position coming from the potentiometer on the hardware system. The potentiometer was mapped from 0 to 360 degrees and must allow for forward and backward motion of the motor. This reference signal is compared to the encoder readings with the difference between them being sent through a proportional controller that has a gain that was found through testing. The resulting position signals will be then compared between the hardware and Simscape model to see differences between the control systems.

Hardware Experimental Results:

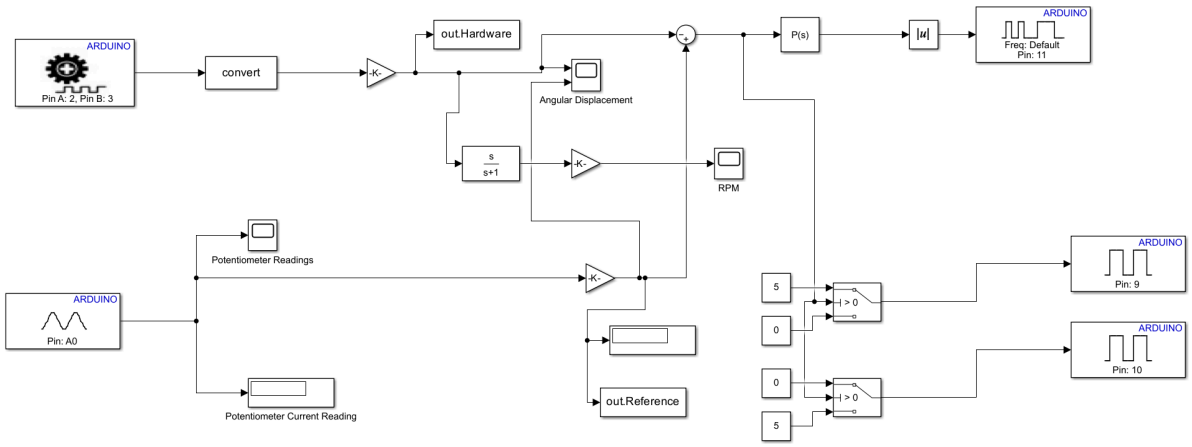

Figure 1: *Hardware Model*

The Simscape model for the hardware model located in Figure 1 above showcases that from the potentiometer readings from analog pin 0 has to go through a gain block before being compared to the encoder readings with the gain being 360/1018. This maps to the full 360 degrees of motion since 1018 is the max potentiometer reading the hardware system allows in the current set up. These readings are then saved to a variable block that allows for the same readings to be used for the Simscape model. There is an absolute value block to make it usable for the arduino so the PWM signal is always positive. And to allow for the back and forth motion of the system two switch blocks are used to control the H-bridge thus allowing for when the error is positive pin 9 is powered allowing the motor to go forward and when the error is negative pin 10 is powered so the motor goes backward. Finally the proportional controller gain was found by testing multiple gains ranging from 0.7 to 7 and seeing the results from there. In the figures below we can see multiple different graphs that showcase this. With the results of a test with a Kp within the controller of 0.7 being shown in Figure 2 below. This results in values that do not get close to the reference value and thus a higher Kp needed to be used. This Kp was increased to 3.5 which is shown in Figure 3 below which gets closer to the reference value but there are still some issues for smaller changes in the reference value. The Kp was then doubled to be a value of 7 with results in Figure 4 below. Through this it is seen that it much closer resembles the reference signal with some differences between it. Thus the chosen Kp value for the hardware system was 7 and the data collected from Figure 5 was under those conditions.

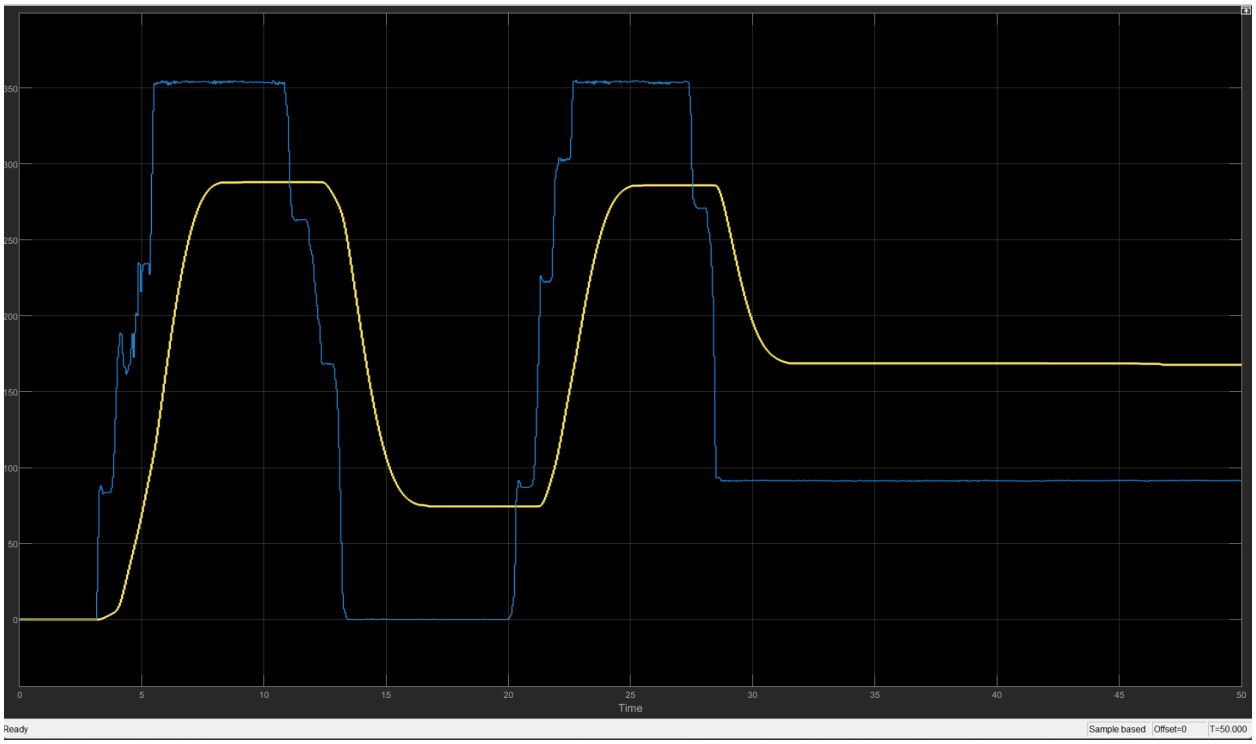

Figure 2: *Position with Kp of 0.7. Blue represents the reference value and yellow is encoder readings.*

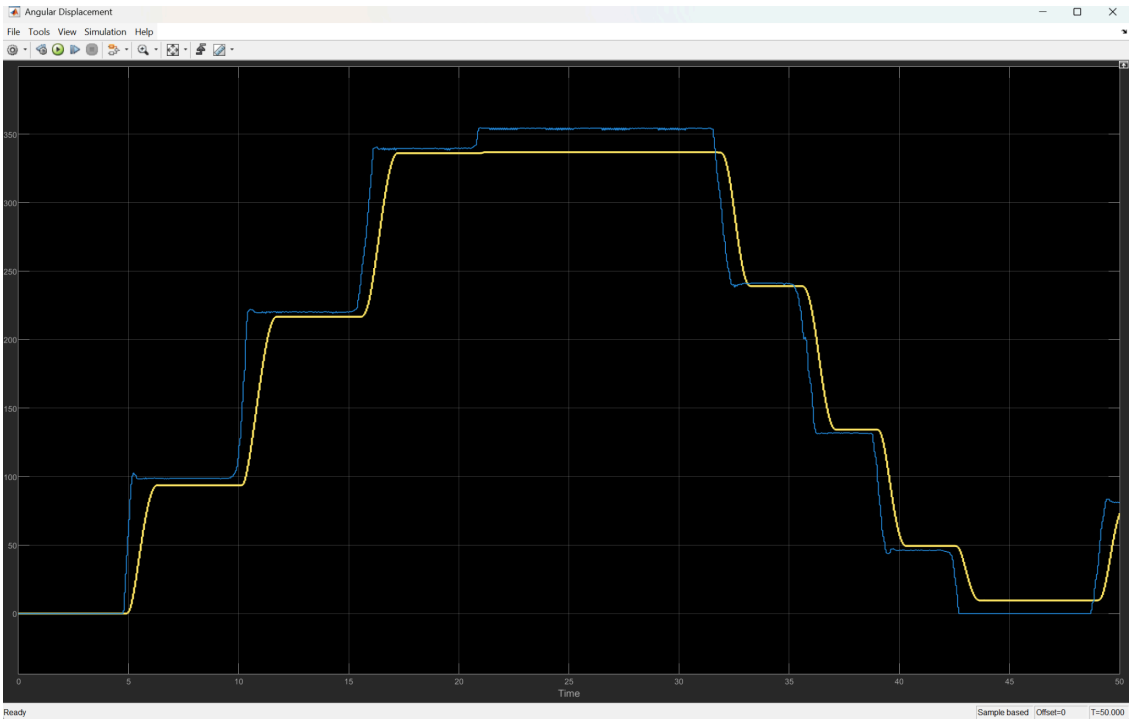

Figure 3: *Position with Kp of 3.5. Blue represents the reference value and yellow is encoder readings.*

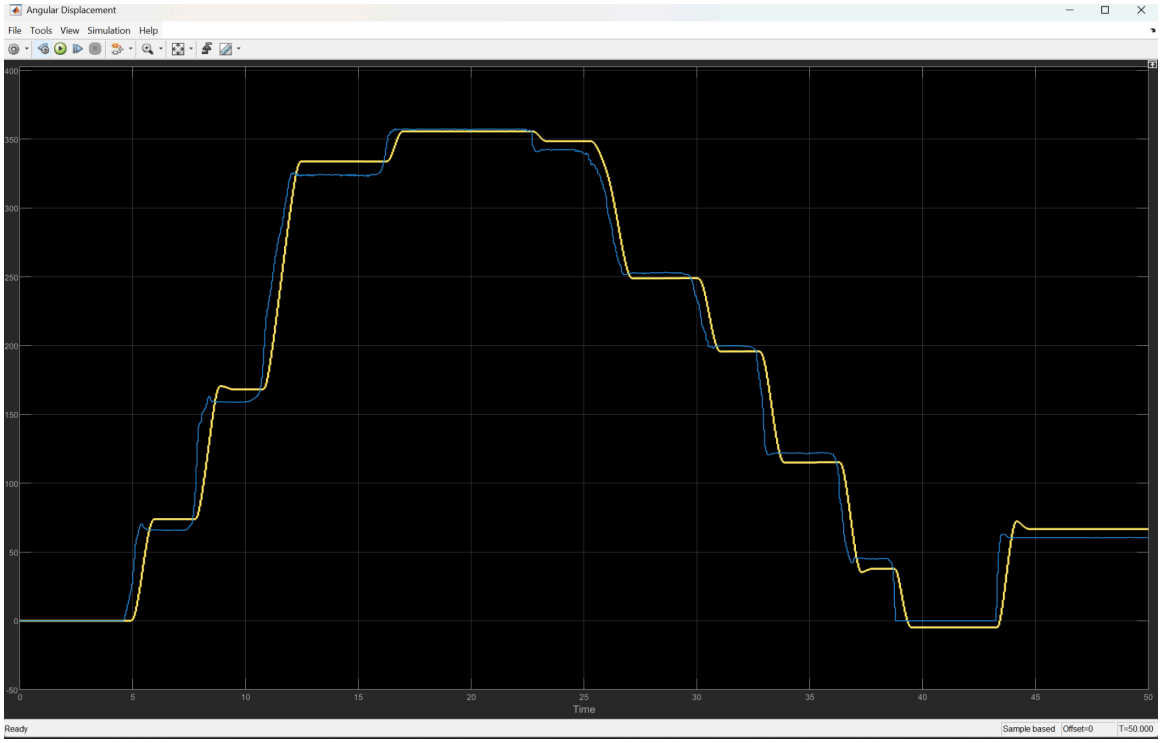

Figure 4: *Position with Kp of 7. Blue represents the reference value and yellow is encoder readings.*

