University of Southern Queensland

School of Engineering

Design & Develop a Low Cost Programmable Logic Controller (PLC) Training Tool, utilising open-source micro-controllers

A dissertation submitted by

Mr Shane Fulcher

In fulfilment of the requirements of

ENG4111 and ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Honours) (Instrumentation Control & Automation)

Submitted October 2023

Abstract

Programmable Logic Controllers (PLC) have increased in usage since their introduction, and the need for adequately trained programmers has risen along with this increased usage.

The aim for this project was to design and develop a low-cost PLC training tool that makes use of open source microcontrollers to simulate industrial devices commonly found in an industrial control system. Physical requirements for this tool were that it was not to cost more than $1000 AUD, it had to be reasonably portable and that it had to be safe for users. Program requirements for the tool were that it had to allow for programming in at least 3 programming languages, the microcontrollers had to be able to simulate at least 3 standard industrial devices and that users had to be able to easily select these devices.

Previously constructed PLC training tools often fall into one of two categories – advanced pieces of equipment making use of analogue signals and with a professional look to them, but are very expensive, or much cheaper options but are significantly more simplistic in the options available to the user.

The technical requirements for this tool were laid out, detailing the types of industrial devices the microcontrollers were to simulate, and how many inputs and outputs these would require. These requirements led to the selection of a Delta ES3 Series DVP32ES311T PLC for use with the tool. This PLC has capacity for the digital I/O requirements but needed an analogue extension module included to ensure the ability for analogue signals to be utilised. The microcontrollers selected for this tool were the Arduino Uno Rev3. The tool was physically constructed, and sample programs written to test operation. Test scenarios were written for users to follow to aid in their learning.

The tool came in over budget by approximately $160 AUD, but otherwise met all other physical and programming requirements. It was constructed in a portable way and is safe for users to utilise. The PLC can be programmed in Ladder, Structured Text and Sequential Function Chart languages, the microcontrollers can be used as one of up to 7 devices and a selector switch and LCD screen has been added to allow for easy selection of these devices.

The tool was given to users with a wide range of prior PLC programming experience to test its effectiveness and overall feedback from these users was positive. Particularly noteworthy from this feedback was that the users all felt as though they had gained some knowledge and confidence from use of the tool, and the more advanced users enjoyed the ability to practice PID control loop tuning from within the training tool.

Candidates Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Shane Fulcher

Student Number: [redacted]

Acknowledgements

I wish to take this opportunity to express my deep gratitude to my supervisor, Mrs Catherine Hills of the School of Engineering. Without her initial input and ongoing support throughout, this project would not be what it is today.

I also wish to thank my employers Adam Kinder & Eilisha Langford for their patience with me over this time, for understanding when I needed to prioritise this project and for being some of my biggest supporters over the last few years.

I would also like to thank my good friend Mr Mitchell Burley for his ever-patient guidance over the years, as well as his unwavering support as a friend.

Finally, I would like to thank my lovely wife Lianne, and my amazing daughter Isla for their love, constant support, and incredible patience with me during this time. I appreciate the sacrifices the two of you have made for me to complete this more than you may ever know.

Contents

1 Introduction

Programmable Logic Controllers (PLC) are an integral part of automation projects across many various industries. Prior to their advent in 1968, the only real way to control ‘automation’ machines was using extensive relay control wiring ('The Evolution of PLCs' 2021). This type of wiring requires a great deal of space and wiring to be operational, in addition to a significant number of locations in the system for physical faults to occur. These issues only worsened as automation needs grew. The number of motors and other devices were required to be operated by the control system increased to a point where it was no longer feasible to simply control them all from these hard-wired relays.

This increasing need led to the design of the PLC, a small industrial computer designed to control any number of automation processes from a single point. They are designed with the intent to be able to handle all manners of harsh environment factors such as moisture & dust. Over the many years since its inception, it has evolved to adapt new and emerging technologies including analogue signals. They are extremely cost effective when compared to previous techniques of automation control, in addition to saving a significant amount of space in the control systems wiring requirements (Fuada et al. 2012).

The introduction of these computers to these automation industries introduced the need for engineers and technicians to have the ability to design these control systems and write the programs to operate them. In addition to this, the need for those engineers and technicians to be able to fault-find and debug this system using the PLC code written previously. Due to this ever-increasing need for the people working with PLCs to have this advanced level of understanding and skill in programming, there has been a significant increase in training courses and tools available for students to utilise (Akparibo et al. 2016) (Artiyasa et al. 2020).

These training courses and tools are extremely beneficial for an inexperienced PLC programmer to be able to learn the techniques required with little to no risk of damage to an operational automation system. They generally allow students to practice their programming skills, and occasionally allow a student to practice wiring the inputs and outputs for the PLC. They range in size and complexity from large systems that take up a full work bench, with many different physical options for inputs and outputs to be utilised, through to systems small enough to fit into a briefcase, with simple push buttons or switches for inputs, and LEDs or similar used for outputs (Pratama et al. 2022). Other options are fully simulated PLC training programs online, where a student can write programs and test them in a simulator without any physical components at all (Kim & Kim 2013).

Unfortunately, these systems are not without their issues which can affect students learning capabilities. Obviously, the larger the training tool means the larger the cost to implement. Some of these larger systems can cost upwards of $3500 USD (approx.. $5500 AUD) to purchase, and they can take up quite a significant amount of space. The smaller options are obviously cheaper, but due to their size they are

quite a bit more simplistic in their design. Quite often they do not allow students to perform their own wiring, and only allow simple switches or LEDs for outputs due to their space constraints. They also rarely offer a programming language other than Ladder logic. Whilst Ladder logic is a highly used method of PLC programming, there are multiple other types of languages available and utilised by manufacturers and as such students need to have the option to learn these languages in a similar fashion to how they can learn Ladder logic.

The purpose of the research in this project was to determine the potential need for a multi-language, low-cost PLC training tool which utilises suitable technology (eg. Microcontrollers) to appropriately simulate physical components and the inputs/outputs of those physical components that a student would find utilised in an industry situation (Yakimov et al. 2019). If this was found to be beneficial then complete a design of this tool, perform a cost analysis of the components required, and construct such a tool for use.

2 Background

Aims

The project aim was to design a relatively low-cost PLC training tool, which utilises microcontrollers to simulate the physical digital inputs and outputs as well as analogue inputs and outputs that can be found on various hardware components in an industrial setting. These components would include Direct On Line (DOL) motors, Variable Speed Drive (VSD) motors, Valves, Level transmitters, etc. The training tool will be designed and constructed to be as portable as possible without losing the capability to interface with at least 3 microcontrollers.

A further aim of this project was to allow students to learn programming across multiple programming languages. A review of current PLC options available will be undertaken to determine which brands and models will allow for this requirement to be met.

Specific requirements of this tool –

- Tool must not cost much more than $1000 AUD to purchase and construct.
- Tool must not be much larger than a standard carry case (approx. 500mm x 300mm x 200mm) – portability was to be considered at every step.
- PLC must be programmable in at least 3 of the programming languages listed in IEC 61131-3 – one of which must be Structured Text.
- PLC CPU ability to be programmed with free open source programming tool – Codesys, OpenPLC, etc. to be considered to assist with the previous point
- Microcontrollers must be able to appropriately simulate at least 3 standard industrial field devices, VSDs, DOL motor control circuits, Valves, Various transmitters, etc.
- Microcontrollers must have the ability to utilise an appropriate amount of I/O to suitable simulate the devices previously mentioned
- User must be able to easily set the particular device in the microcontroller – programming the microcontroller was not to be part of the tools training.
- Construction of tool must be done with safety of the user as paramount. All voltage above 24v must be appropriately covered from contact.

Objectives

Following on from these specific requirements for the tool, the following objectives were proposed for this project:

- Compare various PLC options to find the most suitable when considering the budget available, portability of the PLC, and the programming requirements set out earlier

- Compare various microcontroller options available to find the most suitable to perform as industrial component hardware simulators, while still being portable and within budget. These must also be compatible with the PLC option chosen.
- Acquire resources and construct the physical PLC tool, maintaining the requirement for it to be portable.
- Program the components that require programming prior to the actual tool being able to be utilised by a student for PLC programming.
- Test the effectiveness of the tool and ensure it has met the previously laid out requirements

Table 1 below shows the estimated resources that were required to meet these objectives.

Table 1 – Required Resources

Item	Quantities	Source	Cost	Comment
Computer	1	Student	Nil	Personal laptop for use in writing dissertation and programming components
PLC Programming software	1	Student	Varies depending on PLC	Once PLC is determined, the software to program the specific CPU will be required
Microcontroller programming software	1	Student	Nil	Once microcontroller is determined, the software to program the specific microcontroller will be required
Microsoft Word	1	Student	Nil	Program for writing dissertation
Microsoft Excel	1	Student	Nil	Program for cost analysis of components
PLC	1	Student/USQ	~$500	Once suitable PLC is determined, acquire CPU and necessary expansion cards
Microcontrollers & Microcontroller HMI	4	Student/USQ	~$200	Once suitable microcontroller is determined, acquire hardware
24v Power Supply	1	Student	~$100	24VDC power supply to operate components within case.
Communication cables	2	Student	Nil	Cables to connect PLC & microcontrollers to PC – already in possession
Tool case	1	Student	~$100	Case to hold all components mounted within for portability
Sundries		Student	~$100	Equipment used to mount equipment within case – nuts and bolts, din rail, glands, terminals, cables, plugs, etc.

3 Literature Review

Programmable Logic Controllers

A Programmable Logic Controller (PLC) is a small industrial computing device designed to control and automate a process or machine. The PLC does this by making use of micro-processors with a programmable memory to implement functions such as logic, timers, counters, relays, and mathematics to replace complex relay control (Bolton 2015).

The PLC operates by running a programmed series of instructions or 'logic'. These instructions allow the PLC to evaluate a variety of inputs and in turn control a variety of outputs to affect the process or machines operation. These inputs can include discrete (on/off) inputs such as a push-button or selector switch, or analogue inputs such as a tank level or current motor speed from a Variable Speed Drive (VSD). Similarly, the outputs affected by the PLC can be discrete or analogue with items such as contactors/relays (discrete) or motor speed setpoints or analogue valve position setpoints (analogue).

The physical components of a PLC arrangement can vary quite a bit from process to process. The options available for these configurations can include (Bolton 2015):

- Central Processing Unit (CPU) – The CPU is the only non-optional component in the PLC configuration. It is the 'brains' of the unit, containing the micro-processor, memory, etc. for the instructions to control the PLC.
- Input Modules – These can be discrete or analogue and are designed to be the components of the PLC that receive the information from the real world for the PLC to interpret and use to effect changes on the outputs.
- Output Modules – These can also be discrete or analogue and are designed to be the components of the PLC that control the equipment or processes in the real world.
- Communication Ports/Modules – These are the components of the PLC configuration that allow it to communicate with other devices such as a Human Machine Interface (HMI), or other PLCs, or a SCADA system, or many others. The communication protocols used can vary quite a bit from PLC to PLC, and can include protocols such as Ethernet/IP, EtherCat, Modbus/TCP, Devicenet, and many others.

PLCs are utilised in a wide variety of applications across a wide variety of industries. These can include industries such as (Alphonsus & Abdullah 2016):

- The manufacturing industry – PLCs are used quite a bit in manufacturing processes, from simple motor and valve control, up to control of robotic arms.
- The energy industry – The use of PLCs is fairly important in power plants for the control of generators, turbines, monitoring processes, etc.

- The water and waste treatment industry – PLCs are crucial to this industry, in the monitoring and control of water treatment and filtration, and sewage treatment processes.
- The food and beverage industry – PLCs are utilised extensively in this industry to ensure quality and productivity are at a high standard in their processes.
- The automotive industry – This industry makes use of PLCs to control the assembly lines, as well as robotic arms to manufacture the automotives to ensure quality.

The use of PLCs is not limited to the above industries, as the need for monitoring and control of a process can be found almost anywhere.

PLC Programming Languages

PLCs are programmed from a variety of software options. Each brand of PLC will have its own developed software to allow for users to write the instructions mentioned above. However, there are also open-source software options available, that allow for the programming of a multitude of PLC brands that adhere to the IEC standard for PLC programming.

IEC 61131 is the international standard for Programmable Logic Controllers. Section 3 of this standard provides instruction on a set of 5 specific programming languages that manufacturers must allow programmers to utilise to program their PLCs with one of (IEC 2013). These programming languages are a mix of graphical and textual options, and are listed below:

- Ladder Diagram – Graphical
- Function Block Diagram – Graphical
- Structured Text – Textual
- Instruction List – Textual
- Sequential Function Chart – Graphical

The most commonly used language is Ladder diagram. It is a graphical language designed to look quite similar to relay logics circuit diagrams. It makes use of a series of rungs (hence, ladder) with items such as contacts (inputs) and coils (outputs) on the rungs. It is most commonly used because it is quite flexible and allows for easy troubleshooting. However, it can get quite large and difficult to follow if not implemented well. An example of a simple ladder diagram is shown below in figure 1.

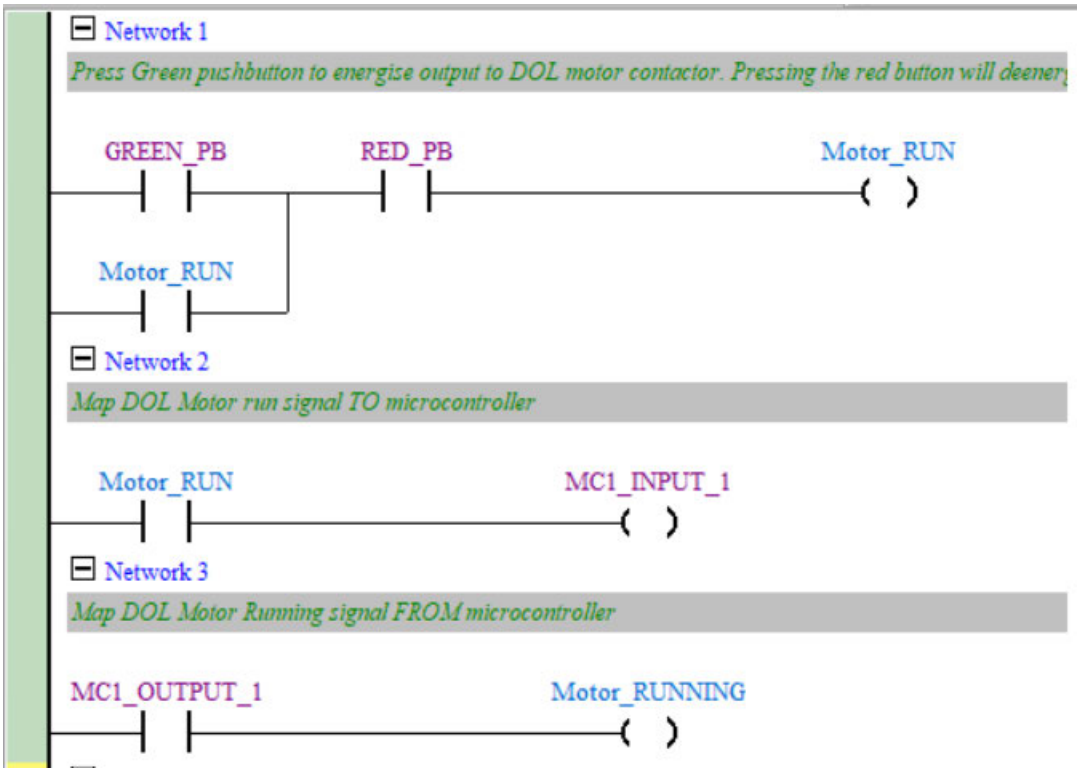


Figure 1 – Example of Ladder Diagram program

Another graphical language in the standard is Function Block Diagram (FBD). FBD consists of a variety of blocks that consist of inputs and outputs, which can be made up of a wide range of data types. These blocks can then be connected to other blocks to complete a program and make it operate the way the user desires. This type of programming is commonly used with safety PLCs and wherever a user needs to reuse code frequently (Faylor 2022). En example of function block diagram programming is shown below in figure 2.

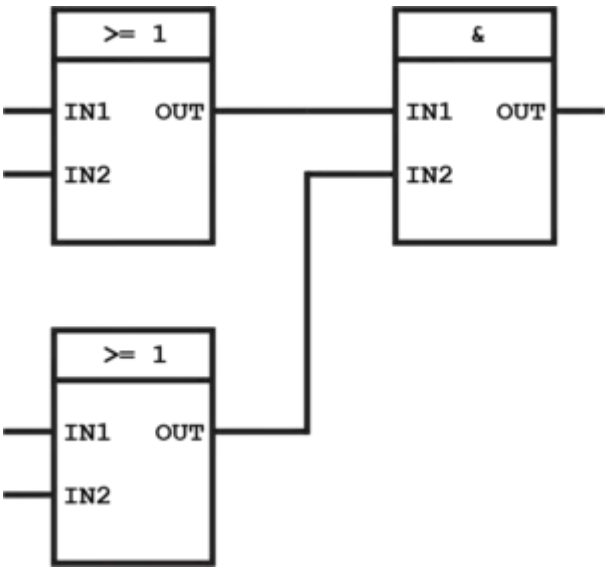


Figure 2 – Example of Function Block Diagram program (Peter 2018)

The first textual programming language is known as Structured Text. Structured Text is a programming style that is fairly comparable to programming in C, or C++. This means that those who have studied or utilised that kind of programming method before will find this language a little easier to implement. In this language, statements such as IF, WHILE, FOR, etc. are used to perform the functions required to operate the process. This language type can be written in a much more concise way than ladder or function block diagram, but can be quite tricky to troubleshoot (Faylor 2022). An example of structured text programming can be seen below.

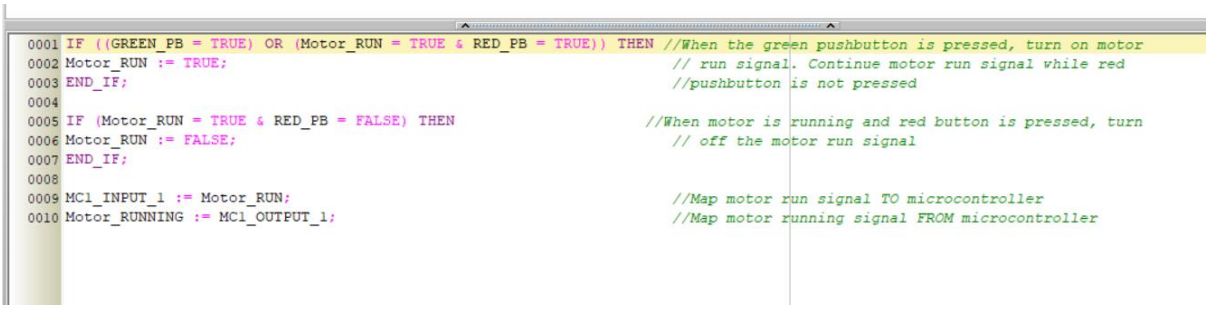


Figure 3 – Example of Structured Text program

The second textual programming language is Instruction List. Instruction List was deprecated in the most recent edition of the IEC 61131, so is unlikely to be utilised by newly made PLCs. However, as it is still used within existing PLCs, it's worth being aware of it. Instruction list is fairly similar to assembler language, and consists of a simple series of instructions (Faylor 2022). An example of an Instruction List program is shown below.

```
LD      TRUE      (*load TRUE in the accumulator*)
ANDN    BOOL1     (*execute AND with the negated value of the BOOL1 variable*)
JMPC    label     (*if the result was TRUE, then jump to the label "label"*)
LDN     BOOL2     (*save the negated value of *)
ST      ERG       (*BOOL2 in ERG*)
label:

LD      BOOL2     (*save the value of *)
ST      ERG       (*BOOL2 in ERG*)
```

Figure 4 – Example of Instruction List program (Beckhoff-Automation 2023)

The final programming language mentioned in the IEC 61131 is a graphical language known as Sequential Function Chart (SFC). This type of programming is based on computer science algorithm flowcharts. It consists of a series of Steps & Transitions. A Step being some action for the PLC to take, and a Transition being the conditions required to move from one Step to the next. A significant benefit of SFC programming is how much it is able to simplify a complex process into smaller, more manageable sections (Faylor 2022). An example of a basic SFC program is shown below in figure 5.

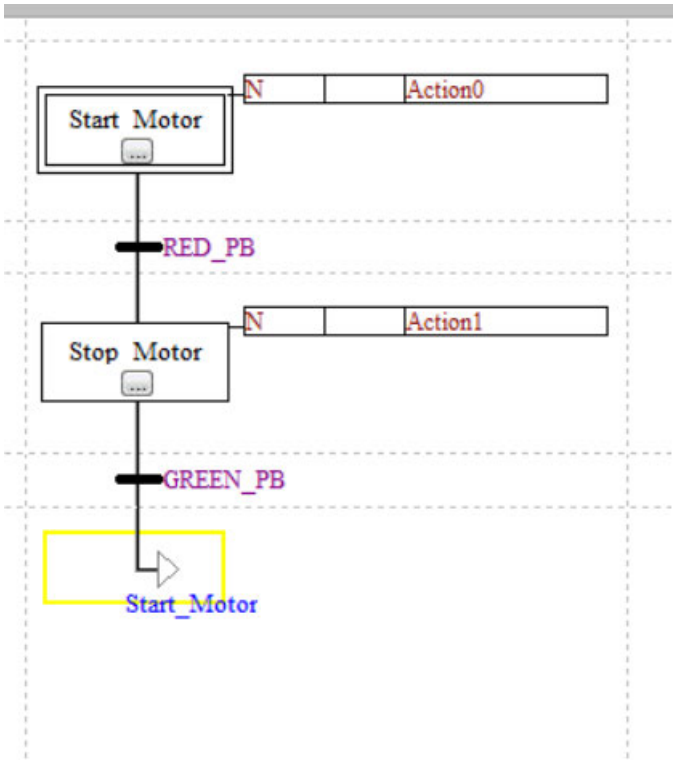


Figure 5 – Example of Sequential Function Chart program

Previous PLC Training Tools

There have been many PLC training tools proposed and designed in the past. (Guo & Pecen 2009) wrote about a junior/senior level PLC training course aimed at being part of a four (4) year electrical engineering degree. This course was to discuss PLC hardware, wiring diagrams, installation and most importantly, PLC programming. The students performed this programming on a program called RSLogix, software released by Rockwell Automation. The entire course takes place in both the classroom (approx. 75%) and the laboratory (approx. 25%), where the programming takes place. Despite learning theoretically about hardware, wiring and installation, very little, if any, of this takes place in the duration of this course. The programming language chosen for this course is exclusively ladder logic.

(Arowolo et al. 2020) discussed a similar concept for a PLC trainer, however their design significantly improved on the student's ability to wire the inputs and outputs themselves. They acknowledged the challenges behind designing a suitable PLC training tool, as many others have made their own versions of this tool in the past but few have put any focus into how to physically wire and install the PLCs themselves. Their training tool was designed to be desktop sized, which allowed for this desire to allow it to be physically wired by the students. Their constructed training tool can be seen below, in Figure 6.

