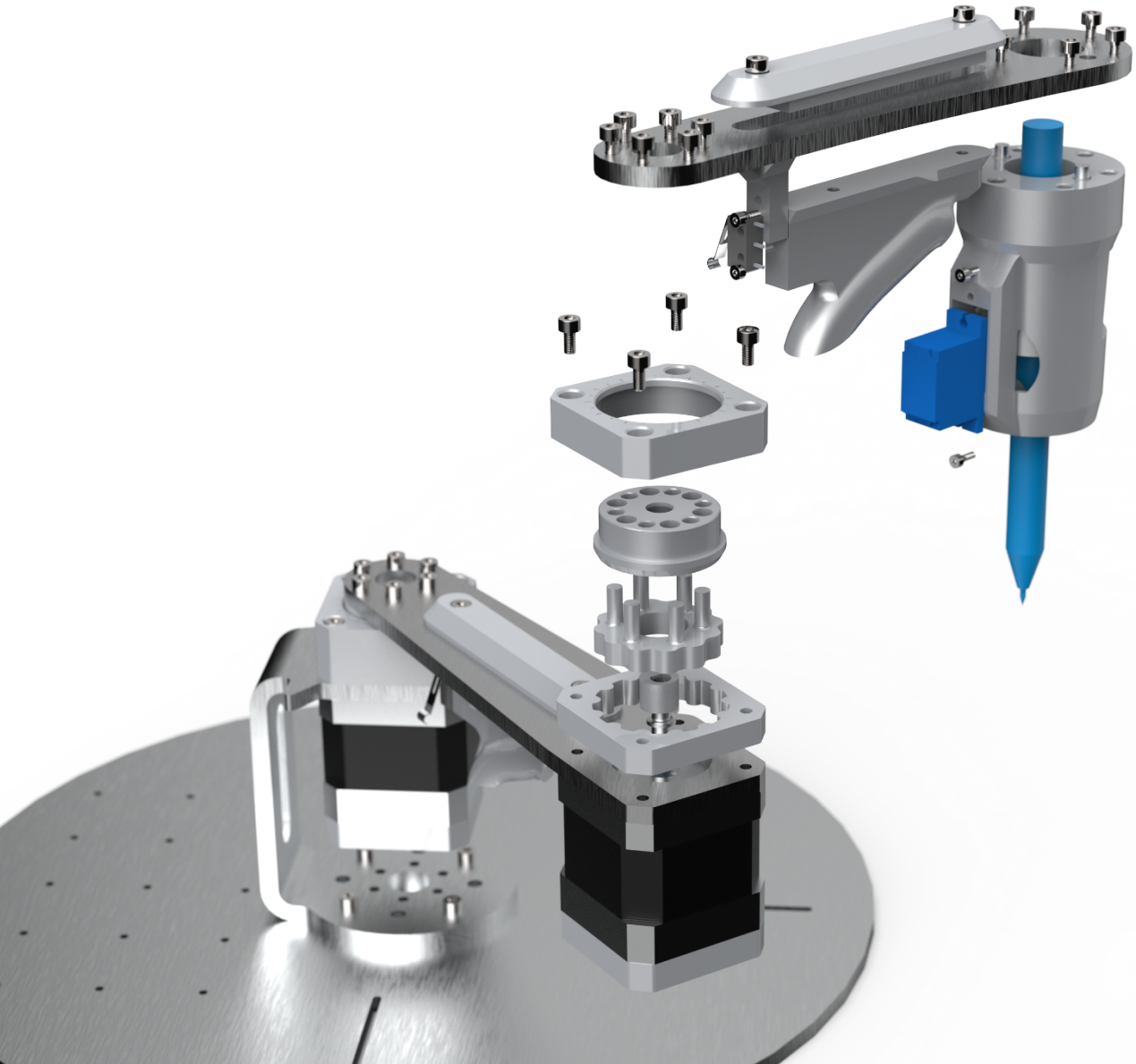
SCARA GUI Report

282778 Mechatronics



Contents

1. Introduction

1.1. Outlines and Scope

The scope of this assignment consists of the design, programming, and integration of a GUI to control a separately developed SCARA robot. Both systems should be designed coherently and work together as one functional system.

The final system has been evaluated against factors such as functionality, safety, accuracy, ingenuity, and ease of use. The GUI needs to show that the student has understood what operational requirements of a system are relevant to a robotic application as well as ensure functional and practical implementation of such.

1.2. Application and Background

SCARA (Selective Compliance Assembly Robot Arm) robots are versatile and are very popular in industry due to high effectiveness at certain tasks yet offering a simple design and straight forward integration. Due to its horizontal constraints with a single vertical moving axis at the end or beginning of the robot, it is often applied in assembly lines to pick and place or hold items. It also offers high efficiency due to the fast speed and accurate movement it can be deployed at. [1-3]

In this case, a pen to draw and write is the application to demonstrate functionality of the robot. Two rotational links (140mm length) and a translational joint (pen up/down) as the end effector have been implemented. The SCARA robot most important components are two stepper motors with self-developed cycloidal gearboxes for rotational movement, a servo motor for moving a pen up and down, sensors such as limit switches and encoders, an ESP32 as the controller, and a full aluminium frame for rigidity.

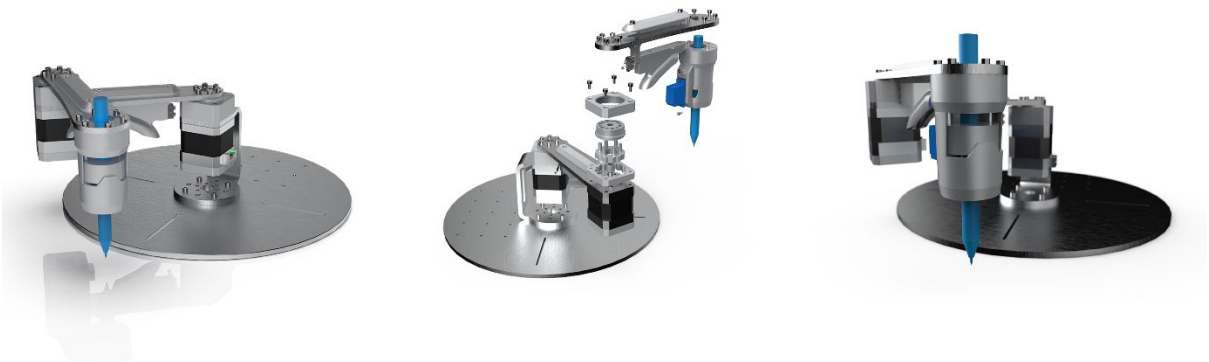


Figure 1 - SCARA Robot Renders

2. Methodology

2.1. Design Process

The design process has been driven by the project deliverables and the overall vision to create a clear and functional GUI that implements all functions required to properly control the SCARA robot. The process started with researching different interfaces currently deployed by leading companies such as ABB for industrial applications and DJI for consumer level control. [4, 5]

The following flowchart depicts the steps taken during the design process of the GUI. First a list of features was accumulated. This list was then grouped into different sections and prioritized. The layout has then been based on the sections and more space was allocated to sections that should be larger in the GUI (such as the plot). After testing for functionality and troubleshooting, the GUI has been reskinned with a custom design and smaller features such as the help and themes implemented.

