2DoF Robot Arm

Sebastian Rosas Guaida, Rodrigo Rivas Gonzalez, Fernando Giovanni Muciño Guerrero

December 1, 2024

Abstract

This project focuses on the development of a two degrees-of-freedom (2-DoF) robotic arm, combining theoretical modeling with practical implementation. By utilizing cascade PID controllers, we achieved precise speed and position control, with the inner loop stabilizing motor speed and the outer loop governing position. The transfer functions of the motors were obtained experimentally and approximated for simplicity, providing the foundation for controller design using tools like Wolfram Mathematica and Simulink.

While simulations showed excellent results, the application to real-world motors revealed challenges such as friction, backlash, and sensor noise, which are often neglected in simulations. Despite these obstacles, the robotic arm successfully demonstrated its ability to follow trajectories, proving the validity of the mechanical and control system design. This project highlights the importance of bridging simulation and reality, providing valuable insights into improving trajectory parameters and controller gains for future iterations.

1 Introduction

A two-DoF robotic arm serves as an essential application of control system design, showcasing the principles of robotic motion and automation. Robotic arms are mechanical systems designed to perform tasks with precision, efficiency, and accuracy, mimicking human arm movements. Widely used in industries like manufacturing, assembly, and logistics, they enhance productivity, reduce errors, and improve safety.

Two-DoF arms, in particular, are effective for tasks requiring motion in two directions, such as pick-and-place operations, sorting, drawing, or material handling. Their simplicity makes them ideal for educational purposes, allowing students to explore robotics fundamentals like kinematics and control. This project aims to design and build a functional and efficient 2-DoF robotic arm, applying theoretical knowledge to create a system capable of practical applications while highlighting its relevance in automation and engineering.

The evolution of robotic arms provides valuable context for their importance in modern engineering. From Leonardo da Vinci's mechanical knight in 1495, which introduced the idea of human-like motion, to the groundbreaking Unimate in 1961, the first industrial robotic arm, each milestone has paved the way for innovation. Developments such as the Rancho arm (1963) for aiding disabled individuals and the Stanford arm (1969), which improved flexibility and control, expanded their capabilities. Today's advancements, like brain-machine interfaces, enable real-time control and adaptability, demonstrating how far robotic arms have come. This project continues that legacy, focusing on a simple yet versatile design that reflects both the history and future potential of robotics.

2 Theoretical Framework

2.1 Project's Foundation

For this project we need first to develop the control of two DC-Motors with encoder, a sensing device that makes the task of controlling speed and position easier. For the programming department Arduino provides an easy-to-go environment, which offers facilities to use hardware like the motor drive modules. For both IDE and coding language we made Arduino our option.

2.2 Approximation of Second Order Systems as First Order

In different engineering models, there are situations where second-order systems can be treated as first-order. This happens when looking at the properties of the transfer function, like the dominance of certain poles, or when analyzing the response graph, which might show that higher-order effects are minimal. For instance, in the case of DC motor control, factors like armature resistance and inductance often have time constants much smaller than the mechanical time constant associated with the motor's inertia and damping. This means that the electrical dynamics settle quickly compared to the slower mechanical response. As a result, the electrical transients can be considered negligible, allowing the system to be approximated as a first-order one dominated by the mechanical dynamics.

2.3 Creation of a PID

The methodology we applied for creating the PID controller of a G(s) plant is the following.

1. Define the System Dynamics

The process begins with identifying the plant's transfer function, which describes the system's input-output relationship. This transfer function provides the foundation for designing a controller tailored to the system's dynamics. The dynamics of the system are present in the denominator of the transfer function. The denominator will then be equaled to the desired characteristic polynomial to obtain the Damping ratio and the natural frequency of the system. This way we can obtain all the surrounding of the system for example the overshoot and the settling time.

2. Specify Desired Closed-Loop Performance

The desired behavior of the system is determined by specifying a characteristic polynomial. This polynomial encapsulates performance criteria such as:

- **Damping ratio (ζ)**: Determines the level of oscillations and overshoot.
- **Natural frequency (ω_n)**: Influences the speed of response.

These parameters define the characteristic equation of the 2nd order closed-loop system:

$$\Delta(s) = s^2 + 2\zeta\omega_n s + \omega_n^2.$$

3. Design the PID Controller

The PID controller is represented in its general form:

$$PID(s) = k\frac{(s + Z_1)(s + Z_2)}{s},$$

where:

- k: Overall controller gain,
- Z_1, Z_2: Zeros of the controller.