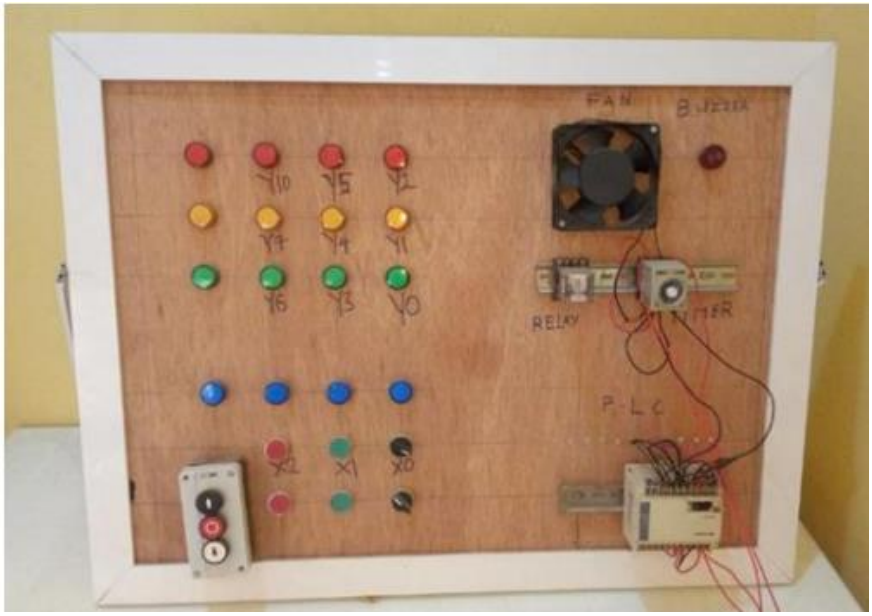


Figure 6 – A Hardware Focussed Desktop PLC Trainer (Arowolo et al. 2020)

As can be seen, the need for the hardware wiring ability has fairly significantly increased the need for the tool to be quite large. Another troublesome part of the design is that the wiring for the Lamp outputs is on the backside of the board compared to the Lamp itself being on the front. So, if adjustments were needed to be made during this training, the entire board will need to be moved and turned around just to make changes to simple wiring. However, giving students the ability to wire their own inputs and outputs gives them an insight into how PLC contacts work that the previous tool (Guo & Pecen 2009) did not. It is not uncommon to see a PLC with inputs that can be either PNP or NPN configured, and this can be a tricky concept for inexperienced technicians and engineers. The benefit of being able to learn that in a specifically designed training tool is fairly significant. The PLC utilised in this tool is a MITSUBISHI FX 1S-30MR-001, which is programmed using software called GX Developer V8. This software allows for programming in multiple languages, but the designers of this tool only intended for it to be utilised only with Ladder Logic.

In 2008, (Barrett 2008) suggested the construction of a PLC training tool with many similar characteristics to the one being proposed here. It was designed to be low cost, as well as portable. They believed that whilst other training methods had their merits, they were not suitable to be the only option available to students wishing to improve on their all-round PLC knowledge. Their constructed tool is shown below, in Figure 7.

Figure 7 – Training System & PLC (Barrett 2008)

The tool is simply a plywood a-frame on its side, with components mounted and wired. The pushbuttons used appear too large to mount to the a-frame, which detracts from the portability of the tool. The PLC they chose to utilise was a Mitsubishi Fx1n. The authors do not mention the software used, but it would be a version of GX Developer, like the previously mentioned tool. However, with this tool the designers fully intended for it to be utilised using multiple programming languages including Ladder logic, Instruction List & Sequential Function Chart. As one of the authors objectives was for their tool to be able to be used with Ladder logic as well as at least two other programming languages, this was an important point to make.

Commercially available PLC training tools are also already available in a fairly wide range of variety and prices. Most kits available in the price range that this project was proposing to remain within are limited to digital I/O only, and of that only a small number of them available for use. For $269.95USD, a PLC training tool can be purchased that utilises an Automation Direct CPU (PLC Cable 2023). At this time, that is $405AUD for a training tool that allows for only 5 digital inputs and outputs and can only be programmed in ladder logic. A kit that would come closer to meeting the requirements laid out for this project would cost $1499.95USD (PLC Cable 2023). This kit utilises an Allen Bradley Micro850 CPU, which will allow for the use of multiple programming languages, The kit uses 8 digital inputs and outputs, as well as 2 analogue inputs and outputs. However, at a price point of $2253AUD this kit is well outside the proposed price point for this project.

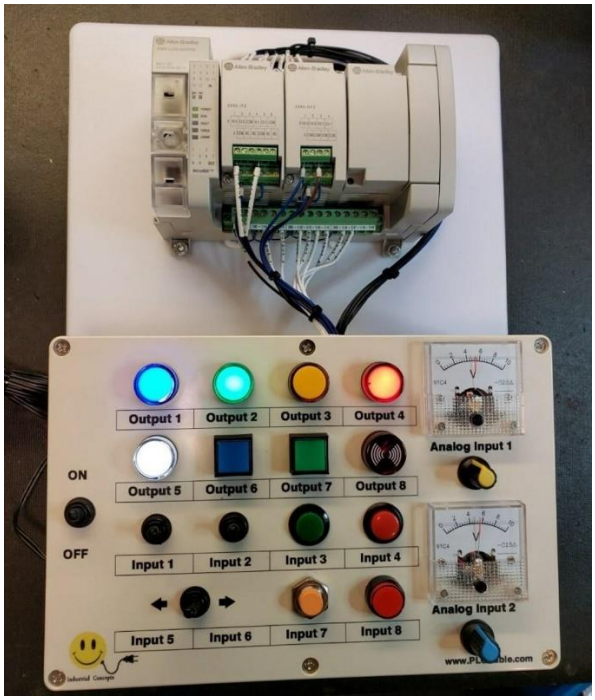


Figure 8 – Allen Bradley Deluxe Analog Trainer (PLC Cable 2023)

There have been many PLC training tools designed and constructed in the past, clearly of varying sizes, costs, and methods. However, the vast majority of them appear to focus solely on programming in Ladder Logic only. There is also a lack of focus on the wiring and programming of actual components that would be used in an industrial setting, like VSD motors, valves, etc.

PLC Options

Siemens S7-1200

The Siemens S7-1200 is a compact PLC option consisting of a varying number of digital I/O, and analogue I/O. It is expandable to allow for more of these inputs and outputs, in addition to other options including communication modules and modules dedicated to condition monitoring and weighing technology, amongst many others. It utilises software called Totally Integrated Automation (TIA) Portal (Siemens 2022). TIA Portal offers multiple programming languages including Ladder Logic, Function Block Diagram, Sequential Flow Chart, Structured Text, etc. A paid licence is required to use TIA Portal, however without the appropriate subscription level to the Siemens website it isn't immediately clear on the actual price for this licence (Siemens 2022).

Figure 9 – Siemens S7-1200 (Siemens 2022)

Allen Bradley Micro 800 Series

The Micro 800 Series is a range of Micro PLCs released by Allen Bradley. The lower end of the range has a very limited ability to expand on the built in number of I/O, with only 2 ports available for expansion modules. This would include the addition of an analogue I/O module. However, as the range moves into the higher end the PLC becomes quite a bit more modular. This means an end user has quite a bit of freedom to simply purchase the modules that they need for their particular project (Rockwell Automation 2022). They vary in physical size from 10 I/O points up to 48 I/O points in the base, as well as the ability to add on expansion modules to that base. These expansion modules include digital I/O, analogue I/O, relay outputs as well as thermocouple/RTD inputs. The software required to program this controller is called Connected Components Workbench. This software allows for programming in

multiple languages, including Function Block Diagram, Structured Text & Ladder Logic. There is no licence required to utilise this software for the Standard version, but it does not allow for runtime downloads or controller simulation (Rockwell Automation 2022). There is a 'developer version' but it costs the user. The price of the PLC itself seems to range from around $350 for the lower end models through to $700 or higher for the more advanced models.

Figure 10 – Allen Bradley Micro 830 PLC (Rockwell Automation 2022)

Delta Electronics DVP-ES3 Series

The DVP-ES3 series of PLC released by Delta Electronics has a range of I/O options available, from as little as 8 digital inputs and outputs up to 40. There are no in-built analogue I/O, but there this series of PLC does have the ability to have extension modules connected and there is a range of analogue I/O extension modules available (Delta Electronics 2023). The programming software for this PLC is called ISPSoft. This software is free to download and use, and depending on the PLC model being used allows for multiple programming languages to be used including ladder, FBD & structured text. Delta Electronics is also listed as a compatible manufacturer on the Codesys website (Codesys 2023). This PLC can be found online for around $500 to $1000 depending on the chosen model. An analogue extension module would also be required which can be found for a little over $100.

Figure 11 – Delta Electronics DVP-ES3 Series PLC (Delta Electronics 2023)

Automation Direct H2 Series

The H2 series of PLC are a pair of controllers released by Automation Direct. It is entirely modular, in that a base must be purchased and populated with the CPU and any other cards required. What this means is that the number of I/O available is a very wide range of options based on the specific configuration installed (Automation Direct 2022). The largest base has 9 slots available, so there is quite a number of available configurations. The software available for programming these controllers is called Do-More Designer. This software is completely free to use, with no licencing required. Do-More Designer only allows for Ladder Logic, or a slight mix of Stage and Ladder Logic, as a programming language (Automation Direct 2022). The cost of the PLC itself will obviously vary depending on the configuration desired. A base CPU will cost approximately $395, while a base to mount it in will cost at least $187 for a 3 slot unit. Digital I/O cards then cost upwards of $100, so in total a user would be looking at paying at least $700 to get this PLC close to meeting the objectives of this project.

Figure 12 – Automation Direct H2 Series PLC (Automation Direct 2022)

Beckhoff CX7000 Series

The CX7000 series is a range of Embedded PLCs from a manufacturer called Beckhoff. The base CPU comes with 8 digital inputs, and 4 digital outputs. It can be expanded to include analogue inputs, analogue outputs, encoder cards, etc. There doesn't seem to be a limit on the number of expansion cards that can be added, except for the limit of 1.5A on the bus from the CPU (Beckhoff Automation 2022). It does not require a base, the modules simply fit together and are mounted onto din-rail. The software used to program this PLC is called Beckhoff Automation TwinCat 3. This software allows for a large range of programming languages to be used, include Instruction List, Structure Text, Ladder Logic, Function Block Diagram, Sequential Function Chart and Continuous Function Chart. The basic version of this software is free to all users, and that version would be sufficient to perform the work required of this project. The base CPU can be found available for around $2200, and this will then require further expansion modules to meet objectives.

Figure 13 – Beckhoff CX7000 Series PLC (Beckhoff Automation 2022)

Microcontroller Options

The options for Microcontrollers that are available are increasing seemingly every day. For the purpose of this proposal, only Microcontrollers that have a suitable number of digital I/O & analogue I/O will be assessed.

Arduino

Arduino is a range of open source electronics hardware and software equipment, mainly aimed at building digital devices (Arduino 2022). They have a varying range of digital inputs and outputs,

analogue inputs and, depending on the model, pulse width modulating outputs or analogue outputs. These microcontrollers operate at a small range of very low voltages, down to 3.3v. They are programmed using software called Arduino IDE. Arduino IDE utilised a programming language very similar to C++, and the programs written in this software are called 'Sketches'. As mentioned earlier, this software is open source so does not cost the user anything to purchase or use (Arduino 2022). Due to its low operating voltage, this unit would likely need other components such as relays between itself and any PLC it is to be connected to to avoid damage to the microcontroller.

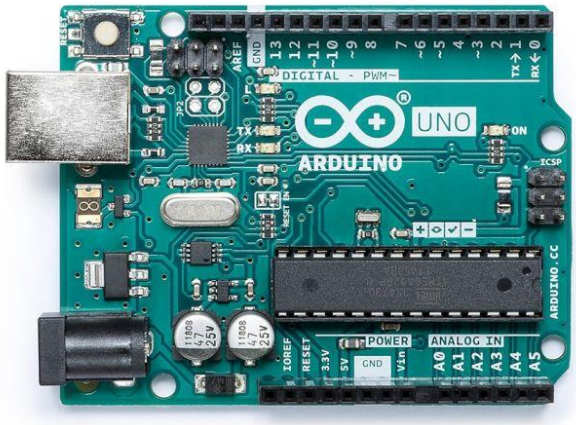


Figure 14 – Arduino Uno R3 (Arduino 2022)

Teensy 32 Bit

The Teensy range of Microcontrollers was developed and released by PJRC over the last few years (Favela & Kaye 2021). The latest release in this range is the Teensy 4.1. It comes constructed with ethernet capability, 40 digital I/O pins, 31 PWM output pins and 14 analogue input pins. Using a low pass filter, the PWM output pins can be used as analogue outputs. The Teensy range is programmed using the Arduino IDE software, with an add-on called Teensyduino. As such, it uses the same programming language as Arduino – the modified version of C++. This also means that the software for these devices is open source and therefore free to use. The Teensy range has more accuracy and processing power than the Arduino range does, if an application required those characteristics.

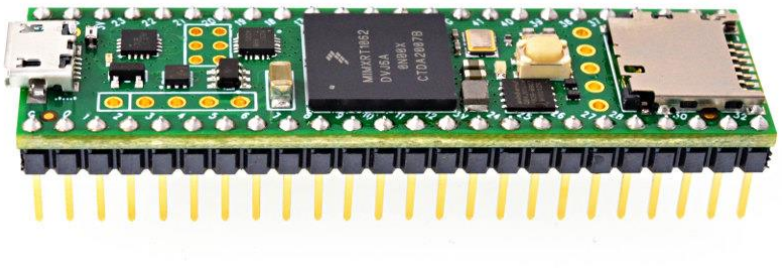


Figure 15 – Teensy 4.1 (PJRC 2022)

Launchpad MSP430

The Launchpad MSP430 range of microcontrollers was released by Texas Instruments. It is a 16-bit microcontroller, designed to utilise certain features that are "not commonly available with other microcontrollers" (TI 2022). These features include having a full system on a single chip, extremely low power consumption (only 4.2nW per instruction), very high speed (approx. 300ns per instruction), etc. The software required to program this range of microcontrollers is called TI Code Composer Studio IDE. It seems to be a free software for use with writing code for many TI products. The programming languages for this software are C/C++. The cost of purchasing these microcontrollers seems to be in the range of $20 to $30AUD.

Figure 16 – TI Launchpad MSP430 (TI 2022)

PID Loop Control

Proportional-Integral-Derivative (PID) control loops are a widely utilised method of controlling processes and machinery commonly found in industrial automation situations.

PID control loops are a feedback style control system that makes use of three different primary control actions to affect a process variable such as temperature, level, pressure, etc. to hold that variable as

close as possible to a desired setpoint. These three primary control actions are obviously a proportional action, integral action and a derivative action (NI 2023).

Proportional control: The proportional control component of these controllers is directly proportional to the error. The error in these control loops is the difference between the desired setpoint and the actual value of the process variable. So if the difference between the setpoint and the process variable is great, then the response from the proportional component will also be great. As that error reduces, so will the response from the proportional component.

Integral control: The integral control component of these controllers considers the amount of time the error has been in place. The longer an error has existed, the more this component of the controller will increase the effect on the overall output of the controller to reduce any long term steady state errors.

Derivative control: The derivative component of these controllers attempts to predict the future trend of the error by measuring the rate of change of the error. This is then used to counteract any sudden changes and dampen oscillations in the process variable. This component is more rarely used than the proportional and integral components.

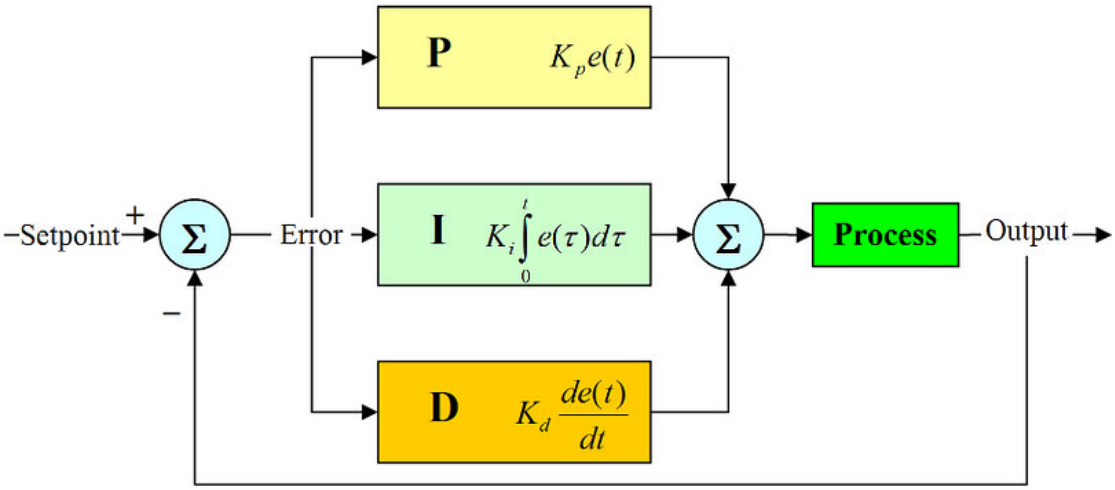


Figure 17 – Flowchart of PID Control Loop algorithm (Technology Robotix Society 2019)

In a PID control loop, the process variable is measured constantly, and the controller compares this value to the current setpoint which calculates the current error of the system. The controller then calculates the response from each of the three components detailed above and sums them to determine the appropriate output to affect the process variable and attempt to reduce the error. In real world terms, this may mean increasing the speed of a pump, or opening a valve further in response to this change in the controller (Technology Robotix Society 2019).

How these three components behave depends heavily on the tuning of the control loop to use optimal parameters. This process is undertaken to find the most appropriate response of the process variable to changes in the desired setpoint.

The purpose of tuning is to reduce or eliminate certain types of error in the response. These qualities of the response can be seen in the figure below, and are things like (NI 2023):

- Overshoot – This is commonly found when the proportional component is too large and causes the process variable to go past the setpoint and will then need to come back the other way to reach the desired setpoint.
- Rise & Settling Time – This is the time it takes to reach the window in which the process variable is considered at the target setpoint. These can be reduced through altering these components, but with a faster response time comes the risk of overshoot.
- Steady-State Error – This is an error that has existed for too long with no movement towards the setpoint. This occurs when the integral portion of the controller is too small and ineffectual.
- Oscillations – This response is when the process variable continually swings too far past the setpoint, and then swinging too far back the previous way. This effect will never allow the process variable to meet the setpoint. It is caused by the gains from the three components being too great or too small.

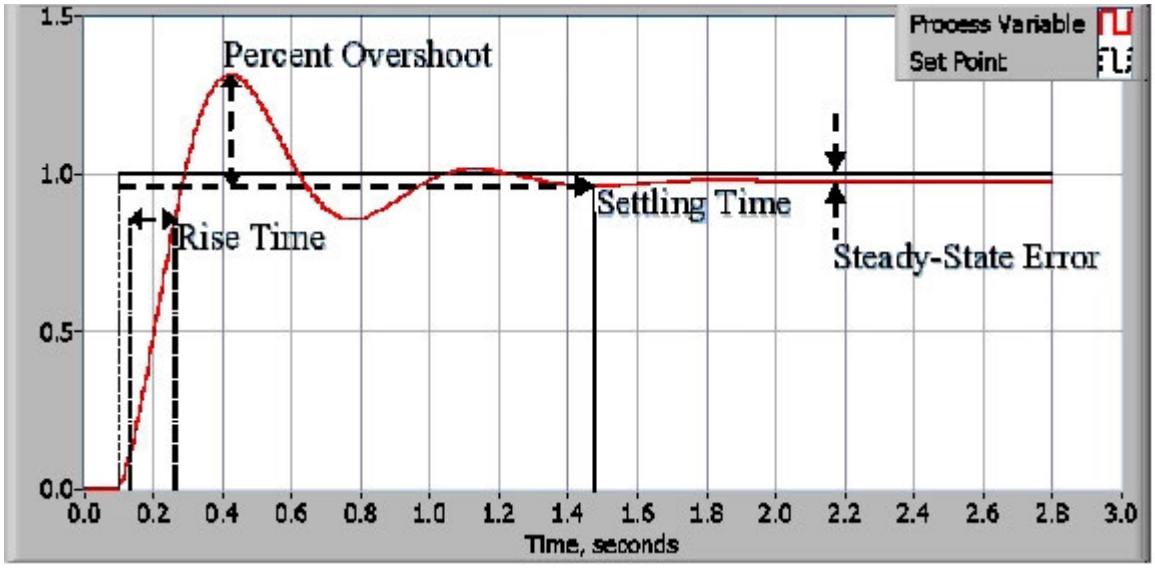


Figure 18 – Response of a typical PID Control Loop (NI 2023)

There are a variety of methods for tuning a PID control loop. These include basic methods like trial and error, to advanced methods that involve mathematic equations to determine the most suitable parameters, all the way to auto-tuning algorithms built into the controllers themselves (NI 2023).

The basic method of trial and error can be quite an effective method for tuning the parameters, but can be quite difficult and risky to perform on a live process. It is done by simply observing the response of the process variable to a change in setpoint, and making changes to the parameters based on that observation. The concern is if the parameters currently in use are too great or small, the process variable may respond in a way that damages physical components.

More advanced methods that make use of mathematical equations to determine suitable parameters allow the tuning of the controller to begin from a point where the parameters are already quite close to being correct for the desired response. One of these methods is known as the Ziegler-Nichols open loop method (Kuphaldt 2009). This method is popular as it involves a systemic approach to determine the appropriate PID parameters through manually driving the process variables response to a step change in setpoint and measuring some of the qualities of that response. These qualities are then used in mathematical equations to find these PID controller parameters. The equations differ depending on if the controller is Proportional only, Proportional-Integral or a full PID controller. The equations for each parameter for a full PID controller are shown below. K_p is the parameter for proportional gain, T_i is the parameter for integral gain and T_d is the parameter for derivative gain. R is the reaction rate in %/min and L is the deadtime in minutes. Reaction rate is the rate in which the process variable is rising or following in response to the step change, and the dead time is the time in which it takes the process variable to change once the step change has been made (Kuphaldt 2009).

$$K_p = 1.2 * \frac{\Delta m}{RL}$$

$$T_i = 2L$$

$$T_d = 0.5L$$

Finally, there are controllers that have an auto-tune feature built into them. These auto-tuners work by performing a frequency-response estimation experiment on the live plant. The response to this experiment on the plant allows the controller to tune the PID parameters to suitable gain levels with little to no input from outside (Mathworks 2023).