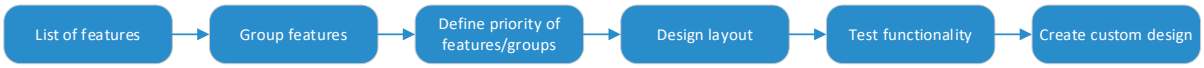


Figure 2 - Design process - flowchart

2.2. Overview

The GUI has been developed to incorporate many useful features as well as be easy and clear to interact with. The design philosophy has been inspired by a clean and modern direction.

The GUI is a web-based interface that wirelessly connects to the MCU of the robot. This allows access from several devices such as a web browser on your computer or a tablet. It also offers the ability to have several GUIs running simultaneously or the same one mirrored on a separate device for monitoring. For testing and validation, the ESP32 has been connected to a local hotspot created by a computer which allowed control from all devices connected to the same network. HTML and JavaScript have been used to implement this web interface.

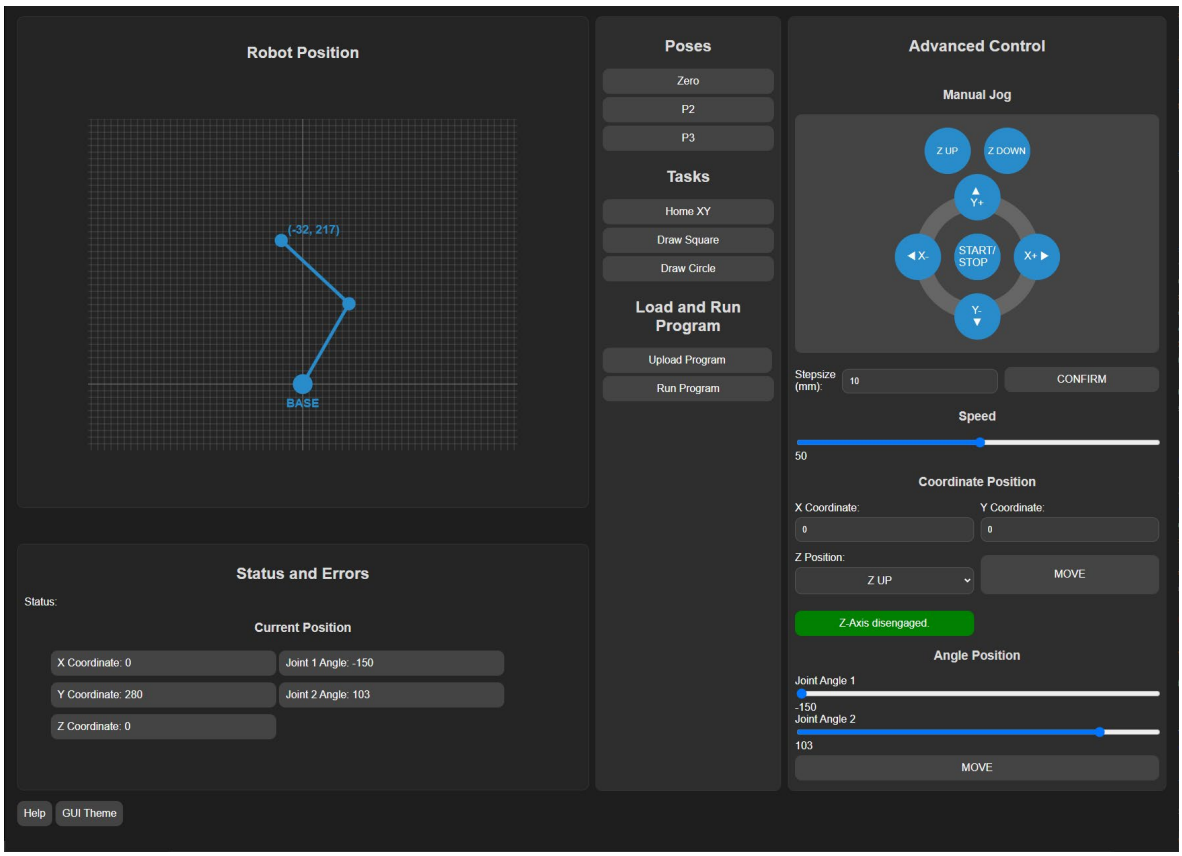


Figure 3 - GUI in web-browser

The features of this GUI include a real-time 2D Plot of the robot position, status, and error output to the user, display of current position and joint angles, available preset poses to move to and tasks to

complete, a file upload to run custom tasks, manual jogging control for axis, an e-stop, step size and speed control, manual coordinate input for moving, a help section and a GUI theme toggle.

The GUI has been divided into three sections to accommodate all features. The robot positioning section is on the left, the commands, and tasks in the middle and the manual positioning is on the right.

