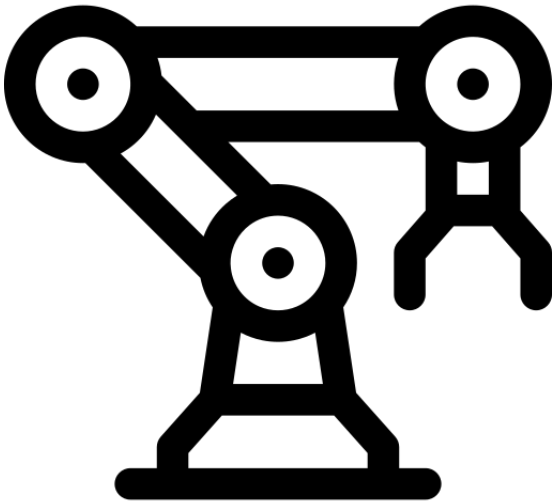


SCARA GUI

Selective Compliant Articulated Robotic Arm Graphical User Interface

282778 Mechatronics

Assessment 4: “SCARA robot GUI project”

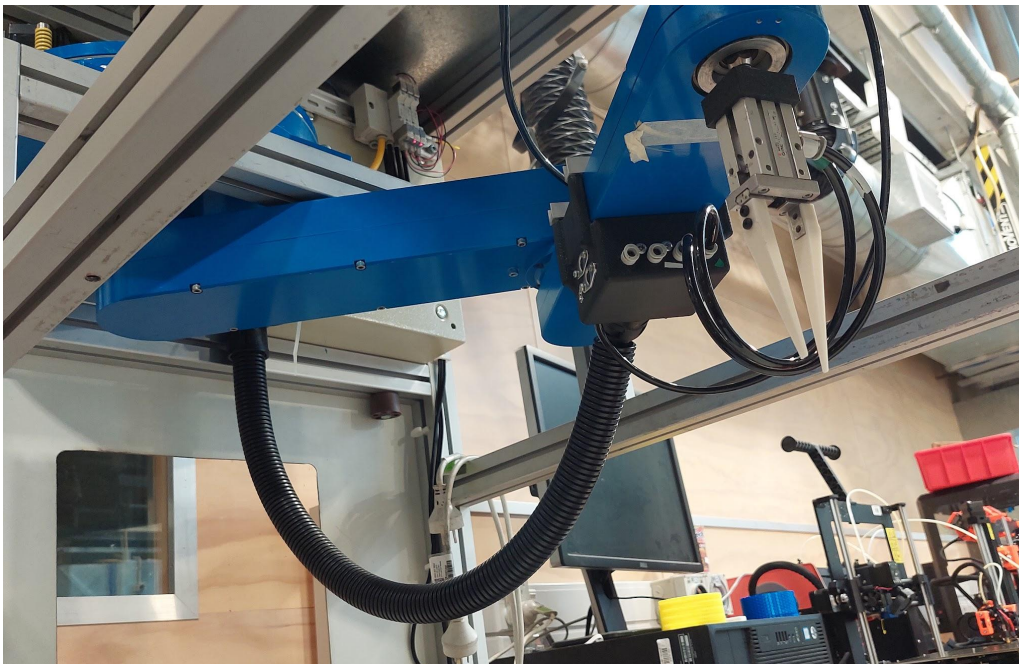
Jamie Churchouse - 20007137

Contents

Note that underlined text denotes a hyperlink.

Save the trees - don't print this, please!

1 Introduction



SCARA

The School of Food and Advanced Technology (SFAT) possesses three Selective Compliant Articulated Robotic Arms (SCARAs). These SCARAs are controlled with an Arduino Due microcontroller board which actuates stepper motors to move the links of the arm, as well as solenoids to actuate the pneumatic piston and End Effector (EE). The objective of this project is to create a Windows application to convey commands to the SCARA via serial communication (a form of PC to microcontroller communication) which can be operated by the user with an easy to interpret, and use, Graphical User Interface (GUI).

The SCARA has three revolute Degrees Of Freedom (DOF); these are the joints at the base of each arm, and the joint at the end of the second arm that holds the EE assembly. The pneumatic piston at the end of the second arm has only the up and down state, and the EE has only open and closed states. These are the main functions of the SCARA; move to an X,Y coordinate, rotate the EE assembly, extend / retract the piston, and open / close the EE. The SCARA comes with firmware pre-installed on the microcontroller - including pre-defined commands - and accepts commands as a slave from the computer who acts as its master.

2 Methodology

2.1 GUI Requirements

This application is required to enable the user to both send valid commands, and manipulate the robot's settings, all while being intuitive to use and pleasing to look at. This breaks down to four parts:

- Interface - UI is aesthetically appealing, intuitive to use, not cluttered
- Commands - can send every type of command
- Settings - can change every setting easily
- Validation - only valid settings / commands can be saved / sent

The commands that the SCARA can accept, along with the returns for each of them, can be found in appendix 1.

2.2 Visual Studio

The project brief recommended the QT Creator (QTC) IDE with Python to develop this programme, however, I have gained much experience over the last six months using Microsoft's Visual Studio (VS) IDE. VS is also far more popular than QTC, and is used for many, if not most, software development worldwide. VS's default language is the popular C# which has a great variety of documentation sources, and there is a large variety of NuGet packages available from within the IDE itself.

For this project, I opted to use WPF for the GUI design rather than WinForms. This is because WPF is more widely used nowadays and GUIs made with WinForms don't tend to scale well as easily. It also means that assigning C# functions as event handlers is merely a single parameter in the XAML code for buttons and text boxes.