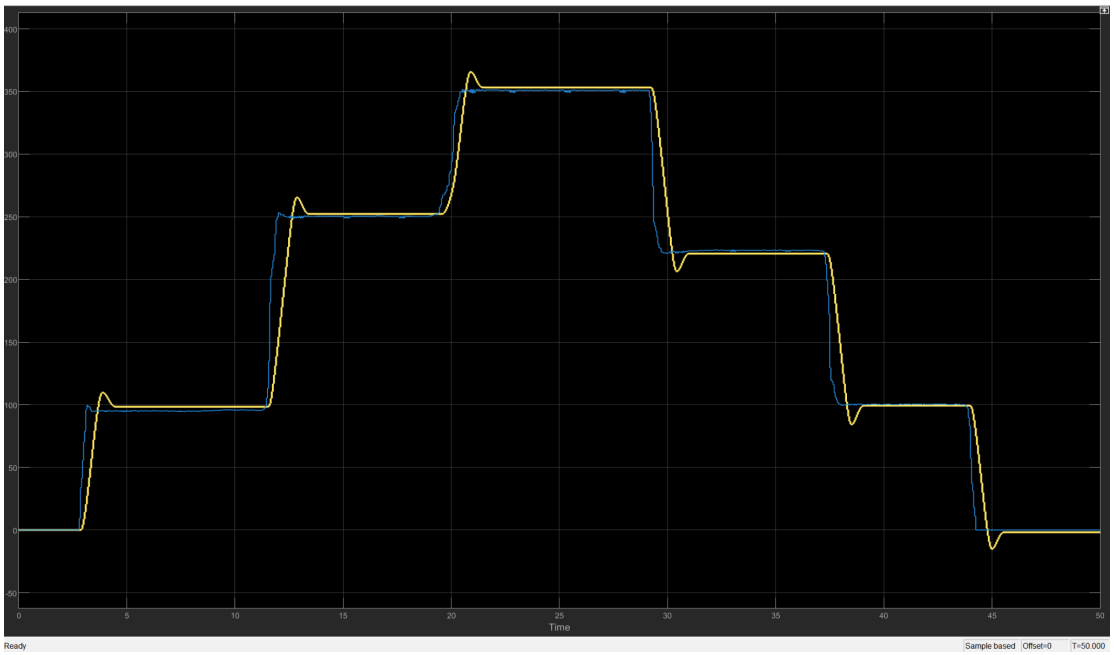

Figure 5: *Position, Blue represents the reference value and yellow is encoder readings.*

It is shown in the Figure 5 above that the current system settles at a position slightly higher than the reference when the position increases but settles below the reference when moving the motor backwards. Using these results we see the following results in Table 1 below with all the steady state errors calculated being the absolute value. Through this table we can see that the steady state never quite reaches the reference value but the steady state errors are relatively small with the largest being 2.59 degrees. This error was at a reference of 223 degrees and taking the average of all six reference angles used the system had an average steady-state error of 2.05 degrees. So, the steady-state error is minimal for the system. The overshoot is also calculated in Table 1 with the maximum overshoot being at the reference value of 0 with a percentage of 13.2%. With the smallest overshoot being 8.20% at a reference of 250 degrees thus all overshoots are between those two values with a range of 4.98% between them. After these overshoots there is no undershoot afterwards and it can reach the steady state values recorded in Table 1

Table 1: *Reference Vs Actual Comparison*

Reference	Steady State	Overshoot Value	Overshoot %	Steady-State Error
95.8	98.4	109.7	11.5	2.52
250	252	264.9	8.20	2.23
351	353	365.5	12.3	1.98
223	221	208	9.68	2.59
100	99.3	84.9	11.9	1.16
0	-1.79	-15.1	13.2	1.79

Simscape Experimental Results

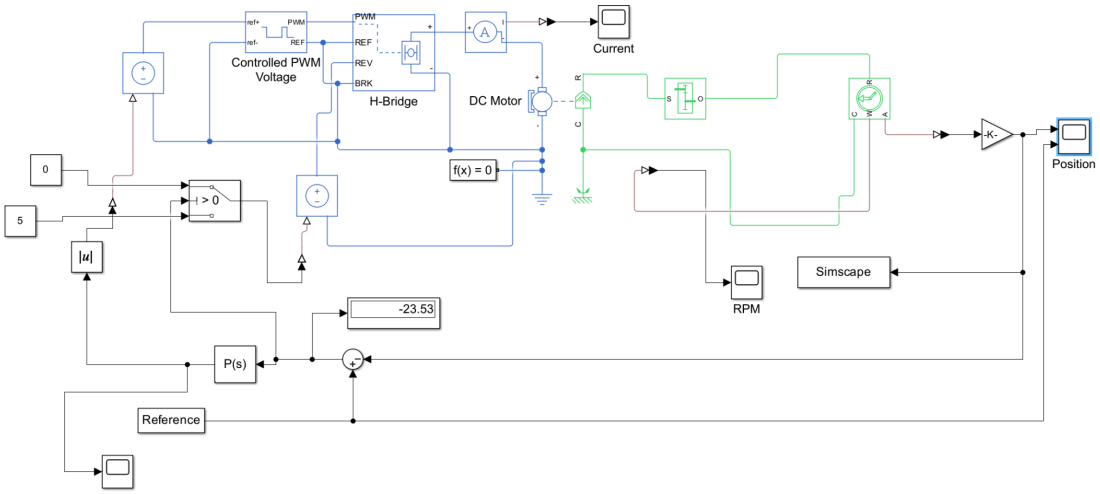

Figure 6: *Simscape Model*

In Figure 6 above the Simscape model for the system is showcased through it it can be seen that only one switch is required to control the H-bridge sending a value of 5 V when the error is negative to reverse the motion. As well as the resulting value from the proportional controller being mapped to 0 to 5 V instead of to a PWM signal since that is generated later in the model. The reference signal and proportional controller gain both come from the hardware experiment to allow for comparison between the systems. The proportional controller gain has to be multiplied by 5/255 to convert to the simscape model since the value is being fed into a voltage which comes out to be 0.139 for a Kp value. In Figure 7 below it is showcased what happens when a constant value is chosen for the reference. From this it can be seen that the first overshoot of the position is around 39.8% since the steady state value here is the reference and it takes until 25.7 seconds before the overshooting of the position becomes to merge with the reference signal.

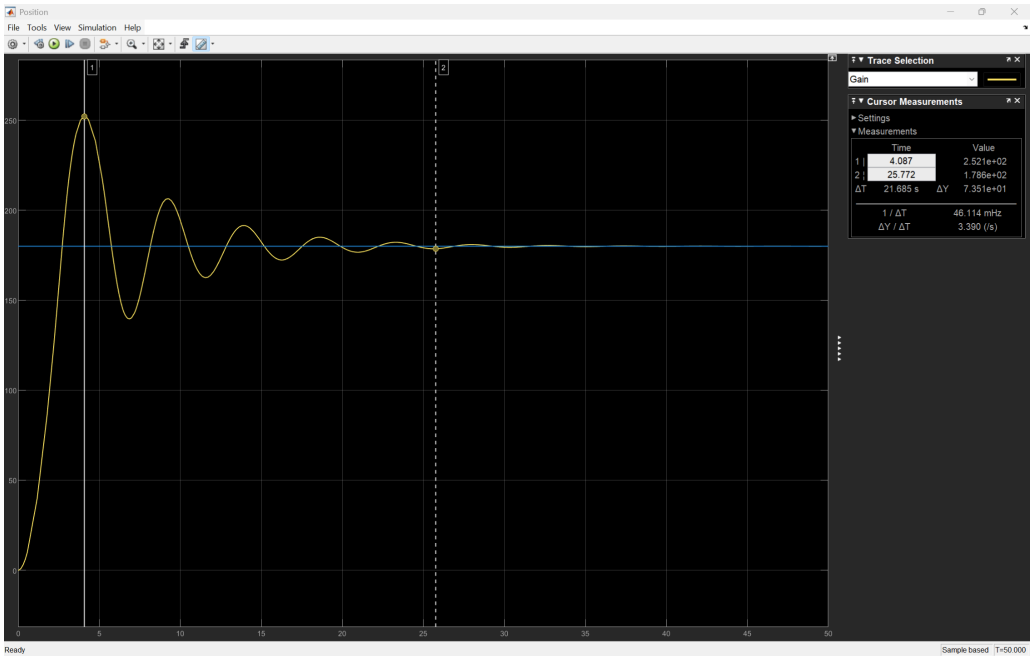

Figure 7: *Simscape Position with Constant Reference. Blue represents the reference value and yellow is encoder readings.*

Meanwhile in Figure 8 below it can be seen that with the use of the potentiometer readings from the hardware model in a time frame of 50 seconds it is unable to reach steady state value since the reference signal is being changed too frequently to achieve that value. Now it can be seen in Table 2 below for these results that the overshoot percentages are very large with the smallest being 41.1% and largest being 57.0%. Now this is with the assumption that the steady state would eventually reach the reference value as seen in Figure 7 above since in the time frame tested the signal did not have enough time to reach steady state. This makes the assumption that there is no steady state error as time goes to infinity although the overshoot percentage has a range of 15.9 when the smallest measured overshoot and largest measured overshoot is compared. With the steady time being around 35 seconds before it stops overshooting and undershooting the reference as seen in Figure 7.