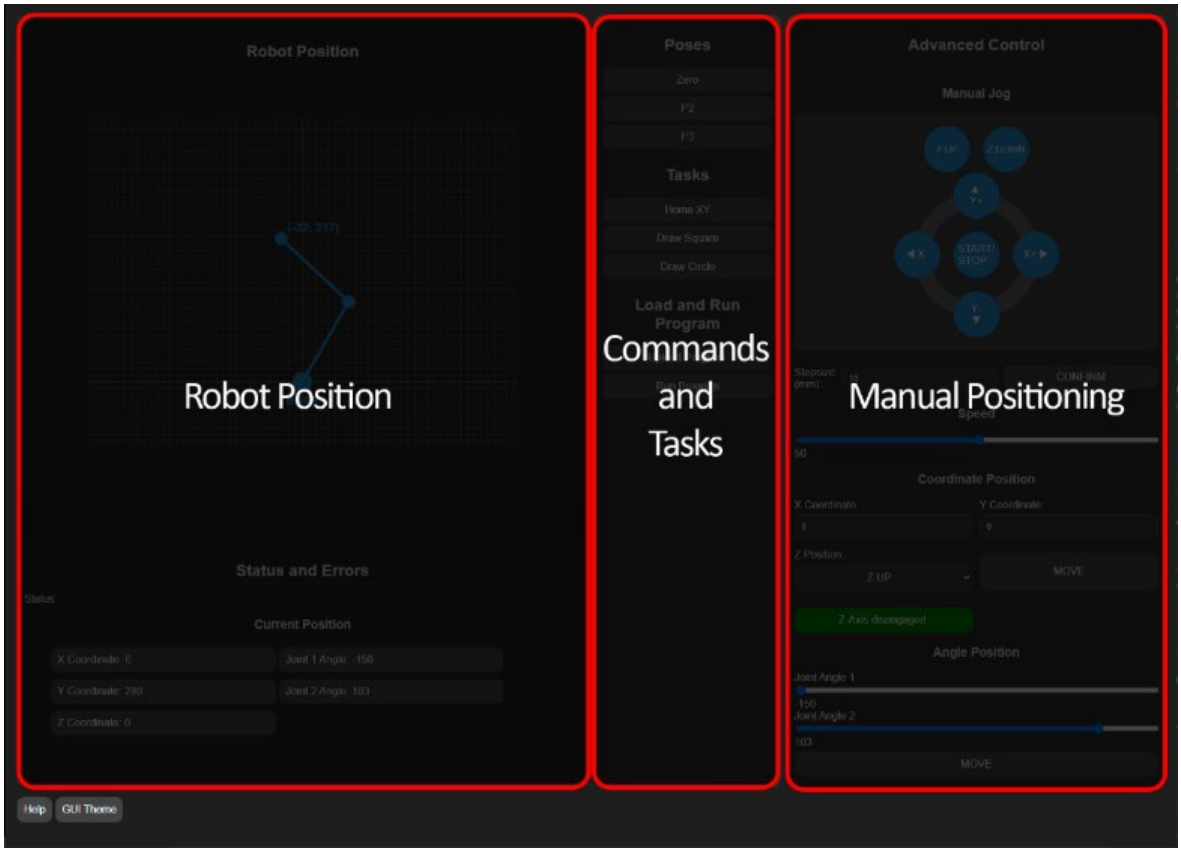


Figure 4 - GUI layout

2.2.1. Robot Position

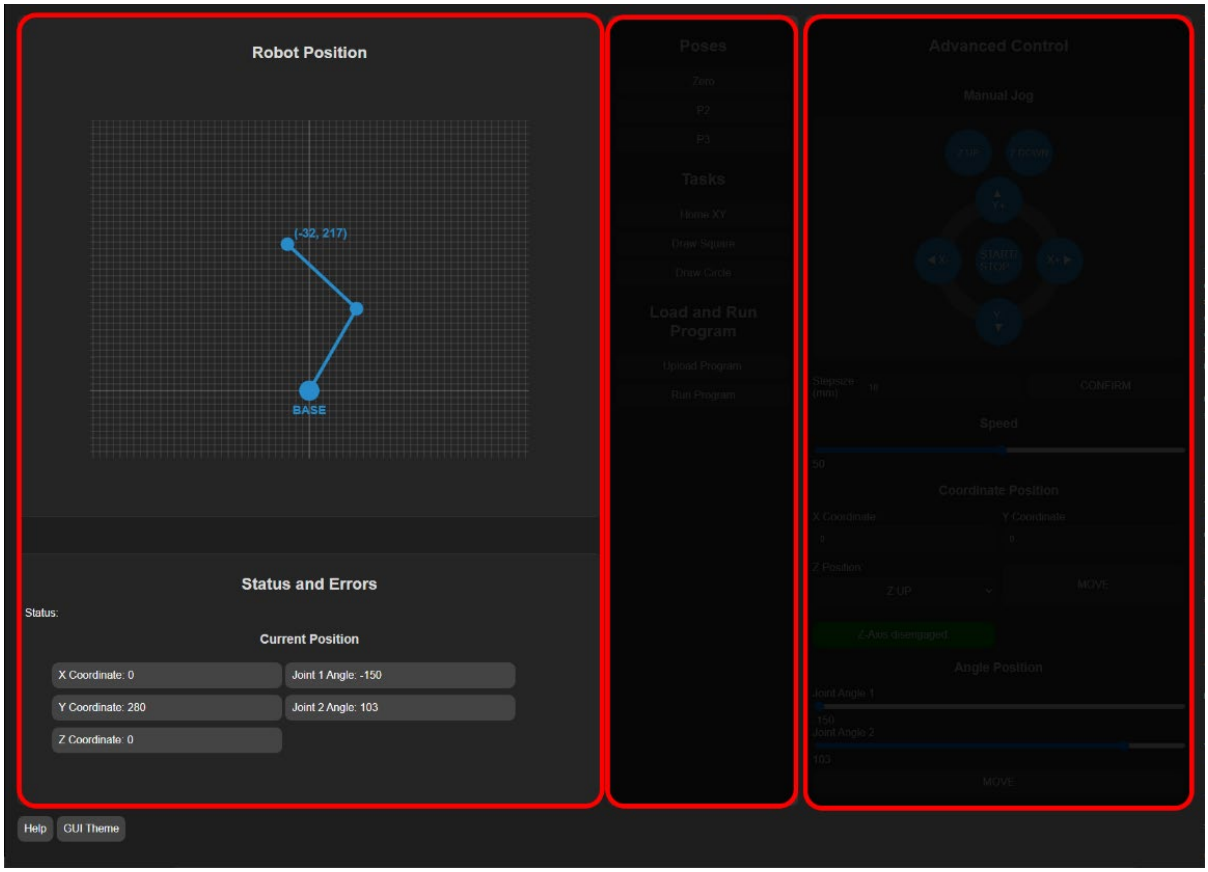


Figure 5 - GUI layout - robot position

The robot position section inherits all information on where the robot is in space. The 2D top-down plot shows the links, joints, and end-effector of the robot arm. It updates in real time when the robot is moving and displays the X, Y position of the robot at the end-effector.

The next section underneath gives the user feedback on the status of the robot and if any errors are occurring. Messages such as “Homing steppers”, “Finished homing”, “Moving to position” and “Invalid move entered” have been implemented to show exactly what the status is and why some input may not be executed.

The current robot parameters such as the XYZ coordinates, and joint angles are displayed underneath the Status and Error section.