4 Methodology

The system proposed in this project was intended to have a central PLC in place, interfaced with multiple microcontrollers that are electrically behaving like a device found in an industrial environment. The microcontrollers were to be programmed to send and receive the signals that one would expect from those devices, which allowed a PLC programmer to write a program that would be similar to those that would need to be utilised in a real world environment. An example of this arrangement is shown below. The PLC is interfacing with the three microcontrollers and each of them is behaving as a different device that would be normally found in a situation often utilised in an industrial environment. In this example, it is a VSD pump being used to fill a Vessel with a discrete valve in place to drain the vessel. The PLC would be able to send the signals to open & close the valve, or run the pump and at what speed, and it would receive things back like whether the valve is opened or closed, if the pump is running and at what speed it is running, or the current level of the vessel. With these signals in place, a user of the tool would then be able to write a program that controls the level of that vessel using the other two devices.

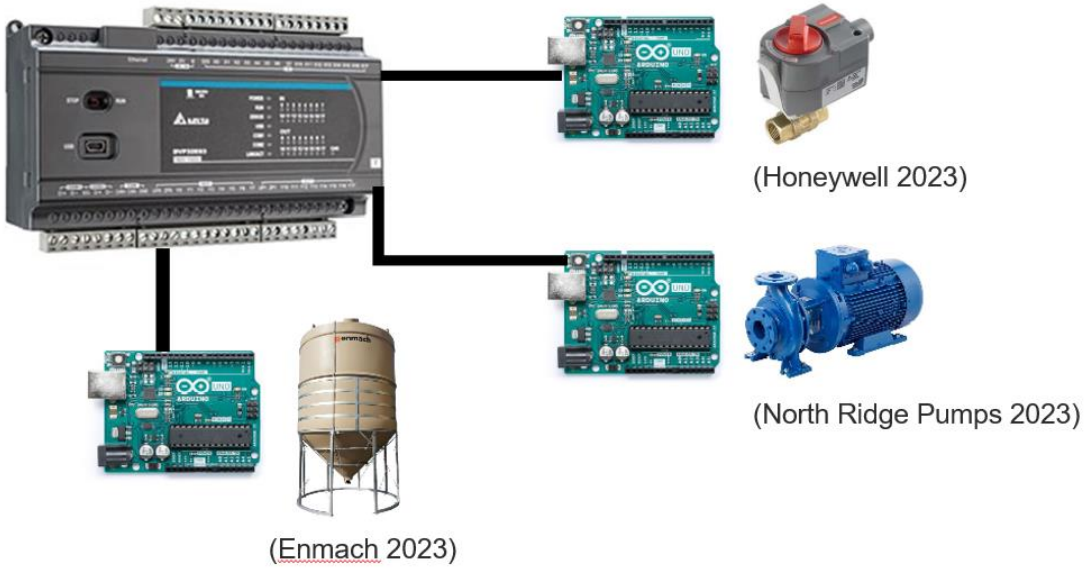


Figure 19 – Arrangement Option of Proposed Training Tool

Technical Requirements

Before any decisions could be made regarding the particular hardware and software for use in this tool, the specific technical requirements needed to be defined. When considering the technical requirements of the components in this system, the minimum number of I/O needed on both the PLC and the microcontrollers would come down to the kinds of devices being simulated by those microcontrollers.

These devices will include the following

- Valves
    - Discrete (Open/Closed)
    - Analogue (Positional)
- Vessel
- Motors
    - Direct On Line (DOL)
    - Variable Speed Drives (VSD)
- Heater

Due to the compact nature of the tool, these devices will not act simply as a single final control device. They will encompass the device as an overall system, and include all of the I/O that one would expect around that device (e.g. Discrete Valve will include reed switch feedback, etc.).

Table 2 – Standard I/O Requirements for Common Devices (Kuphaldt 2009)

Device	D/I	D/O	A/I	A/O	Comments
Discrete Valve	1	1	0	0	Discrete signal to operate, discrete signal back to indicate successful movement.
Analogue Valve	0	0	1	1	Analogue signal to change position setpoint (SP). Analogue signal back to indicate current position.
Vessel	1	2	1	1	Discrete signals for Hi Hi & Lo Lo signals as well as method of filling/emptying ON, analogue signal back to indicate current level or weight and analogue signal in to indicate setpoint.
DOL Motor	1	2	0	0	Discrete signal to run motor, discrete signals back to indicate motor running or motor TOL fault.
VSD Motor	3	2	1	1	Discrete signals in to run motor forward & backward and to reset any faults present. Discrete signals back to indicate the drive is running and any faults are present. Analogue signal in to change speed SP. Analogue signal out to indicate motor speed or current (DO Supply 2022)
Heater	0	0	1	1	Analogue signal to change temperature SP. Analogue signal back to indicate current temperature.

As can be seen from the above Table 2, the device with the most I/O requirements was a VSD (Kopczyk 2018). This means that the minimum I/O requirements for the microcontrollers is the below.

- 3 Digital Inputs
- 2 Digital Outputs
- 1 Analogue Input
- 1 Analogue Output

Using this information, the minimum I/O requirements for the PLC can now be calculated. This tool was to incorporate at least 3 microcontrollers, so this calculation will be based off that. For just the microcontroller indication & control alone the PLC will be required to have the below.

- 6 Digital Inputs
- 9 Digital Outputs
- 3 Analogue Inputs
- 3 Analogue Outputs

The decision was made to add two push buttons and a two-position selector switch to the tool to allow the user to have external influence on the program as well. These will add 3 digital inputs to that total.

These requirements should be fairly easily met by most PLCs available if the correct model was selected. An issue may arise with the analogue I/O, as that many analogue signals can push the cost of the PLC up higher than this project will allow.

Suitable Components

Suitability of components mainly came down to the below –

- Ability to meet technical requirements laid out above.
- Price within allowable range to meet tool objectives.
- Leadtime within reasonable range to ensure project will remain on schedule.

Consideration of the previously researched components was then undertaken to determine which, if any, would be suitable to meet the above requirements. The following major items were selected for use in this tool.

The PLC selected for this project was a Delta ES3 Series, part number DVP32ES311T (Delta Electronics 2023). This PLC operates off a 24vDC supply, has 16 digital inputs and 16 digital transistor outputs. It connects to the PC for programming via a standard ethernet cable. An analogue extension module was required as well, to achieve the analogue signal requirements. This module part number is a Delta DVP0XA-E2. This extension allows for 4 analogue inputs (current or voltage), as well as 2 analogue outputs (current or voltage). This will mean that there will be one microcontroller that does not have access to an analogue output signal from the PLC.

The software for this PLC, as mentioned earlier, is called ISPSoft and it requires no licencing to use. It allows for multiple programming languages to be used, depending on the model of PLC chosen. This particular model will allow for most of the languages to be used, including Ladder Diagram, Structured Text & Sequential Function Chart which will allow it to meet one of the main requirements laid out for this project which was to be allow users to write programs in at least 3 programming languages.

On top of that, Delta Electronics are compatible for use with the open-source PLC programming software Codesys. This software will also allow for programming in many different programming languages, so the option will be there for both tool designer and tool user to choose between two different software's based on personal preference.

One issue that arose was interfacing between the PLC and the microcontrollers will require some sort of relay between on the digital I/O. This was because the PLC is operating on 24vDC and the microcontrollers will be outputting and expecting to receive 5vDC. This was a minor issue, as relays are quite small and inexpensive – but still a consideration that had to be made.

The price for these PLC units was $756.40, which was a little higher than estimated in the resource planning section. However, the configuration met the requirements of the project perfectly and the lead-time was found to be very suitable. Reasonable lead-time was found to be one of the most important factors when considering components for this project.

The microcontroller selected for this project was Arduino Uno R3 (Arduino 2023). This microcontroller has 14 digital I/O pins available. Of this 14 however, 6 of them are available as Pulse Width Modulated outputs which will be used as analogue outputs from the microcontroller for this tools purposes. There are 6 analogue inputs pins available also. The microcontroller will need 7-12v to operate, so some circuitry will be required to supply them from the 24vdc power supply.

This unit can be programmed by the standard Arduino programming software, Arduino IDE (currently version 2.1.0). This software is freely available from the Arduino website to download and use.

In addition to the three microcontrollers needed for this project, the decision was made to include a small HMI for use with these microcontrollers. This was to make it simpler and clearer for the tool user to select the operation mode for each microcontroller. After some investigation into the options available, an RGB LCD Shield that is compatible with the Arduino Uno R3 was chosen. This shield was constructed by Adafruit, and it has a 16x2 LCD and up to 5 keypad buttons available for use and control. It connects to the microcontroller via the two I2C pins, which were not going to be utilised otherwise in this project so nothing was lost I/O wise through the addition of this device.

The cost of the microcontrollers was $16.90 each, and the LCD shield was $49.20. The total order for microcontrollers, LCD shield and jumper wires for use in connection to/from these devices was

$135.23. This amount comes in at nearly half of the estimated amount in the resources section above, which helps offset the PLC price going over its estimate.

Electrical Schematics

See Appendix I for completed electrical schematics of the tool.

Operational Requirements

The I/O allocation for each virtual device will simply begin at the first of each type, no matter what device was selected. This will allow for simplicity in the interconnections between PLC & Microcontrollers, since they are to be connected permanently. Table 3, 4, 5 & 6 below will detail the specific interconnections between the PLC and each individual microcontroller.

Table 3 – Overall Interconnection Between PLC terminals & all Microcontroller pins

PLC Connection		Micro 1	Micro 2	Micro 3
X0	----------	D2		
X1	----------	D3		
X2	----------	----------	D2	
X3	----------	----------	D3	
X4	----------	----------	----------	D2
X5	----------	----------	----------	D3
Y0	----------	D8		
Y1	----------	D9		
Y2	----------	D10		
Y3	----------	----------	D8	
Y4	----------	----------	D9	
Y5	----------	----------	D10	
Y6	----------	----------	----------	D8
Y7	----------	----------	----------	D9
Y8	----------	----------	----------	D10
V1+	----------	D11		
V2+	----------	----------	D11	
V3+	----------	----------	----------	D11
VO1	----------	A0		
VO2	----------	----------	A0	

Table 4 – Interconnection Between Microcontroller 1 & PLC

Micro Controller 1	M/C I/O Type	Interface Relay Number	PLC Connection	PLC I/O Type
D2	Discrete Output	R1	X0	Discrete Input
D3	Discrete Output	R2	X1	Discrete Input
D8	Discrete Input	R7	Y0	Discrete Output
D9	Discrete Input	R8	Y1	Discrete Output
D10	Discrete Input	R9	Y2	Discrete Output
D11	PWM Output	Direct	V1+	Analogue Input
A0	Analogue Input	Direct	V01	Analogue Output

Table 5 – Interconnection Between Microcontroller 2 & PLC

Micro Controller 2	M/C I/O Type	Interface Relay Number	PLC Connection	PLC I/O Type
D2	Discrete Output	R3	X2	Discrete Input
D3	Discrete Output	R4	X3	Discrete Input
D8	Discrete Input	R10	Y3	Discrete Output
D9	Discrete Input	R11	Y4	Discrete Output
D10	Discrete Input	R12	Y5	Discrete Output
D11	PWM Output	Direct	V2+	Analogue Input
A0	Analogue Input	Direct	V02	Analogue Output

Table 6 – Interconnection Between Microcontroller 3 & PLC

Micro Controller 3	M/C I/O Type	Interface Relay Number	PLC Connection	PLC I/O Type
D2	Discrete Output	R5	X4	Discrete Input
D3	Discrete Output	R6	X5	Discrete Input
D8	Discrete Input	R13	Y6	Discrete Output
D9	Discrete Input	R14	Y7	Discrete Output
D10	Discrete Input	R15	Y10	Discrete Output
D11	PWM Output	Direct	V1+	Analogue Input

Snippets of these interconnections can be seen in the below figures of the electrical schematics showing the ways that the PLC and one of the microcontrollers interconnect. As can be seen from the first schematic figure 20, the PLC makes use of the 24VDC power supply for its inputs and outputs, which goes through the normally open contacts on relays R1 to R6. These relays are on the digital output pins of the microcontrollers, which allows for discrete signals to be passed from the microcontrollers to the PLC, and can be seen physically in the second figure 23 showing the actual tool. The PLC outputs are connected in NPN configuration to relays R7 to R15. As can be seen in the second schematic figure 22, the microcontrollers utilise the normally open contacts on these relays to energise their digital input pins. This allows for discrete signals from the PLC to be received by the microcontrollers. These relays can be seen in figure 21 below showing the actual tool.

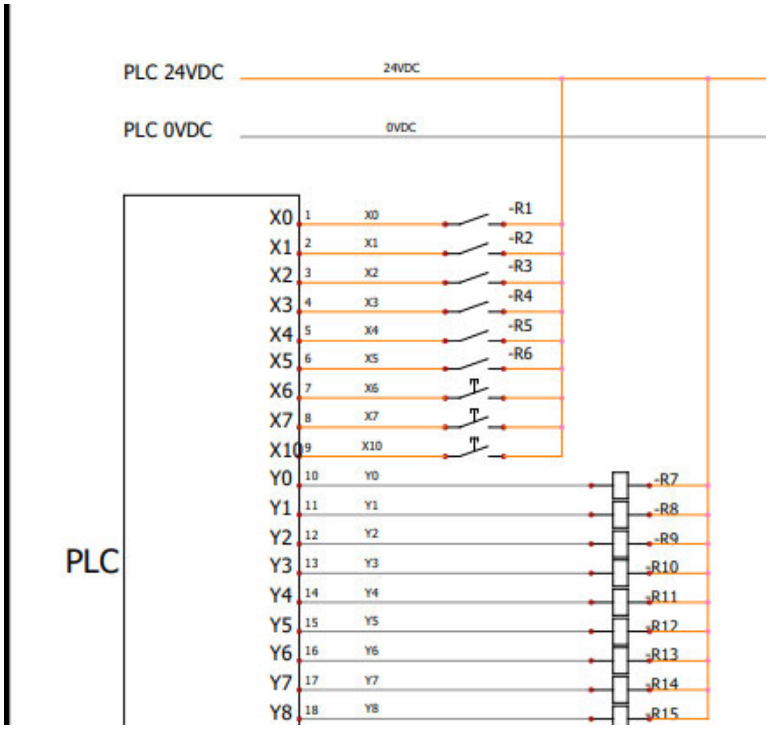


Figure 20 = Electrical Schematic snippet of PLC Wiring

Figure 21 – Relays R7 – R15 in Training Tool

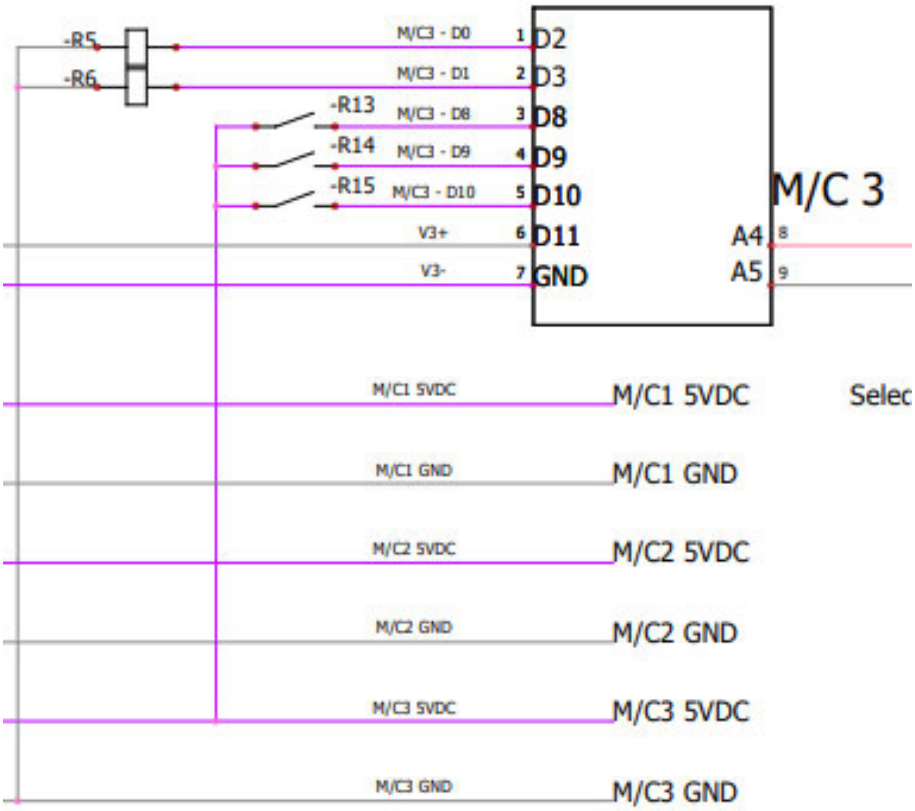


Figure 22 – Electrical Schematic snippet of Microcontroller Wiring

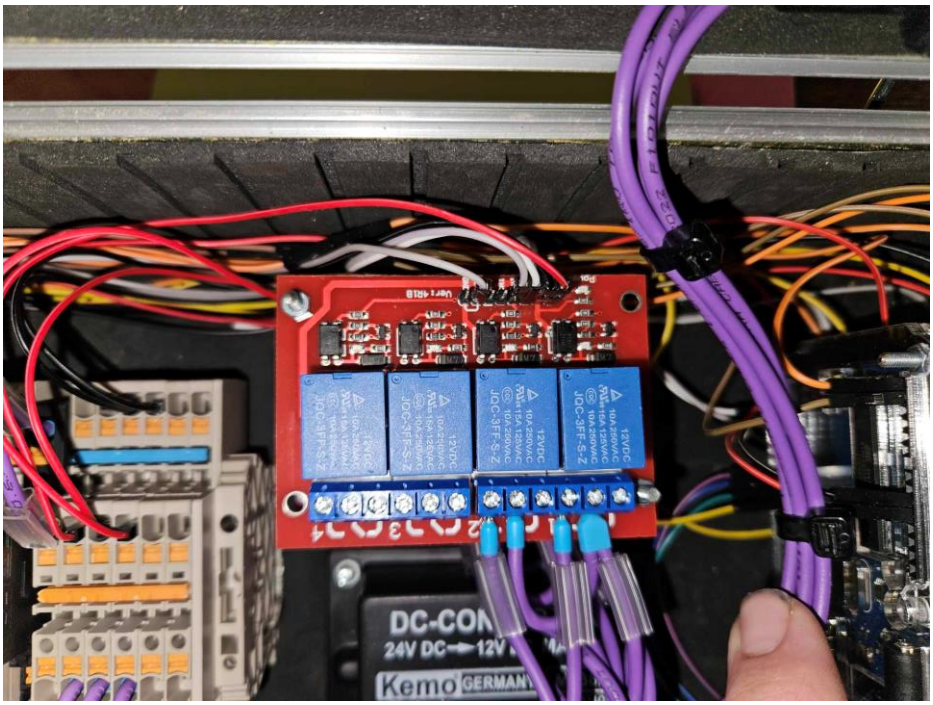


Figure 23 – Relays R1 – R6 in Training Tool

The device behaviour of the microcontrollers will closely mimic the typical behaviour of the simulated devices (Kuphaldt 2009).

A discrete valve will receive an ‘open’ signal and will return an ‘open’ signal in a reasonable amount of time afterwards.

A position valve will receive an analogue signal which will be a position setpoint from 0 to 100% open. It will then return an analogue signal to the PLC which will be a current position value of 0 to 100%. These analogue signals will be able to be utilised in PID loop tuning.

A vessel will receive a discrete signal to indicate that the method of filling/emptying it is on (pump running, valve open, etc.). It will send back discrete signals for HiHi and LoLo signals. An analogue signal will be received to control the setpoint of the vessel level/weight. An analogue signal will be sent back to indicate the current value of vessel level/weight. These analogue signals will be able to be utilised in PID loop tuning.

A DOL motor will receive a ‘run’ signal, and will return a ‘running’ signal in a reasonable amount of time afterwards. It will also send back a Thermal Overload ‘healthy’ signal.

A VSD motor will receive ‘run forwards’, ‘run backwards’ and ‘fault reset’ signals. It will send back ‘running’ and ‘fault’ discrete signals. Analogue signals will be sent back and forth to indicate a speed setpoint, as well as a current speed. These analogue signals will be able to be utilised in PID loop tuning.

A heater will receive an analogue signal to indicate the temperature setpoint, and will return another analogue signal to indicate the current temperature. These analogue signals will be able to be utilised in PID loop tuning.

In addition to the above, the decision was made to include a functionality called Fault Mode. This can be selected from the HMI, and when it is on the microcontrollers will choose a random amount of time and after this random amount of time it will cause a ‘fault’ and send appropriate signals for that fault. This allows the user to program in fault responses to their code and see if in action. These faults can be seen in Table 7 below.

Table 7 – Faults for Microcontroller Devices

Device	Faults
Discrete Valve	Valve Not Opened – Valve has received open signal but opened feedback not received back. Valve Not Closed – Valve has lost open signal but opened feedback is still received.
Analogue Valve	Valve Not in Position – Valve position does not match position setpoint after period of time.
DOL Motor	TOL Fault – Motor has caused Thermal Overload trip. Motor Not Running – Motor has received run signal but running feedback not received.
VSD Motor	VSD Fault – VSD has faulted. VSD Not Running – VSD has received run signal but current speed does not match speed setpoint after period of time.
Vessel	Nil.
Heater	Temperature Fault – Heater has received on signal, but temperature has not risen.

Construction of the Tool

Once all the pieces of hardware for the tool had arrived, the layout of the tool was tested and trialled before finally settling on the most suitable one. These components were then physically mounted into the case onto din-rail, or bolted down as the situation called for.

The PLC, power supply, 24vdc relays and terminals were mounted to the case via din-rail. This allowed for a quick and easy mounting point where the component allowed for it.

As the microcontrollers were to be mounted together, they were put together into their cases for protection from one another. 3mm Threaded rod was used to ensure lengths needed to bolt certain components in place was adequate. They were then bolted into position, with their programming port easily accessible for the tool designer to access.

The LCD screen was disassembled when it arrived, so it needed soldering to ensure its operation. While it was designed to be utilised with a single microcontroller, this project required it to be able to be connected to three through a selector switch. As such, this soldering was done to ensure it could be used as a stand-alone unit and not need to be mounted to the microcontroller. This meant determining which pins did what, and soldering wires to those points on the LCD screen card.

The microcontroller selector switch used with the LCD screen was mounted onto one of the Perspex covers. This allows for user access to the switch in the only position that it would fit within the case.

The Arduino relay modules were bolted together and then to the case with the threaded rod as well.

The pushbuttons and selector switch were mounted onto a length of Perspex and then also bolted into place using the 3mm threaded rod.

Once all components were mounted in place, the electrical schematics were followed to complete the wiring. Finally, when the wiring was completed and had been tested as correct, the Perspex covers were mounted into position to provide safety to the user from the components with accessible 240 volts.