2.3 Development Process

The flow of this project is fairly straightforward and is more or less the order outlined above. First the UI needs to be designed and then implemented in XAML code. Next, communication needs to be established with the microcontroller. To aid in this, I took the code from the actual SCARA, removed all the functionality that pertains to movement, and refactored the code so that the serial commands act identically to the real SCARA but have no action. This code was then uploaded to an Arduino Uno which then acts as my emulator for testing purposes. Once communication is established, adding the event handlers for button pushes is next, along with the code to send commands with their parameters. After careful contemplation, I established that creating a new form for the settings would be the cleanest route, so the settings window and its functionality is next. Once everything is working, all user inputs need to be validated before being stored as settings or being sent to the SCARA.

Testing is critical for a project like this. After the serial communication is established to be operational, every function is tested immediately following its creation. Another important consideration when working on programming projects is that of backups and version history. For this, I created a GitHub repository where all the files are committed and pushed to, to allow cross device development, and version history - which is very useful for experimenting with features I'm not sure how successful they'd be.

2.4 Interface

The first stage in designing the UI, is establishing what elements are needed in order to fulfil the functions that are required. I have established these to be:

- Status indicators
- Connect button
- Emergency stop button
- Inputs and buttons for movement commands
- Toggles for pneumatics
- Labels / buttons for the extra settings
- Output log window

Supplementary to these, I decided to give the user the ability to decide which outputs they'd like to see, an easy way to locate the output file, and add functionality to resize the window without the elements clipping each other or becoming distorted. Screenshots of the GUI can be found in appendix 2.

2.5 Commands

The full code is available on my GitHub, so I will only highlight a few of the more important sections of code here.

Initialisation

Along with initialising the GUI, the initialisation function reads settings and shows or hides corresponding elements respectively such as which extra settings are enabled, and what the dimensions of the window are. It also runs the logging initialisation, and the serial initialisation, along with updating the font sizes.

Connect

When the *Connect* button is clicked, the *ScanAndConnect* function is run. This function confirms that the port is closed, and then proceeds to scan all COM ports into a list. For each port in this list, the description is tested for an Arduino associated term. Provided a port is found meeting the criteria, the function attempts to open the COM port for serial communication at the predefined baudrate. If this fails, the loop continues until there are no more ports in the list. Either way, the function verifies that the port is open, and if it is, sends some initial commands and shows the warning message to the user (appendix 3.3), otherwise the user is advised of the failure to connect.

Move

Upon the *Move* function being activated, the contents of each input box, for X, Y, and W (“W” for *w*rist), are tested for validation. Provided this succeeds, these values are injected into a string, and this string is passed to the serial send function.

Sending Data

This function checks the quality of the command and then attempts to send it to the serial port accounting for failures in both sections. Should the transmission fail, the user is advised, the status bar updated to reflect a connection issue.

2.6 Settings

Another feature of VS I like is the easy ability to save settings inside the project. These settings act like variables, but are still available next time the programme is launched. In the menu bar, under *Options*, *Advanced*, launches the settings window as seen in appendix 3.2. The values of each setting are populated from the project settings. The tabs *Visibility*, *Pneumatics*, *Limits*, and *Serial* are visible:

Visibility:	the visibility of the “settings” commands.
Pneumatics:	the characters and delays for the pneumatic actuations.
Limits:	the minimum and maximum X, Y, and W positions.
Serial:	the baud rate and the timeout time.

When any box is interacted with, the *Confirm* button is enabled which allows the user to save their settings. These settings are validated before proceeding to save them.

2.7 Validation & Error Handling

There are two user inputs that are subject to validation in this programme: on the inputs from the settings window, and on the inputs from the command window. The validations involve checking if the value is not null, parsing the string from the textbox to an integer, and then checking if it falls between a minimum and a maximum value. The limits for the settings are hard coded and are at limits that the robot simply cannot reach anyway, whereas the limits for the commands are the ones set in the settings window.

As mentioned in the above discussion, all aspects of this code have been designed to handle errors and exceptions. Everything from failing to send a packet via serial, to parsing strings to integers, are handled with *try*, *throw*, *catch* statements. If any of these errors occur, the user is notified via either a Message Box, or with a message in the output log box.

3 Results

3.1 Design Process

The process of developing this programme was very smooth thanks to the stipulated design process as stated earlier in this report. Continuous use of GitHub to commit changes in stages was of enormous help and saved me many hours of trying to go back and figure out why some features weren't working. It helped very much that I am very familiar with the VS IDE as I didn't need to spend much time figuring out how to do common things.

Throughout the development process, I found many opportunities to include subtle, but very useful, *quality-of-life* features. The first one is the ability to resize the window - which doesn't sound too impressive until you make a GUI and experience the battle to make everything work together for yourself. When the user has selected no extra commands to be visible, there is a large empty space below the "move" and "pneumatics" control blocks. I added in logic to show the client's company logo in that space in the event this occurs.

I realised early on in the development of this project, that there were different types of messages that would be shown to the user in the output box, and wanted to give the option to the user as to which of these they saw. The settings window is intended to be a "set-and-forget" tool that only needs to be opened rarely, so the options for enabling or disabling message types is in the *Options*, *Outputs* section on the menu bar as seen in appendix 3.1.

The last subtle feature I'd like to point out, is that buttons are disabled when their output would be invalid; i.e. the *Connect* button is disabled during the connection process, and the control buttons are disabled when the SCARA is not connected. The cursor changes to the "wait cursor" during the connection process also.

3.2 Testing

As with any engineering project, it is imperative that this solution is tested thoroughly. Throughout the testing of this project, the debug logs have been of enormous help in finding bugs and rectifying them. An example of the debug logs can be found in appendix 3.4. Putting effort into setting up the emulator on the Arduino Uno was also a great help to the testing process. It allowed me to test every aspect of the programme without having to be on site with the SCARA itself. I already knew the commands I were sending would result in actions from the machine, so once development had been completed off site, the first test run with the real machine resulted in perfect operation (appendix 4).

A video that demonstrates the programme in operation can be found on YouTube at this link.