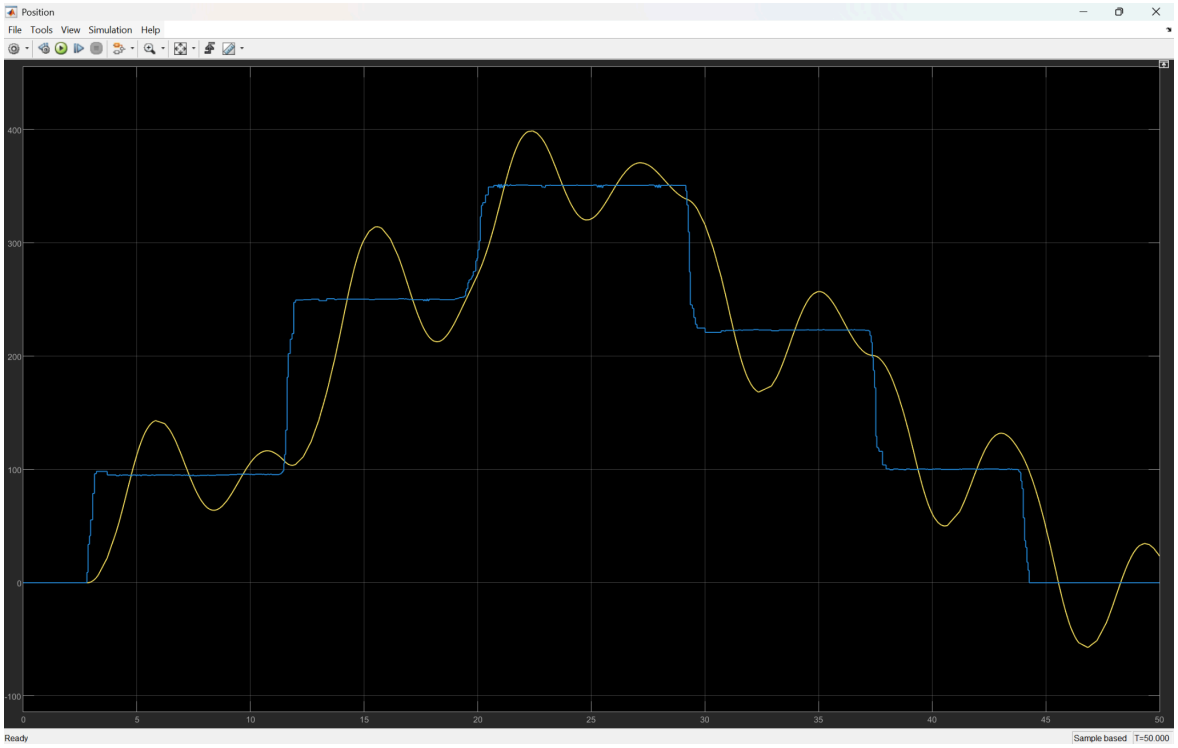

Figure 8: *Position, Blue represents the reference value and yellow is encoder readings.*

Table 2: *Reference Vs. Actual*

Reference	Overshoot Value	Overshoot %
95.8	143	49.4
250	314	41.6
351	398	46.7
223	168	42.8
100	50	41.1
0	-57.3	57.0

Comparison:

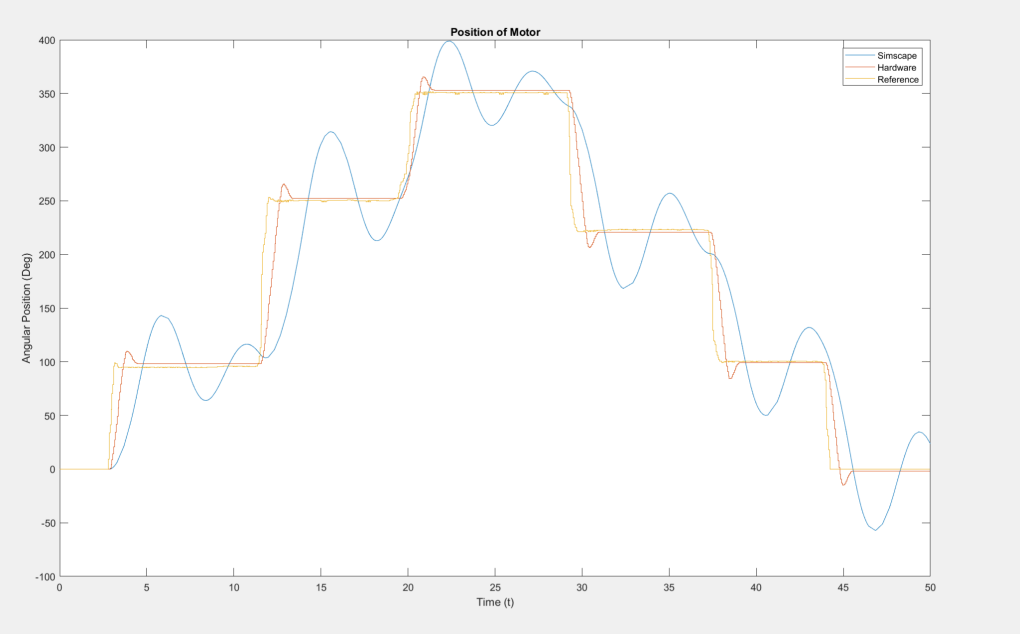

Figure 9: *Position of Motor Graph with Simscape results, Hardware results, and Reference values*

Using Figure 9 above it can be seen the simscape results, hardware results, and reference values are all plotted on a single graph. With this result it can be seen that while the hardware results have a lower overshoot there is a difference between the reference and steady state values of the hardware. Meanwhile the simscape results have larger overshoot values and do not reach steady state within the time frame; it is shown previously in Figure 7 that it does approach the reference value as a steady state but it takes around 25.7 seconds before it begins approaching and takes a full 35 seconds before it settles at the reference value. Meanwhile the hardware model does settle in the time frame shown in Figure 9 and has an average settling time of 1.89 seconds calculated through the settling times located in Table 3 below. Thus it can be seen that the Simscape model is not a good representation for the amount of time it takes to settle compared to the hardware model. These differences are caused by how the motor was modeled previously in respect to DC gain of the system, not the time constant. With the overshoot percentages from Table 1 and Table 2 it shows that the hardware consistently has lower overshoot percentages and is able to reach steady state at each reference value. The hardware also does not show an undershoot in the graph following it and insteads stops at a steady state value unlike the Simscape results. There is also steady-state error in the hardware model with it being an average 2.05 degrees difference between the reference and actual steady state. This is not the

case for the Simscape model since it was shown previously the steady state value of it is the reference value thus it has a steady-state error of zero.

Table 3: *Steady State Time*

Reference	Ts (sec)
74.6	1.70
236	1.95
353	2.20
283	1.85
205	1.85
100	1.80

Conclusion for Position Control:

The objective of this section was to implement and test the use of a closed loop position control system under a varying reference position from the potentiometer. The experimental data was collected to determine the Kp value to be used for the proportional controller as well as taking the potentiometer reference values to be used for the Simscape model. This data was used for comparison between the two systems.

The results showed for every case the overshoot value of the simscape model was greater than the hardware model as an example for a reference value of 95.8 degrees the simscape model had an overshoot of 49.4% and the hardware model had an overshoot of 11.5%. With that said, in both cases the overshoot at the beginning of the motors reach higher values compared to the other overshoot percentages. The other difference between the models is that for Simscape to reach steady state fully it takes around 35 seconds. Meanwhile for the hardware system it reaches the steady state value within the recorded data in Figure 9 and at an average time of 1.89 seconds which is around 18.5 times faster than the digital model. With the steady state error either being above or below the reference value with an average of 2.05 degrees compared to the model's zero steady state error. With these facts stated it can also be seen in earlier figures that while it is known that the Simscape will eventually reach the reference value that is not the case for the hardware instead settling close to reference but not being the exact reference. These values will either be greater or less than reference depending on the motion of the motor itself.

Overall, I would say that both the Hardware and Simscape models have a position control system implemented with greater improvements that could be made to the simscape model to more accurately model the hardware. The differences could be

caused by how the motor was previously modeled from the gain of the transfer function in the earlier parts of this project this affected the time constant and thus the settling time which is the cause of the overshoots being much larger. With the overall modeling to get close to the desired position being a success.

Closed Loop Project Objective 2:

Project Brief:

- ➔ Developed a **Simscape model** to implement a closed-loop DC motor speed control system as defined in Control Objective (ii).
- ➔ The Simscape model includes:
  - ◆ DC motor with gearbox
  - ◆ PWM signal generation
  - ◆ H-bridge for bidirectional control
  - ◆ PI-based feedback loop for speed tracking
- ➔ Demonstrated **speed tracking** in simulation by adjusting the reference input and observing motor response.
- ➔ Built a corresponding **Simulink model for hardware implementation** using Arduino.
  - ◆ Potentiometer used as reference input
  - ◆ Encoder used for real-time speed feedback
  - ◆ Control loop executed in real time using Simulink and PWM output through an H-bridge
- ➔ **Compared simulation results with hardware performance** to evaluate tracking accuracy and system behavior.
- ➔ Compiled a **detailed and organized technical report** covering modeling, implementation, simulation, and comparison.

Introduction:

This project focuses on the design and implementation of a closed-loop DC motor speed control system using Simulink, Simscape, and real-time hardware integration with Arduino. The primary objective is to ensure that the motor accurately tracks a variable reference speed while maintaining a consistent direction of rotation, as outlined in Control Objective (ii). A PI controller is developed to minimize the error between the measured speed—obtained from encoder feedback—and the reference input, which is set dynamically using a potentiometer.

The system is initially modeled and simulated in Simscape, which includes components such as the DC motor, gearbox, H-bridge driver, and PWM control. The same control strategy is then deployed on the hardware platform using Simulink and an Arduino microcontroller. Performance is evaluated by comparing the motor's response in both simulated and real environments to verify the accuracy and reliability of the control loop. This report documents the modeling, simulation, hardware implementation, and analysis.

Simscape Simulation

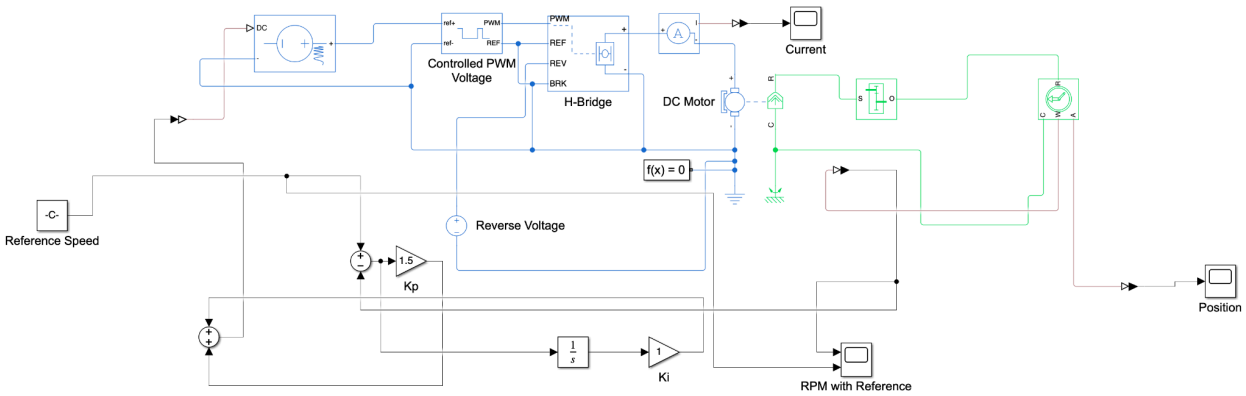

Figure 10: *Simscape model implementing Control Objective (ii) for DC motor speed tracking using PI control, encoder feedback, PWM, gearbox, and H-Bridge.*

A complete electromechanical model of the DC motor speed control system was developed in Simscape to implement Control Objective (ii), which aims to achieve real-time speed tracking. The model includes all major physical and control components needed to simulate actual hardware behavior: a DC motor, gearbox, encoder feedback, a PI controller, PWM generator, and an H-Bridge circuit for bidirectional motor control.