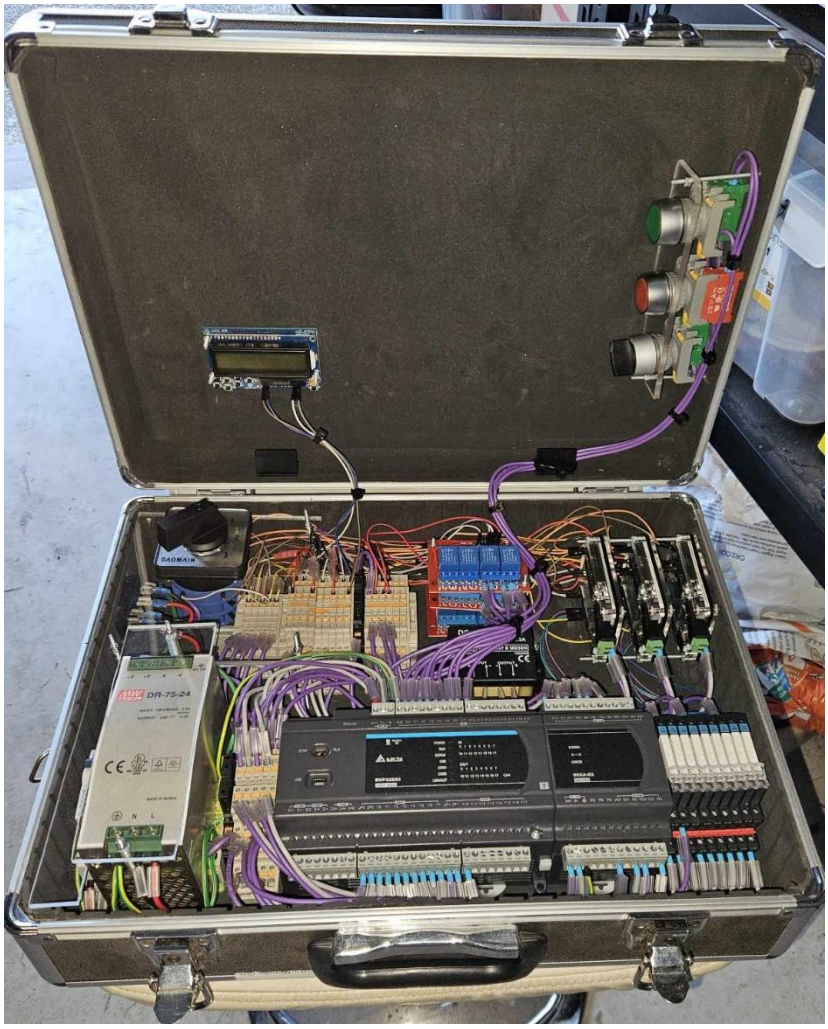


Figure 24 – Constructed Training Tool

The figures shown in Appendix H show further images of the physically completed tool.

Program Microcontrollers & PLC

The programming of the Arduino microcontrollers was be done via their software available, Arduino IDE (Arduino 2023). The Arduino website contains a vast amount of information regarding programming their products including clear descriptions of what each function is called, how to call it in a programmers code and what it does.

Adafruit have also released documents detailing how to use their LCD screen that was utilized in this project (Adafruit 2023). They have made available schematics for the unit, which were helpful in identifying the pins to solder cables to to make the unit standalone. They also have sample programs available which assisted with incorporating the unit into the tools microcontroller program.

There was a slight difference in the programs between microcontroller one, and microcontrollers two & three. The program in microcontroller one does not allow for selection of any analogue devices since the PLC selection did not allow for 3 analogue outputs and microcontroller one was the one to miss out on that connection. Otherwise, the programs across the microcontrollers was identical.

The program for operating the microcontroller as a DOL motor is shown below. As digital input one on the microcontroller is energized, there is a one second delay to simulate the delay in a contactor coming in and then digital output one is energized to indicate that the contactor has come in and the drive is running. When digital input one is deenergized, all the digital outputs are deenergized to ensure readiness for use on the next occasion.

```
  if (currentSelectionInt == 1){ //DOL Motor selected

    if (digitalRead(diginput_one) == LOW){
    delay(1000);
    digitalWrite(digoutput_one, HIGH); // write output to HIGH state to
simulate 'running' signal from contactor
    Serial.println("Contactor running");
    }

    if(digitalRead(diginput_one) == HIGH) {
    delay(1000);
    digitalWrite(digoutput_one, LOW); //Set digital output low when run signal
is removed
    digitalWrite(digoutput_two, LOW); //Set digital output low when run signal
is removed
  }
  }
```

The program written for operation of a VSD motor is shown below. Unlike the DOL motor above, this program receives a speed setpoint from the PLC and this is mapped to allow this value to scale from the analogue input range to suit the PWM output range. Once digital input one is energized, digital

output one is immediately energized to indicate the drive is running. The ‘speed of the motor’ is then altered based on what its current value is compared to the speed setpoint being received on analogue input one. If the motor speed is lower than the setpoint, it will add 5 to the output value each loop of the code until it meets or exceeds the setpoint value. If the motor speed is higher than the setpoint it will remove 5 from the output value every loop of the code until it is no longer higher than the setpoint. If at any point digital input one is deenergized, simulating losing the run signal, the output value will reduce by 5 each loop of the code until it has reached zero. This simulates the drive ramping down to a stop.

```
  if (currentSelectionInt == 2){ //VSD Motor selected
// read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range

  if(digitalRead(diginput_one) == LOW){
    digitalWrite(digoutput_one, HIGH);
  // If the run signal is on and the output speed is lower than the speed SP,
the output speed of the motor will ramp up to speed SP
  if(outputValue <= sensorValue){

    outputValue = outputValue + 5;

  }

  // If the run signal is on and the output speed is higher than the speed SP,
the output speed of the motor will ramp down to speed SP
  if(outputValue >= sensorValue){

    outputValue = outputValue - 5;
  }
  }
  if(digitalRead(diginput_one) == HIGH){
    digitalWrite(digoutput_one, LOW);
 // If the run signal is off and the output speed is higher than 0, the output
value will ramp down to 0
  if(outputValue > 0){

    outputValue = outputValue - 5;

  }
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);
```

```
  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }
```

The program for operating the microcontroller as a discrete valve is shown below. As digital input one on the microcontroller is energized, there is a three second delay to simulate the delay in a valve opening and then digital output one is energized to indicate that the valve has opened. When digital input one is deenergized, again there is a 3 second delay to simulate the speed of a valve closing and then all the digital outputs are deenergized.

```
 if (currentSelectionInt == 3){ //Discrete Valve selected
    if (digitalRead(diginput_one) == LOW){
    delay(3000);
    digitalWrite(digoutput_one, HIGH); // write output to HIGH state to
simulate 'open' signal from valve position
    Serial.println("Valve Open");
    }

 if(digitalRead(diginput_one) == HIGH) {
    delay(3000);
    digitalWrite(digoutput_one, LOW); //Set digital output low when run signal
is removed
    digitalWrite(digoutput_two, LOW); //Set digital output low when run signal
is removed
  }
 }
```

The program written for operation of an analogue, or positional, valve is shown below. Unlike the discrete valve above, this program receives a position setpoint from the PLC and this is mapped to allow this value to scale from the analogue input range to suit the PWM output range. The position of the valve is altered based on what its current position is compared to the position setpoint being received on analogue input one. If the valve position is lower than the setpoint, it will add 10 to the output value, and then wait for a one second delay each loop of the code until it meets or exceeds the setpoint value. If the valve position is higher than the setpoint it will remove 10 from the output value and then wait a one second delay every loop of the code until it is no longer higher than the setpoint.

```
  if (currentSelectionInt == 4){ //Analogue Valve selected

// read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range
```

```
  if(outputValue <= sensorValue && outputValue <= 255){

    outputValue = outputValue + 10;
    delay(1000);
  }

  if(outputValue >= sensorValue && outputValue >= 0){

    outputValue = outputValue - 10;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }
```

The program for the Vessel was split into two, as the control of a vessel being filled by a DOL pump was quite different to the control of a vessel being filled by a VSD pump. The program written for a Vessel – DOL is shown below. In this code digital input one is to indicate that the DOL pump is running, and digital input two is to indicate if the discrete valve on the vessel is open or closed. If digital input one is energized and digital input two is not energized, the output value indicating the level of the vessel will increase by 2 and then wait a one second delay each loop of the code. If both digital inputs one and two are energized, then the output value will still increase but more slowly to simulate the valve being open at the same time as the pump trying to fill the vessel. The output value now only increases by 1 each loop instead of 2. If digital input one is deenergized and digital input two is energized simulating the pump being off but the valve being open, the output value will reduce by 1 and then wait a one second delay each loop of the code. This simulates the level of the vessel lowering as the pump is no longer filling it. If both digital inputs are deenergized then the output value simulating the vessel level will hold at where it is. This simulates that there is nothing filling and nothing emptying the vessel at that time.

```
if (currentSelectionInt == 5){ //Vessel - DOL selected

  // If the pump is on, and the outlet valve is closed the level will rise
quickly
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == HIGH){
    outputValue = outputValue + 2;
    delay(1000);
```

```
  }

// If the pump is on, and the outlet valve is open the level will rise slowly
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == LOW){
    outputValue = outputValue + 1;
    delay(1000);
  }

// If the pump is off, and the outlet valve is open the level will drop slowly
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == LOW){

    outputValue = outputValue - 1;
    delay(1000);
  }

// If the pump is off, and the outlet valve is closed the level will remain
the same
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == HIGH){

    outputValue = outputValue;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }
```

The program written for the Vessel -VSD is shown below. It operates quite similarly to the above Vessel – DOL in some ways, but this needs to take into account the speed of the VSD pump being used to fill the vessel. If digital input one is energized and digital input two is deenergized, the analogue input value indicating the pump speed is converted into a percentage value, and then used to calculate the increase to the output value indicating vessel level. This obviously means that if the pump is running at 100% it the level of the vessel will increase faster than if it was running at 25%. When both digital inputs one and two are energized, this same calculation is in place but the output value is then reduced by 1 to simulate the loss of level from the valve being opened. If digital input one is deenergized and digital input two is energized simulating the pump being off but the valve being open, the output value will reduce by 1 and then wait a one second delay each loop of the code. This simulates the level of the vessel lowering as the pump is no longer filling it. If both digital inputs are deenergized then the output value simulating the vessel level will hold at where it is. This simulates that there is nothing filling and nothing emptying the vessel at that time.

```
if (currentSelectionInt == 6){ //Vessel - VSD selected

 // read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range

  // If the pump is on, and the outlet valve is closed the level will rise in
accordance with the speed of the pump
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == HIGH){
    sensorValue = sensorValue/100;
    if(outputValue <= 251)
    {
    outputValue = outputValue + (2 * sensorValue);
    }
    delay(1000);
  }

// If the pump is on, and the outlet valve is open the level will raise in
accordance with the speed of the pump, and lower with the outflow of the valve
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == LOW){
    sensorValue = sensorValue/100;
    if(outputValue <= 252)
    {

    outputValue = outputValue + (2 * sensorValue);

    }
    if(outputValue > 0)
    {
    outputValue = outputValue - 1;
    }
    delay(1000);
  }

// If the pump is off, and the outlet valve is open the level will lower
slowly with the outflow of the valve
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == LOW){
    if(outputValue > 0)
    {
      outputValue = outputValue - 1;
    }
    delay(1000);
  }

// If the pump is off, and the outlet valve is closed the level will remain
the same
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == HIGH){
```

```
    outputValue = outputValue;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }
```

The program written for the control of a Heater is shown below. If digital input one is energized, the output value simulating the current temperature will increase. This simulates the heater being turned on by the PLC. When digital input one is deenergized the output value will slowly reduce by 1 and wait a one second delay for each loop of the code until it reaches a value that simulates 'ambient' temperature. This is to simulate the fact that these temperature probes should always be reading something due to ambient temperature of the surrounds.

```
if (currentSelectionInt == 7){ //Heater selected

  if(digitalRead(diginput_one) == LOW){
  // If the run signal is on the heater will raise the temperature
    delay(1000);
    outputValue = outputValue + 1;
  }

  if(digitalRead(diginput_one) == HIGH){
 // If the run signal is off, the temperature will lower to ambient
  if(outputValue > 51){
    delay(1000);
    outputValue = outputValue - 1;
  }
  if(outputValue < 51){
    outputValue = 51;
  }
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }
```

Appendix C below has the complete current program that was most recently downloaded to the Arduino microcontrollers in the tool.

Delta Electronics provide a very detailed user manual for the use of ISPSoft, the software used to program the model of PLC chosen for this project (Delta Electronics 2020). This manual was used to configure the initial connection to the PLC to a computer for use of the tool. The default IP address for these PLCs was 192.168.1.5 and the decision was made to keep this value in place for the tool. This was to ensure easier connection to the tool in the event instructions or information regarding this tool were lost and reconnection was required with no knowledge of prior communication information.

Develop Training Scenarios

Prior to the tool being given out for evaluation by users, instructions for the training tool needed to be written. The full PLC Training Tool Instruction document can be seen in Appendix F.

These instructions begin with Connection & Initial Configuration sections where the user was directed to download and install the software required to connect to and program the PLC to a computer available to them. The user was then instructed on how to make connection to the PLC using an ethernet connection. They were given the IP address of the PLC and brief instruction on how to change their own IP address to suit if needed. Once they are connected to the PLC, they were then finally instructed to perform an upload from the PLC which will provide them with all the written sample programs and I/O mapping.

The next section of the instruction was Programming, in which the user was then provided with basic instructions on how to program using ISPSoft, and how to download their program to the PLC once they were ready to do so.

The last section of the instructions was Scenarios. In this section users were given a particular device or arrangement that they were to write a program for. Each device or arrangement comes with setup instructions, in which the user was told how to set up the microcontrollers correctly for that particular arrangement, and a brief description on how that microcontroller should behave with that set up. After the setup, the user was then given instruction on how to write their PLC program to make use of the device or arrangement they have chosen.

Table 8 – Training Scenarios

Scenario	Requirements	Language
DOL Motor	Select appropriate devices on microcontrollers. Write program to start/stop motor with green & red pushbuttons.	Ladder Diagram (LD), Structured Text (ST)
VSD Motor	Select appropriate devices on microcontrollers. Write program to start/stop motor with green & red pushbuttons. Map analogue signals to send speed setpoint and receive current speed.	LD, ST
Discrete Valve	Select appropriate devices on microcontrollers. Write program to open/close valve with selector switch.	LD, ST
Analogue Valve	Select appropriate devices on microcontrollers. Map analogue signals to send valve position setpoint and receive current valve position.	LD, ST
Vessel – VSD	Select appropriate devices on microcontrollers. Write program to start/stop motor with green & red pushbuttons. Write program to start/stop motor with green & red pushbuttons. Map analogue signals to send speed setpoint, and receive current speed. Complete PID control function block and add suitable parameters to tune loop.	LD, ST

Test Operation

Once communication between PLC and computer was configured, a small program was written to test the operation of all the inputs and outputs on the PLC and microcontrollers. This was to ensure that the interfacing between all devices has been wired correctly prior to testing microcontroller device operation.

Once all the digital and analogue inputs and outputs had been confirmed as operating correctly the tool was physically ready for use.

5 Results & Design

Tool Construction

Once the construction of the tool prototype was completed, a review of that process could be undertaken. The components were able to be mounted into the case used and wired up suitably. The tool was made safe with the inclusion of the Perspex covers as designed. All components in the case were mounted securely and should not move in transit. All cables in the tool were terminated with bootlaces to ensure a good connection, and cable label sleeves were added to each wire.

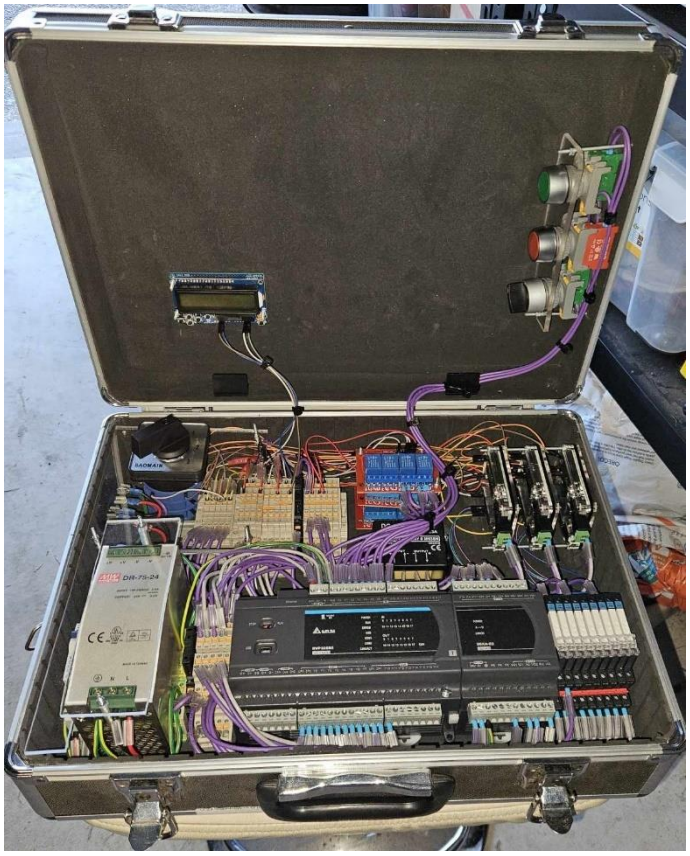


Figure 25 – Completed Training Tool

Whilst all the components were able to fit into the case used to house them all, overall the they were packed in quite tightly to be able to do so. This has led to a general inability to keep the wiring of the tool neat and tidy. It also led to components being mounted in positions where they would fit, not where it would make logical sense to have them. The selector switch for choosing which microcontroller is being used on the LCD screen was mounted quite far away from the microcontrollers where it would have made logical sense for it to be. This had the flow on effect of the LCD screen being away from the microcontrollers as well.

Another effect this had was on the decision to not allow the tool to be rewireable by the tool user. This means that the inputs and outputs were fixed and will not allow a user to gain experience in that side of the process.

The cable label sleeves have yet to be populated with labels, as time did not allow.

Overall, construction also went slightly over budget. The requirement laid out earlier was for it not to cost over $1000, however by the time this prototype was completely built the amount spent had risen to $1166.29. This was mostly due to the PLC costing more than anticipated in the original calculations.

Tool Operation

Each microcontroller device was selected on the LCD screen to have a basic test PLC program written about it to prove each ‘device’ operates as per its description laid out previously. These programs were written in both ladder logic and structured text to ensure that what the tool was asking users to do was actually able to be achieved. These complete sample programs can be seen in Appendix D & Appendix E.

The sample programs were successful in identifying areas of the microcontroller device programming that worked as designed, as well as areas in the program that required either immediate correction or improvement at a later date.

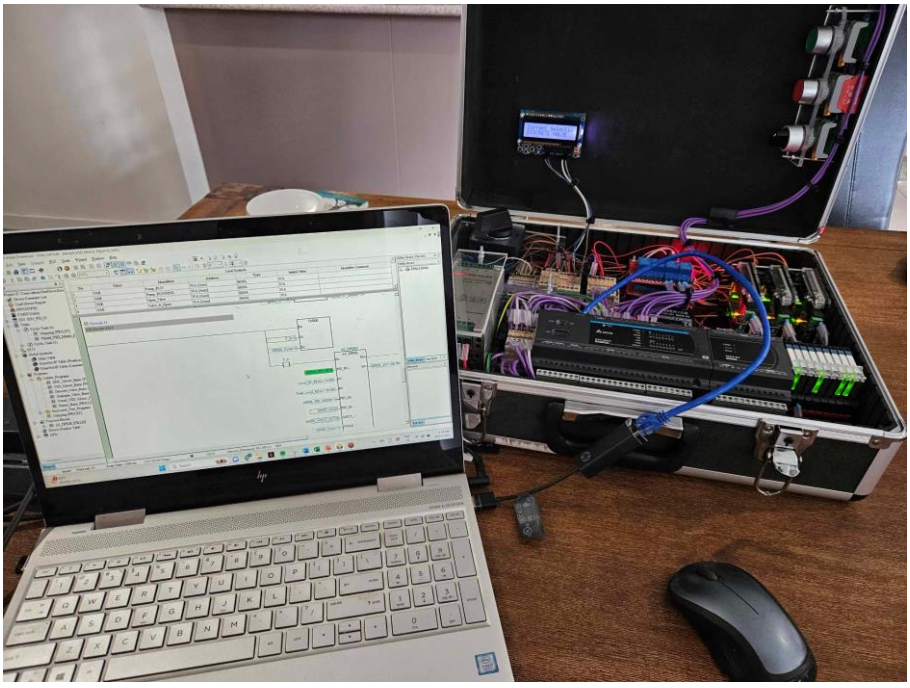


Figure 26 – Testing Operation of PLC Training Tool

The DOL motor in normal operation behaved exactly as designed and required no further corrections or updates. The green pushbutton was used to start the motor, and the red pushbutton was used to stop

it. These signals were mapped across to and from the microcontroller. This can all be seen in the figure below.

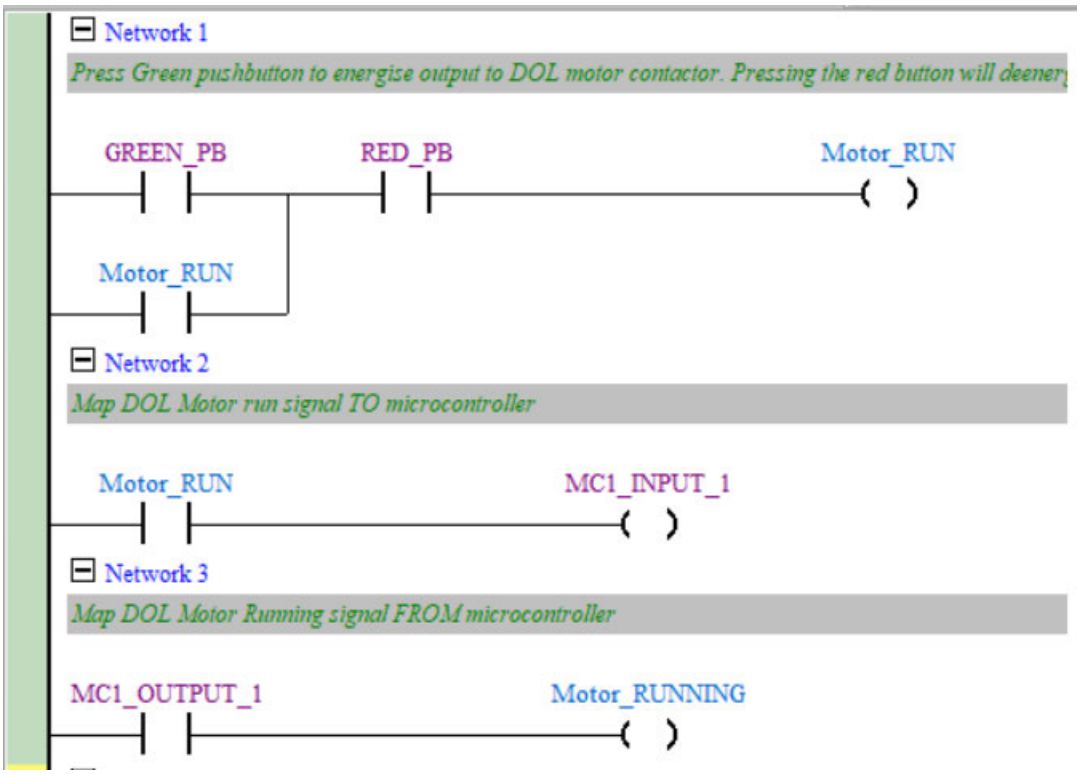


Figure 27 – Snippet of DOL Motor Sample Program

The VSD motor in normal operation behaved roughly as designed, but the ramping up and down of the speed was too slow for the purposes of this tool. This was due to a simple delay in the microcontroller program being too large, so lowering that had the desired effect of decreasing the ramp time. All other signals were deemed correct. The figure below shows the scaling of the analogue signals in this sample program to and from the microcontroller used.