4 Discussion

4.1 The Objective

The objective of this project is to create a GUI to control the SFAT's SCARAs and all their features. The solution I have presented offers the user quick access to the most commonly used commands, and the ability to select whether the lesser used commands are visible. The GUI has a colour, layout, and style scheme in line with many common Windows applications, and it is resizable. The user is always up to date on what is happening in the background with the output box, and can diagnose issues with the easy to access output logs. While I am not a graphic designer, I am very happy with the aesthetics of the interface. For these reasons, I am very pleased to say that my solution meets the requirements of the brief.

4.2 Strengths & Limitations

The biggest limitation to what this programme is able to offer the user, is the limited number of features the SCARA itself offers. If there was the ability to read the time the robot would take to complete a movement, or find the maximum speed or trajectory, then I would've been able to provide this information to the user.

Within the programme itself, there are a few things I'd like to have optimised had I been given more resources; UI scaling isn't perfect, colour scheme and layout could be improved, and it is unclear what coordinates correspond to physical locations. All these aspects have been deliberately left in their unoptimised state because the return on time investment is not favourable at this stage. I also feel that some more background optimisation could be made if I learn more about how Windows handles processes and so on.

A few features I'd like to highlight as strong points of this programme are the one-click connect button, the simple UI that is free of clutter, user options that are restored on reload, and reliability. I believe my philosophy of trying to make the simplest user experience has come through to show in the design of the programme, and I feel that is intuitive to use without having to read the instruction manual first.

4.3 Future Features

If I had more time to work on this project, I would implement a map system to select the desired EE location on. This map would generate the required MOVE command automatically, and also show the user where elements in the machine are; such as conveyor belt, trays, scales, etc. there is also the possibility to have a graphical representation of the SCARA's links shown too.

The second feature I would like to add is the ability to create automation sequences. The user can create a pose of the robot, and then save it. These poses can be strung together and then run like an automated system.

5 Conclusion

5.1 Findings

The objective of this project is to create a GUI for the SFAT's SCARAs. I created a WPF application in VS using C# and XAML which is available on GitHub. The programme I created uses serial communication to send the SCARA commands from the user's Windows computer. The GUI features a one click connect button, a controls panel, an output log box, and a settings menu to configure the pneumatic and movement capabilities of the SCARA, and control the visibility of some lesser used controls.

5.2 Reflection

I had the opportunity to learn all about VS during my internship at Levno over the summer, and I had a head start on this assignment since I made a GUI for the first SCARA assignment in this course. This project, however, was my first time using WPF for the GUI as I had only used WinForms until now.

I really enjoyed this project, even though I have only spent six days on it. The project has strengthened my understanding of how VS operates, and the need for writing readable and reliable code. This project has been entirely documented on GitHub, and it is interesting to review some of the commit comments as I experienced disappointment and joy throughout the development process.