The reference speed input is provided through a constant block, representing the desired speed in RPM. This signal is compared against the measured speed (derived from the motor's angular velocity) using a summation block. The difference, or tracking error, is processed by a PI controller, which generates the voltage command for the motor. The proportional and integral gains (Kp and Ki) that were tuned manually through repeated testing in the Simscape simulation. To reduce overshoot, I initially started with a moderate Kp like 0.1 value and gradually increased Ki in small steps. When I noticed excessive overshoot, I reduced Kp slightly to dampen the response. To improve settling time, I fine-tuned Ki by increasing it until the system reached the steady-state faster without causing a lot of oscillations. I continued adjusting both gains while observing the motor’s step response, aiming for a balance where the system responded quickly, minimized overshoot, and settled with minimal steady-state error. PWM voltage control is applied through a Controlled PWM Voltage block, which drives the H-Bridge. The H-Bridge allows current to flow through the motor in either direction, enabling both forward and reverse operation based on the sign of the control signal. In this case we had to only focus on maintaining one direction. The encoder feedback loop closes the system, continuously feeding back the motor's speed to ensure accurate tracking.

The motor speed output is compared to the reference within the model and plotted to confirm tracking behavior. When the system settles and both reference and actual speeds align, minimal steady-state error is confirmed visually through the plotted response. This simulation provides a reliable benchmark for expected real-time performance, which is later compared with the results from hardware implementation.

Simscape Simulation Experiment Results:

Table 4: *Simulation results from the Simscape model showing reference speed, 90% target RPM, time to reach 90%, and percentage overshoot for various reference speeds.*

SimScape Model			
Reference Speed (RPM)	**90% of RPM**	**Time to 90% (s)**	**Overshoot (%)**
4.927	4.434	0.577	8.333
10.000	9.000	0.939	17.172
15.015	13.513	1.627	29.333
19.853	17.868	2.447	33.503
25.513	22.962	3.812	20.079

Simscape Plots:

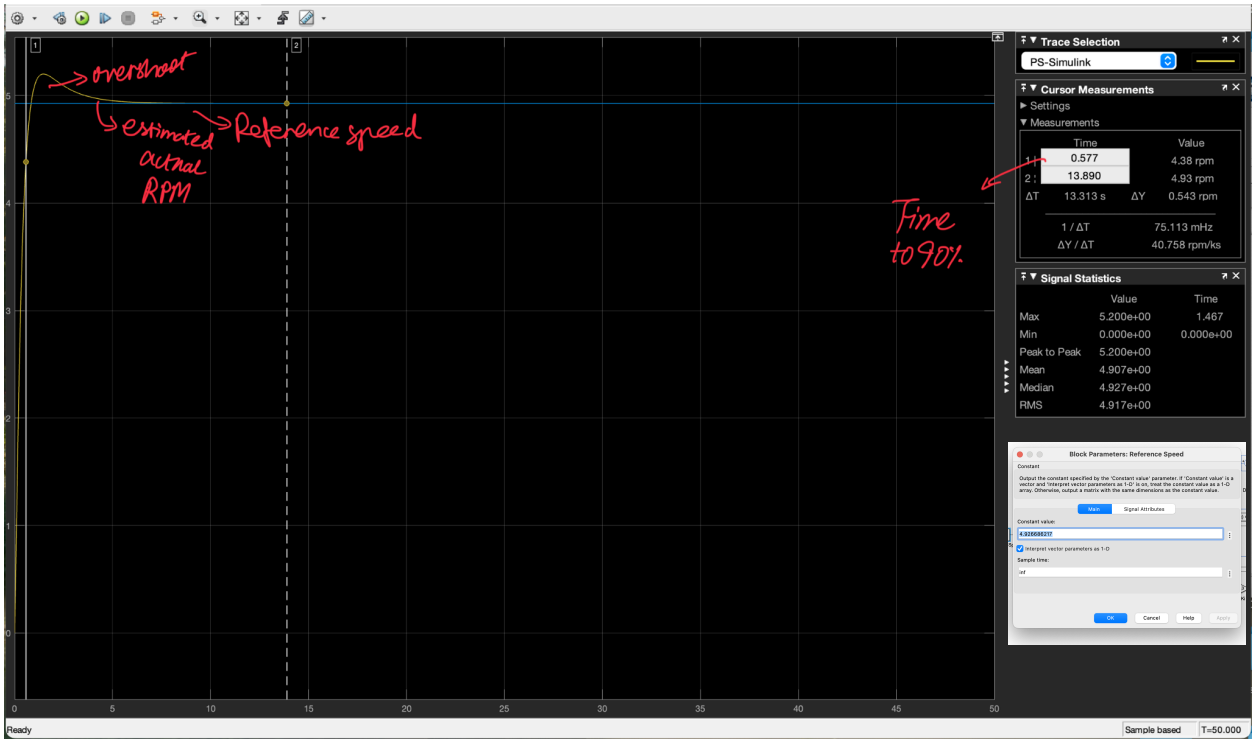

Figure 11: *Simulink plot for a reference speed of 4.927 RPM showing the motor's response, including overshoot and time to reach 90% of the reference speed.*

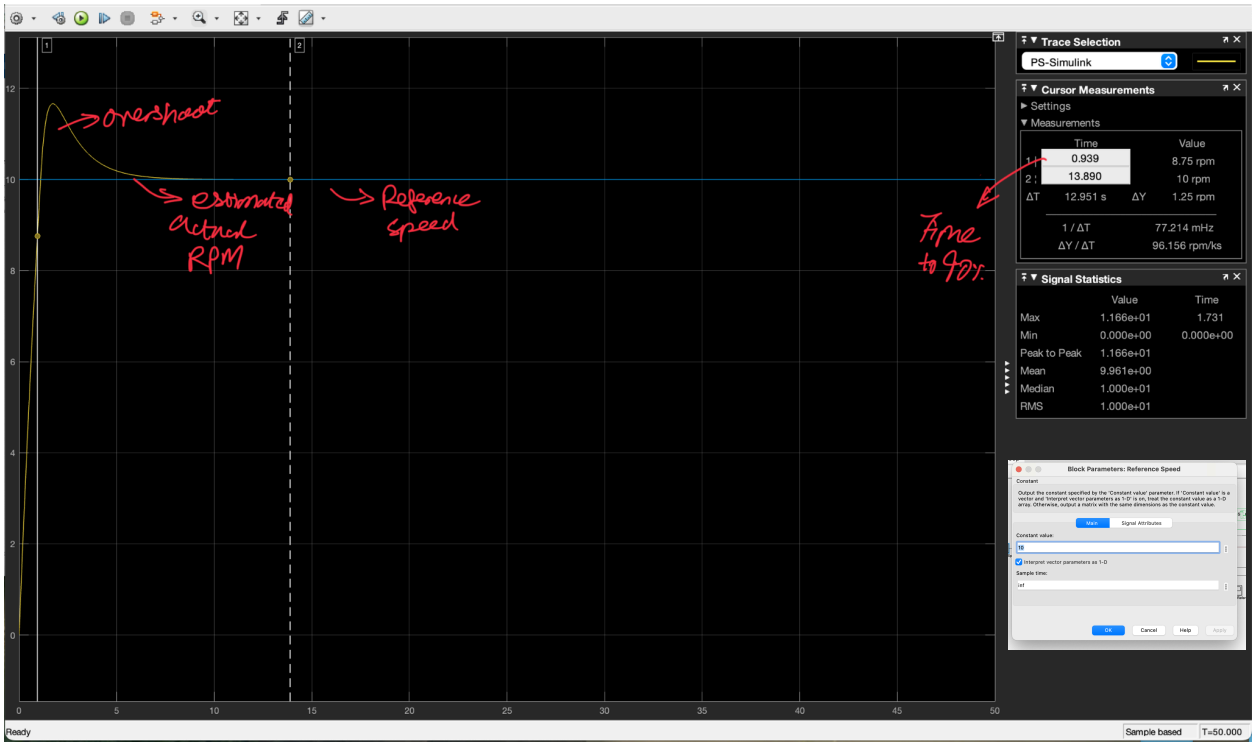

Figure 12: *Simulink plot for a reference speed of 10.000 RPM showing the motor's response, including overshoot and time to reach 90% of the reference speed.*

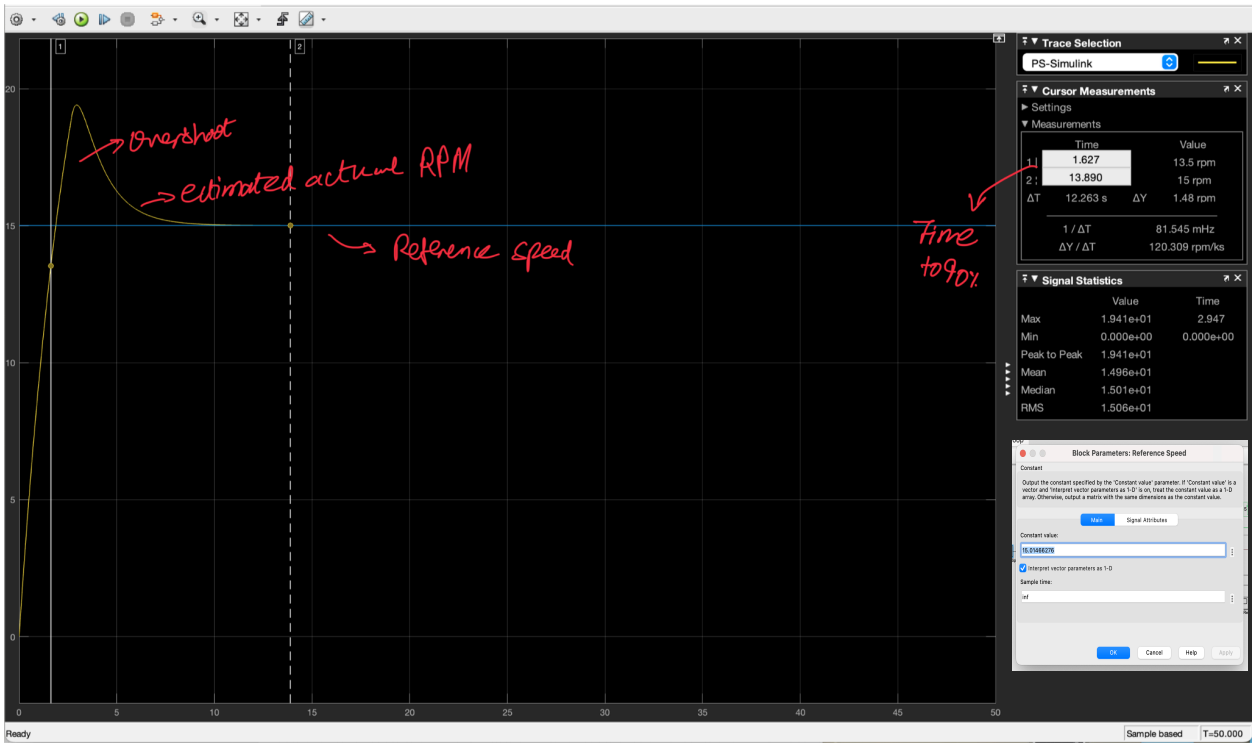

Figure 13: *Simulink plot for a reference speed of 15.015 RPM showing the motor's response, including overshoot and time to reach 90% of the reference speed.*

