Project Report: Serial Arm Control in Real Time

Course Code: MEE428 Real Time Control

Group Members: Mehmet Akif Demirlek, Mehmet Berke Parlat

Instructors: Özgün Başer

Introduction

The primary goal of this project is to control a two degrees of freedom serial arm mechanism using MATLAB Simulink and Arduino hardware. This involves creating a block diagram in Simulink, deploying it to an Arduino board, and adjusting PID parameters in real-time using OptiTrack cameras to observe the end effector's motion.

Objectives

- Implement motion control for a serial arm mechanism.
- Observe the effects of adjusting PID controller gains in real-time.
- Design and utilize a MATLAB App for real-time control and visualization.

Methodology

Hardware and Software Requirements

Software:

- MATLAB
- MATLAB Simulink Support Package for Arduino Hardware
- Arduino IDE

Hardware:

- Arduino UNO
- Potentiometer
- DC Motor with Encoder
- DC Motor Driver

Step-by-Step Implementation

Reading Potentiometer and Driving DC Motor in Real Time (Homework 1)

Objective: Read the analog input value of a potentiometer and drive a DC motor in real time.
Procedure:
- Connect a potentiometer and a DC motor driver to appropriate pins of Arduino UNO.
- Use MATLAB Simulink to read the analog input value of the potentiometer in real-time.
- Set the DC motor voltage according to the 10-bit analog value of the potentiometer.
- Display the 10-bit analog value and its corresponding voltage value on real-time plots.

Reading Encoder and Driving DC Motor in Real Time (Homework 2)

Objective: Enhance the previous setup by incorporating encoder pulse counting.
Procedure:
- Add an encoder to the DC motor setup and connect it to the Arduino UNO.
- Use an external interrupt block in Simulink to count the encoder pulses and determine the motor's real-time velocity.
- Display the 10-bit analog value, PWM percentage, encoder pulses, and determined velocity on real-time plots.
- Implement a bi-directional absolute position real-time graph.

Reading Encoder and Observing Data on MATLAB App (Homework 3)

Objective: Create a MATLAB App to interface with the Arduino and display motor position data.
Procedure:
- Develop a MATLAB App with buttons to start and stop the connection with Arduino.
- Implement serial communication to read motor position data from Arduino and display it in real-time on the App.
- Ensure the App can visualize the data in a user-friendly manner.

DC Motor Position Control with Encoder/Potentiometer (Homework 4)

Objective: Implement a PID controller for DC motor position control using a potentiometer or encoder.
Procedure:
- Use the feedback sensor (potentiometer or encoder) to read the position of the DC motor.
- Implement a PID controller in Simulink to control the motor position.
- Tune the PID controller in real-time and display the reference and actual positions on plots.
- Show at least three different plots representing two bad tunings and one fine tuning.

Final Project: Serial Arm Control in Real Time

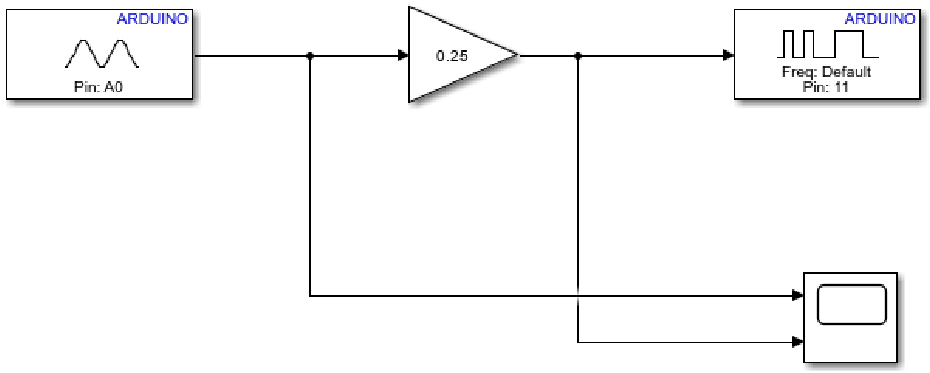
Objective: Control a serial arm mechanism with real-time adjustments of PID parameters.
Procedure:
- Design and implement a Simulink model to control the serial arm mechanism.
- Develop a MATLAB App with a GUI that allows real-time adjustment of PID control parameters and visualization of the serial arm's path.