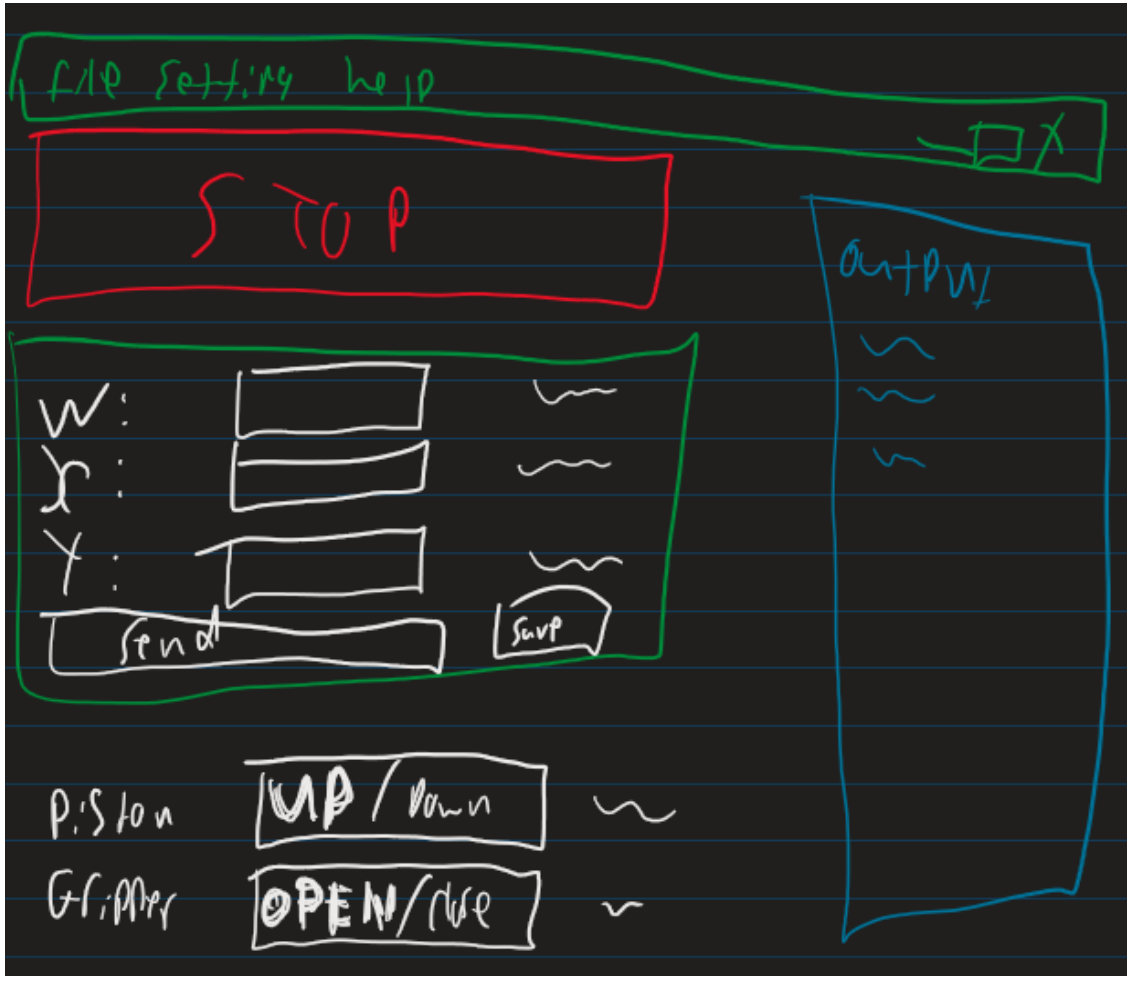
Appendix

Appendix 1: SCARA Functions

Type	Name	Command	Parameters	Returns	Description
Commands	Stop Robot	STOP	-	STOPPED	Stop the robot immediately. This command is processed differently to all other commands in that it is processed on receive rather than added to a buffer.
	Ping	PING	-	PONG	Ping the device to test communication.
	Home	HOME	-	HOME	Home the robot to 0,0,0
	Move to	MOVE	,*x*,*y*,*w*	MOVE: *x y w*	Move to coordinates *x*,*y* and rotate the EE *w*.
	Air control	AIR	,*c*[,*w*]	AIR: *c*[*d*]	Activate air function *c* with optional delay *d*.
	Wait for	WAIT	,*w*	WAIT: *d*	Wait a time in milliseconds, *w*, before proceeding.
	Ding	DING	,*msg*	DONG: *msg*	Send a message, *msg*, when the robot reaches that command.
Settings	Clear buffer	CLEAR	-	CLEAR	Clear the buffer.
	Echo commands	ECHO	,*b*	ECHO *b*	Turn on/off (*b*) confirmation of command receive messages.
	Set speeds	SPEEDSET	,*v*,*a*	SPEEDSET: *v a*	Set the velocity, *v*, and acceleration, *a*, values of the robot.
	Read offset	ROFFSET	-	OFFSET: *v*	Read the offset of the stepper motors, *v*.
	Set offset	SOFFSET	,*v*		Set the offset of the stepper motors, *v*.
	Get ID	ID	-	ID: *v*	Read the ID of the robot, *v*.
	Get proximity	PROX	-	PROX: *v1*,*v2*,*v3*	Read the values of the proximity sensors, *v1*,*v2*,*v3*.
ERROR	Negative acknowledge	-	-	NACK: *cmd*	Command was received, *cmd*, but is invalid.