2.2.2. Commands and Tasks

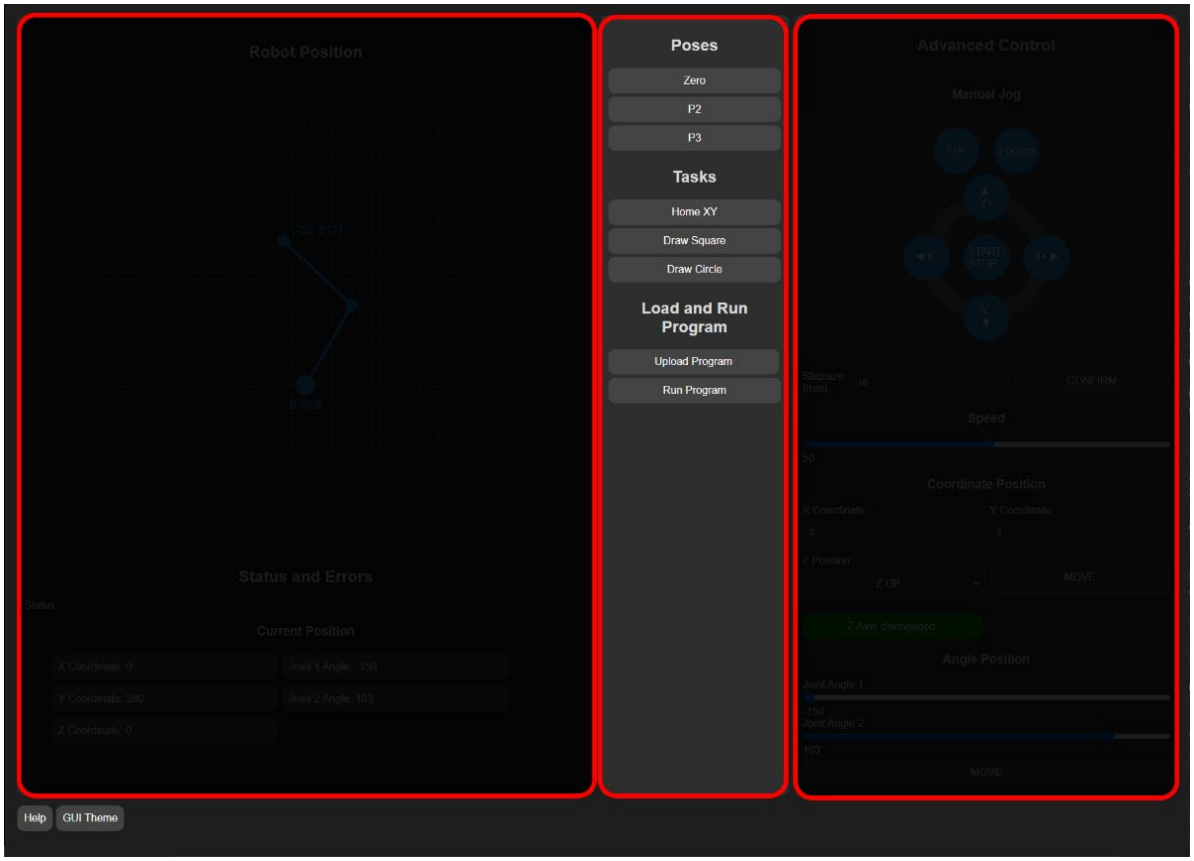


Figure 6 - GUI layout - poses and tasks panel

The commands and tasks section is divided into the “Poses”, “Tasks”, and “Load and Run Program” sections. The poses allow the user to move the robot to preset positions such as the zero position. The P2 and P3 poses are space holders for custom poses required for the user. The tasks section can execute tasks such as homing or running a specific path such as a square. The load and run program section allows the user to upload a .txt file with G-Code to run a custom task.

2.2.3. Manual Positioning

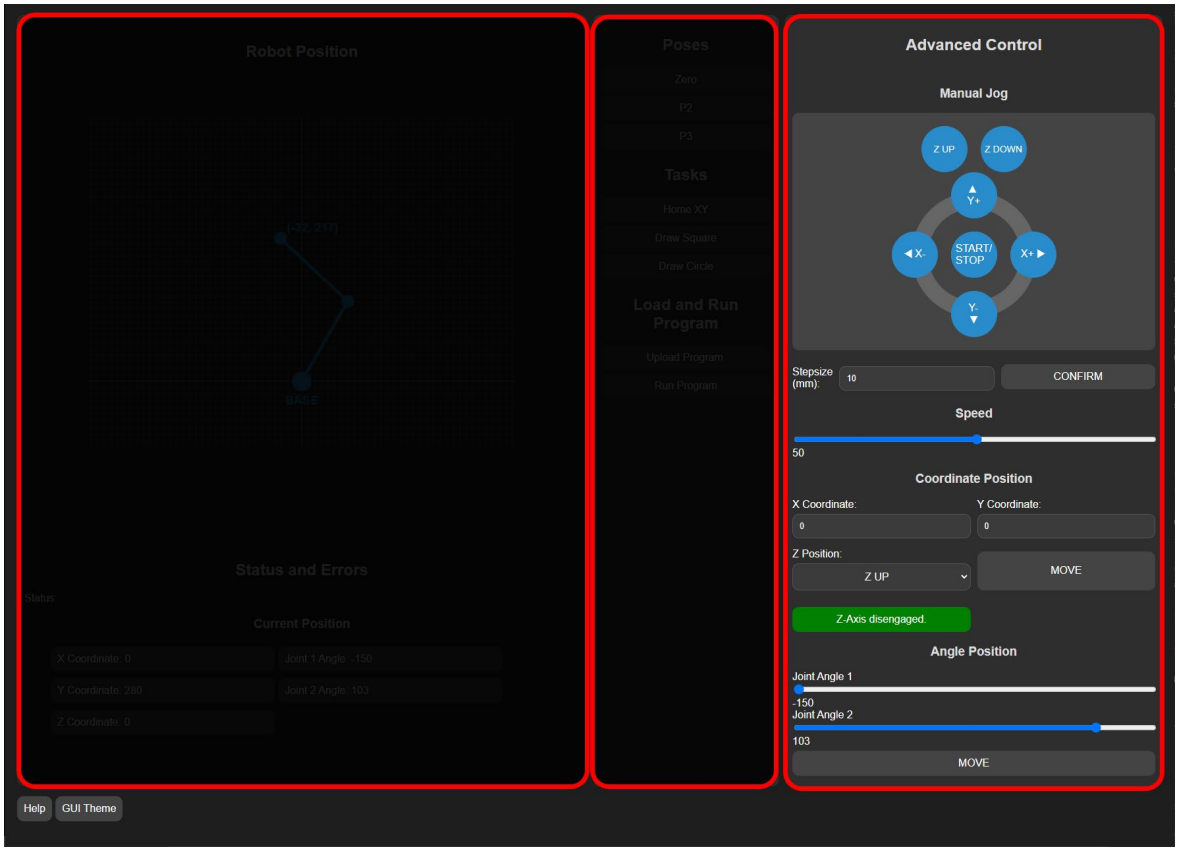


Figure 7 - GUI layout - advanced control panel

The first section allows manual jogging in XYZ space and includes an e-stop/pause button in the middle that works for not only the jogging, but all moves the robot is doing. A physical e-stop that cuts all power to the robot is also implemented.

Underneath is an input box that allows the user to input the step size of the jogging in a range from 1-50mm. The confirm button next to it must be pressed to change to the new step size.

A slider to control the speed from 1-100 (% of speed range) is implemented next.

To enter a position, two input boxes (XY coordinates) and a drop down (Z up/down) and a “MOVE” button have been added. The input boxes only allow numbers, and the position is only sent to the robot once the move button has been clicked. To increase safety and help the user, a feedback box has been placed under the Z-Axis drop down menu. This shows if the Z-axis is currently engaged or not.

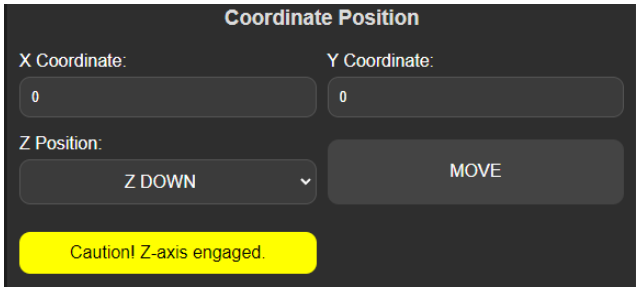


Figure 8 - Z-axis feedback

The “Angle Position” sliders with a “MOVE” button have been implemented in the GUI with a range of -180 to 180 degrees but are not yet implemented into the robot’s moving functions.