- Use OptiTrack cameras to track the end effector and verify the motion paths.
- The GUI should include controls for connecting/disconnecting Arduino, starting/stopping the motor, and adjusting PID gains.

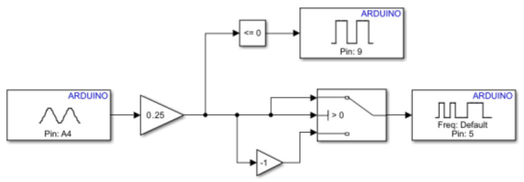
Results

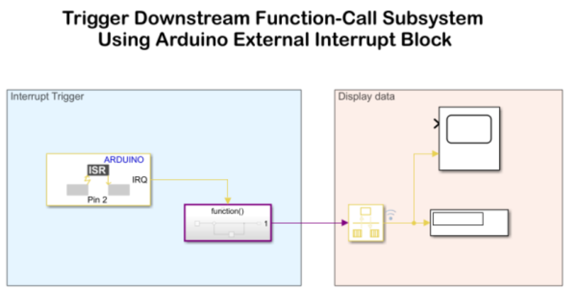
Simulink Design and Implementation

Controlled the velocity of the DC motor using a potentiometer by converting the analog signal to a digital signal.
Managed the direction and speed of the motor by comparing the signal with a threshold and utilizing a switch block to handle positive and negative values.
Counted encoder pulses using an external interrupt block and converted them to motor position in degrees.
Designed a PID controller that minimized the error between the desired and actual positions of the motor.
Developed a MATLAB App that provided a real-time interface to control and visualize the serial arm mechanism.

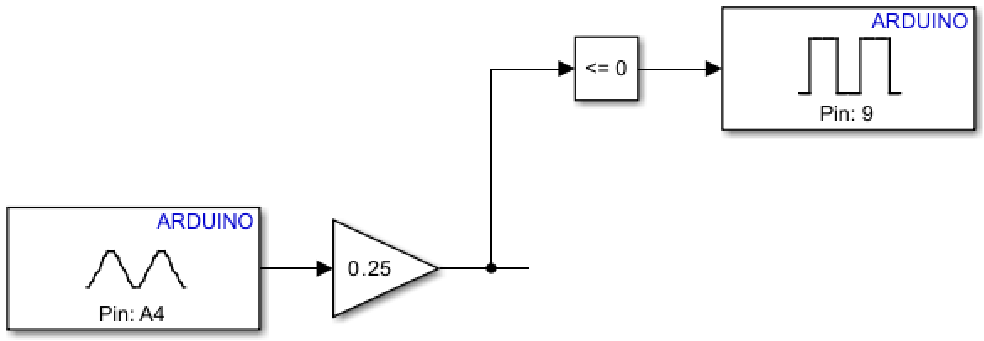


First, we controlled the velocity of the DC motor using a potentiometer. We converted the analog signal that we received from Arduino into a digital signal by multiplying it with 0.25 gain. Then, we observed that the motor velocity changed according to the magnitude of this digital signal.

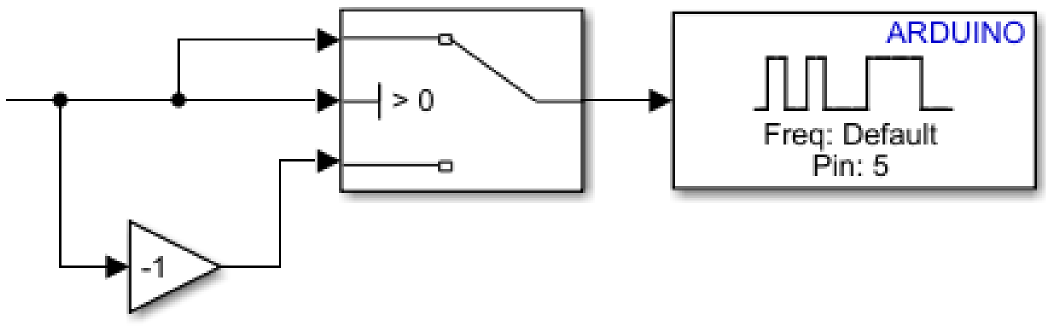



Trigger Downstream Function-Call Subsystem
Using Arduino External Interrupt Block
Interrupt Trigger
ARDUINO
ISR
IRQ
Pin 2
function()
Display data