4. Solve for Controller Parameters

To determine the controller parameters (k, Z_1, Z_2), the open-loop transfer function, combined with the PID controller, is used to form the closed-loop characteristic equation. This equation is matched to the desired characteristic polynomial, and the parameters are solved analytically.

5. Compute PID Gains

Once the controller parameters are determined, the proportional (k_p), integral (k_i), and derivative (k_d) gains are calculated:

- k_p: Proportional gain, which responds to the current error.
- k_i: Integral gain, which corrects accumulated error over time.
- k_d: Derivative gain, which predicts future error for enhanced stability.

6. Validate and Tune the Controller

We decided to take this method among others like Zieger-Nichols or experimentally doing it with Matlab Control System Designer because it minimizes the need for extensive simulations or experimental adjustments, reducing the reliance on trial-and-error.

2.4 Application of a PID

For the 2-DoF robotic arm, the PID controller is used to regulate the DC motors controlling the arm's links:

- **Position and Speed Control:**
  - The inner loop uses a PID to control the motor speed.
  - The outer loop uses another PID to control the position, ensuring precise movement to the desired angles.
- **Cascade Control:** Speed control stabilizes the motor, while position control ensures the arm accurately follows the desired trajectory, such as drawing shapes or performing pick-and-place tasks, we create the cascade PID in cascade using the output of the firts PID controller as the input of the second one.

- **Practical Implementation:**
  - Encoders measure the motor's position and speed.
  - The error (setpoint minus actual value) is processed by the PID, which generates a control signal (voltage or PWM) to adjust the motor's movement.

By integrating PID controllers, the robotic arm achieves precise, stable, and reliable operation, essential for its real-world applications. This approach simplifies the system's design while ensuring accurate performance.

2.5 Tools

For this project we made use of Wolfram Mathematica for the algebraic calculations, as it has a powerful ability to handle analytical solutions and optimizing parameters, as the ones needed for creating a PID.

On the other hand we chose Simulink for the later stages of the project because it has a great capacity to model dynamic systems, and perform real-time simulations. Its compatibility with Arduino also made possible to apply control to the robot arm in an effective, easy-to-go way.

3 DC Motor Controller

3.1 Obtaining the transfer function

For this case we need to control both speed and position, then it's important to mention each one has their own transfer function, and this can also be proved with the units of measurements. For obtaining the speed transfer function we made use of an oscilloscope, that delivers the response graph of the plant, from which can be directly obtained the time constant, and with a couple more steps the gain.

To run the motor we made a program wrote in Arduino Language but executed in an ESP32, this because Arduino Boards doesn't have an ADC. By setting the PWM value to 244, we coded a program to make the motor function at almost its full speed, which showed at a rate of 96200 bauds in the serial monitor.

Before plotting the graph, we can expect that the inductance and resistance from the armor are negligible, reducing the total amount of parameters to consider in the model.

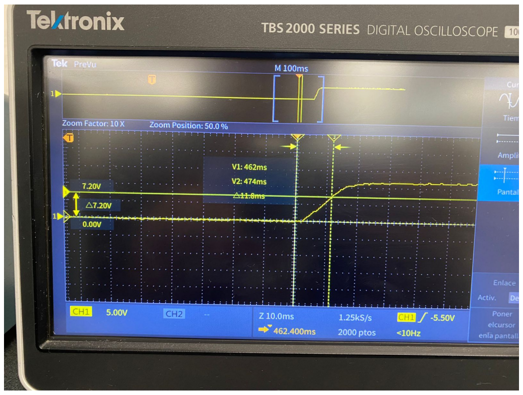

Figure 1: Graph for the first motor transfer function

Because of the form of the response graph we can conclude we have a second order transfer function that can be approximated to a first order. We can observe a second order phenomena in the way the response rises from the origin, it has a 'belly', but the rest of the response has the form of a first order plant.

According to control theory, we can define the time constant as the time it takes for the step response to rise up to 63.2 percent. In this case the percentage corresponds to 7.2 volts in a time of 0.0118 seconds. On the other hand, the serial monitor from the Arduino program printed a speed of 258 rpm. By this point is important to remember the form of a first order transfer function.

$$G(s) = \frac{K}{\tau s + 1} \tag{1}$$

To calculate K we must make a relation between the input and output from the system; how many volts are feeding the actuator, and how many revolutions per time the motor is producing. For making the units of measurement for our plant coherent, we chose to keep the time constant in seconds, as consequence, choosing to measure the output in revolutions-per-second (rps).