2.2.4. Help and GUI Theme

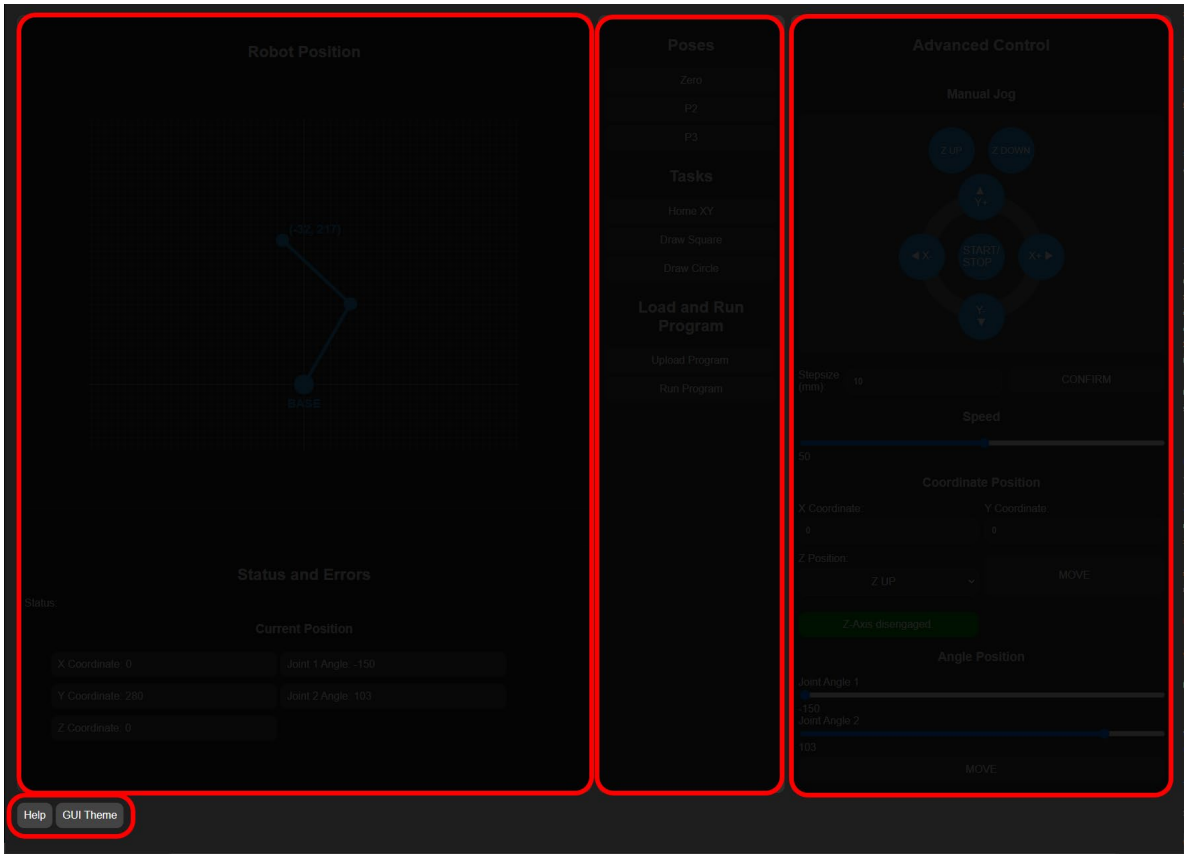


Figure 9 - GUI layout - help and theme button

A dedicated help button has been implemented to aid with usability. Whenever the button is pressed, a pop-up message appears, giving instructions on how the GUI is laid out.

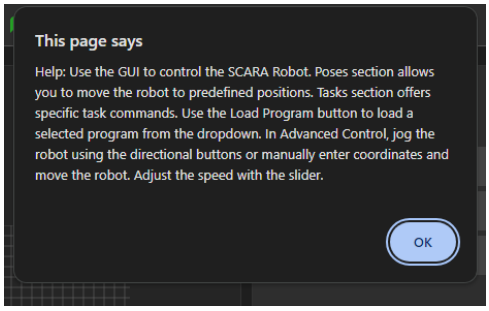


Figure 10 - Help prompt

Another way to help the user understand functionality is by hovering over buttons and other features. Each item has a unique brief description, giving insights of what it does.

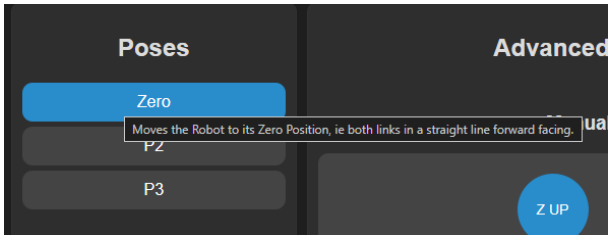


Figure 11 - Description when hovering over element

The GUI theme button allows the user to toggle between a dark and light mode of the user interface.

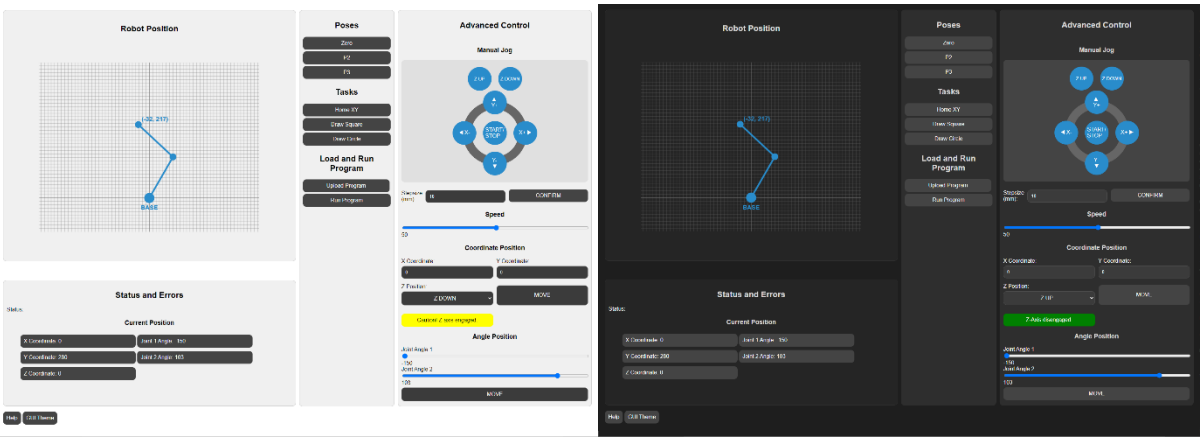


Figure 12 - GUI light and dark theme

2.3. Implementation and workings

The main_Nik.thml consist of several parts, the CSS Styles, HTML key elements and JavaScript integration. The CSS section ranges from line 8-473, HTML from 1-7 and 474-616, and JavaScript from 617-1146. The GUI uses the “Websocket.h” to establish a connection with ESP32 which then passes information back and forth from the GUI and the processing in “main.cpp”. “Scara_new.h” is used to initialize all values on start up in “main.cpp”. A flow diagram showing the program structure can be found under 7.1. Appendix A – Program Structure Flow Diagram.