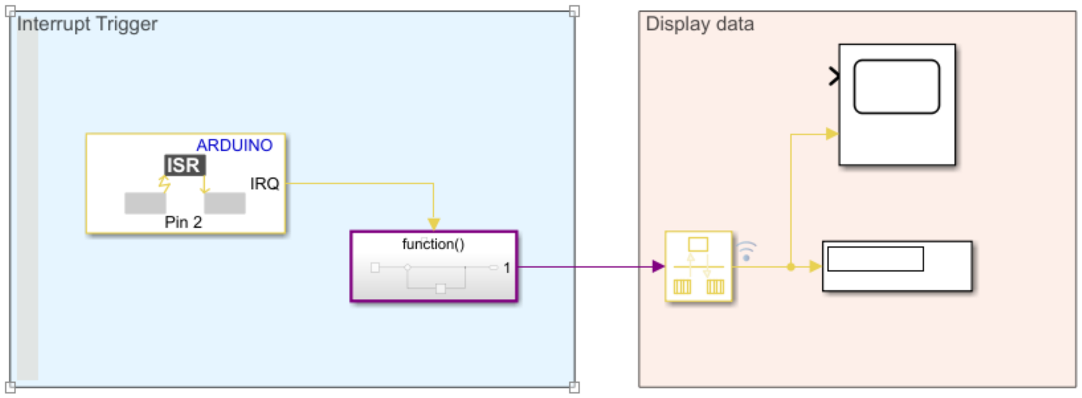
We controlled the direction and speed of the motor using a potentiometer. It was also used to count the encoder pulses of the dc motor with the external interrupt block.



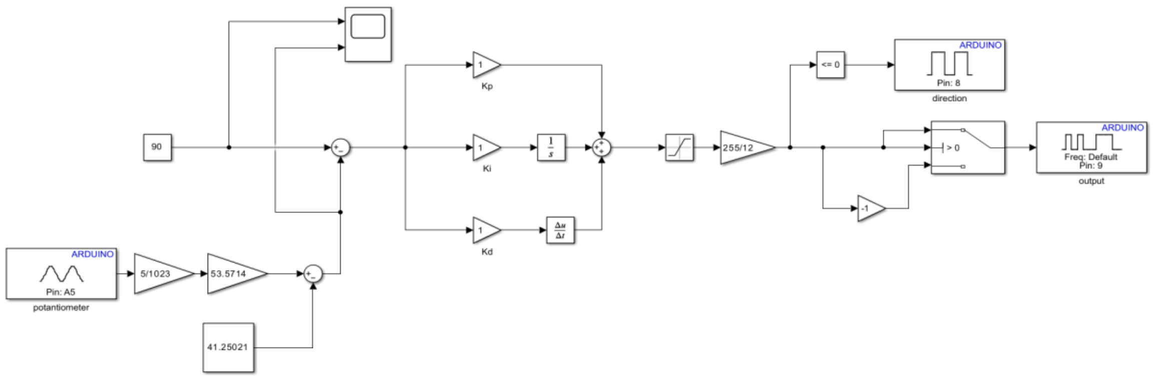
After converting the analog signal to digital signal, we compared this signal with the "compare to zero" block. This block checks whether the incoming signal is less than zero and the signal is sent to the Arduino digital output pin. If the compared signal is less than zero, the motor starts rotating in one direction (clockwise or counterclockwise); If the signal is greater than zero, the motor starts rotating in the other direction.