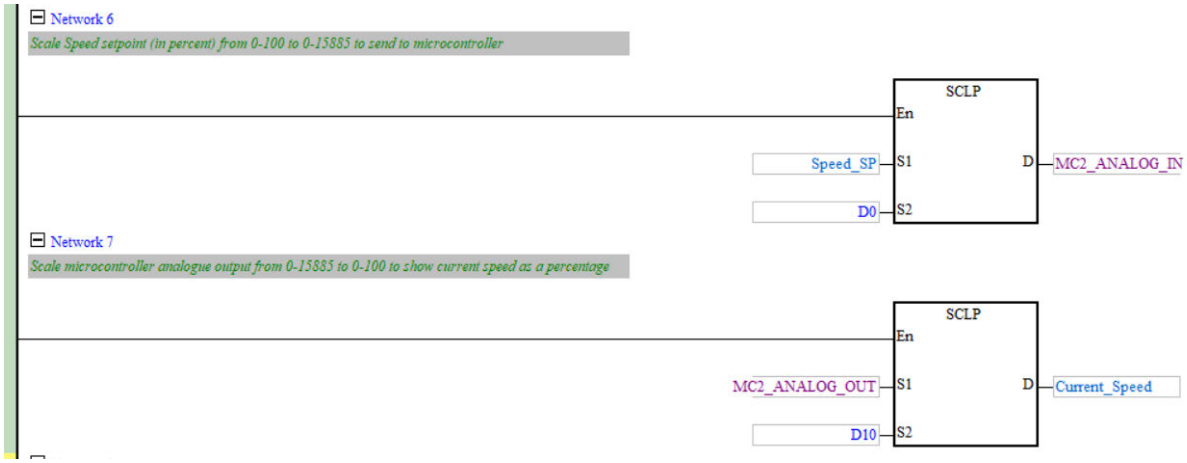


Figure 28 – Snippet of VSD Motor Sample Program

The discrete valve in normal operation behaved exactly as designed and required no further corrections or updates. The selector switch was used to open the valve. This signal was mapped across to and from the microcontroller. This signal mapping can be seen in the figure below.

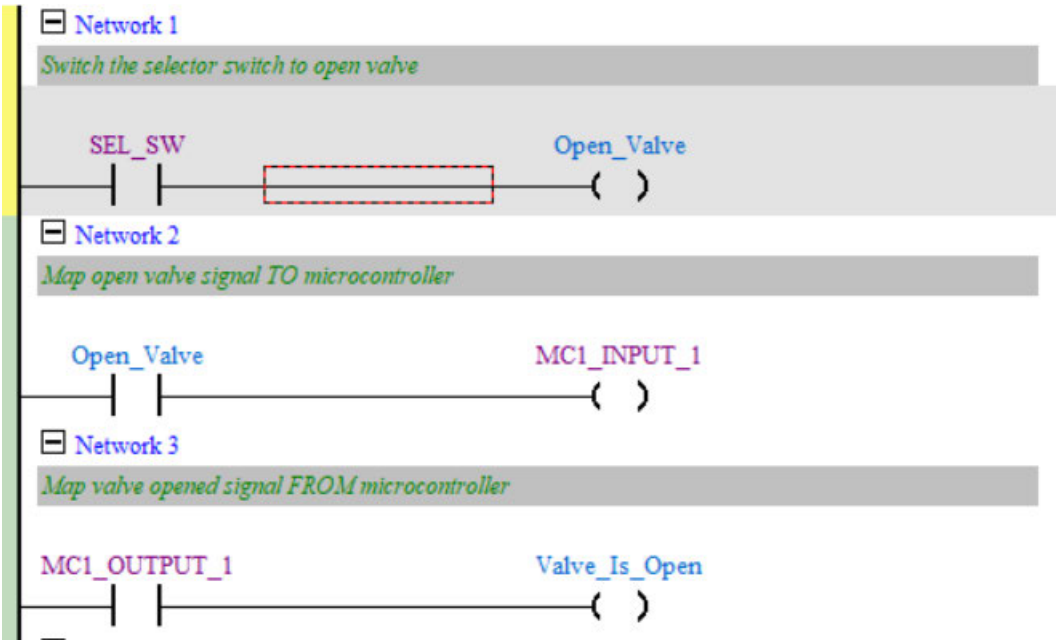


Figure 29 – Snippet of Discrete Valve Sample Program

The analogue valve in normal operation behaved roughly as designed, but it had similar issues to the VSD motor in that the time taken for the valve to change positions was far too slow for the purposes of this tool. The delay mentioned previously was delayed for this device and that corrected this issue. All other signals were deemed correct. The figure below shows the scaling of the analogue signals in this sample program to and from the microcontroller used.

Figure 30 – Snippet of Analogue Valve Sample Program

The vessel in normal operation did not work as designed. Whilst it seemed to work fine for a DOL motor being used to fill it, it was found that the rate at which the vessel would fill did not vary with the

speed of the motor if used in conjunction with a VSD motor. This was a major problem when considering this needed to be able to be utilized in a PID loop tuning scenario. The decision was made to separate out the vessel device into two – one being utilized with a DOL motor, and one being utilized with a VSD motor. The vessel with the DOL motor was left the same, as it behaved as expected. The program for the vessel with VSD motor was altered to consider the analogue signal as a percentage, and then use that percentage to update the level of the vessel if the motor running signal was received. This had the desired affect of varying the rate of change in the level of the vessel.

The heater in normal operation behaved as designed and required no further corrections or updates. The selector switch is turned on, and heating enable is used to control the temperature based on the setpoint. A snippet of this code is shown in the figure below.

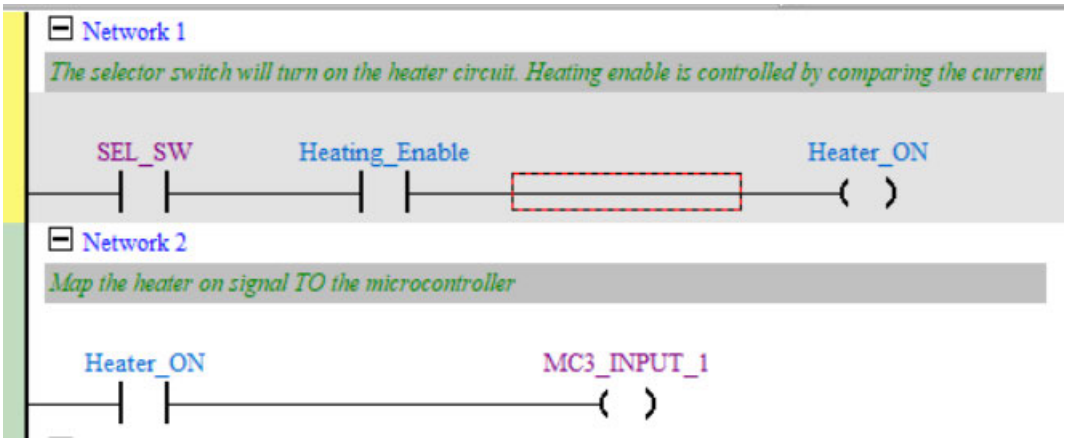


Figure 31 – Snippet of Heater Sample PLC Program

The addition of Fault Mode was never fully implemented to the programming of the microcontrollers, so the operation of it was unable to be tested.

The sample programs previously mentioned were written and tested in both ladder and structured text programming language. However, with the Delta PLC used in this prototype the user was also able to write a program in function chart logic, both sequential and continuous. This fills the previously laid out requirement for this tool to be programmable in at least 3 programming languages.

The final program written for testing the operation of the tool was a combination of VSD motor, Discrete valve, and a Vessel. This was to simulate a VSD controlled pump filling a vessel, with a discrete valve on the outflow of that vessel. This kind of setup allows the user to practice programming and tuning a PID control loop, which is a great feature to be able to utilise with no risk to any live process components. This program allows the user to input a level setpoint for the vessel, and see the pump ramp up or down to try and meet that setpoint. The figure below shows the function block used to control the PID control loop used in this sample program.

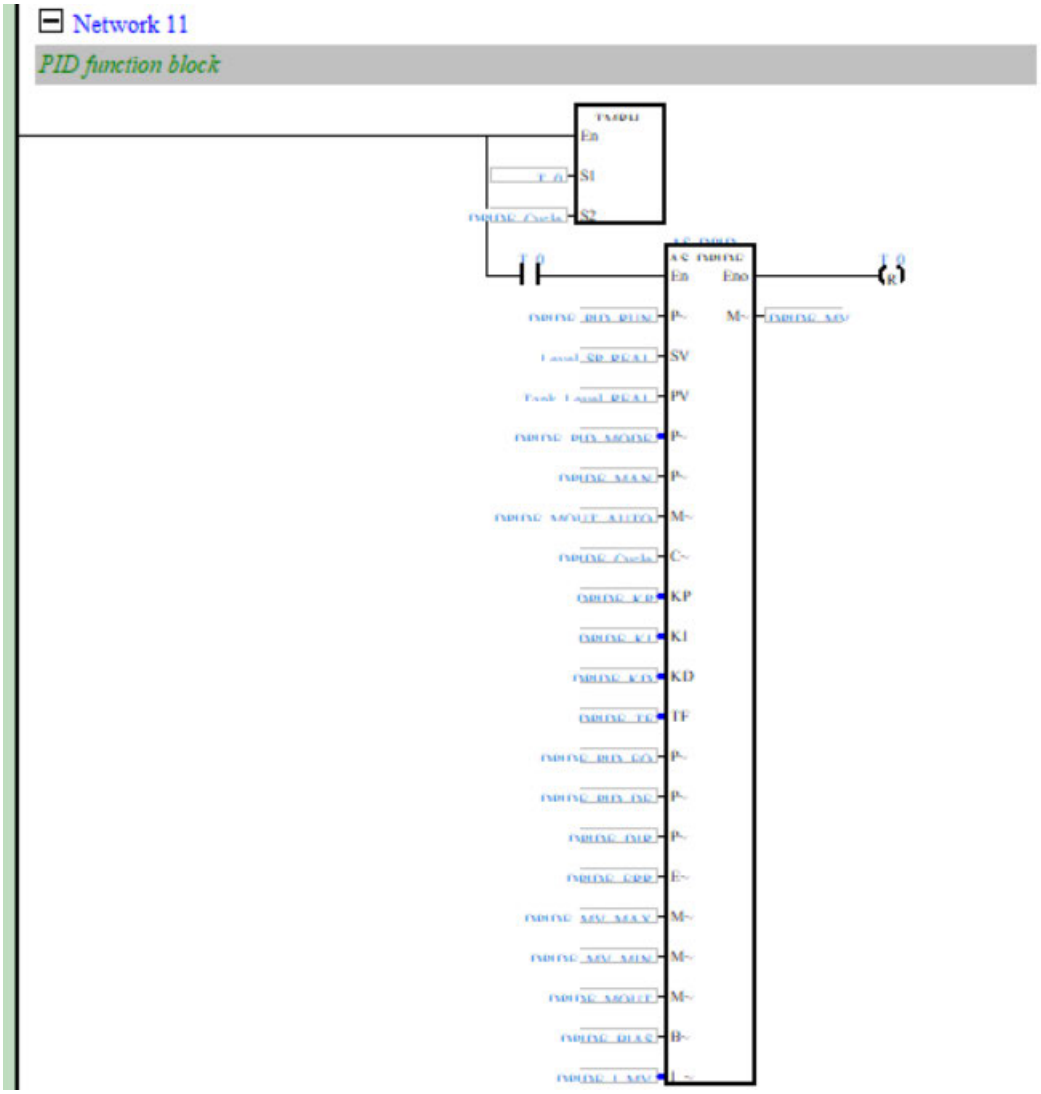


Figure 32 – Snippet of Vessel – VSD PID Loop Tuning Sample Program

A trend of this program working is shown below in Figure 33. The control loop was tuned using the Ziegler-Nichols open loop tuning method. The equations used for finding the gain values for this tuning are shown below.

$$K_p = 1.2 * \frac{\Delta m}{RL} = 1.2 * 25 = 30$$

$$T_i = 2 * L = 2 * 0.1665 = 0.333$$

$$K_i = \frac{1}{T_i} = \frac{1}{0.333} = 3.003$$

$$T_d = 0.5 * L = 0.5 * 0.1665 = 0.08325$$

T_i was changed to K_i as this was the parameter that the PID Control function block in the PLC software was expecting. This was simply a matter of calculating the inverse of T_i.

These parameters were then input to the function block, and the response was observed through the trend wizard in the PLC software. They were then slightly altered to refine the response to the most suitable it could be.

The yellow line on the trend is the setpoint for the vessel level, and the orange line is the ‘actual’ vessel level. This trend shows the process variables lag when the setpoint is changed, as well as the settling time from overshoot to acceptable levels of oscillation around the setpoint.

Figure 33 – Trend of PID Loop Control Tuning Scenario

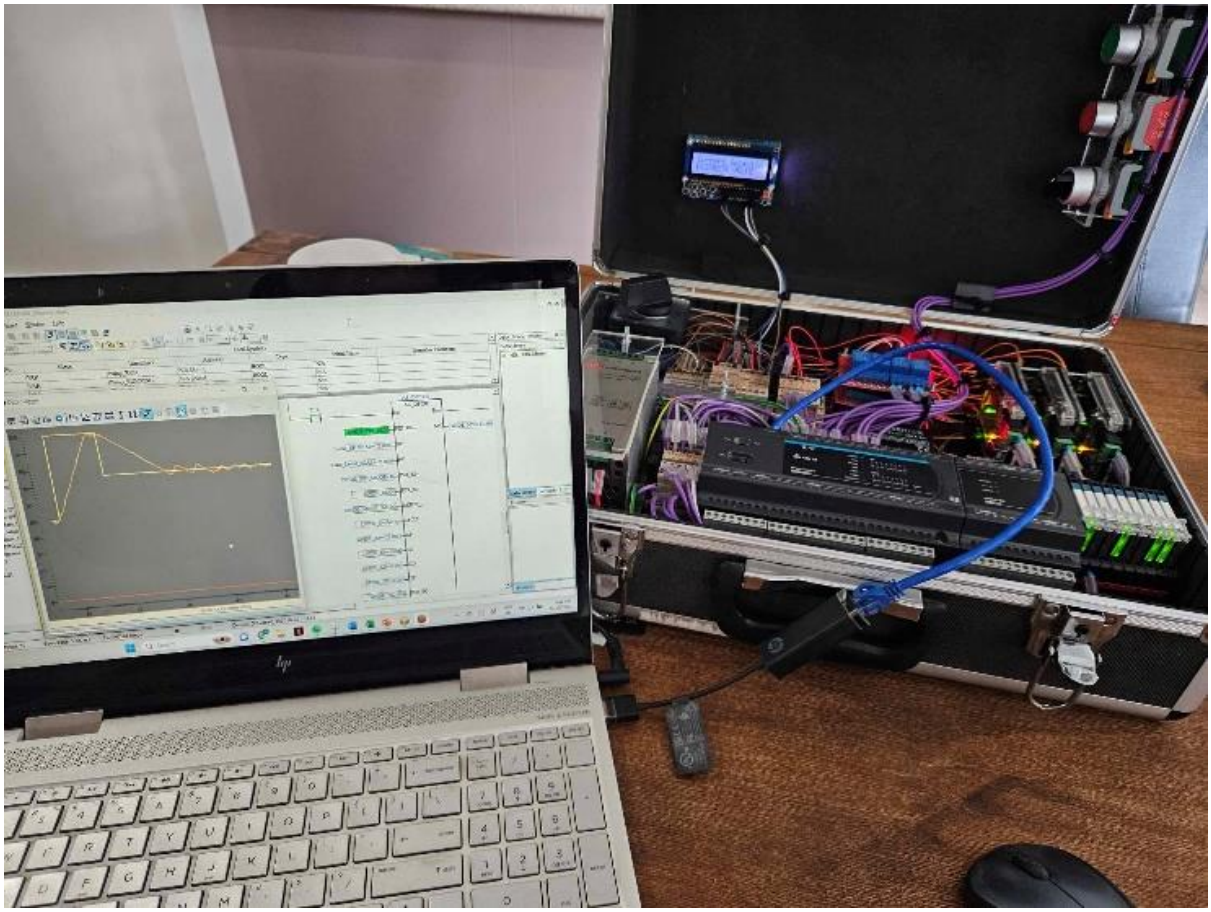


Figure 34 – Tuning PID Loop within Training Tool

The instructions for how to complete each of these programs were included in the PLC Training Tool Instructions for users of the tool to be emulate, as well as these written programs were included on the PLC when users initially perform an upload as sample programs for them to check their work against, or use for visual assistance.

User Feedback

On top of the Training Tool Instructions, all users of the training tool were also given a feedback form for them to fill out before and after their use of the tool. This User Feedback Form is shown in Appendix G.

This feedback form has two sections, one for completion prior to use of the tool and the other for completion once the use of the tool has concluded.

The section for completion prior to use of the tool asks questions about the users' current level of skill and confidence with PLC programming in general, as well as specifically about the different languages they were able to utilise with the PLC in this tool. This section also asks the users to write down what they were hoping to achieve from the use of this tool, which will help in evaluating the tool once they were completed.

The section for completion once the use of the tool has concluded asks the user to rate their experience and then a series of questions about the tool itself. They were asked if they found the instructions clear and easy to follow, if they feel their knowledge and familiarity with writing PLC programs was any different after using the tool, if there was anything they found particularly difficult whilst using the tool, how they found the operation of the microcontrollers when considering its simulation of a real world devices behaviour. After each of these questions the user was given space to provide more detailed feedback if they would like. Finally, a space was left at the bottom for the user to provide any general comments if they would like.

The tool was given to 4 users for them to use and then provide their feedback. Due to time constraints, each user was only given the prototype to use for one week. These responses are shown in the table below, and discussed further after.

Table 9 – User Feedback Form Responses

	User 1	User 2	User 3	User 4
Prior to Use				
Current PLC Programming Ability?	Advanced	Beginner	Intermediate	Intermediate
How often do you perform PLC Programming?	Regularly	Never	Occasionally	Regularly
Confidence Level – Ladder Diagram?	Very confident	Not confident	Very confident	Very confident
Confidence Level – Structured Text?	Very confident	Not confident	Not confident	Not confident
What would you like to gain from using this tool?	Would like to gain experience in different PLC and software than they've worked with in the past	To gain a basic knowledge in general programming as it was something they'd like to do more of in the future	To learn about programming in different programming languages and using PLC without affecting real process	The opportunity to program in structured text, while also being able to utilise 'real world' signals for their program
After Use is Completed				
Overall experience with tool?	Positive	Positive	Positive	Positive
Were instructions clear and easy to follow?	Yes	Yes	Yes	Yes
Any change in knowledge and familiarity with PLC Programming now?	Very little change	Definitely, with no experience before this use there was a large change	Yes more comfort with structured text programming, and programming in general	Yes more knowledge with structured text
Anything you found particularly difficult?	None	The structured text programming	PID loop control scenario was difficult to	Some difficulty with the

		was quite difficult to see and understand	implement correctly, needed the sample program	structured text scenarios
Opinion on operation of microcontrollers? Did they behave as expected?	Behaved as expected for the most part. Some responses were a little quicker than the 'real-world' equivalent	Behaved roughly as expected	Behaved roughly as expected	Behaved roughly as expected
Any further comments?	Really enjoyed being able to tune PID control loop and to trend the results	Found the tool very instructive and felt as though they could learn quite a bit more with more time with the tool	Nil	Loved being able to simulate 'real-world' devices in a safe environment, helped them with PLC fault finding as well

When asked to rate their current ability with PLC programming, user A described themselves as advanced, user C & D as intermediate and user B as a beginner. User A & D reported that they perform PLC programming regularly, user C reported programming occasionally, with user B reporting that they have never performed PLC programming.

Users A, C & D all reported as being very confident with programming using Ladder diagram, and user B obviously not confident having never programmed at all before. Only user A reported as being very confident with using structured text as a programming language, with all three other users reporting that they were not confident using structured text.

When describing what they would like to achieve from the use of this tool, User A mentioned that they would like to gain experience in a different PLC & software than they have used in the past, since they have never used a Delta PLC before. User B wrote that they would like to gain a basic knowledge in general programming as it was something they'd like to do more of in the future. User C & D both

indicated that they were looking forward to the opportunity to program in structured text, while also being able to utilise ‘real world’ signals for their program.

Once the use of the tool was completed for each user, they then provided the following answers. All four users reported that their overall experience with the training tool was positive. Similarly, all four users reported that the instructions given were clear and easy to follow.

When asked if their knowledge and familiarity with writing PLC programs had improved after using the tool user A reported that they noticed very little change for them. They noted that they did learn more about programming Delta PLCs, their overall PLC programming knowledge was largely unaffected. Users B, C & D all reported as the use of the tool having a positive effect on their knowledge and familiarity with programming. Users C & D focussed their gains in knowledge on writing in structured text specifically. User B wrote more about their gains in overall knowledge regarding PLCs, as well as gains in programming using ladder diagram logic.

The users were then asked to comment on any difficulties they encountered with using the tool. User A wrote that they did not encounter any difficulties. User B wrote that they struggled quite a bit with programming using structured text. They went on to mention that the training tool instructions did not really clearly describe how to program using this language, and that they felt like if they had more time they would probably have felt better about it. User C mentioned that they found the PID control loop scenario a little difficult to implement correctly, and that they needed to copy the sample program to make theirs work properly. User D wrote about how they also found the use of structured text programming difficult at times, especially when it came to implementing functions within the structured text program.

When describing how they found the operation of the microcontrollers, specifically their ability to appropriately simulate industrial devices, User A reported that they did behave as they had expected them to for the most part. They mentioned that the responses of some of the devices were a little quicker than a ‘real world’ version of that device, but acknowledged that was likely due to being able to see changes in the signals in a more suitable timeframe for training purposes. User B, C & D all reported that they behaved roughly as they’d expected them to based on their knowledge of those devices.