2.3.1. CSS Styles

The CSS section ranges from line 8 to 473.

Global styles set the foundational look and feel of the webpage. Background and text colours are set to provide high contrast and readability, particularly in dark mode. The grid layout is established with six columns and specified row settings, to organize the structure. These global styles should ensure consistency across the entire interface.

As two styles, a dark and a light mode, have been implemented, the styles need defined individually for varying colours.

Dark mode styles are defined to provide a comfortable viewing experience in low-light environments. Background colours are set to shades of dark grey or black, while text colours are white or light grey to ensure readability. Buttons change colour whenever hovering over them to

Light mode styles cater to users who prefer a traditional bright interface. Background colours are white or light grey. Elements such as buttons and input fields have distinct borders and backgrounds to differentiate them from the rest of the content.

Common styles apply to interactive elements like buttons and containers across both light and dark modes. Buttons have consistent styling with padding, borders, and hover effects to enhance user interaction. Containers use Flexbox for aligning items within them, and grid properties for the overall page layout.

2.3.2. HTML Key Elements

The HTML section ties the CSS and JavaScript together and acts as the actual structure and body of the interface. The code has been structed based on the visual layout including all its elements.

The robot position in the top left corner utilizes a <canvas> with the dimensions of 650x500px to draw plot the robot.

```
<canvas id="robotCanvas" width="650" height="500"></canvas>
```

The section underneath showing the status and current parameters is based on the assigned variables.

```
  <p>Status: <span id="mystatus"></span></p>
      <div class="output-box-style">X Coordinate: <span id="myX">0</span></div>
      <div class="output-box-style">Joint 1 Angle: <span
id="myang1">0</span></div>
      <div class="output-box-style">Y Coordinate: <span id="myY">0</span></div>
      <div class="output-box-style">Joint 2 Angle: <span
id="myang2">0</span></div>
      <div class="output-box-style">Z Coordinate: <span
id="myservo_state">0</span></div>
```

The “Help” and “GUI Theme” buttons underneath are integrated using the <button> element.

```
  <button id="help-button" title="Press to show instructions." class="button-
group">Help</button>
  <button id="theme-toggle-button" title="Toggle light/dark theme" class="button-
group">GUI Theme</button>
```

In the middle section the poses, tasks, and load program are located. The <button> element has been utilized for such as well. the oneclick function has been used to associate certain backend functions with a button. For example, the “Zero” button is linked via oneclick=”controller()” to call the function anytime the button is pressed.

```
<button id="zero-pose"
                title="Moves the Robot to its Zero Position, ie both links in a
straight line forward facing."
                onclick="controller()">Zero</button>
    <button id="pose-2" title="Moves the Robot to programmed Position
1.">P2</button>
    <button id="pose-3" title="Moves the Robot to programmed Position 2."
onclick="drawRobotArm()">P3</button>
```

```
<button id="home-xy" title="Homes all three axis of the robot."
onclick="home()">Home XY</button>
    <button id="draw-rectangle"
                title="Draws a pre-programmed rectangle with the dimensions of xx
by xx mm starting from XY position."
```

```
                onclick="preset2()">Draw Square</button>
    <button id="draw-circle"
                title="Draws a pre-programmed circle with the diameter of xx mm
starting from XY position.">Draw
    Circle</button>
```

The file upload is done using the <input> element that opens the file explorer anytime it is pressed. The accepted file format is set to .txt. The “Run Program” button can then be used to execute the uploaded file using the “uploadFile()” function.

```
<input type="file" id="inputFile"
                title="Select program of .txt file format to be loaded into the
robot. Press 'Run Program' to execute."
                accept=".txt">
    <label class="upload-button" for="inputFile">Upload Program</label>
    <button id="run-program" onclick="uploadFile()" title="Executes the loaded
file.">Run Program</button>
  </div>
```

The manual robot control on the right side is done using generic containers which are linked to JavaScript object called “jogButtons” and the specific function to move the robot into the desired position.

```
        <div id="jog-z-up" title="Moves Z-Axis (Pen) down." onclick="servoup()">Z
UP</div>
        <div id="jog-z-down" title="Moves Z-Axis (Pen) up."
onclick="servodown()">Z DOWN</div>
      </div>
      <div class="jog-wheel">
        <div id="jog-y-pos" title="Moves in Y+ direction at for set increment per
click." class="up"
                        onclick="move1()">&#9650;<br />
          Y+</div>
        <div id="jog-y-neg" title="Moves in Y- direction at for set increment per
click." class="down"
                        onclick="move4()">Y-<br />
          &#9660;</div>
        <div id="jog-x-neg" title="Moves in X- direction at for set increment per
click." class="left"
                        onclick="move2()">&#9664; X-</div>
        <div id="jog-x-pos" title="Moves in X+ direction at for set increment per
click." class="right"
                        onclick="move3()">X+ &#9654; </div>
        <div id="jog-stop" title="Stops any movement." class="center"
onclick="e_stop()">
          <div>START/<br />
            STOP</div>
```

The step size is controlled using an input that allows numbers from 1-50 (mm). The function “vars()” controls such and is called whenever the “CONFIRM” button is pressed.

```
<label for="stepsize">Stepsize (mm): </label>
        <input type="number" id="jog_var" min="1" max="50" step="any" value="10"
name="stepsize"
                        class="input-box">
        <button id="stepsizebutton" title="Confirms stepsize change (10mm
Standard)."
                        class="stepsize-position button-group"
onclick="vars()">CONFIRM</button>
```

The speed is controlled by a slider with a range from 1-100.

```
<input type="range" id="speedslider" class="speed-slider" min="1" max="100">
      <span id="speed-value">50</span> </div>
```

To send the robot to set coordinates, the input fields for “X Coordinate”, “Y Coordinate”, and “Z Position” can be used and the “MOVE” button initiates the move using the “handleMove()” function.

```
<label for="armAngle">X Coordinate: </label>
        <input type="number" id="armAngle" min="0" max="360" step="any" value="0"
name="x-coordinate"
                        class="input-box">
      </div>
      <div>
        <label for="lineAngle">Y Coordinate: </label>
        <input type="number" id="lineAngle" name="y-coordinate" min="0" max="360"
step="any" value="0"
                        class="input-box">
      </div>
      <div>
        <label for="z-position">Z Position: </label>
        <select id="z-position" title="Moves Z-Axis (Pen) up or down. CAREFUL,
'DOWN' is engaged."
                        name="z-position" class="dropdown">
          <option value="z-up">Z UP</option>
          <option value="z-down">Z DOWN</option>
        </select>
      </div>
      <button id="go-to" title="Moves robot to specified coordinates."
class="coordinate-button"
                    onclick="handleMove()">MOVE</button>
```

The angle sliders for “Joint Angle 1” and “Joint Angle 2” with a range from -180 to 180 degrees are implemented but not currently used. The function would be similar to the coordinates where the direction would be confirmed using a “MOVE” button.

```
<label for="angle1-slider">Joint Angle 1</label>
      <input type="range" id="angle1-slider" class="speed-slider" min="-150"
max="150">
      <span id="angle1-value">0</span> </div>
    <div>
      <label for="angle2-slider">Joint Angle 2</label>
      <input type="range" id="angle2-slider" class="speed-slider" min="-150"
max="150">
      <span id="angle2-value">0</span>
      <button id="go-to" title="Moves robot to specified coordinates."
class="coordinate-button"
                    onclick="drawRobotArm()">MOVE</button>
```

2.3.3. JavaScript Integration

The JavaScript provides the execution and handling of functions and data. In particular, the WebSocket communication to send and receive status updates from the robot, the UI interactivity by using event listeners to handle tasks when the user presses buttons or moves a slider, robot control for jogging and movement, file handling for uploading and reading programs, and drawing of the robot plot.