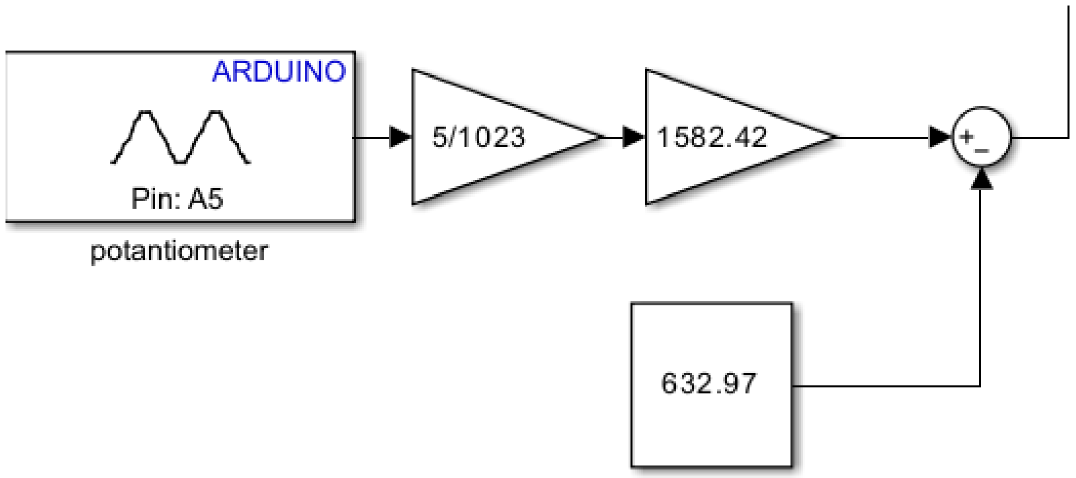
The switch block used here performs a kind of absolute value mathematical operation. In other words, it ensures that the incoming digital signal (negative or positive) passes through the switch block and comes out as a positive signal. This signal is sent to the PWM output of the motor and the speed of the motor is determined depending on the amplitude of the signal.



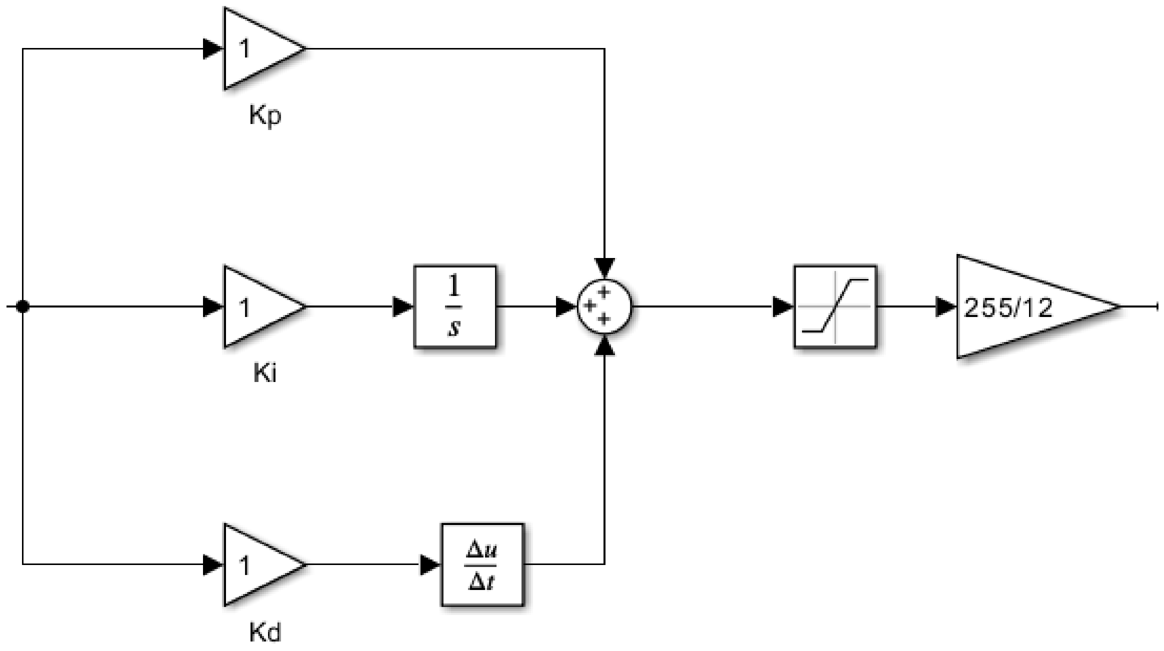
Here, we aimed to count the number of DC motor encoder pulses using the external interrupt block. At each rising edge, we counted the number of pulses with the subsystem within the Interrupt trigger block.