Placing our required relation into an algebraic expression we obtain:

$$12[V] * K = 15480[rps] \tag{2}$$

$$K = 1290[\frac{rps}{V}] \tag{3}$$

Replacing values into the general form of a first order transfer function we obtain:

$$G(s) = \frac{1290}{0.0118s + 1} \tag{4}$$

3.2 Creating a PID Controller

We used Wolfram Mathematica to create a PID controller for the transfer function developed in the section 3.1 using the equation:

$$PID(s) = k_p + \frac{k_i}{s} + k_d s$$

where k_p is the proportional gain, k_i is the integral gain, and k_d is the derivative gain. By analyzing the desired closed-loop response, the following PID controller equation was developed:

$$PID(s) = \frac{k(s + Z_1)(s + Z_2)}{s}$$

Expanding and simplifying this equation yielded the following gains:

- $k_p = 23.564$, with units $\frac{1}{\text{second}}$,
- $k_i = 203.431$, with units $\frac{1}{\text{seconds}^2}$,
- $k_d = 0.999999$, which is dimensionless.

The proportional gain, k_p, determines the reaction to the current error, while k_i eliminates steady-state error by addressing past errors, and k_d predicts future errors to enhance system stability.

To validate the design, the characteristic equation of the closed-loop system was matched with a desired second-order polynomial to ensure specific damping ($\zeta = 0.826$) and natural frequency ($\omega_n = 14.2629\,\text{rad/s}$).

3.3 Transfer function for a Second DC Motor

Replicating the methodology explained in the section 3.1, we used an oscilloscope to graph the angular speed of the motor, then from the step response, obtaining the time constant and finally generating an equation for calculating the gain.

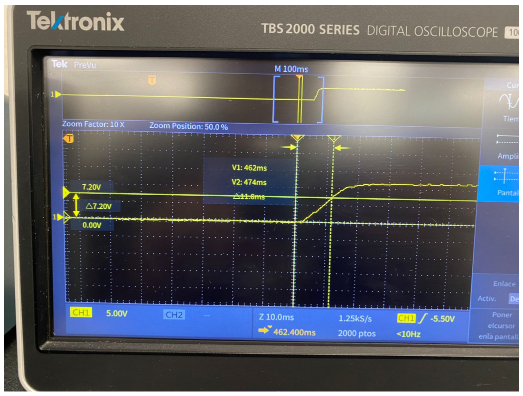

Figure 2: Graph for the second motor transfer function

For this motor the step response rise up to 7.2 V in a time of 0.0098 seconds, running at a speed of 250 rpm (15000 rps), thus, we obtain the following equation for calculating gain:

$$12[V] * K = 15000[rps] \tag{5}$$

$$K = 1250[\frac{rps}{V}] \tag{6}$$

Therefore our transfer function for the second DC motor is:

$$G(s) = \frac{1250}{0.0098s + 1} \tag{7}$$

3.4 Creating a Cascade PID Controller

For creating a position PID control, we, in part, repeated what was done in Subsection 3.2, but in this case, we created a Cascade PID. This approach leverages two nested PID controllers, with the inner loop dedicated to speed control and the outer loop to position control.

The first step involved creating a closed-loop system for speed control by combining the motor's transfer function with the speed PID controller. This inner loop stabilizes and regulates the motor's speed, ensuring rapid and precise adjustments to disturbances. The resulting closed-loop transfer function for the speed system forms the "new plant," which acts as the basis for the outer position control loop.

For the position control loop, a second PID controller was designed. The plant for this controller is the closed-loop system of the speed control (new plant). The PID controller for position control was structured as:

$$PID_{\text{position}}(s) = k\frac{(s + Z_3)(s + Z_4)}{s},$$

where k is the gain, and Z_3, Z_4 are the zeros of the position controller.

The characteristic polynomial for the closed-loop system, including the position PID controller, was matched to a desired third-order characteristic equation to ensure the system meets specified transient and steady-state performance criteria. The desired polynomial was defined as:

$$\Delta(s) = s^3 + 2\zeta\omega_n s^2 + \omega_n^2 s + p_3,$$

where:

- ζ: Damping ratio,
- ω_n: Natural frequency,
- p_3: Additional pole to increase system stability.