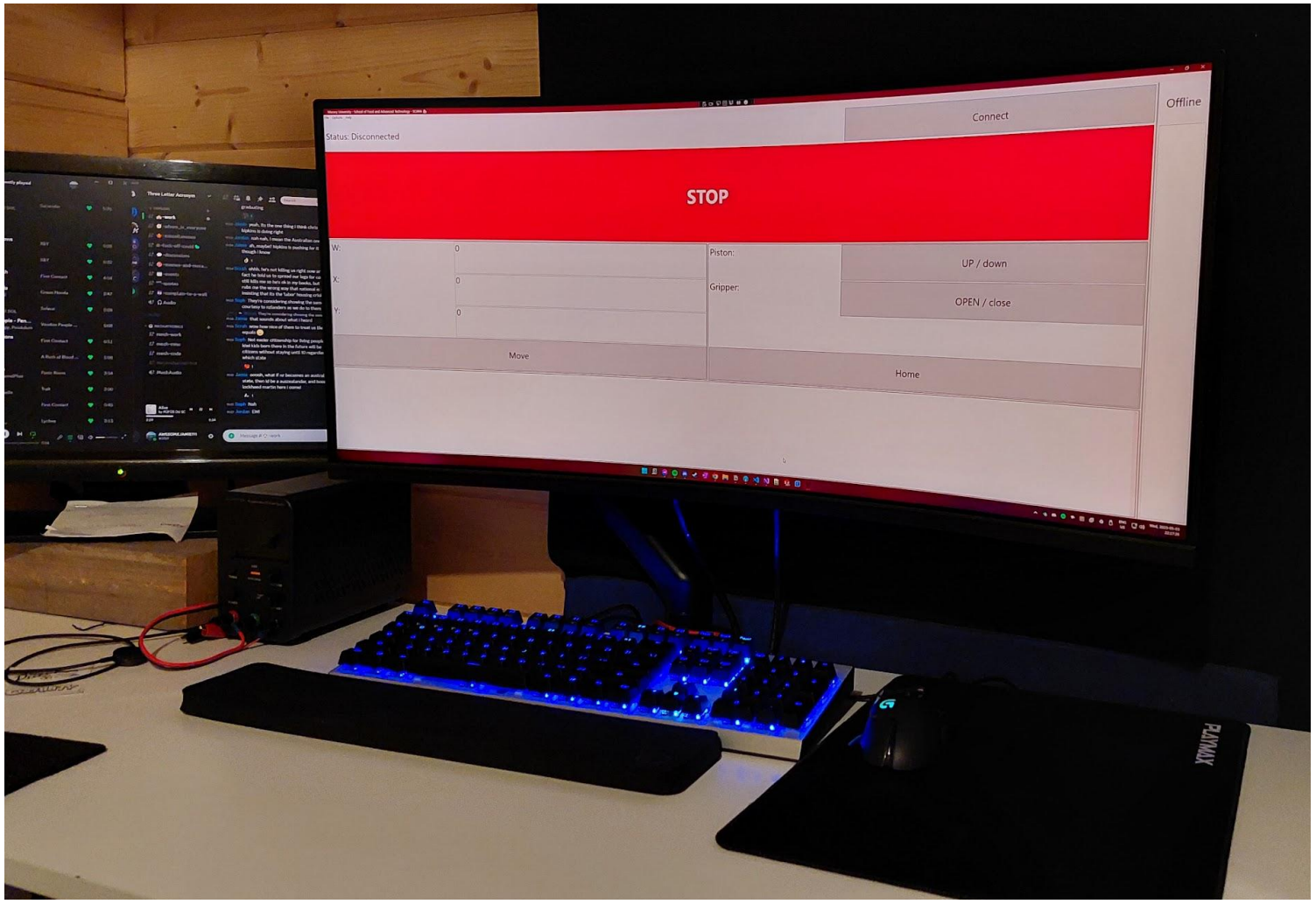
Appendix 2: Main Window GUI

A 2.1: Initial Design in OneNote



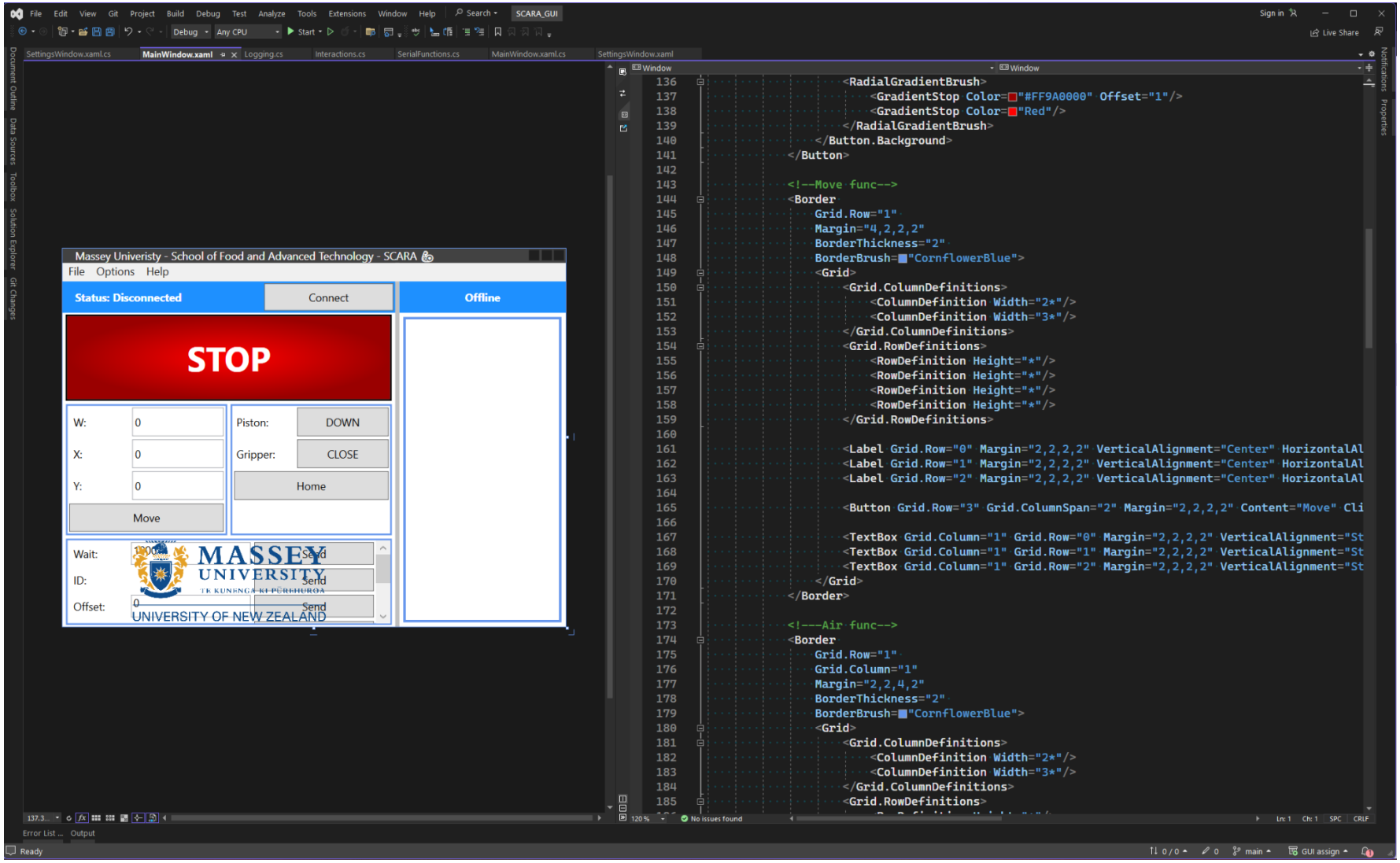
A 2.2: Work In Progress - A

Testing scaling on my ultrawide monitor

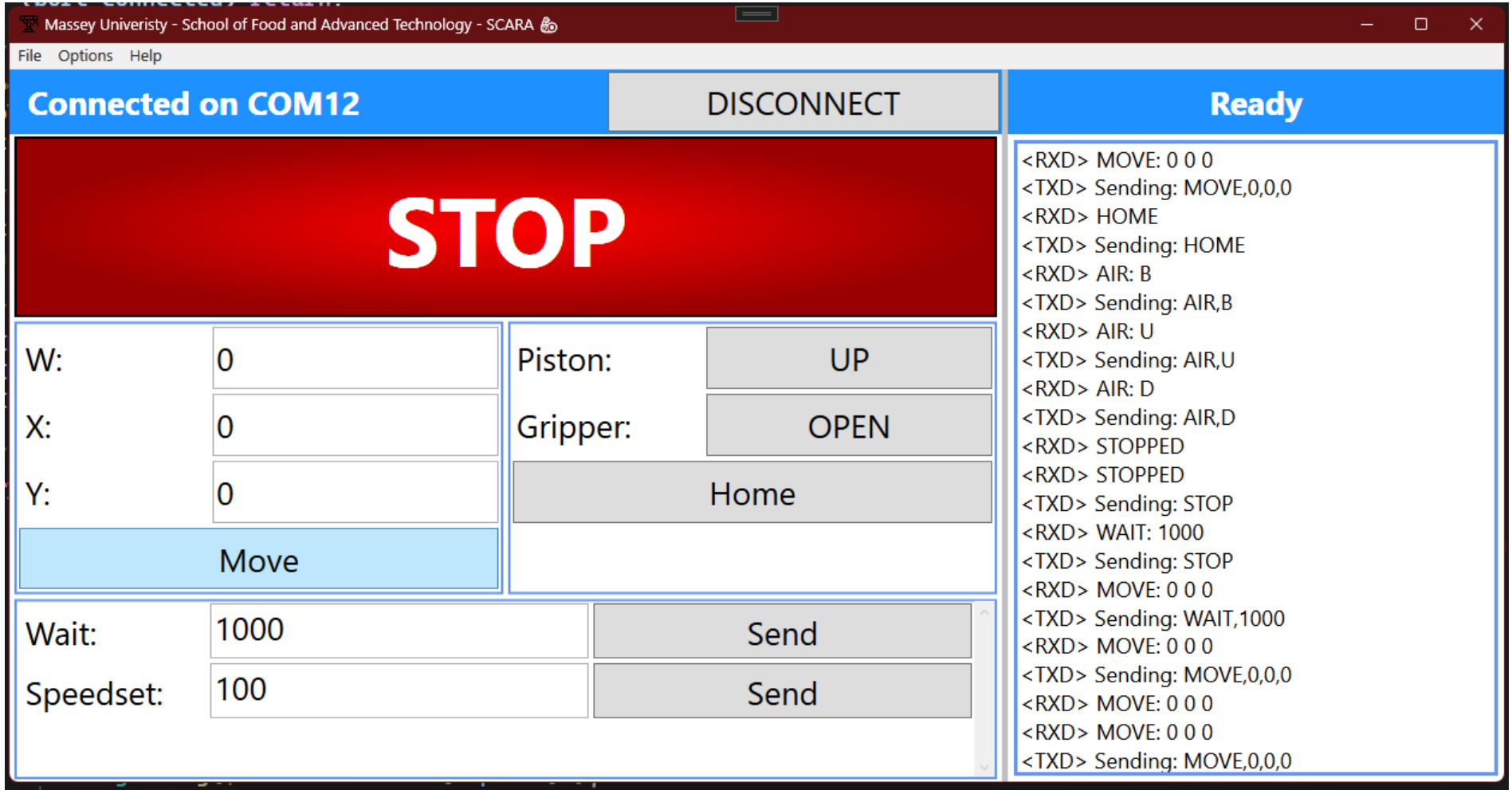


A 2.3: Work In Progress - B

Screenshot of the VS designer and the XAML file

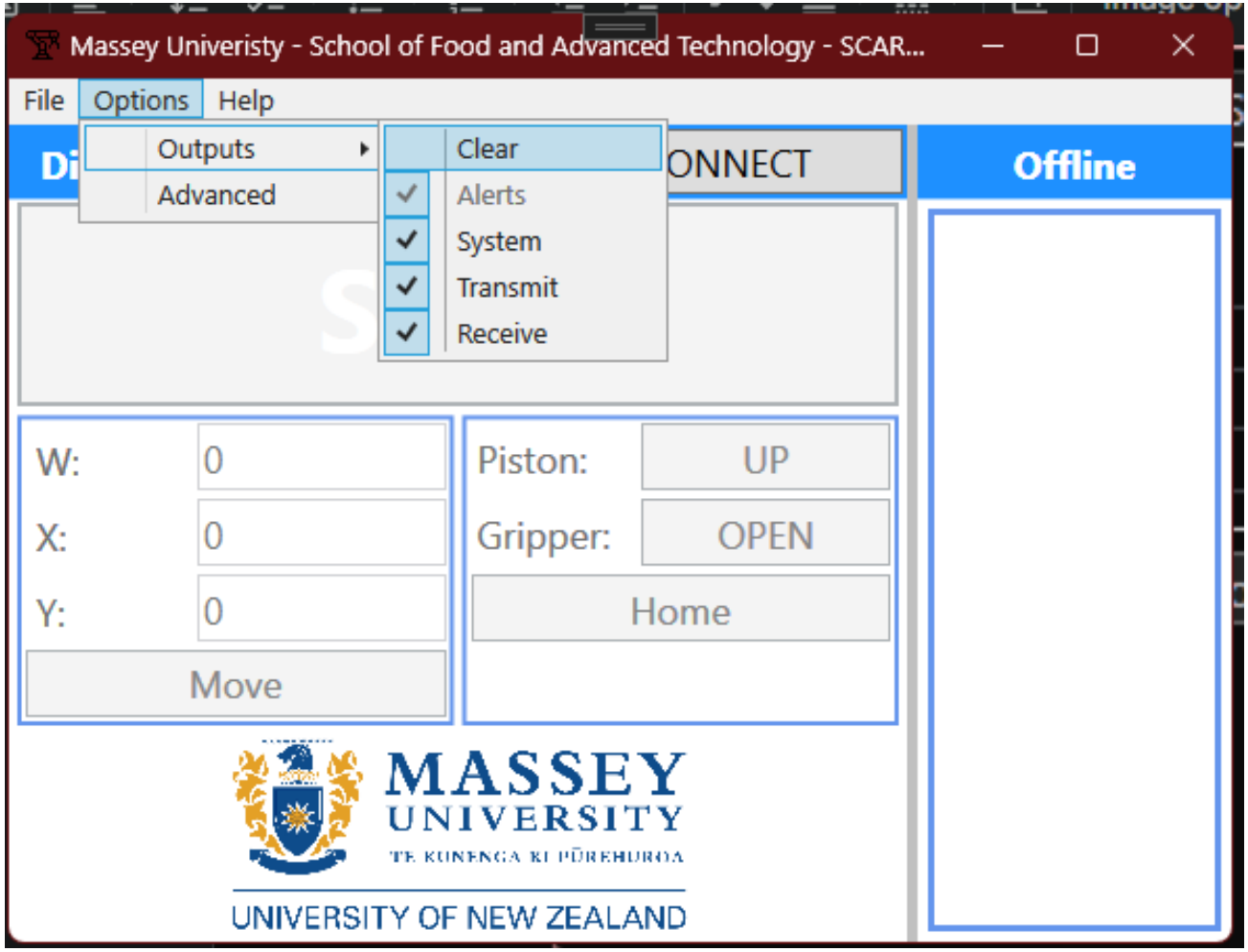


A 2.4: Production GUI

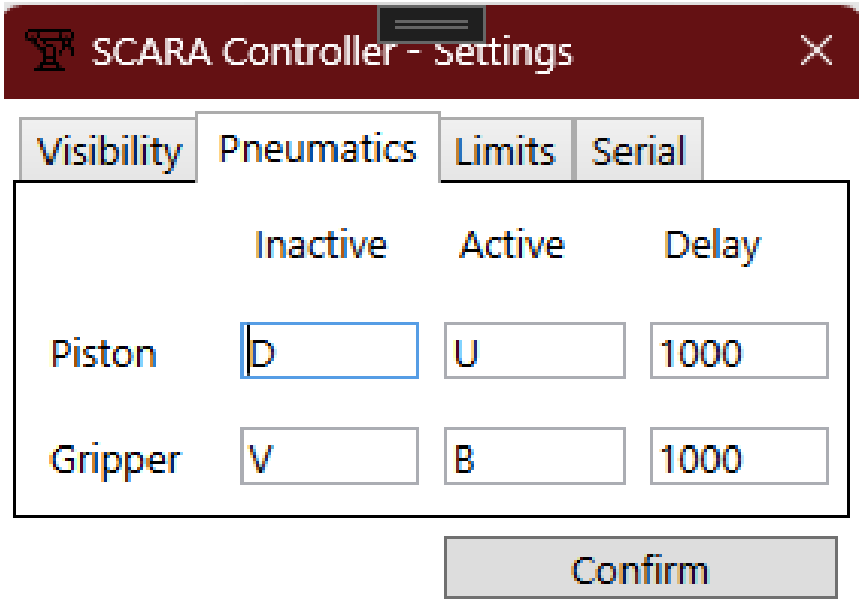


Appendix 3: Additional Features

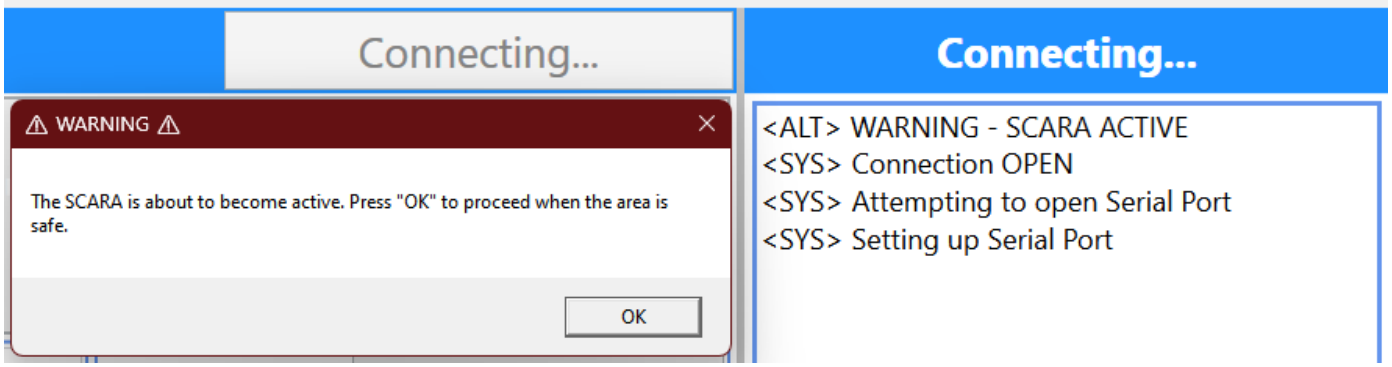
A 3.1: Main Window Options



A 3.2: Settings Window



A 3.3: Activation Warning Message Box



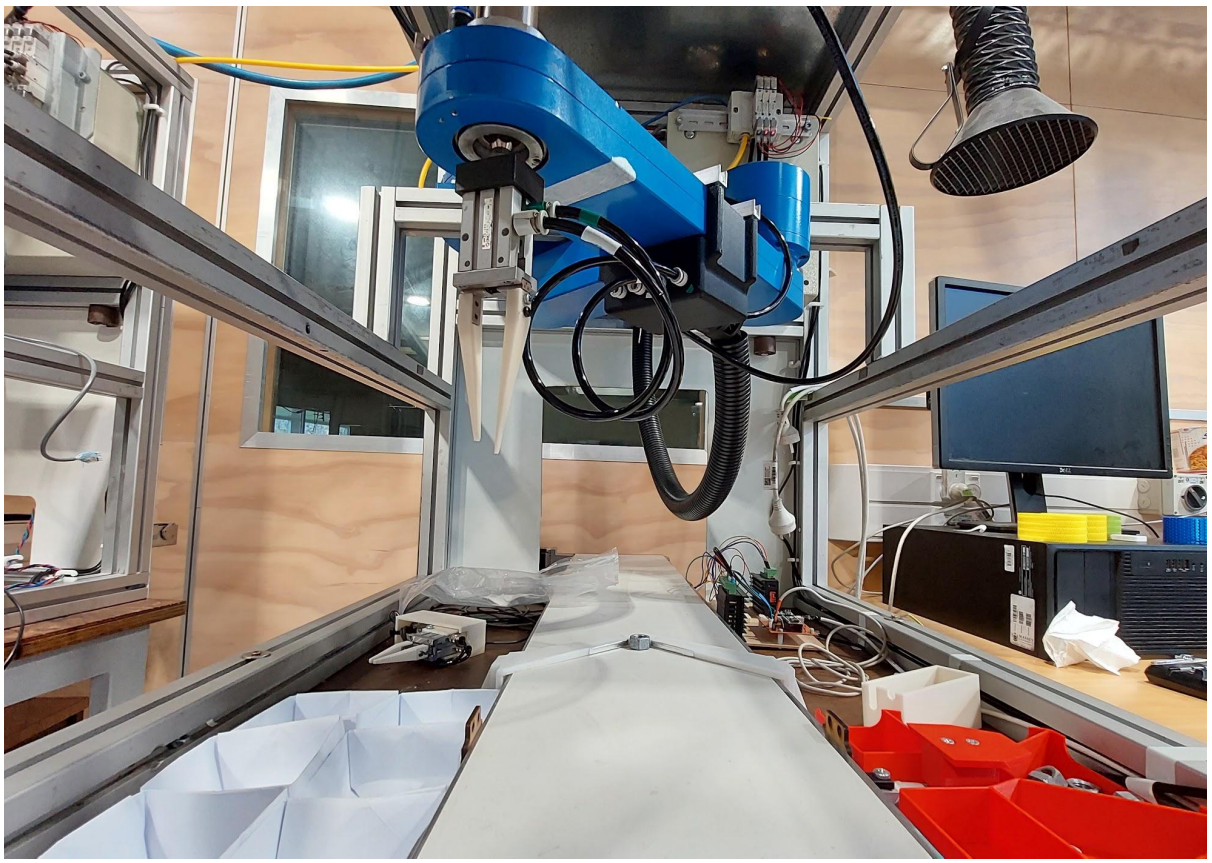
A 3.4: Generated Log File

GUI assign > SCARA_GUI > SCARA_GUI > bin > Debug > logs > 2023-05-06-21-10-10_SCARA_GUI.log

```
2023-05-06 21:10:10.824 +12:00 [INF] This programme was developed by J. P. Churchouse