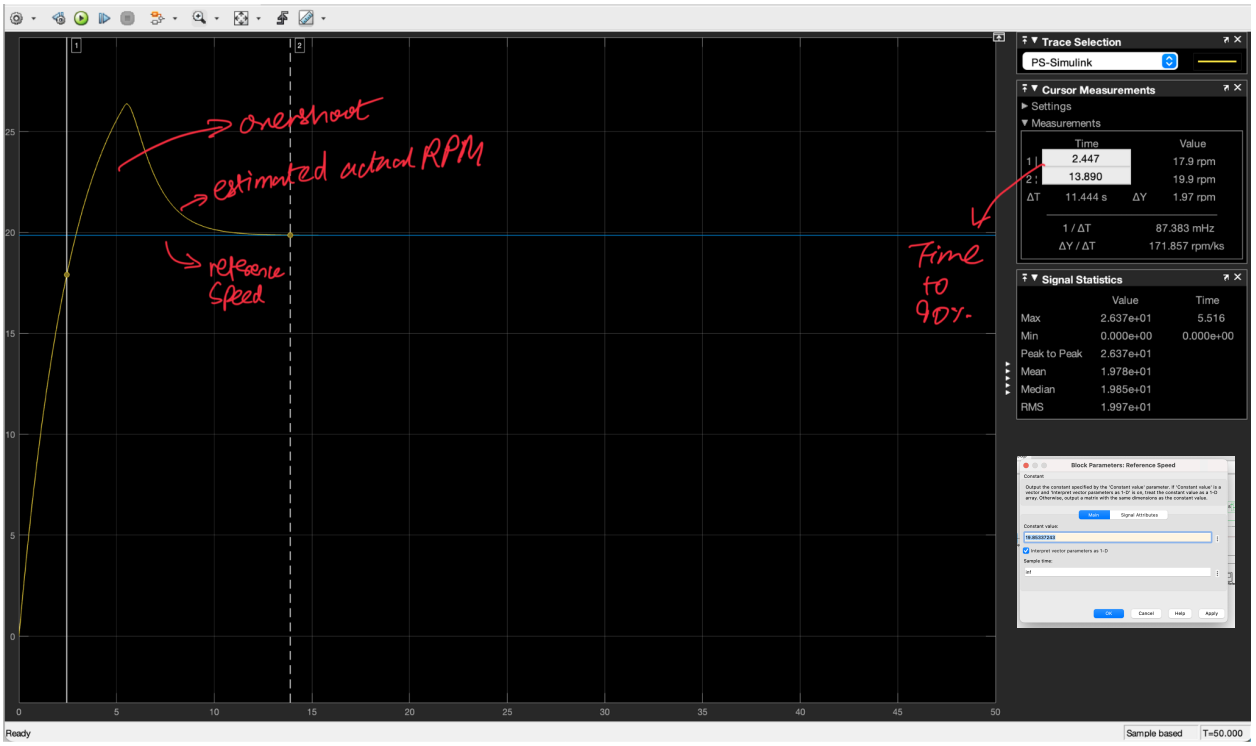

Figure 14: *Simulink plot for a reference speed of 19.853 RPM showing the motor's response, including overshoot and time to reach 90% of the reference speed.*

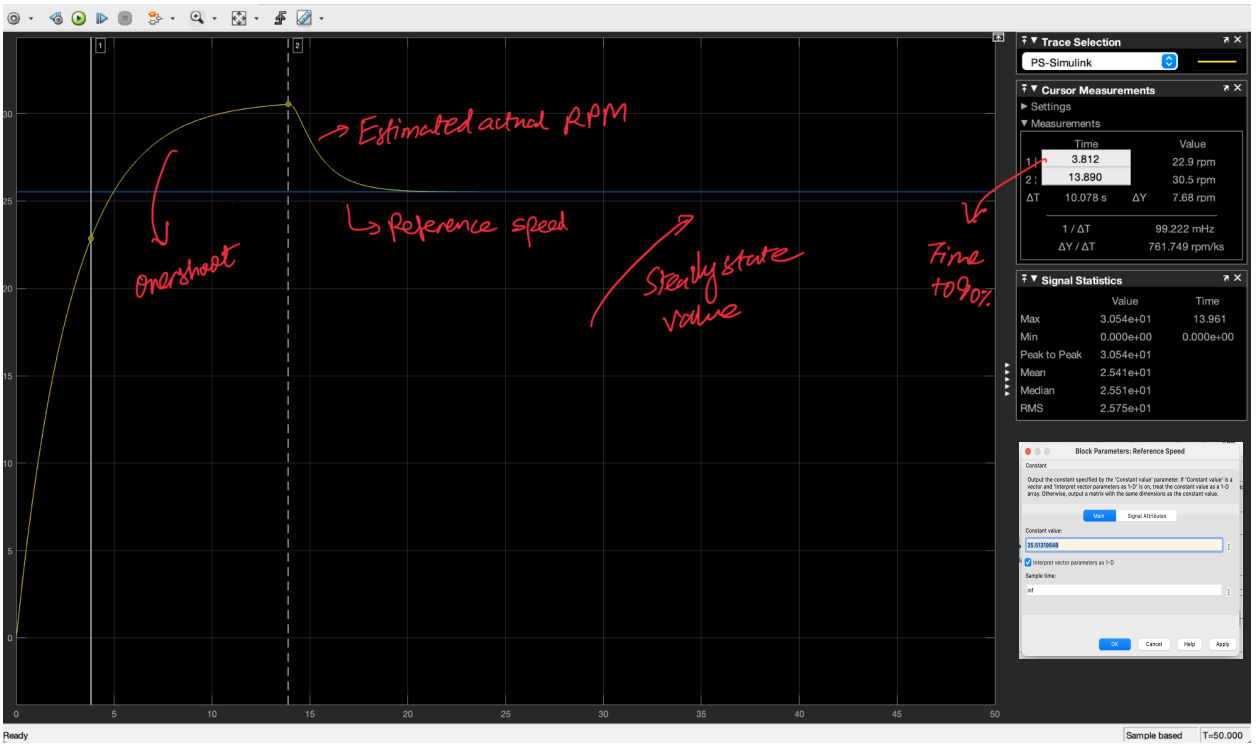

Figure 15: *Simulink plot for a reference speed of 25.513 RPM showing the motor's response, including overshoot and time to reach 90% of the reference speed.*

To evaluate the performance of the Simscape-based closed-loop speed control system, simulations were conducted for five different reference speeds: **4.927 RPM, 10.000 RPM, 15.015 RPM, 19.853 RPM**, and **25.513 RPM**. Each simulation was used to extract the **time to reach 90%** of the reference speed and the **overshoot percentage**, which are indicators of system responsiveness and stability.

The **time to 90%** was calculated as the time taken by the motor to reach 90% of the reference RPM after the step change in input. This metric reflects how quickly the system responds to new commands. As shown in the results:

- At **4.927 RPM**, 90% of the target is **4.434 RPM**, reached in **0.577 seconds**.
- At **10.000 RPM**, 90% is **9.000 RPM**, reached in **0.939 seconds**.
- At **15.015 RPM**, 90% is **13.513 RPM**, reached in **1.627 seconds**.
- At **19.853 RPM**, 90% is **17.868 RPM**, reached in **2.447 seconds**.
- At **25.513 RPM**, 90% is **22.962 RPM**, reached in **3.812 seconds**.

A clear trend is observed: as the reference speed increases, the time taken to reach 90% also increases. This suggests that the system experiences progressively slower acceleration at higher speeds, possibly due to limitations from the motor dynamics.

The **overshoot percentage** was calculated using the formula:

Overshoot (%) = ((Initial peak - Steady-State Value) / Steady-State Value) × 100

This value indicates how much the system output exceeds the final steady-state speed before settling down. The measured overshoot values were:

- At **4.927 RPM**, the overshoot was **8.333%**.
- At **10.000 RPM**, the overshoot increased to **17.172%**.
- At **15.015 RPM**, it rose further to **29.333%**.
- At **19.853 RPM**, it peaked at **33.503%**.
- At **25.513 RPM**, the overshoot dropped to **20.079%**.

These results show that overshoot tends to increase with reference speed initially, indicating a more aggressive response by the controller. However, at the highest speed setting (25.513 RPM), the overshoot begins to reduce—possibly due to saturation in control output or internal damping effects modeled in Simscape.

Together, these trends suggest that the PI controller tuned in Simscape effectively tracks the reference RPM across a range of inputs. The rise time and overshoot behaviors are consistent and predictable, validating that the control system is functioning as expected. The Simscape model therefore serves as a useful environment to test controller performance before hardware deployment.

Simulink Hardware Implementation:

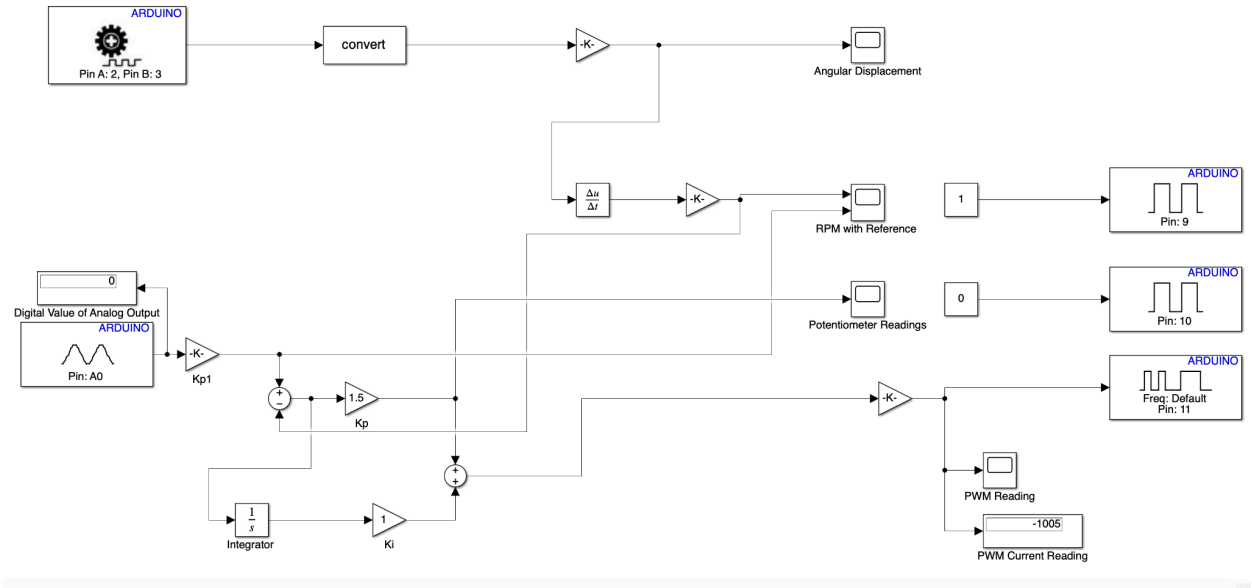

Figure 16: *Simulink model for real-time DC motor speed control using encoder feedback, PI controller, and PWM output to Arduino.*

The Simulink hardware model implements the real-time control of a DC motor using encoder feedback, a PI controller, and PWM output signals sent to an Arduino. This setup is used to validate the same control strategy developed in the Simscape model on actual hardware.