The denominators of the closed-loop system and the desired polynomial were expanded and their coefficients equated. Using this method, the values for k, Z_3, and Z_4 were determined as:

$$k = 7.2727, \quad Z_3 = 19.1868 - 75.7177i, \quad Z_4 = 19.1868 + 75.7177i.$$

Expanding the numerator of the position PID controller yielded the final proportional (k_p), integral (k_i), and derivative (k_d) gains as:

- $k_p = 279.079$,
- $k_i = 44,372.9$,
- $k_d = 1.0$.

4 2DoF Robot Arm

4.1 Creating the plant

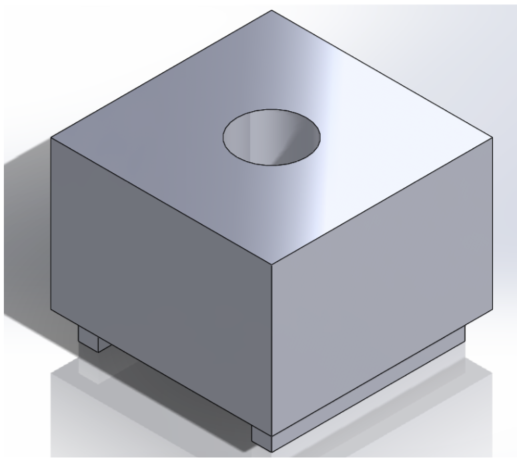

Figure 3: The base structure

The base provides stability to the entire structure and anchors the arm; it houses the first motor for the rotational movement of the first link of the arm. The base allows the arm to remain stable, even when extended. And the circular cut in the center allows the motor to snap in and not have any play. The height of the base was planned so that both the encoder and the motor shaft come out, allowing the encoder to be connected to their respective connections and the motor shaft to be connected to the first link.

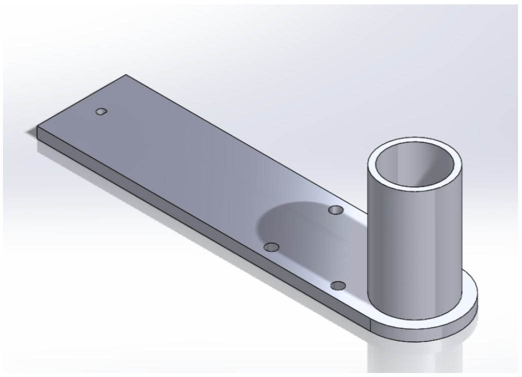

Figure 4: The first arm link

The motor is connected to the first arm, responsible for transferring the movement from the base to the second link. The length ensures that the reach of the arm is sufficient for its application. This arm will have another support for the second motor, the height of the support will allow the encoder and the motor shaft to protrude as well. To ensure that the movement of this link is correct and does not produce much moment when mounting the second motor, it was decided to implement an idler wheel, preventing the piece from not being able to support the weight and bending.

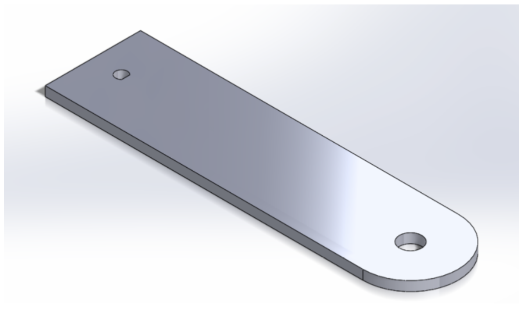

Figure 5: The second arm link

The second link was designed solely to hold what would be the pen that would draw the desired figure. However, the use of the idler wheel only allows the movement of this link to be from 90 to -90 degrees.

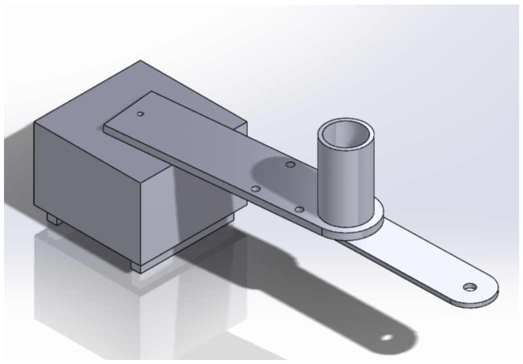

Figure 6: Final assembly

The assembly ensures that each joint and motor shaft are precisely aligned to ensure smooth movement. The system has limitations in movement due to the wiring and the use of the idler wheel. However, for the use for which it was designed, the system has the necessary characteristics.