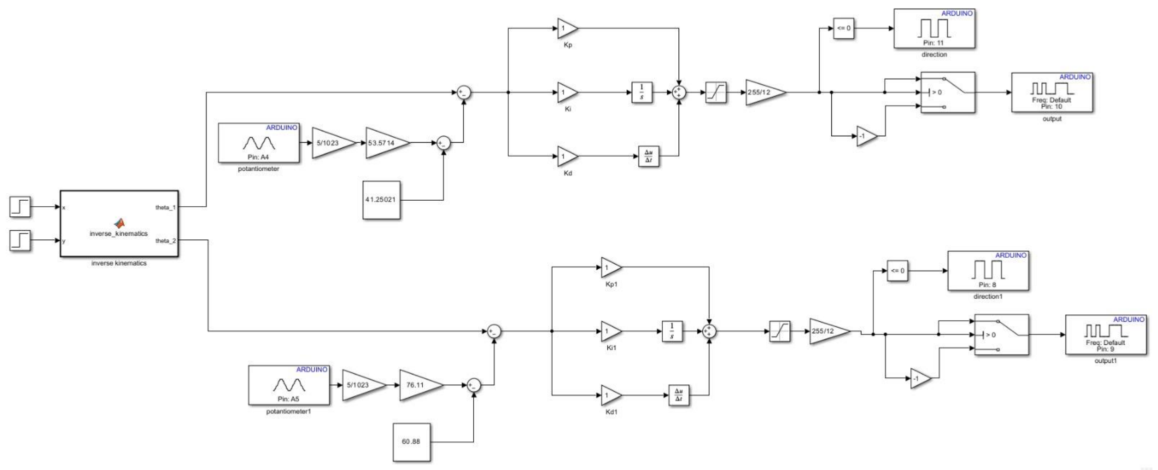
Here we used a rotary potentiometer that rotates synchronously with the rotation of the motor. This potentiometer acts as the feedback of the system. In addition, we wanted to ensure that the motor reaches the desired position with the least error and in the fastest way by evaluating the signal coming from the feedback with the PID controller.

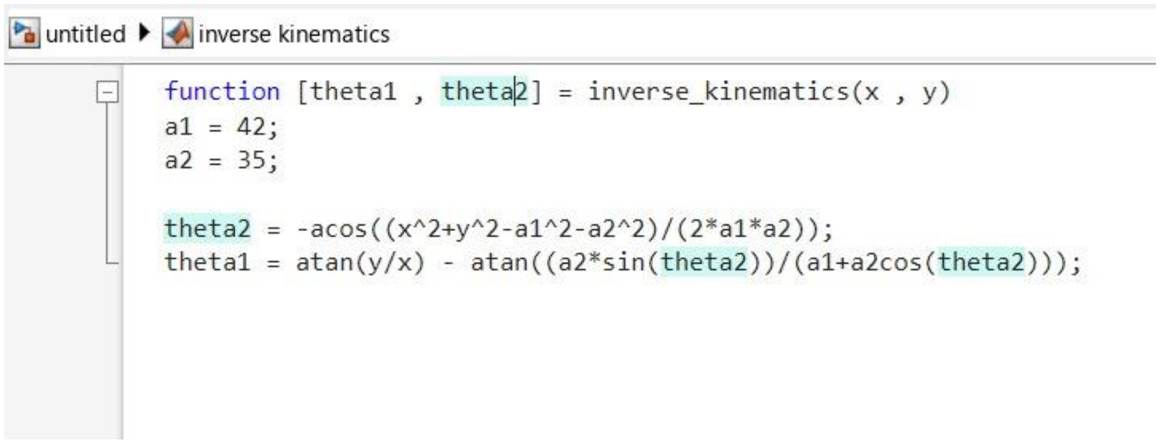


This is the feedback of the system. First, we converted the analog signal coming from the potentiometer output into a digital signal. Next, we added the voltage-degree calibration gains of the potentiometer. Thus, we observed what position the engine was in.