Some of the main functions executed within this section are “drawRobotArm()”, “uploadFile()”, “movearm()”, “initWebSocket()”, and several event listeners.

The “drawRobotArm()” defines a canvas and then takes in the calculated angles from the ESP32 and converts them to radians. It then calculates the position of each link and draws the coordinate grid, links and end-effector position using OpenCV functions.

```
        function drawRobotArm() {
            const canvas = document.getElementById('robotCanvas');
            const ctx = canvas.getContext('2d');
            const angle1 = document.getElementById("myang1").innerHTML;
            const angle2 = document.getElementById("myang2").innerHTML;

            const length = 140;  // link length in mm
            const base = { x: 325, y: 400 };

            // Convert angles from degrees to radians, with 0 degrees pointing
upwards
            const angle1Rad = (angle1 - 270) * (Math.PI / 180);
            const angle2Rad = (angle2 - 180) * (Math.PI / 180);

            // Calculate the position of the first joint
            const joint1 = {
                x: base.x + length * Math.cos(angle1Rad),
                y: base.y + length * Math.sin(angle1Rad)
            };
```

```
            // Calculate the position of the end effector
            const joint2 = {
                x: joint1.x + length * Math.cos(angle1Rad + angle2Rad),
                y: joint1.y + length * Math.sin(angle1Rad + angle2Rad)
            };

            ctx.clearRect(0, 0, canvas.width, canvas.height);
            drawGrid(ctx, canvas.width, canvas.height, base.x, base.y, 10);

            ctx.strokeStyle = '#298ccb';
            ctx.fillStyle = '#298ccb';

            // Draw the first link
            ctx.beginPath();
            ctx.moveTo(base.x, base.y);
            ctx.lineTo(joint1.x, joint1.y);
            ctx.stroke();

            // Draw the second link
            ctx.beginPath();
            ctx.moveTo(joint1.x, joint1.y);
            ctx.lineTo(joint2.x, joint2.y);
            ctx.stroke();

            // Draw the joints
            ctx.beginPath();
            ctx.arc(base.x, base.y, 15, 0, 2 * Math.PI);
            ctx.fill();

            ctx.beginPath();
            ctx.arc(joint1.x, joint1.y, 10, 0, 2 * Math.PI);
            ctx.fill();

            ctx.beginPath();
            ctx.arc(joint2.x, joint2.y, 10, 0, 2 * Math.PI);
            ctx.fill();

            ctx.font = "bold 18px Arial";
            ctx.lineWidth = 3;

            // Display the end effector's (x, y) position rounded to 2 decimals
            const endEffectorX = (joint2.x - base.x).toFixed(0);
            const endEffectorY = (-(joint2.y - base.y)).toFixed(0);
            ctx.fillText(`(${endEffectorX}, ${endEffectorY})`, joint2.x + 10,
joint2.y - 10);
            ctx.fillText("BASE", base.x - 25, base.y + 35);
        }
```

The "uploadFile()" function handles the file upload process by allowing the user to select a file, reading its contents, and sending the data to an ESP32 device in chunks. It utilizes a FileReader to read the file's text and the WebSocket connection to communicate with the ESP32.

```
        function uploadFile() {
            const input = document.getElementById('inputFile');
            if (!input.files.length) {
                alert('Please select a file first');
                return;
            }

            const file = input.files[0];
            const reader = new FileReader();

            reader.onload = function(event) {
                const text = event.target.result;
                const lines = text.split('\n');

                const sendChunk = (startIndex, endIndex) => {
                const chunk = lines.slice(startIndex, endIndex);
                chunk.forEach(line => {
                    sendDataToESP32(currentLine, line);
                    currentLine++;
                });

                // Wait for a response from the ESP32 before sending the next
chunk
                websocket.onmessage = function(event) {
                    if (event.data === "chunk_received") {
                    if (endIndex < lines.length) {
                        sendChunk(endIndex, endIndex + chunkSize);
                    } else {
                        console.log("All lines sent!");
                    }
                    }
                };
                };

                sendChunk(0, chunkSize);
            };

            reader.readAsText(file);
            }
```

The "movearm()" function adjusts the position and rotation of the robotic arm based on the given angles. It calculates the new angles, applies the transformations, and then determines the end point's position.

```
        function movearm() {
```

```
            const armAngle = ang1 + 90;
            const lineAngle = ang2 + 90;
            arm.style.transform = `rotate(${armAngle}deg)`;
            line.style.transform = `rotate(${lineAngle + armAngle}deg)`;

            // Calculate the position of the end point
            const armLength = parseFloat(window.getComputedStyle(arm).width);
            const armRotation = armAngle * (Math.PI / 180);
            const lineX = Math.cos(armRotation) * armLength + arm.offsetLeft;
            const lineY = Math.sin(armRotation) * armLength + arm.offsetTop;

            // Set the position
            line.style.left = lineX + 'px';
            line.style.top = lineY + 'px';
        }
```

The "initWebSocket()" function initializes a WebSocket connection, sets up event handlers for the connection, and logs the connection attempt.

```
        function initWebSocket() {
            console.log('Trying to open a WebSocket connection...');
            websocket = new WebSocket(gateway);
            websocket.onopen = onOpen;
            websocket.onclose = onClose;
            websocket.onmessage = onMessage;
        }
```

2.4. Testing and Validation

To test the functionality of the GUI, the robot has been connected wirelessly via a local network and all functions have been tested several times. To test the robot accuracy, the GUI was used to move the robot to certain positions and record the position of the robot with a camera. For such the robot has been moved to three different positions, XY 0,280, XY 100,200, XY 100,200.

3. Results

The functionally of the GUI has been tested on the actual robot. The features seen in the GUI all work besides the joint angle sliders. The design represents a clearly structed interface that lets the user navigate without many instructions. If questions arise, the user can use the help button or hover over elements to get instructions of their use. The theme can be switched between a dark and light mode based on user preferences. Videos have been provided showing the workings of the robot in conjunction with the GUI.