The model begins by reading the desired reference speed through a potentiometer input on analog pin A0. This reference value, obtained from the potentiometer, is then compared to the actual motor speed measured by the encoder. To evaluate the system's performance across a range of operating conditions, five distinct reference speeds were selected by adjusting the potentiometer from its minimum to maximum

position. These settings represent evenly spaced intervals within the motor's achievable speed range, allowing for consistent testing and comparison of the control response at low, mid, and high speeds, measured by counting encoder pulses on digital pins 2 and 3. The error between the reference and actual speed is fed into a PI controller, which adjusts the motor input via PWM. The proportional (Kp) and integral (Ki) gains used in this model are carried over from the Simscape simulations, appropriately converted for hardware implementation.

To maintain consistency with the Simscape simulations, the simulation time for the hardware model is set to 50 seconds. This allows for a direct comparison between the physical and simulated responses. The PWM signal is generated on digital pin 11 of the Arduino, with pins 9 and 10 used to control motor direction. The RPM, reference speed, and PWM duty cycle are visualized using scopes for performance analysis.

This implementation allows observation of how well the physical system can track the reference speed in real-time and serves as a direct comparison point for validating the controller design against simulation results.

Simulink Hardware Experiment Results:

Table 5: *Experimental results from the physical model showing reference speed, 90% target RPM, time to reach 90%, and percentage overshoot for various potentiometer settings.*

Physical Model			
Reference Speed (RPM)	**90% of RPM**	**Time to 90% (s)**	**Overshoot (%)**
4.927	4.434	0.042	11.111
10.000	9.000	0.042	8.333
15.015	13.513	0.125	18.182
19.853	17.868	0.167	18.182
25.513	22.962	0.250	12.000

Simulink Plots:

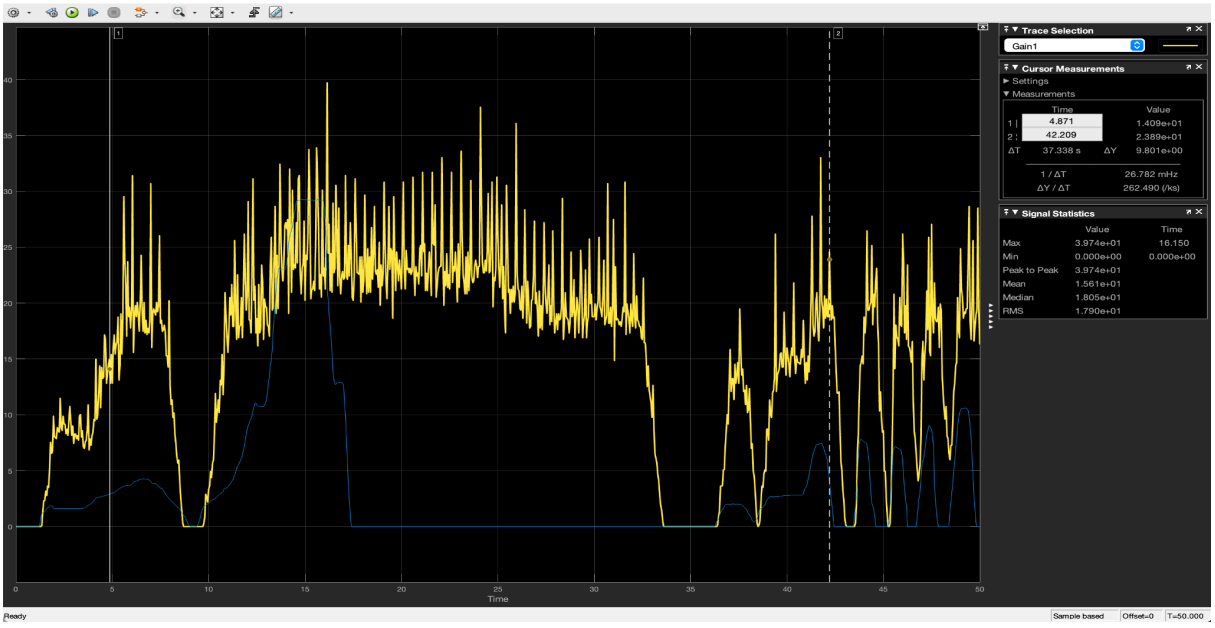

Figure 17: *This plot displays motor performance over time, with time on the x-axis and RPM on the y-axis. The blue line represents the reference speed set by the potentiometer, while the yellow line indicates the actual motor RPM. The graph demonstrates that the motor's response effectively tracks the reference input, accurately following both increases and decreases in the reference input. This validates the system's ability to respond dynamically to real-time control inputs.*

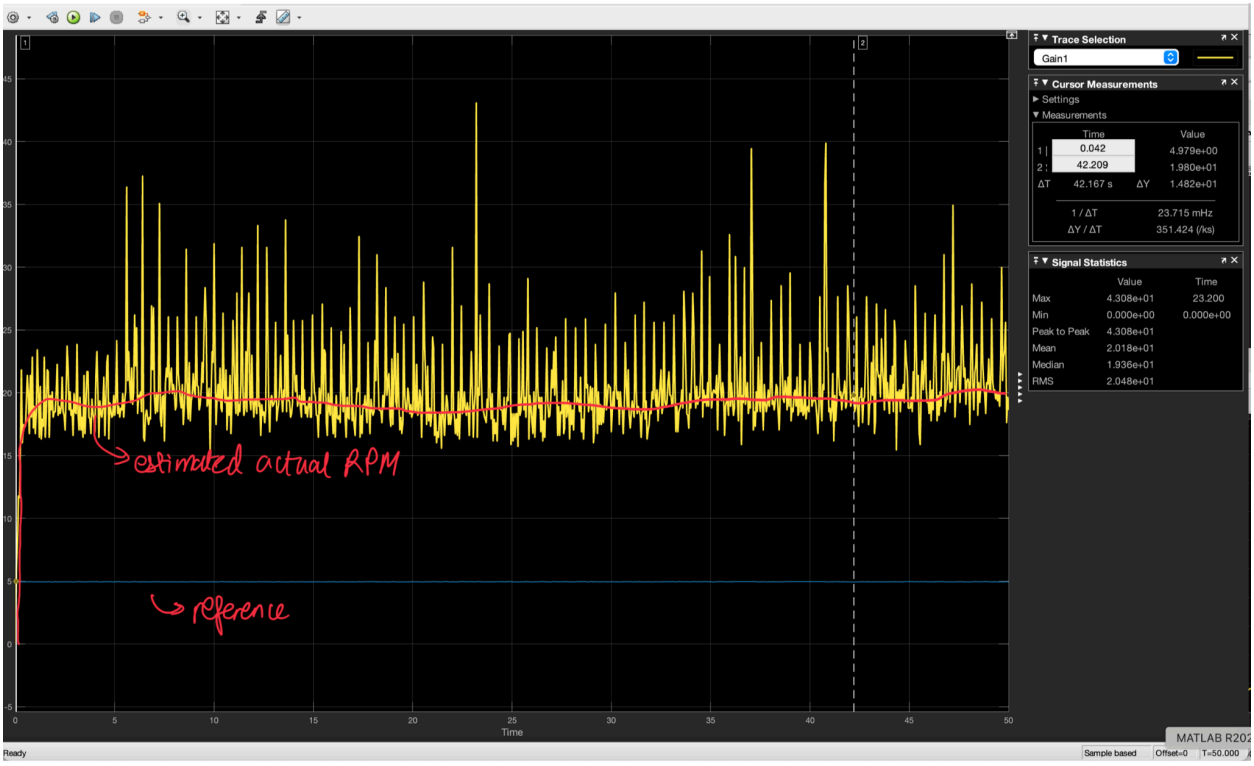

Figure 18: *Motor response for a reference speed of 4.93 RPM. The system reaches 90% in 0.042 s with an overshoot of 11.11%.*

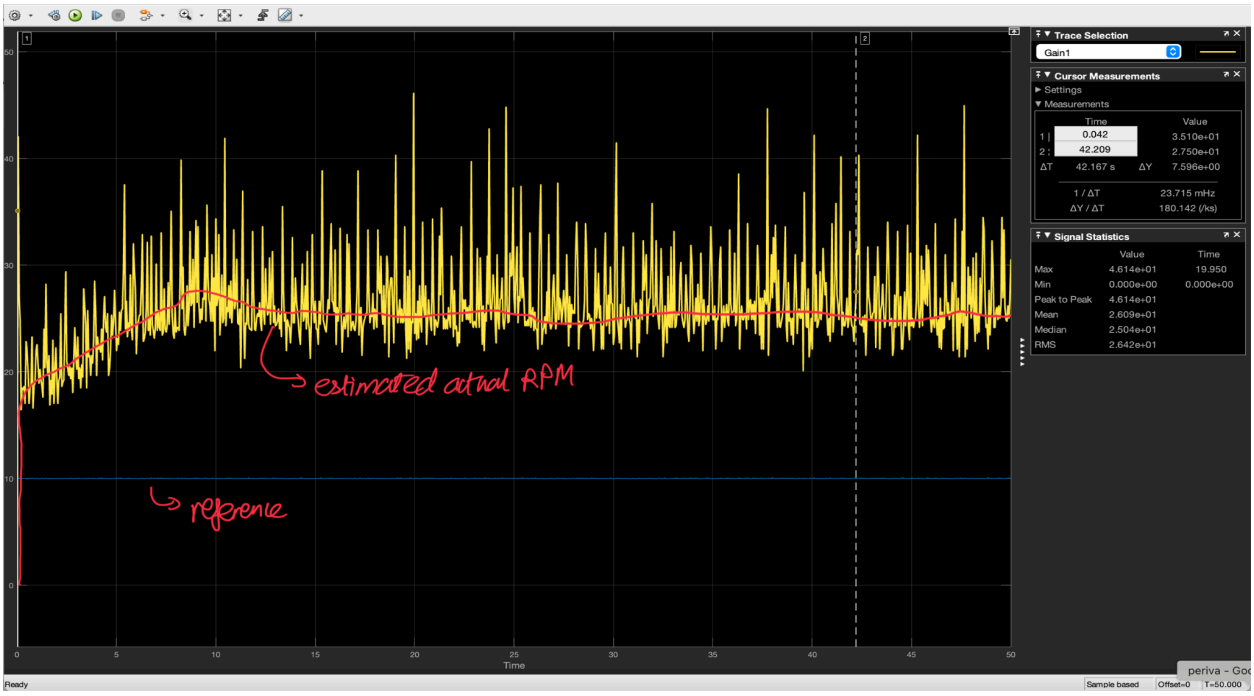

Figure 19: *Motor response for a reference speed of 10.00 RPM. The system reaches 90% in 0.042 s with an overshoot of 8.33%.*