Here, we designed a PID controller that evaluates the error between the feedback signal from the potentiometer and the reference angle. Next, we converted the controller output signal to a PWM signal with a gain of 255/12.



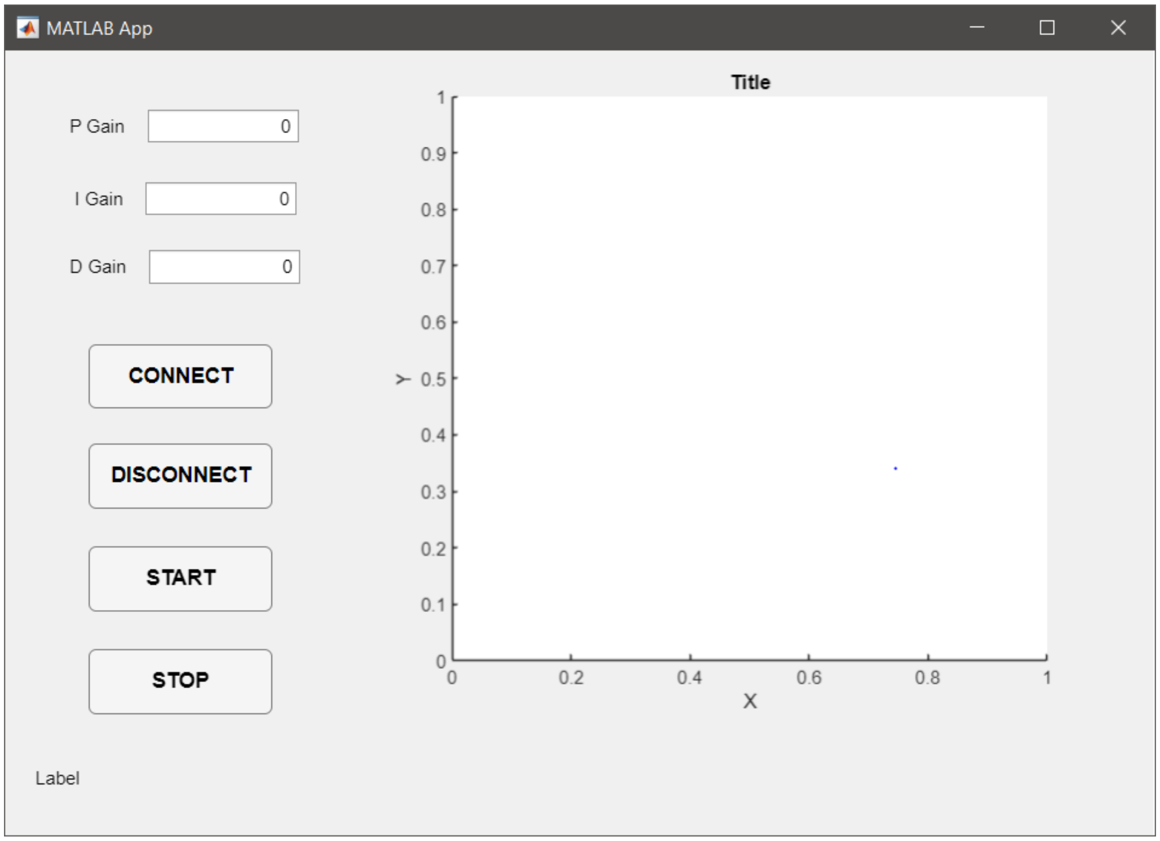


```
function [theta1 , theta2] = inverse_kinematics(x , y)
a1 = 42;
a2 = 35;

theta2 = -acos((x^2+y^2-a1^2-a2^2)/(2*a1*a2));
theta1 = atan(y/x) - atan((a2*sin(theta2))/(a1+a2cos(theta2)));
```

MATLAB App Interface

The MATLAB App included UI controls such as buttons for connecting/disconnecting Arduino, starting/stopping the motor, and text boxes for displaying real-time data.
The app allowed real-time intervention of control gains and visualization of the serial arm's reference and actual paths.