2023-05-06 21:10:10.835 +12:00 [INF] Started programme at time: 2023-05-06-21-10-10
2023-05-06 21:10:10.850 +12:00 [DBG] Resize - H: 656 W: 1260 S: 27
2023-05-06 21:10:10.851 +12:00 [INF] Ready
2023-05-06 21:10:10.899 +12:00 [DBG] Resize - H: 656 W: 1260 S: 27
2023-05-06 21:10:35.090 +12:00 [INF] <SYS> Setting up Serial Port
2023-05-06 21:10:35.581 +12:00 [DBG] Found device: Communications Port (COM1)
2023-05-06 21:10:35.582 +12:00 [DBG] Found device: Arduino Uno (COM12)
2023-05-06 21:10:35.582 +12:00 [INF] <SYS> Attempting to open Serial Port
2023-05-06 21:10:35.644 +12:00 [INF] <SYS> Connection OPEN
2023-05-06 21:10:35.645 +12:00 [DBG] Baudrate: 115200
2023-05-06 21:10:35.823 +12:00 [INF] <ALT> WARNING - SCARA ACTIVE
2023-05-06 21:10:36.952 +12:00 [INF] <TXD> Sending: ECHO,1
2023-05-06 21:10:36.955 +12:00 [INF] <TXD> Sending: HOME
2023-05-06 21:10:42.587 +12:00 [INF] <TXD> Sending: AIR,D
2023-05-06 21:10:42.592 +12:00 [INF] Received: "AIR: D"
2023-05-06 21:10:42.592 +12:00 [INF] <RXD> AIR: D
2023-05-06 21:10:43.762 +12:00 [INF] <TXD> Sending: AIR,U
2023-05-06 21:10:43.768 +12:00 [INF] Received: "AIR: U"
2023-05-06 21:10:43.768 +12:00 [INF] <RXD> AIR: U
2023-05-06 21:10:45.101 +12:00 [INF] <TXD> Sending: HOME
2023-05-06 21:10:45.107 +12:00 [INF] Received: "AIR: D"
2023-05-06 21:10:45.107 +12:00 [INF] <RXD> AIR: D
2023-05-06 21:10:45.760 +12:00 [INF] <TXD> Sending: HOME
2023-05-06 21:10:45.767 +12:00 [INF] Received: "AIR: U"
2023-05-06 21:10:45.767 +12:00 [INF] <RXD> AIR: U
2023-05-06 21:10:46.972 +12:00 [INF] <TXD> Sending: HOME
2023-05-06 21:10:46.976 +12:00 [INF] Received: "HOME"