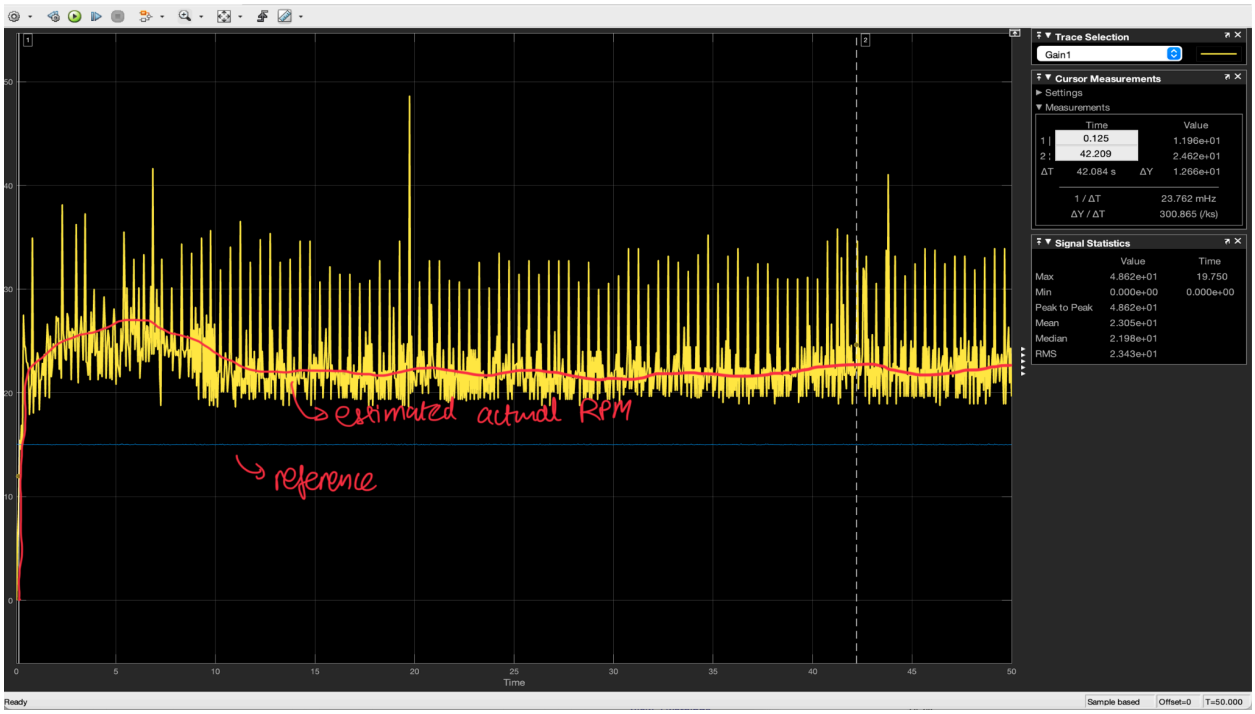

Figure 20: *Motor response for a reference speed of 15.02 RPM. The system reaches 90% in 0.125 s with an overshoot of 18.18%.*

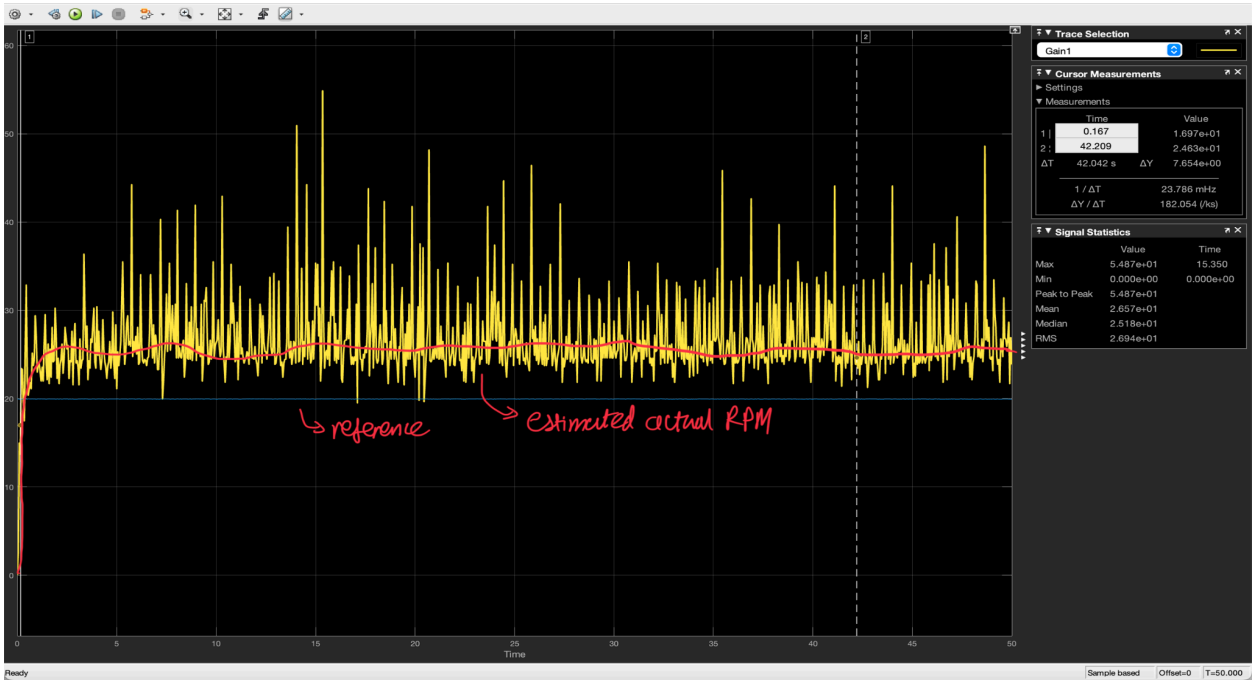

Figure 21: *Motor response for a reference speed of 19.85 RPM. The system reaches 90% in 0.167 s with an overshoot of 18.18%.*

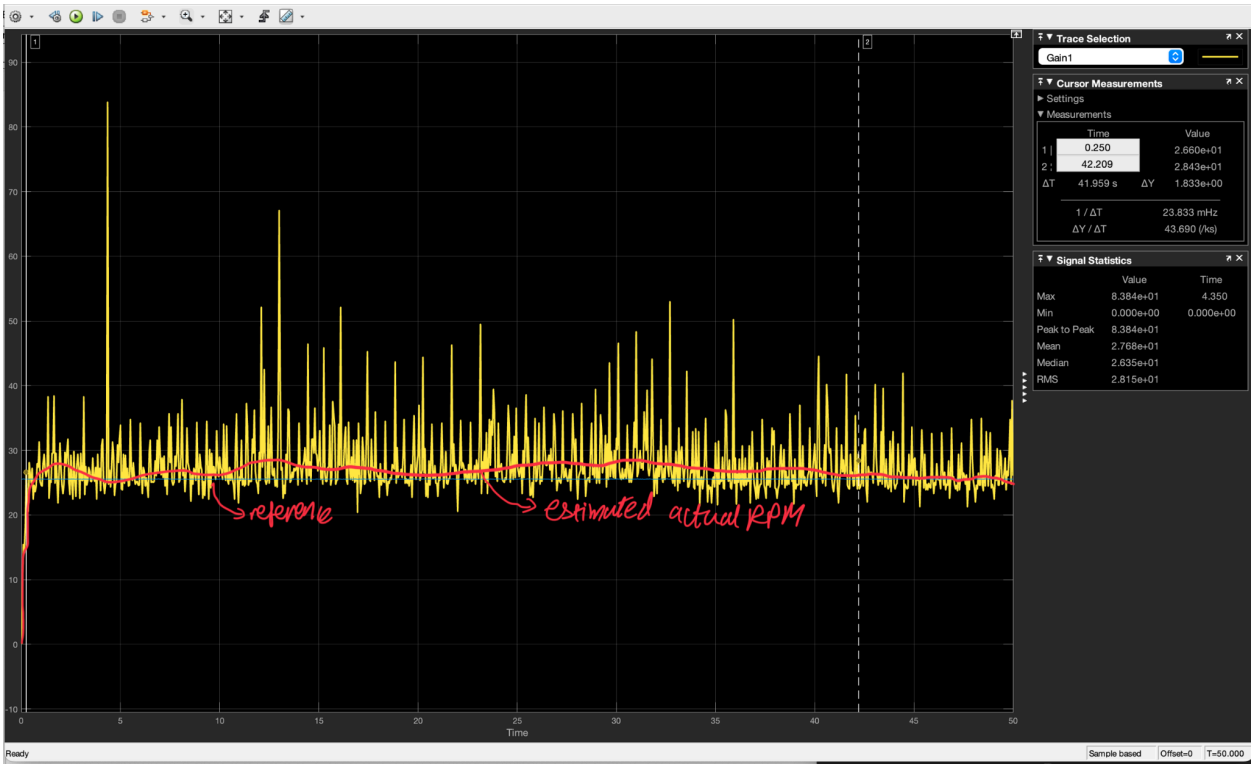

Figure 22: Motor response for a reference speed of 25.51 RPM. The system reaches 90% in 0.250 s with an overshoot of 12.00%.

To validate real-time hardware performance, the Simulink model was deployed on an Arduino using a fixed simulation time of **50 seconds**, consistent with the Simscape environment. The same PI controller gains used in Simscape (Kp = 1.5, Ki = 1) were converted and applied for hardware implementation. A potentiometer was used to define five reference speeds: **4.927, 10.000, 15.015, 19.853, and 25.513 RPM**. The system response was recorded using scope data and analyzed.

The following performance metrics were calculated for each reference input:

- **90% of RPM**: This value was used to determine rise time in each case:
    - 90% of **4.927 RPM = 4.434 RPM**
    - 90% of **10.000 RPM = 9.000 RPM**
    - 90% of **15.015 RPM = 13.513 RPM**

    - 90% of **19.853 RPM = 17.868 RPM**
    - 90% of **25.513 RPM = 22.962 RPM**

- **Time to 90%**: This is the time it took for the motor to reach 90% of the reference speed after the step input:
    - **0.042 s** at 4.927 RPM
    - **0.042 s** at 10.000 RPM
    - **0.125 s** at 15.015 RPM
    - **0.167 s** at 19.853 RPM
    - **0.250 s** at 25.513 RPM
- The increasing time to reach 90% with higher reference speeds indicates the system's response slows slightly as more energy is required to reach higher RPMs.
- The **overshoot percentage** was calculated using the formula:

    Overshoot (%) = ((Initial peak - Steady-State Value) / Steady-State Value) × 100

    Experimental overshoot values:

    - **11.111%** at 4.927 RPM
    - **8.333%** at 10.000 RPM — lowest overshoot
    - **18.182%** at both 15.015 and 19.853 RPM — moderate and consistent
    - **12.000%** at 25.513 RPM — slightly lower again at the highest speed

Overall, the hardware results show that the system consistently achieved fast rise times and moderate overshoot across all tested reference speeds. For instance, at the two lowest speeds (4.927 and 10.000 RPM), the motor reached 90% in just **0.042 seconds**, while overshoot remained modest at **11.111%** and **8.333%**, respectively. At mid-range values like 15.015 and 19.853 RPM, the rise times increased to **0.125 s** and **0.167 s**, with overshoot stabilizing at **18.182%**, indicating good responsiveness without instability. Even at the highest tested speed of 25.513 RPM, the system maintained a controlled rise time of **0.250 s** and a reduced overshoot of **12.000%**. This suggests that real-world physical effects such as motor friction, voltage limitations, and signal smoothing contribute to the controller's behavior, helping it stay stable and responsive across the range of tested inputs.

Comparison Between Simscape Simulation and Simulink Hardware Implementation:

To evaluate the effectiveness of the speed control system in both simulation and real-time environments, results were compared for five reference speeds: 4.927, 10.000, 15.015, 19.853, and 25.513 RPM. The same PI controller gains were used in both setups, with appropriate scaling for the physical model. Two key performance metrics were analyzed: **time to reach 90% of the reference speed** and **percentage overshoot**.