The users were then finally asked for any final overall comments regarding their experience with the tool. User A commented that they really liked the fact that they were able to practice their PID control loop tuning with the ability to trend results quite similar to what they would see in the real world. User B commented that they found the use of the tool to be very instructive and that they felt as though they’d learned a lot from it but that they would like to be able to use it for longer next time. User C did not leave any further comments. User D mentioned that they loved being able to simulate ‘real-world’ devices when writing these programs, and how they felt that really helped them visualise how their code

should work, and what was wrong with it when it didn’t work properly. They also liked how they could practice programming with no chance of damage to any live plant.

It should be noted that all of these users were previously known to the author of this report, and all except one had known skills with regards to programming PLCs. These results may have been quite different had the users all been people who had not ever utilised a PLC or this kind of programming before.

6 Conclusions

Summary & Conclusion

Through research of the existing and available options on the market for PLC training tools, it became clear that there was a potential need for a tool with the features that this tool could have in this area, with most current options either far too expensive for users to afford, or too simplistic to be as useful as they likely need to be. The kind of tool proposed in this project did not yet exist, and could open up the options for training to all kinds of people who could benefit from being able to expose themselves to these methods of programming in a safe, easy to use, affordable environment.

The original aim of this project was to design a low-cost PLC training tool which utilises microcontrollers to simulate physical devices often found in an industrial environment. The requirements for this design were as below –

- Tool must not cost much more than $1000 AUD to purchase and construct.
- Tool must not be much larger than a standard carry case (approx. 500mm x 300mm x 200mm) – portability was to be considered at every step.
- PLC must be programmable in at least 3 of the programming languages listed in IEC 61131-3 – one of which must be Structured Text.
- PLC CPU ability to be programmed with free open source programming tool – Codesys, OpenPLC, etc. to be considered to assist with the previous point
- Microcontrollers must be able to appropriately simulate at least 3 standard industrial field devices, VSDs, DOL motor control circuits, Valves, Various transmitters, etc.
- Microcontrollers must have the ability to utilise an appropriate amount of I/O to suitable simulate the devices previously mentioned
- User must be able to easily set the particular device in the microcontroller – programming the microcontroller was not to be part of the tools training.
- Construction of tool must be done with safety of the user as paramount. All voltage above 24v must be appropriately covered from contact.

The cost of the tool to construct was intended to remain below $1000. However, the final cost came in at $1166.29 which brought the project to $166.29 over budget. This was largely due to the price of the PLC being significantly more than expected. After going through the process of finding a suitable PLC, it was found that this cost being higher than expected was because the extra costs associated with analogue signals had not been considered.

The prototype was constructed in a carry case measuring 400mm x 300mm x 100mm. This allows the tool to be very portable. However, some of the limitations found when mounting the gear in the case would likely have been less of an issue had the case used been slightly larger. Some of the components

in the case were mounted where they would fit, not necessarily where they make logical sense to be placed. An example of this was the microcontroller selector switch and the LCD screen. They were mounted on the other side of the case to where the microcontrollers were as that was the only location available. However, had the case been slightly longer it could have retained its portable nature but given the space for these components to be placed in their more logical locations.

Once construction was completed, all accessible components with voltage levels above 24vDC were covered with custom Perspex covers. These were bolted into place using nylock nuts which will require the use of tools to remove. This ensured that a user must be intentionally removing the cover, and not simply by accident through use.

The PLC chosen for this prototype was capable of being programmed in 3 of the 5 languages found in the international standard for PLCs, IEC 61131. These were ladder diagram, structured text & sequential function chart. Whilst this PLC can be programmed using open source programming software like Codesys, since Delta offer software for their equipment that allows for these different programming languages to be used and that this software is free to download and use, the decision was made to simply use their software ISPSoft.

The microcontrollers in this tool were able to suitably simulate one of up to seven devices or arrangements commonly found in an industrial environment. These devices include –

- DOL Motor
- VSD Motor
- Discrete Valve
- Positional Valve
- Vessel – DOL Motor
- Vessel – VSD Motor
- Heater

These devices can easily be selected through the use of an LCD screen with associated buttons, as well as a selector switch to choose with microcontroller to select the device on.

Training scenarios were developed for users to follow to assist with their use of the tool. These scenarios mainly involved the use of a single device, with the intention that the user would be able to combine them into any number of custom programs once they felt comfortable. However, there was once scenario devised which called for the user to implement a PID control loop to control the level of a vessel.

All of these scenarios were combined into part of the Training Tool Instructions, which were are provided to any user of the tool. This formed an integral part of the PLC training tool, especially for less advanced/experienced PLC users, as it also instructed users on how to connect to the tool and how to use the particular PLC programming software chosen.

These training tool instructions were given to four users for them to use the tool and evaluate its effectiveness. To assist in this, they were also provided with a feedback form. These users provided generally positive feedback, and most of these users have reported that they did increase their knowledge and familiarity with writing a program for a PLC. The less experienced users found it difficult to translate this program into structured text, but felt that they would be able to gain those skills. All users were positive regarding the operation of the microcontrollers – in particular the ability to perform PID loop tuning using the analogue signals.

Further Work

The actual implementation of Fault Mode would bring with it another level of programming for users, to write programs for, and test the operation of, faults with the devices they were using in their code. As fault monitoring and rectification is a very large part of any PLC program, this would be quite valuable for any user of this tool.

Due to the size constraints that came with the size of the carry case used, this prototype was constructed in a way that does not allow users to perform any physical wiring of their own to more customise the tool to their needs. This can be especially important as learning about physically wiring a PLC can be nearly as important as learning how to program them. With PNP, NPN inputs, transistor outputs and relay outputs, current analogue and voltage analogue inputs and outputs, and all the various ways to wire these up it could be quite useful for users to be able to do this themselves. In a second version of this tool, with a larger carry case used, this could be quite a useful option to add for any electricians undertaking the training to give them this experience and exposure.

The microcontrollers are currently limited to seven devices, but there is really no major limit to the number of devices they can simulate other than the amount of I/O required to simulate those devices and interface them with the PLC used. Further programming of the microcontrollers to allow for more device options would open up the ability for users to construct more device combinations, perhaps in more relevant scenarios to the users own work or training interests.

Giving the user the ability to alter the time response of a system under PID loop control would also allow for more customisation of the training experience to suit systems that a user may already come into contact with and desire further experience with. A potential complication of this is that it would currently require the user to be given access and control over the microcontroller code, which if used

incorrectly could render the tool useless. Perhaps a new method of communication between the PLC and microcontrollers could be investigated to update these parameters as a variable on the PLC.

7 References

Adafruit 2023, *Adafruit RGB LCD Shield Library*, Github, viewed 14th June 2023, <https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library/tree/master>.

Akparibo, AR, Appiah, A & Fosu-Antwi, O 2016, 'Development of a Programmable Logic Controller Training Platform for the Industrial Control of Processes', *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 15, no. 1, pp. 186-96.

Alphonsus, ER & Abdullah, MO 2016, 'A review on the applications of programmable logic controllers (PLCs)', *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185-205.

Arduino 2023, *Arduino MKR WiFi 1010*, Arduino CC, viewed 06/05/2023, <https://store-usa.arduino.cc/products/arduino-mkr-wifi-1010>.

Arowolo, MO, Adekunle, AA & Opeyemi, MO 2020, 'Design and Implementation of a PLC Trainer Workstation', *Advances in Science, Technology and Engineering Systems Journal, Federal University Oye, Nigeria*.

Artiyasa, M, Destria, N & Desima, M 2020, 'Creating kit and plc application with industrial applications for practice learning of plc technology in electronics Nusaputra university Sukabumi', *Journal of Physics: Conference Series*, IOP Publishing, p. 012010.

Barrett, M 2008, 'The design of a portable programmable logic controller (PLC) training system for use outside of the automation laboratory', *International Symposium for Engineering Education*, pp. 1-5.

Beckhoff-Automation 2023, *Instruction List (IL)*, Beckhoff Information System, viewed 15/10/2023, <https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/925244555.html&id=>.

Bolton, W 2015, *Programmable Logic Controllers*, 6th edn, Newnes.

Codesys 2023, *CODESYS INSIDE*, Codesys Group, viewed 03/03/2023, <https://www.codesys.com/the-system/codesys-inside.html>.

Delta Electronics 2020, *ISPSoft User Manual*, 7th edn, Delta Electronics.

Delta Electronics 2023, *DVP-ES3 Series*, Delta Electronics, viewed 03/03/2023, <https://www.deltaww.com/en-US/products/PLC-Programmable-Logic-Controllers/5551>.

DO Supply 2022, *What Does I/O do on a VFD?*, DO Supply, viewed 12/03/2023, <https://www.dosupply.com/tech/2022/09/07/what-does-i-o-do-on-a-vfd/>.

'The Evolution of PLCs', 2021, *PLC Technician Training*, 07/12/2021, viewed 28/09/2022, <https://www.plctechnician.com/news-blog/evolution-plcs>.

Favela, C & Kaye, N 2021, *The 10 Best Arduino Alternatives of 2021*, All3DP, viewed 05/10/2022, <https://all3dp.com/2/best-arduino-alternatives/>.

Faylor, J 2022, *PLC Programming Languages: Go Beyond Ladder Logic*, inductive automation, viewed 15/10/2023, <https://inductiveautomation.com/blog/plc-programming-languages-go-beyond-ladder-logic>.

Fuada, S, Huda, M & Aprilowena, R 2012, 'A Study Basic Programmable Logic Controller (PLC) for Effective Learning', *Int. Journal of Computers & Technology (IJCT)*, vol. 3, no. 3, pp. 470-3.

Guo, L & Pecen, R 2009, 'Design projects in a programmable logic controller (PLC) course in electrical engineering technology', *The technology interface journal*, vol. 10, no. 1.

IEC 2013, *Programmable Controllers*, Programming Languages, IEC.

Kim, YS & Kim, H-M 2013, 'Design of a New Virtual Interaction Based PLC Training Using Virtual Sensors and Actuators: System and Its Application', *International Journal of Distributed Sensor Networks*, vol. 9, no. 7, p. 505920.

Kopczyk, J 2018, *Engineering Essentials: A Closer Look at the I's and O's of VFDs*, Machine Design, viewed 12/03/2023, <https://www.machinedesign.com/mechanical-motion-systems/article/21836794/engineering-essentials-a-closer-look-at-the-is-and-os-of-vfds>.

Kuphaldt, TR 2009, *Lessons in industrial instrumentation*, vol. 4, Creative Commons Attribution/PAControl. com.

Mathworks 2023, *How PID Autotuning Works*, Mathworks, viewed 15/10/2023, <https://www.mathworks.com/help/slcontrol/ug/how-pid-autotuning-works.html>.

NI 2023, *The PID Controller & Theory Explained*, Emerson, viewed 15/10/2023, <https://www.ni.com/en/shop/labview/pid-theory-explained.html>.

Peter 2018, *Function Block Diagram (FBD) PLC Programming Tutorial for Beginners*, PLC Academy, viewed 15/10/2023, <https://www.plcacademy.com/function-block-diagram-programming/>.

PLC Cable 2023, *PLC Trainers (Kits)*, PLC Cable, viewed 04/03/2023, <https://www.plccable.com/plc-trainers-kits/>.

Pratama, H, Azman, M, Zakaria, N & Khairudin, M 2022, 'The effectiveness of the kit portable PLC on electrical motors course among vocational school students in Aceh, Indonesia', *Kompleksnoe Ispolzovanie Mineralnogo Syra*, vol. 320, no. 1, pp. 75-87.

Technology Robotix Society 2019, *PID Control*, Medium, viewed 15/10/2023, <https://medium.com/autonomous-robotics/pid-control-85596db59f35>.

Yakimov, P, Iovev, A, Tuliev, N & Balkanska, E 2019, 'Development of hardware and software methods and tools for a successful PLC training', *2019 X National Conference with International Participation (ELECTRONICA)*, IEEE, pp. 1-4.

Appendix A – Project Specification

ENG4111/4112 Research Project

Project Specification

For: Shane Fulcher

Title: Design & Develop a Low Cost Programmable Logic Controller (PLC) Training Tool, utilising open-source micro-controllers.

Major: Instrumentation Control & Automation

Supervisors: Mrs. Catherine Hills

Enrollment: ENG4111 – EXT S1, 2023

ENG4112 – EXT S2, 2023

Project Aim: to design a relatively low-cost PLC training tool, which utilises microcontrollers to simulate the physical digital inputs and outputs as well as analogue inputs and outputs that can be found on various hardware components in an industrial setting. Also, to allow students to learn PLC programming across multiple programming languages.

Programme: Version 1, 28th February 2023

1. Research the field including PLC training and any existing solutions. Investigate the requirements of the system and virtual devices.

2. Define the technical requirements of the system

3. Research components available and make decisions on most suitable to meet project requirements & construct electrical schematic of components chosen for use in tool

4. Check availability lead times on hardware components chosen, order PLC and microcontrollers & order remaining hardware components

5. Define operational requirements for virtual devices including I/O allocation, connections and device behaviour. Define program configuration options and operating modes.

6. Program microcontrollers using appropriate programming software, connect to PLC CPU and ensure correct operation, write basic PLC program to test all inputs and outputs of tool are operational

7. Construct PLC tool - Mount components into case in a logical and secure manner & wire components together according to electrical schematic.

8. Use program previously written to test wiring and labelling of all I/O to CPU, test operation of microcontroller programs operates as per design, write programs for the scenarios constructed earlier and test their operation

As time and resources permit:

9. Develop a range of training scenarios that can be used for testing.

10. Extensive bench testing of the tool in simple and complex applications, to program these scenarios, and others they would like to try, and record the results

Appendix B – Risk Assessment

NUMBER	RISK DESCRIPTION	TREND	CURRENT	RESIDUAL
2442	Construct PLC Training Tool		Medium	Not Assessed

RISK OWNER	RISK IDENTIFIED ON	LAST REVIEWED ON	NEXT SCHEDULED REVIEW
Shane Fulcher	18/05/2023	18/05/2023	18/11/2023

THIS IS A RESTRICTED RISK ASSESSMENT
Not relevant to individuals not involved.

RISK FACTOR(S)	EXISTING CONTROL(S)	PROPOSED CONTROL(S)	OWNER	DUE DATE
Cuts & Abrasions from use of hand tools & power tools	**Control:** When using these tools ensure appropriate PPE is utilised. Gloves, long sleeve shirt and pants, steel cap boots, eye protection.			

Electric Shock from power supplying tool	**Control:** Isolate electricity whenever work is being completed on tool. Test for dead, and lock out. **Control:** Mount suitable covers over accessible conductive parts.	
Burns from contact with hot components after work like heat shrinking & cutting/grinding	**Control:** Ensure components are allowed to cool before continuing to work with them. **Control:** Wear suitable gloves when performing these hot works	

Appendix C – Arduino Code

```
#include <Wire.h>
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

// this constant won't change:
const int digoutput_one = 2;  // the pin that is allocated to the first output
relay
const int digoutput_two = 3; // the pin that is allocated to the second output
relay
const int diginput_one = 8; // the pin that is allocated to the first input
const int diginput_two = 9; // the pin that is allocated to the second input
const int diginput_three = 10; // the pin that is allocated to the third input
const int analoutput_one = 11; // the pin that is allocated to the first
analogue (PWM) output
const int analinput_one = A0; // the pin that is allocated to the first
analogue input
const long dol_interval = 1500;  // 1 second interval (milliseconds)
const long valve_interval = 3000;  // 3 second interval (milliseconds)

// These #defines set the backlight color
#define OFF 0x0
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7

// Variables will change:
bool Init;
int sensorInput = 0;  // input value from the analogue input
int sensorValue = 0;  // scaled value of the input
int outputValue = 0;  // value output to the PWM (analog out)
bool faultMode;
bool TOL_Fault;
bool position_Fault;
int none;
unsigned long initMillis;
unsigned long currentMillis;
unsigned long dol_interval_check;
long random_number; // this will be utilised for fault times

int currentSelectionInt;
```

```
int newSelectionInt;

char *deviceOptions[] = {"NONE", "DOL MOTOR", "VSD MOTOR", "DISCRETE VALVE",
"ANALOGUE VALVE", "VESSEL - DOL", "VESSEL - VSD", "HEATER", "FAULT MODE ON?",
"FAULT MODE OFF?"};

void setup() {
  // initialize the following pins as inputs. They are initialised as with the
Pull Up resistor in use, so 'Low' = True and 'High' = False.
  pinMode(diginput_one, INPUT_PULLUP);
  pinMode(diginput_two, INPUT_PULLUP);
  pinMode(diginput_three, INPUT_PULLUP);
  // initialize the following pins as outputs:
  pinMode(digoutput_one, OUTPUT);
  pinMode(digoutput_two, OUTPUT);

  // initialise serial communications
  Serial.begin(9600);

  // initiailise lcd screen
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Current Selection");
  lcd.setCursor(0,1);
  lcd.print(deviceOptions[currentSelectionInt]);
  lcd.setBacklight(WHITE);

  // initialise selection values
  currentSelectionInt = 0;
  newSelectionInt = 0;

  // initialise fault mode numbers
  randomSeed(1);
  random_number = random(30000, 180000); // Generate random number for fault
generation
}

void loop() {
  // main code, runs repeatedly
while(1){
  uint8_t buttons = lcd.readButtons();
  delay(100); //This is to allow the microcontroller selector switch to work.
// The following code is to display the current selection on the screen, and
allow the user to select a different device
  if (buttons) {
    none = 0;
    lcd.clear();
```

```
    lcd.setCursor(0,0);

    if (buttons & BUTTON_LEFT && newSelectionInt >= 1) {
      --newSelectionInt;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("New Selection?");
      lcd.setCursor(0,1);
      lcd.print(deviceOptions[newSelectionInt]);
    }
    if (buttons & BUTTON_RIGHT && newSelectionInt <= 8) {
      ++newSelectionInt;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("New Selection?");
      lcd.setCursor(0,1);
      lcd.print(deviceOptions[newSelectionInt]);
    }
    if (buttons & BUTTON_SELECT) {
      currentSelectionInt = newSelectionInt;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Current Selection");
      lcd.setCursor(0,1);
      lcd.print(deviceOptions[currentSelectionInt]);
      Serial.println(deviceOptions[currentSelectionInt]);
    }
  }

// The remainder of this code is the operation of the arduino based on the
selection made above

  if (currentSelectionInt == 0){ // This is when no device is selected. Turn
off both digital outputs and zero analogue output
    digitalWrite(digoutput_one, LOW); //Set digital output low when run signal
is removed
    digitalWrite(digoutput_two, LOW); //Set digital output low when run signal
is removed
    analogWrite(analoutput_one, 0);

  }
  if (currentSelectionInt == 1){ //DOL Motor selected
    Serial.println(currentSelectionInt);  //print the current selection on the
serial monitor

    if (digitalRead(diginput_one) == LOW){
    delay(1000);
```

```
    digitalWrite(digoutput_one, HIGH); // write output to HIGH state to
simulate 'running' signal from contactor
    Serial.println("Contactor running");
    }

    if(digitalRead(diginput_one) == HIGH) {
    delay(1000);
    digitalWrite(digoutput_one, LOW); //Set digital output low when run signal
is removed
    digitalWrite(digoutput_two, LOW); //Set digital output low when run signal
is removed
    TOL_Fault = LOW;
  }
  }

  if (currentSelectionInt == 2){ //VSD Motor selected

  Serial.println(currentSelectionInt);

// read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range

  if(digitalRead(diginput_one) == LOW){
    digitalWrite(digoutput_one, HIGH);
  // If the run signal is on and the output speed is lower than the speed SP,
the output speed of the motor will ramp up to speed SP
  if(outputValue <= sensorValue){

    outputValue = outputValue + 5;

  }

  // If the run signal is on and the output speed is higher than the speed SP,
the output speed of the motor will ramp down to speed SP
  if(outputValue >= sensorValue){

    outputValue = outputValue - 5;

  }
  }

  if(digitalRead(diginput_one) == HIGH){
    digitalWrite(digoutput_one, LOW);
 // If the run signal is off and the output speed is higher than 0, the output
value will ramp down to 0
  if(outputValue > 0){
```

```
    outputValue = outputValue - 5;

  }
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }

 if (currentSelectionInt == 3){ //Discrete Valve selected
    Serial.println(currentSelectionInt);

    if (digitalRead(diginput_one) == LOW){

    if (init == LOW) {
    initMillis = millis(); // take the initial time of the beginning of this
loop
    init == HIGH; // Set Init high so this does not run again
    Serial.println("Initial run");
    }

    currentMillis = millis();

    if (position_Fault != HIGH) {
    delay(3000);
    digitalWrite(digoutput_one, HIGH); // write output to HIGH state to
simulate 'open' signal from valve position
    Serial.println("Valve Open");
    }

    if (position_Fault == HIGH && digitalRead(diginput_two) == LOW) {
    currentMillis = 0; //Reset fault timer
    Init = LOW;
    position_Fault = LOW; //Turn off valve position fault
    digitalWrite(digoutput_two, LOW); //Set valve position fault digital
output LOW
    }
    }

    if(digitalRead(diginput_one) == HIGH) {
    delay(3000);
    digitalWrite(digoutput_one, LOW); //Set digital output low when run signal
is removed
```

```
    digitalWrite(digoutput_two, LOW); //Set digital output low when run signal
is removed
    position_Fault = LOW;
    Serial.println("Reset");
  }
 }
  if (currentSelectionInt == 4){ //Analogue Valve selected

// read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range

  if(outputValue <= sensorValue && outputValue <= 255){

    outputValue = outputValue + 10;
    delay(1000);
  }

  if(outputValue >= sensorValue && outputValue >= 0){

    outputValue = outputValue - 10;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }

if (currentSelectionInt == 5){ //Vessel - DOL selected

  // If the pump is on, and the outlet valve is closed the level will rise
quickly
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == HIGH){
    outputValue = outputValue + 2;
    delay(1000);
  }

// If the pump is on, and the outlet valve is open the level will rise slowly
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == LOW){
    outputValue = outputValue + 1;
    delay(1000);
  }
```