4.2 Implementation of DC Motors

In the electronic part, as already mentioned, two motors with encoder connected to an H bridge were used to control the direction of rotation through the signals that the Arduino sends to the enable pins. An Arduino Mega was used since, as there are two motors, we needed four interruptions, which the Arduino Uno did not allow us to do since it only has two pins capable of doing that, which are pins 2 and 3. While the Mega has pins 2, 3, 18, 19, 20, 21.

4.3 Digital program

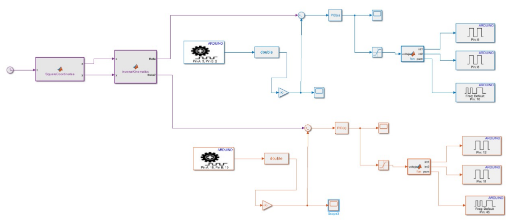

Figure 7: Final program

The programming was done with the help of Arduino libraries in Simulink. Which helps us to use blocks such as the PID to implement them in a simpler way in our system. The communication between Arduino and Simulink was essential to achieve better control in the movement of our motors. For programming such as the coordinates where the system is directed, the inverse kinematics and the direction of our motors; we use the block called Matlab functions. This block allows us to generate lines of code and assign the output we want so we can introduce it to our program generated with blocks.

To obtain the reference of the position in which the motor is located, we use the encoder function for Arduino. This allows us to obtain the changes produced by the encoder signals. In order to obtain a better resolution we use the double function which will allow us to obtain a X4 resolution. The output of these blocks will be given in pulses, so we add a gain which will allow us to go from pulses to degrees. This gain is the division between 360 and the pulses obtained from the enconder when the motor shaft makes a complete turn.

This reference is introduced to a sum to which the desired set point is also introduced. This will allow us to obtain the error that is obtained by subtracting the set point from the obtained reference. This error is connected to the PID block which is found with the gains obtained previously. Later it is introduced to a saturation in which it is defined that the voltage range with which it can operate is from -12 to 12 volts. This voltage is introduced to a matlab function in which the direction of rotation of our motor is defined. From this function three outputs are produced which are connected to the Arduino pins in charge of controlling the motor through the H Bridge that is used.

4.4 Simulink Blocks Specifications

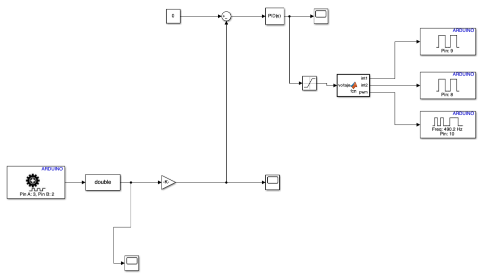

Figure 8: Control program for one motor

First Block

The first code used in the first block is the following:

```
function [x, y] = SquareCoordinates(t)
    vertices = [6, -2; 8, -2; 8, 2; 6, 2];

    numVertices = size(vertices, 1);

    cycleDuration = 5;

    phaseTime = mod(t, cycleDuration);
    vertexIndex = floor((phaseTime / cycleDuration) * numVertices) + 1;
    vertexIndex = mod(vertexIndex - 1, numVertices) + 1;

    x = vertices(vertexIndex, 1);
    y = vertices(vertexIndex, 2);
end
```

This defines that cyclic coordinates of a square are generated according to the time provided by the internal clock of the Simulink, which is the input of this block. The variable `numVertices` contains an array in which the coordinates to which the system will be directed are established. It defines how long the cycle will last in which the square will be made. Finally, a variable is defined to ensure that the cycle is restarted and starts correctly, to determine the outputs of our system which will be connected to the inverse kinematics.

Second Block

The outputs of the generated coordinates are the inputs of the second code which is responsible for converting these coordinates to degrees in order to generate the setpoint of our system and for it to generate the desired square. The code is the following:

```
function [theta, theta2] = inverseKinematics(x,y)
    a1 = 5.0;
    a2 = 5.0;

    D = (x^2 + y^2 - a1^2 - a2^2) / (2 * a1 * a2);

    if abs(D) > 1
        error("Invalid D value. Ensure the target point is within reach.");
    end

    theta2 = atan2(sqrt(1 - D^2), D);
    theta = atan2(y, x) - atan2(a2 * sin(theta2), a1 + a2 * cos(theta2));
    theta2 = (theta2 * 180) / pi;
    theta = (theta * 180) / pi;
end
```

This code uses inverse kinematics and helps us generate the movement of our two motors by defining the degrees at which they have to move.