2023-05-06 21:10:46.976 +12:00 [INF] <RXD> HOME
2023-05-06 21:10:48.025 +12:00 [INF] <TXD> Sending: MOVE,0,0,0
2023-05-06 21:10:48.032 +12:00 [INF] Received: "HOME"
2023-05-06 21:10:48.032 +12:00 [INF] <RXD> HOME
2023-05-06 21:10:50.448 +12:00 [INF] <TXD> Sending: MOVE,0,0,0
2023-05-06 21:10:50.453 +12:00 [INF] Received: "HOME"
2023-05-06 21:10:50.454 +12:00 [INF] <RXD> HOME
2023-05-06 21:10:52.776 +12:00 [INF] <TXD> Sending: MOVE,0,0,0
2023-05-06 21:10:52.780 +12:00 [INF] Received: "MOVE: 0 0 0"
2023-05-06 21:10:52.780 +12:00 [INF] <RXD> MOVE: 0 0 0
2023-05-06 21:10:52.784 +12:00 [INF] Received: "MOVE: 0 0 0"
2023-05-06 21:10:52.784 +12:00 [INF] <RXD> MOVE: 0 0 0
2023-05-06 21:10:55.620 +12:00 [INF] <TXD> Sending: MOVE,0,0,0
```

Appendix 4: Operation

A 4.1: My Programme in Operation



A 4.2: Video Demonstration

https://youtu.be/KZKGZ79S4CI