The inverse kinematics to find the joint angles with the end-effector position may not be part of the GUI directly but has been tested in conjunction with such. The calculated joint angles are accurate and any inaccuracy in position is mostly due to the printed gearboxes in the system. Figure 13 displays the robot at different positions entered through the GUI.

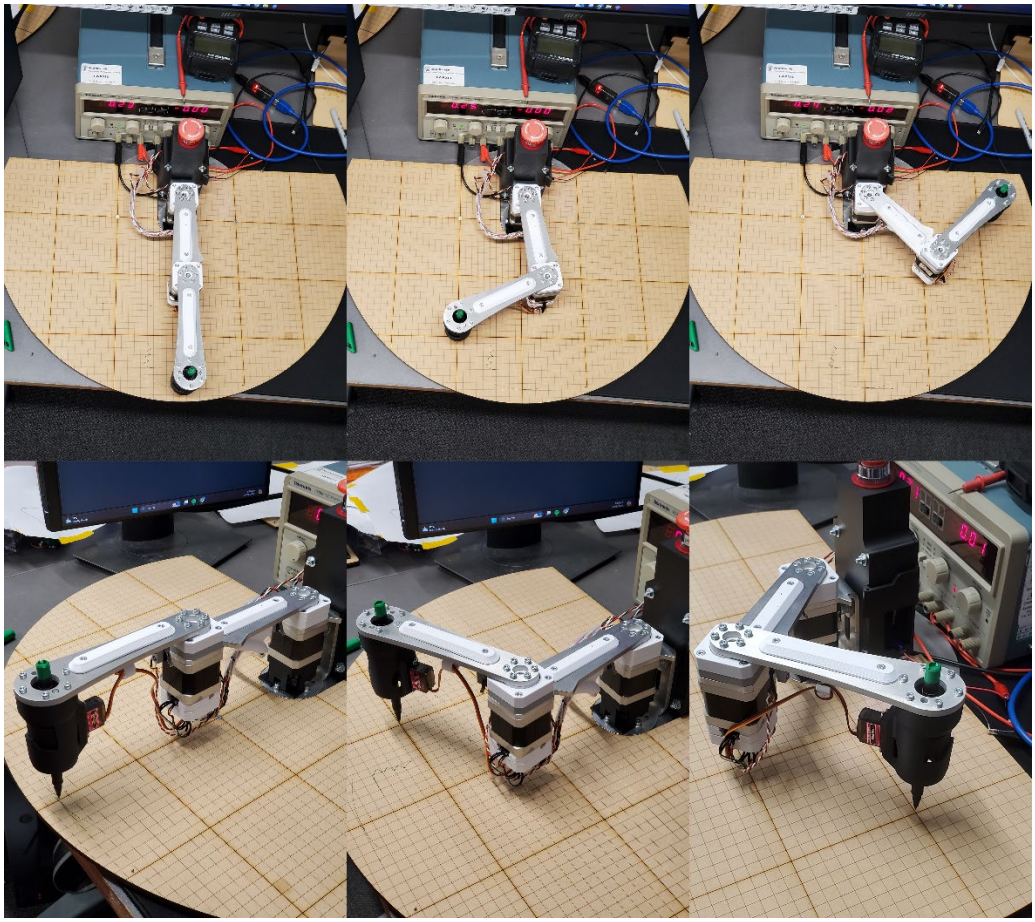


Figure 13 - Coordinate Test (XY: Left 0,280; Middle 100,200; Right 100,200)

The real-time plot of the GUI has been observed to work well but with a certain delay due to the processing speed of the ESP32 and the wireless connection. It is usually around 1 step behind the real time position of the robot when doing tasks such as drawing a square. This should be acceptable for a low speed application such as this.

4. Discussion

4.1. Results and Methods

The development and implementation of the GUI for the SCARA robot have demonstrated significant success in meeting the project's goals. The methods employed, including research into industry-standard interfaces and a long design process, have yielded a functional, intuitive, and visually appealing interface. The GUI effectively allows users to control the SCARA robot, providing real-time feedback and a range of functionalities, from manual positioning to execution of pre-programmed tasks.

During testing, the GUI was thoroughly evaluated for its performance and accuracy. The system's ability to plot real-time positions of the robot, display status and error messages, and provide various control options have proven to be reliable and user-friendly. The integration of the WebSocket protocol ensured seamless communication between the GUI and the robot's controller, automatically connecting to the webserver whenever it is turned on. The use of HTML, CSS, and JavaScript facilitated a responsive and interactive user experience. The few shortcomings observed, such as the

slight delay in real-time plotting and the non-functional joint angle sliders, are areas identified for future improvements.

4.2. Strength and Limitations

One of the key strengths of this project is the clear design of the GUI. This design choice has ensured that the interface functional and easy to navigate. The implementation of a web-based interface allows for flexibility in device usage and supports multiple simultaneous connections, enhancing the overall user experience.

However, there are limitations that need to be addressed. The delay in the real-time plot, caused by the processing speed of the ESP32 and the wireless connection, though minor, affects the precision of real-time feedback. Additionally, the joint angle sliders, which are currently not integrated into the robot's movement functions, represent a gap in the complete functionality of the GUI. The physical limitations of the robot itself, such as the printed gearboxes, also introduce inaccuracies in positioning that the GUI alone cannot rectify.

4.3. Future Development

Several enhancements could further improve the functionality and usability of the GUI. Implementing a feature to save uploaded G-Code files within the GUI would allow users to easily access and execute previously used programs without needing to re-upload them. Another feature could be adding a live camera feed within the GUI would provide users with visual feedback of the robot's operations would enhance the control capabilities as well as enable remote monitoring of the robot.

Developing a live programming feature within the GUI would allow users to program and store multiple locations via coordinates and would improve the system's flexibility and ease of use.

Integrating the joint angle sliders into the robot's movement functions would complete the GUI's current control features and allow for an alternative way of moving the robot.

5. Conclusion

In conclusion, the GUI developed for the SCARA robot has achieved its primary objectives of designing and programing a GUI to control the robot and demonstrated strong functionality and usability, addressing the identified limitations, and suggesting future developments. The user can interact with the robot via various commands such as poses, tasks or manual jogging as well as get help in numerous ways or customize the theme to their liking.

6. References

[1] "SCARA robots (Application examples) - Industrial robots | YAMAHA MOTOR CO., LTD." Yamaha Motor Co., Ltd. https://global.yamaha-motor.com/business/robot/lineup/application/ykxg/ (accessed 23/04, 2024).

[2] A. Owen-Hill. "What is a SCARA robot? The background and benefits." RoboDK blog. https://robodk.com/blog/what-is-a-scara-robot/ (accessed 23/04, 2024).

[3] P. Corke, "Robotics, Vision and Control," *Springer Tracts in Advanced Robotics,* 2017, doi: 10.1007/978-3-319-54413-7.

[4] A. Robotics, "ABB Robotics - Production Screen - HMIs made easy," ed: YouTube, 2013.

[5] I. Services, "DJI Drone Interface," ed: YouTube, 2021.

7. Appendix

7.1. Appendix A – Program Structure Flow Diagram