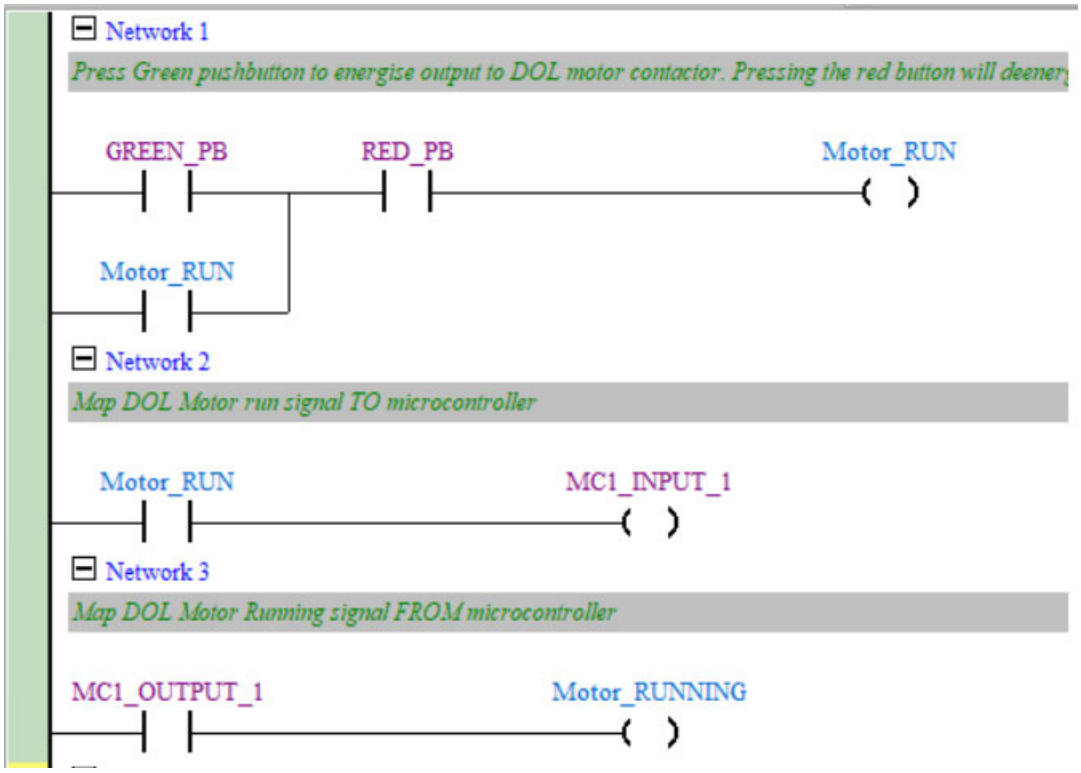
```
// If the pump is off, and the outlet valve is open the level will drop slowly
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == LOW){

    outputValue = outputValue - 1;
    delay(1000);
  }

// If the pump is off, and the outlet valve is closed the level will remain
the same
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == HIGH){

    outputValue = outputValue;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }

if (currentSelectionInt == 6){ //Vessel - VSD selected

 // read the analog in value:
  sensorInput = analogRead(analinput_one);
  sensorValue = map(sensorInput, 0, 1023, 0, 255); //scale analogue input
value to suit output range

  // If the pump is on, and the outlet valve is closed the level will rise in
accordance with the speed of the pump
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == HIGH){
    sensorValue = sensorValue/100;
    if(outputValue <= 251)
    {
    outputValue = outputValue + (2 * sensorValue);
    }
    Serial.println(outputValue);
    Serial.println(sensorValue);
    delay(1000);
  }

// If the pump is on, and the outlet valve is open the level will raise in
accordance with the speed of the pump, and lower with the outflow of the valve
  if(digitalRead(diginput_one) == LOW & digitalRead(diginput_two) == LOW){
    sensorValue = sensorValue/100;
    if(outputValue <= 252)
```

```
    {

    outputValue = outputValue + (2 * sensorValue);

    }
    if(outputValue > 0)
    {
    outputValue = outputValue - 1;
    }
    Serial.println(outputValue);
    delay(1000);
  }

// If the pump is off, and the outlet valve is open the level will lower
slowly with the outflow of the valve
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == LOW){
    if(outputValue > 0)
    {
      outputValue = outputValue - 1;
    }
    Serial.print(outputValue);
    delay(1000);
  }

// If the pump is off, and the outlet valve is closed the level will remain
the same
  if(digitalRead(diginput_one) == HIGH & digitalRead(diginput_two) == HIGH){

    outputValue = outputValue;
    delay(1000);
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }

if (currentSelectionInt == 7){ //Heater selected

  if(digitalRead(diginput_one) == LOW){
  // If the run signal is on and the temperature is lower than the SP, the
heater will raise the temperature to meet the SP
    delay(1000);
    outputValue = outputValue + 1;
  }
```

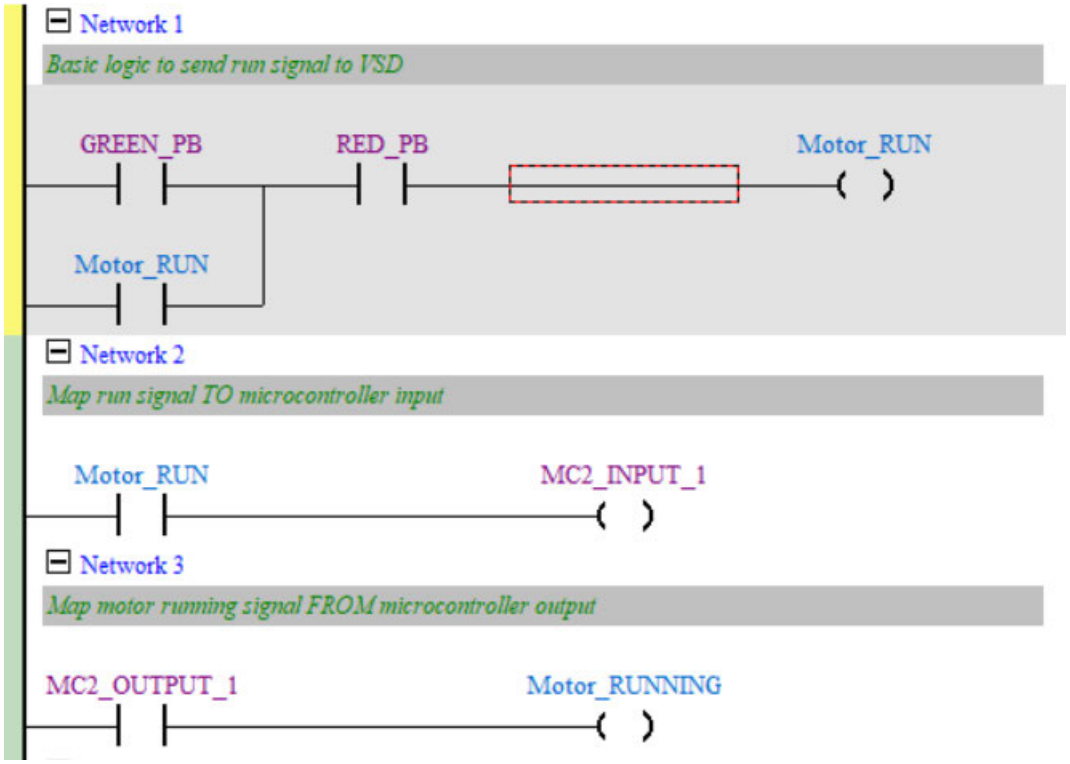
```
  if(digitalRead(diginput_one) == HIGH){
 // If the run signal is off, the temperature will lower to ambient
  if(outputValue > 51){
    delay(1000);
    outputValue = outputValue - 1;
  }
  if(outputValue < 51){
    outputValue = 51;
  }
  }

  // change the analog out value:
  analogWrite(analoutput_one, outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
  }

if (currentSelectionInt == 8){ // Turn on fault mode - when faults will occur
after random period of time
    faultMode = HIGH;
    Serial.print("Fault mode is on");
    currentSelectionInt = 0;
    newSelectionInt = 0;
    delay(100);
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Current Selection");
      lcd.setCursor(0,1);
      lcd.print(deviceOptions[currentSelectionInt]);
  }

if (currentSelectionInt == 9){ // Turn off fault mode
    faultMode = LOW;
    Serial.print("Fault mode is off");
    currentSelectionInt = 0;
    newSelectionInt = 0;
    delay(100);
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Current Selection");
      lcd.setCursor(0,1);
      lcd.print(deviceOptions[currentSelectionInt]);
  }
}
}
```

Appendix D – Sample PLC Ladder Programs

DOL Motor

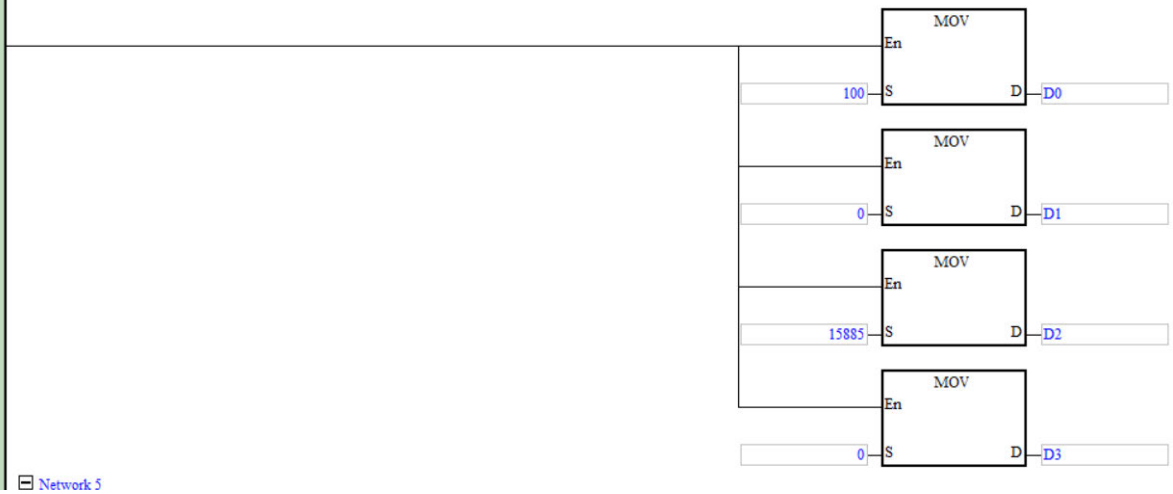


VSD Motor



Network 4

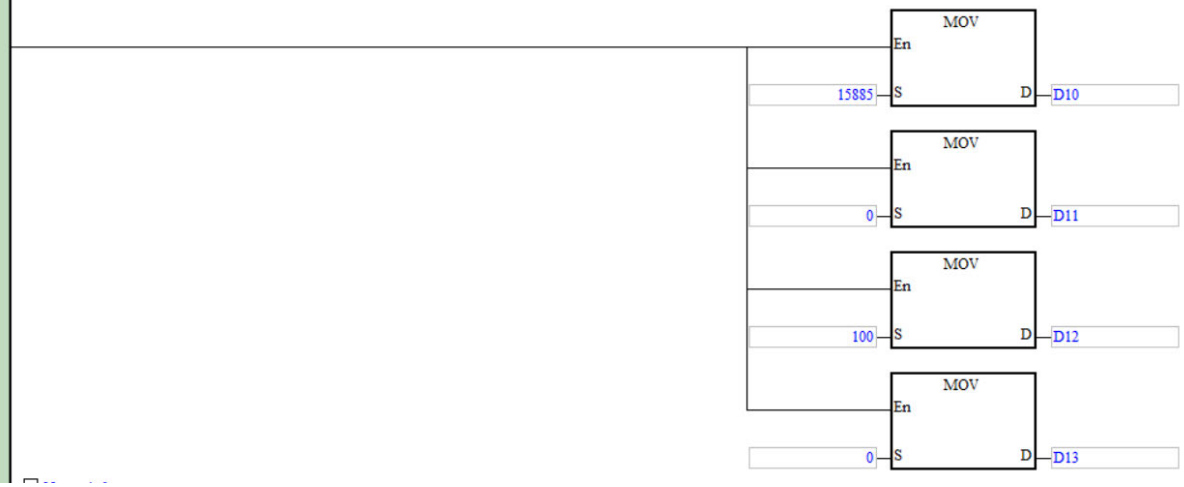
Set up scaling of analogue signal TO microcontroller



Network 5

Network 5

Set up scaling of analogue signal FROM microcontroller



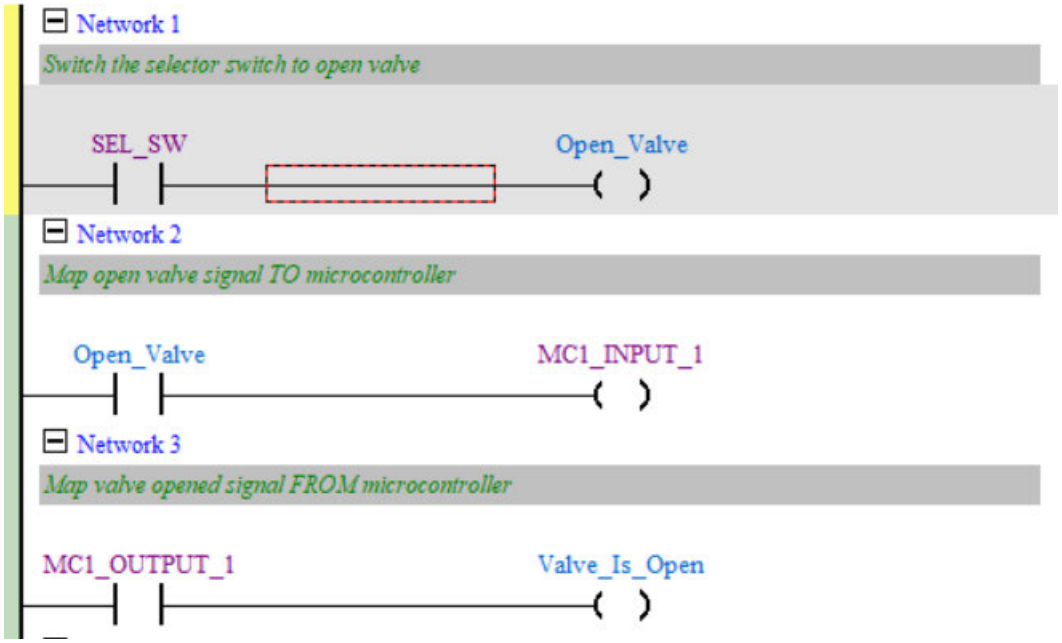
Network 6

Scale Speed setpoint (in percent) from 0-100 to 0-15885 to send to microcontroller

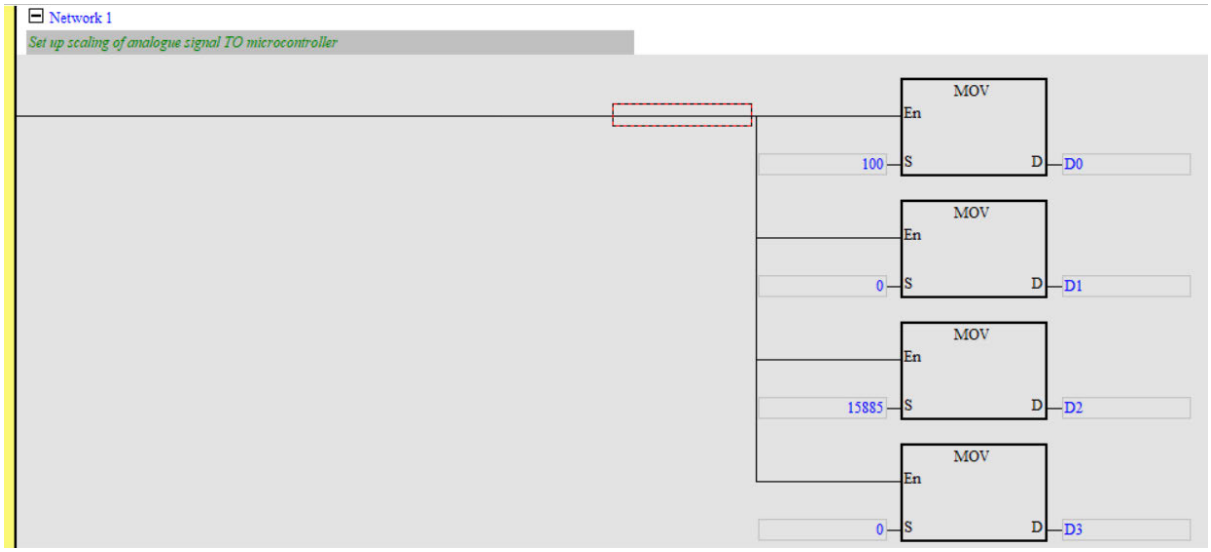
Network 7

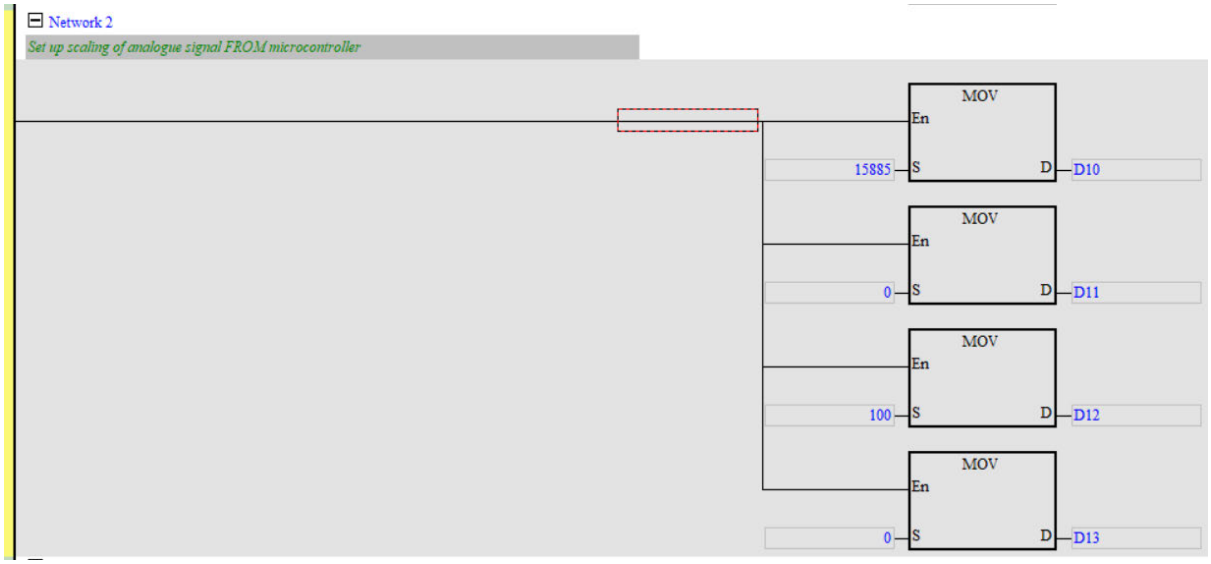
Scale microcontroller analogue output from 0-15885 to 0-100 to show current speed as a percentage

Discrete Valve



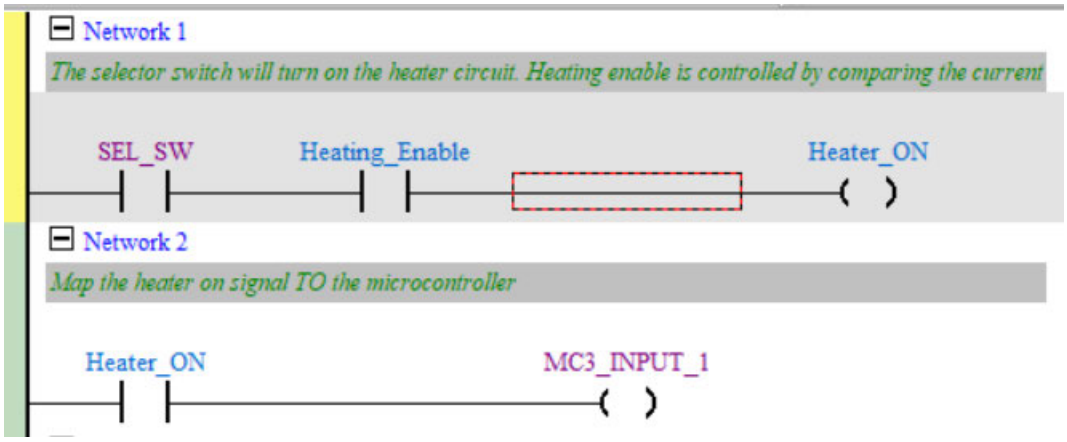
Analogue Valve




Network 2
Set up scaling of analogue signal FROM microcontroller
MOV
En
15885
S
D
D10
MOV
En
0
S
D
D11
MOV
En
100
S
D
D12
MOV
En
0
S
D
D13

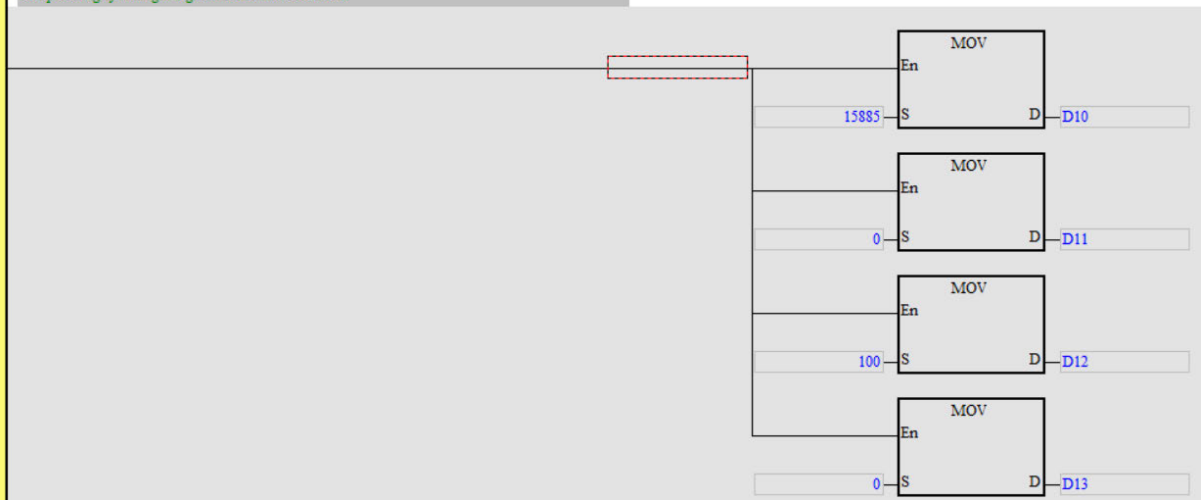

Network 3
Scale valve position setpoint (in percent) from 0-100 to 0-15885 to send to microcontroller
SCLP
En
Valve_Pos_SP
S1
D
MC2_ANALOG_IN
D0
S2
Network 4
Scale microcontroller analogue output from 0-15885 to 0-100 to show current valve position as a percer
SCLP
En
MC2_ANALOG_OUT
S1
D
Actual_Valve_Pos
D10
S2

Heater


Network 1
The selector switch will turn on the heater circuit. Heating enable is controlled by comparing the current
SEL_SW
Heating_Enable
Heater_ON
Network 2
Map the heater on signal TO the microcontroller
Heater_ON
MC3_INPUT_1