Here, we designed a a MATLAB App that has the ability to visualize the serial arm environment with reference path and real path, and ability to intervene the control gains in real-time.

```
classdef MotorControlApp < matlab.apps.AppBase

  % Properties that correspond to app components
  properties (Access = public)
    UIFigure        matlab.ui.Figure
    StartButton      matlab.ui.control.Button
    StopButton       matlab.ui.control.Button
    ReadDataButton    matlab.ui.control.Button
    TextBox          matlab.ui.control.EditField
    SerialPort       matlab.serial.SerialPort
  end

  methods (Access = private)

    % Button pushed function: StartButton
    function StartButtonPushed(app, ~)
      try
        if isempty(app.SerialPort) || strcmp(app.SerialPort.Status, 'closed')
          app.SerialPort = serialport('COM3', 9600);
          configureTerminator(app.SerialPort, "CR/LF");
          configureCallback(app.SerialPort, "terminator", @app.readSerialData);
          app.TextBox.Value = 'Connection Opened';
        end
      catch e
        app.TextBox.Value = ['Error: ', e.message];
      end
    end

    % Button pushed function: StopButton
    function StopButtonPushed(app, ~)
      try
        if ~isempty(app.SerialPort) && strcmp(app.SerialPort.Status, 'open')
          delete(app.SerialPort);
          app.TextBox.Value = 'Connection Closed';
        end
      catch e
        app.TextBox.Value = ['Error: ', e.message];
      end
    end

    % Callback function to read data from serial port
    function readSerialData(app, src, ~)
      data = readline(src);
      app.TextBox.Value = data; % Update the text box with received data
    end

  end

  % Component initialization
  methods (Access = private)

    % Create UIFigure and components
```

```
    function createComponents(app)
      % Create UIFigure and hide until all components are created
      app.UIFigure = uifigure('Visible', 'off');
      app.UIFigure.Position = [100 100 400 300];
      app.UIFigure.Name = 'Motor Control App';

      % Create StartButton
      app.StartButton = uibutton(app.UIFigure, 'push');
      app.StartButton.Position = [100 200 100 22];
      app.StartButton.Text = 'Start Connection';
      app.StartButton.ButtonPushedFcn = createCallbackFcn(app, @StartButtonPushed,
true);

      % Create StopButton
      app.StopButton = uibutton(app.UIFigure, 'push');
      app.StopButton.Position = [210 200 100 22];
      app.StopButton.Text = 'Stop Connection';
      app.StopButton.ButtonPushedFcn = createCallbackFcn(app, @StopButtonPushed,
true);

      % Create TextBox
      app.TextBox = uieditfield(app.UIFigure, 'text');
      app.TextBox.Position = [100 150 210 22];
      app.TextBox.Editable = 'off';
    end
  end
```

```
    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = MotorControlApp
            % Create and configure components
            createComponents(app);

            % Register the app with App Designer
            registerApp(app, app.UIFigure);

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end

        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            if isvalid(app.UIFigure)
                delete(app.UIFigure);
            end
        end
    end
end
```

Conclusion

The project successfully demonstrated the control of a two degrees of freedom serial arm mechanism using MATLAB Simulink and Arduino hardware. Real-time adjustments of PID parameters were effectively managed through a custom MATLAB App, and the system's performance was validated using OptiTrack cameras. The integration of various hardware components and software tools showcased the practical application of real-time control techniques in a complex mechanical system.

Appendices

- Appendix A: Homework 1, 2, 3, and 4 details.
- Appendix B: Simulink block diagrams and MATLAB App code.
- Appendix C: Real-time plots and performance evaluation graphs.

References

- MATLAB Documentation
- Arduino Reference Guide
- OptiTrack User Manual