Last Block

The code to define the direction of the motor is the following:

```
function [intl, int2, pwm] = fcn(voltage)
    if voltage > 0
        intl = 1;
        int2 = 0;
    else
        int1 = 0;
        int2 = 1;
    end
    pwm = 100;
end
```

First, it is defined that this block will have three outputs, which will be the output to activate the enable of the first motor, the output for the enable of the second, and the PWM with which our motors will work. In the code, it is defined that when the value of the voltage or the input that is in the block is greater than 0, the signal will be sent to the H-bridge to turn in one direction. When this condition is not met, the signal will be sent for the motor to turn in the opposite direction. Finally, only the PWM for the system is defined.

5 Analysis of Results

During the early stages of development, we successfully designed and tested system models, including plant dynamics and controllers, using Simulink. These simulations demonstrated excellent results, achieving the desired system performance with precision. However, when these designs were applied to the real-world motors, the expected outcomes were not realized. This gap between simulation and practical implementation highlighted the challenges of transitioning from an idealized environment to real-world conditions, where physical and environmental complexities are unavoidable.

Some of the factors that are left-out by digital simulations, and that could explain the differences are friction, backlash, sensor noise, and load variations. Additionally, the simulations assumed ideal actuators and sensors, neglecting the delays, finite resolution, and noise inherent in real hardware.

The last discussed phenomena had as a consequence the shift from a theoretical to experimental approach, which made the last stages of application a lot more trial-and-error.

The Robot Arm proved to apply correctly the two degrees of freedom. While the mechanic department of the project proved to be successful, the trajectory parameters in Simulink are the improvable part of the project. The coordinates implemented to draw different figures showed that the arm could follow correctly the vertices, but it was unstable when drawing the edges. This also could have a relation with the gains of the motor installed for the movement of the second arm link.

The experimental gains used for the Simulink blocks demonstrates why is important in a control system design to dedicate more time to prepare a solid theoretical backup, so that time can be saved instead of trying to tune with random values. With a stronger core of investigation we could be able to move through a specific range of gains giving a more professional explanation to each of them.

6 Conclusion

The 2-DoF robotic arm project serves as a testament to the integration of theoretical concepts and practical engineering. By employing PID controllers within a cascade control structure, the project achieved significant advancements in both speed and position control. The use of tools like Wolfram Mathematica and Simulink facilitated precise modeling and controller design, ensuring a strong foundation for the control system. However, the transition from simulation to real-world implementation revealed critical challenges, including friction, sensor noise, and actuator limitations. These discrepancies highlighted the importance of accounting for physical non-idealities in control system design.

While the mechanical assembly of the robotic arm proved robust and effective, issues with trajectory stability, particularly along edges, indicated the need for further optimization of controller gains and trajectory parameters. These findings underscore the necessity of a strong theoretical foundation combined with iterative testing to refine system performance. Moving forward, incorporating advanced simulation models that include physical non-idealities and dedicating additional time to theoretical research could enhance the efficiency and accuracy of similar projects. Overall, this project demonstrates the potential of cascade PID control in robotics and provides valuable insights for future endeavors in control system design and implementation.

7 Link for the video of demonstration

https://drive.google.com/file/d/1JOGprkGCAcTE9IUFmGpq4ghyxZcff6FB/edit

References

[1] Universal Robots. (n.d.). Robotic arm. Retrieved December 1, 2024, from https://www.universal-robots.com/in/blog/robotic-arm/

[2] Devonics. (n.d.). The evolution and applications of robotic arms in industry. Retrieved December 1, 2024, from https://www.devonics.com/post/the-evolution-and-applications-of-robotic-arms-in-industry?srsltid=AfmBOorMGSqMI9mKUAkPB7Mnom9xdAQzdZn9ZJwpCa8EbX6IDntFXB6nc

[3] LibreTexts. (n.d.). *An electro-mechanical system model*. In *Introduction to control systems*. Retrieved December 1, 2024, from https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Introduction_to_Control_Systems_(Iqbal)/01:_Mathematical_Models_of_Physical_Systems/1.04:_An_Electro-Mechanical_System_Model

[4] Drela, M. (n.d.). *Motor theory for QPROP*. Retrieved December 1, 2024, from https://web.mit.edu/drela/Public/web/qprop/motor2_theory.pdf

[5] Ogata, K. (2010). *Modern control engineering* (5th ed.). Pearson.

[6] Nise, N. S. (2011). *Control systems engineering* (6th ed.). Wiley.