Time to 90% Comparison

The physical system consistently showed faster response times compared to the Simscape model. For example:

- At 10.000 RPM, the Simscape model took **0.939 s**, while the hardware achieved 90% of the target in just **0.042 s**.

- At 25.513 RPM, Simscape took **3.812 s**, whereas the hardware reached the same threshold in **0.250 s**.

This consistent trend shows that the Simscape model behaves more conservatively, likely because it's hard to capture every detail of a real physical system in simulation. While the motor parameters like inertia and resistance

were matched, certain practical aspects—such as electrical switching behavior, real-time delays, and load variations—are simplified in the model. For example, the H-bridge used in the simulation is an older model and may not accurately reflect the switching dynamics or internal resistance of the actual hardware. In real systems, factors like wiring resistance, slight encoder inaccuracies, mechanical play, and electrical noise all contribute to a response that can't be fully recreated in simulation.

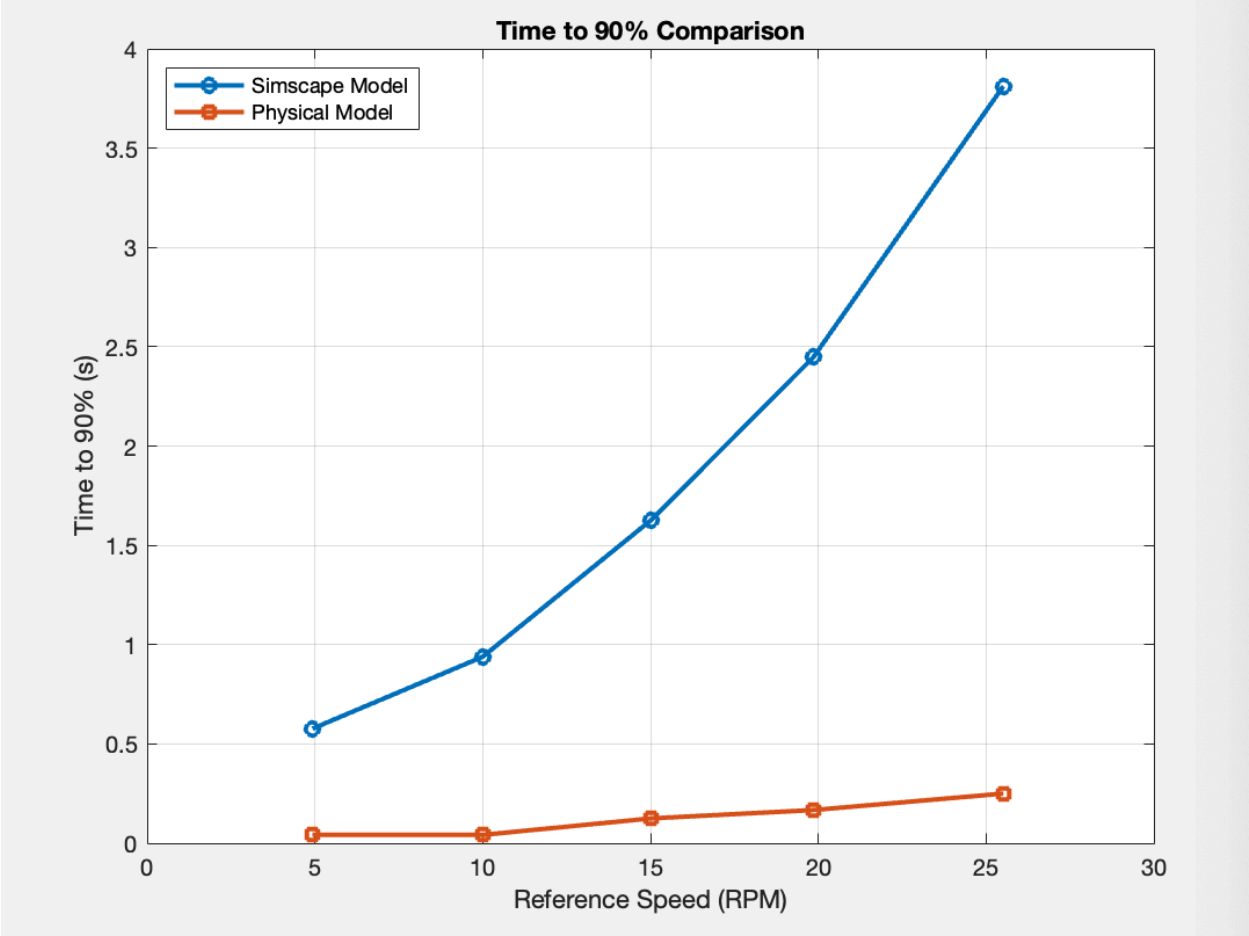

Figure 23: Time to 90% comparison between Simscape and physical models across reference speeds.

Overshoot Comparison

Overshoot behavior was also noticeably different between the two setups:

- At **4.927 RPM**, the Simscape model showed an overshoot of **8.33%**, while the hardware system showed a slightly higher overshoot of **11.11%**.

- At **10.000 RPM**, the Simscape overshoot increased to **17.17%**, whereas the hardware system had a lower overshoot of **8.33%**.

- At **15.015 RPM**, the Simscape model exhibited an overshoot of **29.33%**, while the hardware overshoot was **18.18%**.

- At **19.853 RPM**, the Simscape model peaked at **33.50%**, compared to a stable **18.18%** in the hardware.

- At **25.513 RPM**, the Simscape overshoot reduced to **20.08%**, while the hardware overshoot was **12.00%**.

We see that the overshoot in the hardware remained relatively controlled and consistent, ranging between **8.33% and 18.18%**. This aligns with what we've discussed in class—that real-world systems often exhibit natural damping due to factors like mechanical friction, gear train resistance, and sensor-actuator delays. These physical effects, while often simplified or idealized in simulation, help stabilize the response and reduce overshoot in practice, which is evident in the hardware results.

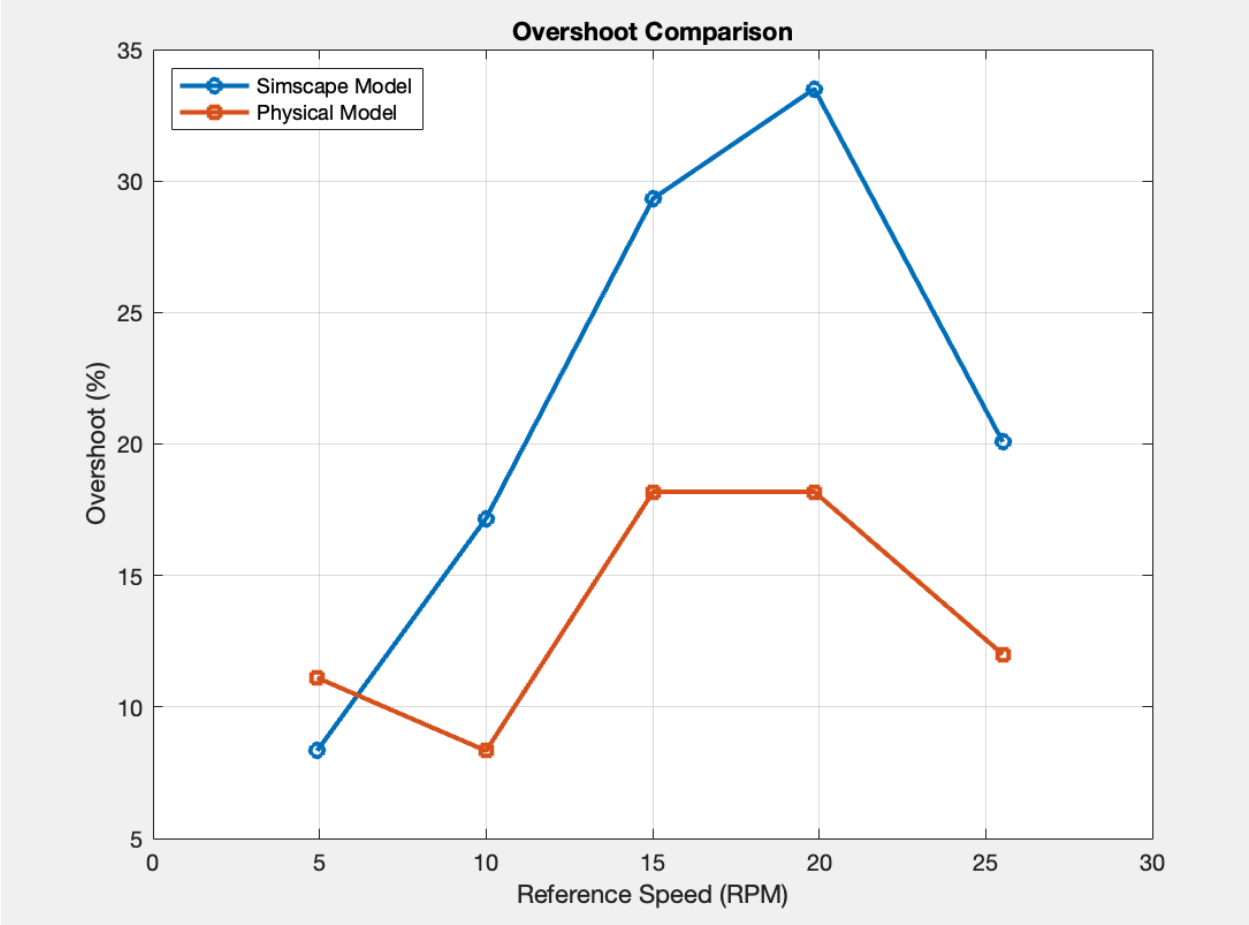

Figure 24: Overshoot comparison between Simscape and physical models across reference speeds.

Trend Consistency and Takeaways

Despite the numerical differences, both the Simscape and hardware models followed the same general trend: as the reference speed increases, both the time to 90% and the overshoot tend to increase. This indicates greater control effort is needed at higher speeds and validates that the PI control structure is functioning as expected in both environments.

The comparison highlights that while Simscape is effective for designing and testing control strategies, real-world validation is essential. Physical systems can behave differently due to damping, noise, and unmodeled dynamics—factors we've explored during class discussions and labs. Combining simulation with hands-on testing provides a more complete understanding of system behavior.

Challenges & Troubleshooting (Learnings from the project)

- **Encoder Noise at Low Speeds:** At low RPMs, encoder readings were noisy and unstable. Applying basic smoothing techniques helped improve accuracy and reduce fluctuations in the speed feedback signal.
- **Potentiometer Scaling:** The raw analog input from the potentiometer didn't directly translate to RPM values. A scaling factor was implemented to map the input voltage to a usable reference speed range.
- **Initial PI Tuning Instability:** Early attempts at tuning the PI controller caused instability in the motor response. Several trial-and-error passes were needed to balance fast response with acceptable overshoot.
- **PWM Signal Dropouts:** During early trials, intermittent PWM failures were traced back to wiring issues. Rechecking and securing connections resolved the problem.
- **Direction Control Glitches:** At certain input levels, the motor direction pins were not switching correctly due to logic mismatches. Revising the logic conditions ensured consistent forward motion during testing.