Network 3

Set up scaling of analogue signal FROM microcontroller

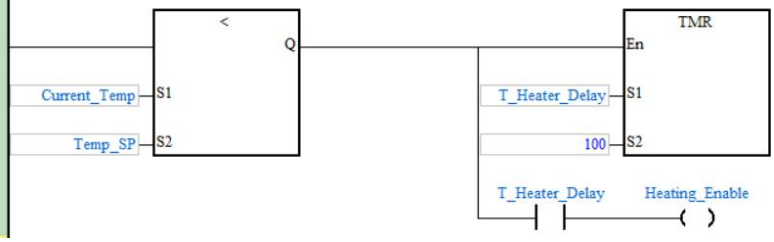


Network 4

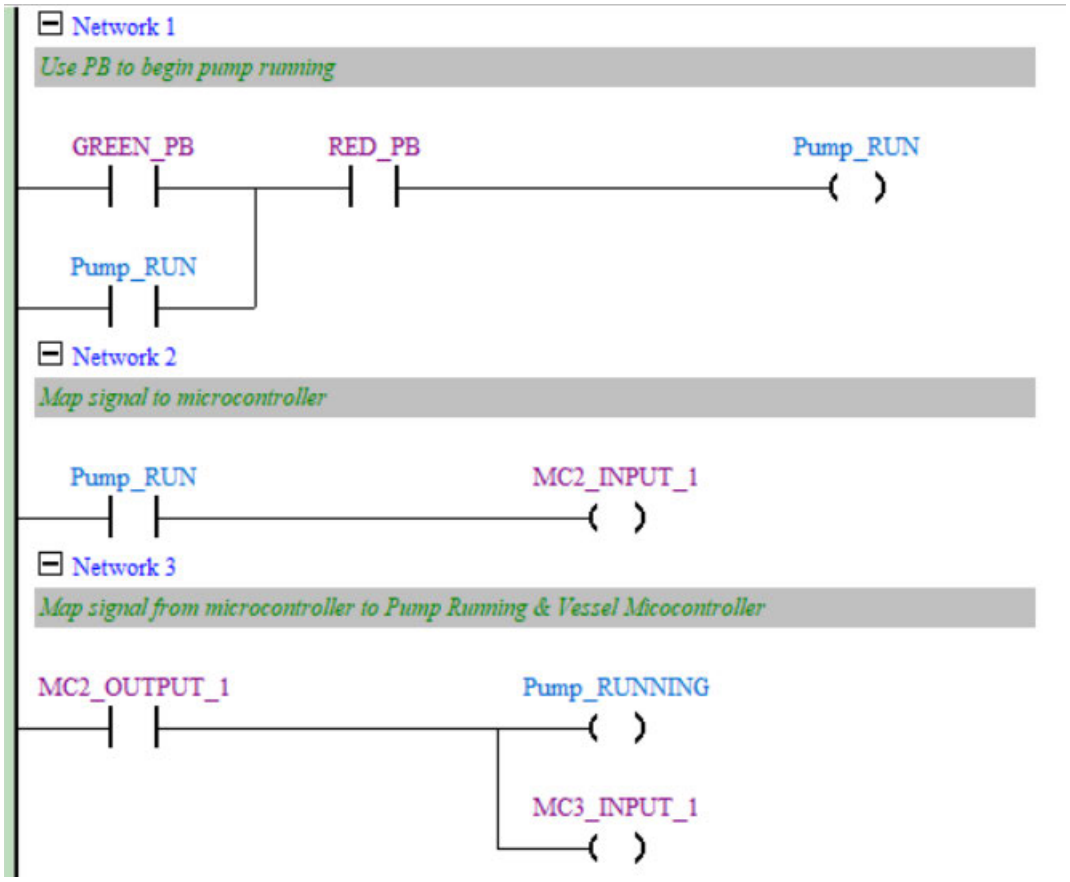
Scale microcontroller analogue output from 0-15885 to 0-100 to show current temperature

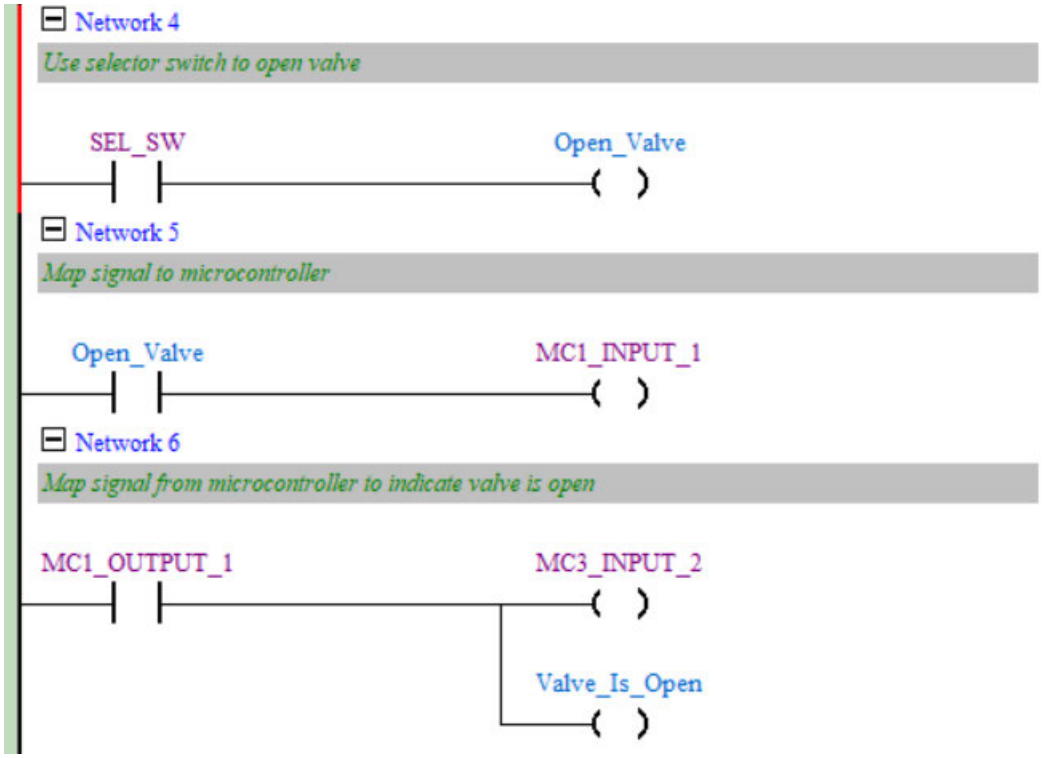
Network 5

Compare the current temperature with the temperature setpoint, and if it is too low for too long, enable h



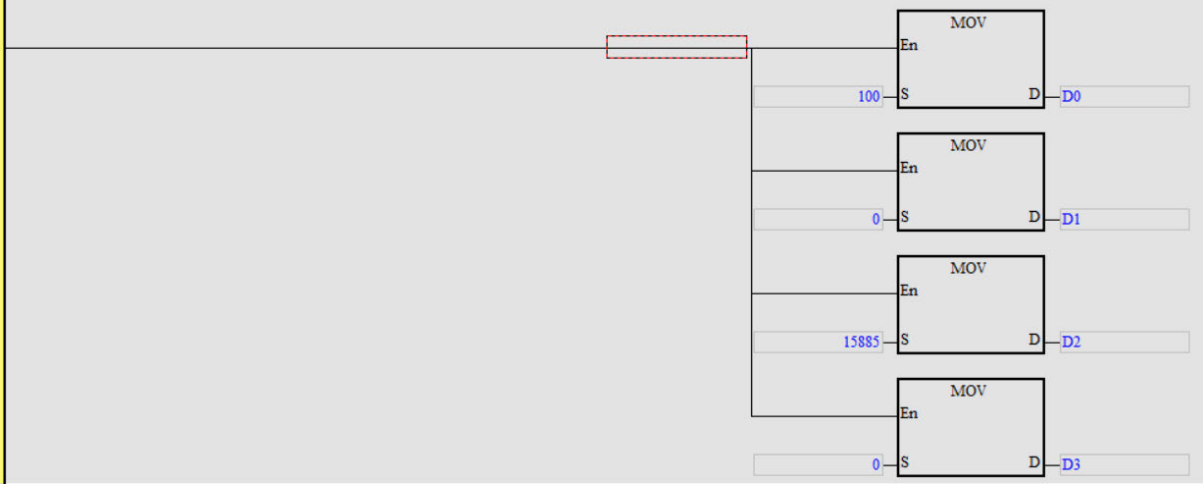
Vessel PID Loop





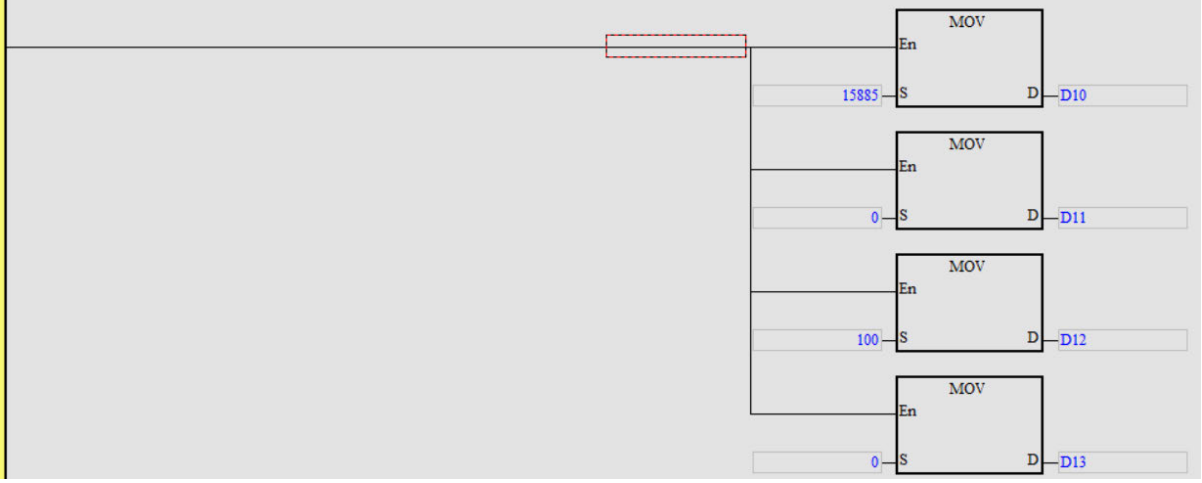
Network 7

Initialise scaling blocks for analogue output signal



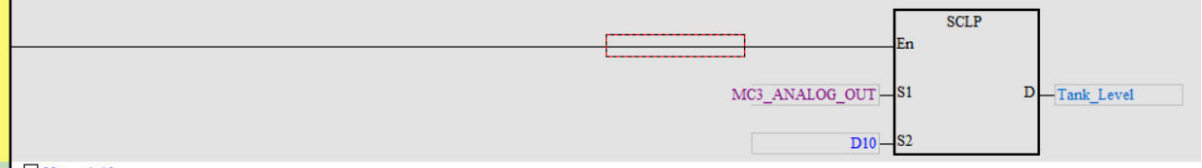
Network 8

Initialise scaling blocks for analogue input signal



Network 9

Scale analogue signal from vessel microcontroller and move into Tank level

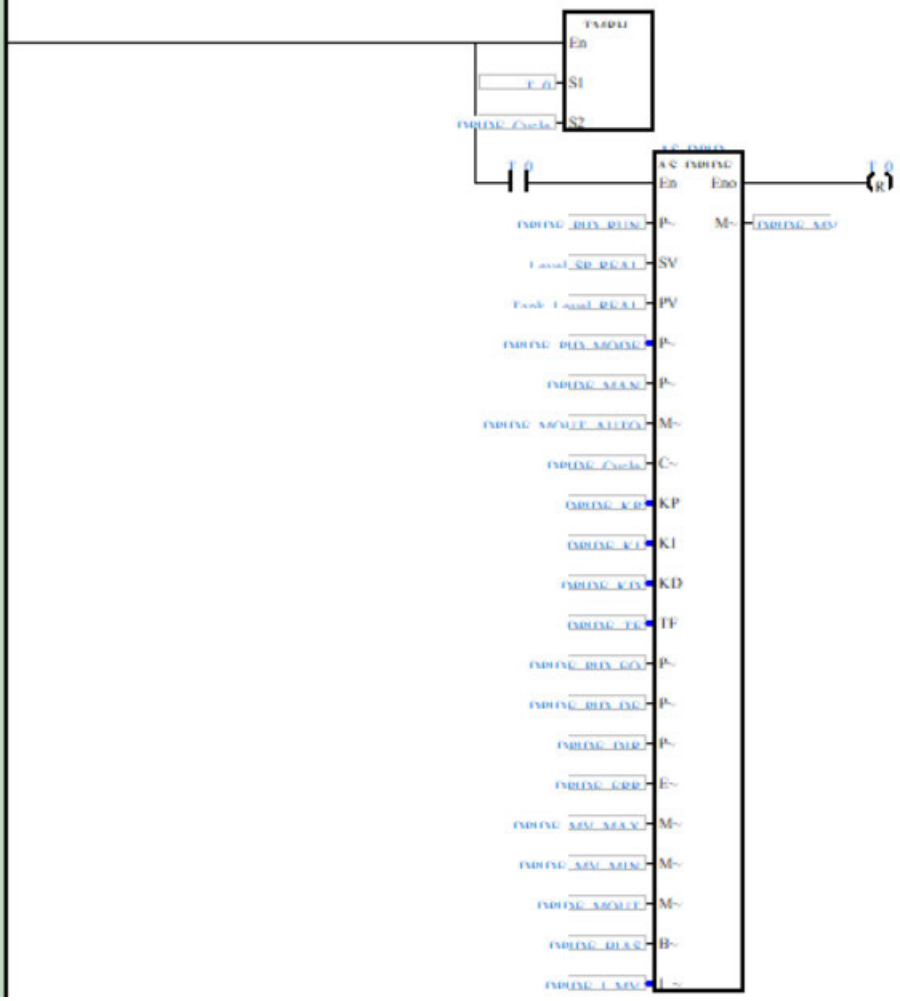


Network 10

Convert Int signals to REAL for PID block

Network 11

PID function block



Network 12

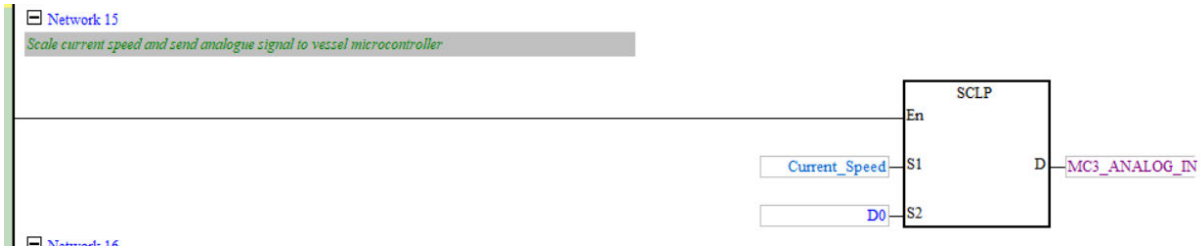
Move PID MV into speed setpoint of VSD pump

Network 13

Scale speed setpoint and send to VSD microcontroller

Network 14

Scale analogue signal from vsd microcontroller and move into Current Speed


Network 15
Scale current speed and send analogue signal to vessel microcontroller
SCLP
En
Current_Speed
S1
D
MC3_ANALOG_IN
D0
S2

Appendix E – Sample PLC Structured Text Programs

DOL Motor

```
IF ((GREEN_PB = TRUE) OR (Motor_RUN = TRUE & RED_PB = FALSE)) THEN //When the green pushbutton is pressed, turn on motor
Motor_RUN := TRUE;                                  // run signal. Continue motor run signal while red
END_IF;                                             //pushbutton is not pressed

IF (Motor_RUN = TRUE & RED_PB = TRUE) THEN                  //When motor is running and red button is pressed, turn
Motor_RUN := FALSE;                                 // off the motor run signal
END_IF;

MC1_INPUT_1 := Motor_RUN;                           //Map motor run signal TO microcontroller
Motor_RUNNING := MC1_OUTPUT_1;                          //Map motor running signal FROM microcontroller
```

VSD Motor

```
//Initalise values for scale functions below
D0 := 100;
```

```
D1 := 0;

D2 :=15885;

D3 :=0;


D10 := 15885;

D11 := 0;

D12 := 100;

D13 := 0;


WHILE(TRUE) DO

SCLP(Speed_SP, D0, MC2_ANALOG_IN);                    //Scale and map speed setpoint TO microcontroller

SCLP(MC2_ANALOG_OUT, D10, Current_Speed);             //Scale and map current speed FROM microcontroller


IF ((GREEN_PB = TRUE) OR (Motor_RUN = TRUE & RED_PB = FALSE)) THEN //When the green pushbutton is pressed, turn on motor

Motor_RUN := TRUE;                                    // run signal. Continue motor run signal while red

END_IF;                                               //pushbutton is not pressed
```

```
IF (Motor_RUN = TRUE & RED_PB = TRUE) THEN                    //When motor is running and red button is pressed, turn
Motor_RUN := FALSE;                                    // off the motor run signal
END_IF;


MC2_INPUT_1 := Motor_RUN;                              //Map motor run signal TO microcontroller
Motor_RUNNING := MC2_OUTPUT_1;                         //Map motor running signal FROM microcontroller


END_WHILE;
```

Discrete Valve

```
IF (SEL_SW = TRUE) THEN                                //When the selector switch is turned, turn on valve
Valve_OPEN := TRUE;                                    // open signal.
END_IF;


IF (Valve_OPEN = TRUE & SEL_SW = FALSE) THEN                  //When valve is opened and selector switch is turned off, turn
Valve_OPEN := FALSE;                                   // off the valve open signal
END_IF;
```

```
MC1_INPUT_1 := Valve_OPEN;                    //Map valve open signal TO microcontroller

Valve_OPENED := MC1_OUTPUT_1;                 //Map valve opened signal FROM microcontroller
```

Analogue Valve

```
//Initalise values for scale functions below

D0 := 100;

D1 := 0;

D2 :=15885;

D3 :=0;


D10 := 15885;

D11 := 0;

D12 := 100;

D13 := 0;


WHILE(TRUE) DO
```

```
SCLP(Valve_Pos_SP, D0, MC2_ANALOG_IN);                    //Scale and map valve position setpoint TO microcontroller

SCLP(MC2_ANALOG_OUT, D10, Current_Valve_Pos);             //Scale and map current valve position FROM microcontroller


END_WHILE;
```

Heater

```
//Initalise values for scale functions below

D10 := 15885;

D11 := 0;

D12 := 100;

D13 := 0;


WHILE(TRUE) DO

SCLP(MC2_ANALOG_OUT, D10, Current_Temperature);           //Scale and map current speed FROM microcontroller


IF ((SEL_SW = TRUE) AND (Heat_ENABLE = TRUE)) THEN        //When the selector switch is turned on and heating enabled

Heater_ON := TRUE;                                        // turn on heater run signal.
```

```
END_IF;

IF (SEL_SW = FALSE OR Heat_ENABLE = FALSE) THEN                 //When selector switch is turned off or heating enable
Heater_ON := FALSE;                                 // is off, switch off the heater on signal
END_IF;

IF (Current_Temperature < Temp_SP) THEN                     //if current temp is lower than setpoint, begin debounce
TMR(T_Heater_DELAY, 100);                               // timer
END_IF;

IF (T_Heater_Delay = TRUE) THEN                             //once debounce timer has completed, enable heating
Heat_ENABLE := TRUE;
ELSE
Heat_ENABLE := FALSE;
END_IF;

MC3_INPUT_1 := Heater_ON;                               //Map heater on signal TO microcontroller
```

```
END_WHILE;
```

Vessel PID Loop

```
//Initalise values for scale functions below

D0 := 100;

D1 := 0;

D2 :=15885;

D3 :=0;


D10 := 15885;

D11 := 0;

D12 := 100;

D13 := 0;


WHILE(TRUE) DO

SCLP(Speed_SP, D0, MC2_ANALOG_IN);                    //Scale and map speed setpoint TO microcontroller
```

```
SCLP(MC2_ANALOG_OUT, D10, Current_Speed);                //Scale and map current speed FROM microcontroller

SCLP(MC3_ANALOG_OUT, D10, Tank_Level);                   //Scale and map tank level FROM microcontroller

FLT(Level_SP, Level_SP_REAL);                            //Convert int to real for PID function block

FLT(Tank_Level, Tank_Level_REAL);                        //Convert int to real for PID function block

INT(DPIDE_MV, Speed_SP);                                 //Convert real to int from PID function block

IF ((GREEN_PB = TRUE) OR (Pump_RUN = TRUE & RED_PB = FALSE)) THEN //When the green pushbutton is pressed, turn on motor

Pump_RUN := TRUE;                                        // run signal. Continue motor run signal while red

ELSE                                                     //pushbutton is not pressed

Pump_RUN := FALSE;

END_IF;

IF (SEL_SW = TRUE)THEN                                   //When the selector switch is on, turn on valve

Open_Valve := TRUE;                                      // open signal
```

```
ELSE
Open_Valve := FALSE;
END_IF;

TMRH(T_0, DPIDE_Cycle);

IF (T_0 = TRUE) THEN
AS_DPIDE1( PID_RUN := DPIDE_PID_RUN ,
        SV := Level_SP_REAL ,
        PV := Tank_Level_REAL ,
        PID_MODE := DPIDE_PID_MODE ,
        PID_MAN := DPIDE_MAN ,
        MOUT_AUTO := DPIDE_MOUT_AUTO ,
        CYCLE := DPIDE_Cycle ,
        KP := DPIDE_KP ,
        KI := DPIDE_KI ,
        KD := DPIDE_KD ,
```

```
        TF := DPIDE_TF ,
        PID_EQ := DPIDE_PID_EQ ,
        PID_DE := DPIDE_PID_DE ,
        PID_DIR := DPIDE_DIR ,
        ERR := DPIDE_ERR ,
        MV_MAX := DPIDE_MV_MAX ,
        MV_MIN := DPIDE_MV_MIN ,
        MOUT := DPIDE_MOUT ,
        BIAS := DPIDE_BIAS ,
        I_MV := DPIDE_I_MV ,
        MV => DPIDE_MV );

T_0 := FALSE;
END_IF;

MC1_INPUT_1 := Open_Valve;                    //Map valve open signal TO microcontroller
Valve_Is_Open := MC1_OUTPUT_1;                //Map valve opened signal FROM microcontroller
```

```
MC3_INPUT_2 := Valve_Is_Open;                    //Map valve opened signal TO Microcontroller

MC2_INPUT_1 := Pump_RUN;                         //Map pump run signal TO microcontroller

Pump_RUNNING := MC2_OUTPUT_1;                    //Map pump running signal FROM microcontroller

MC3_INPUT_1 := Pump_RUNNING;                     //Map pump running signal TO microcontroller


END_WHILE;
```

Appendix F – PLC Training Tool Instructions

PLC Training Tool Instructions

Connection

The PLC used in this tool is a Delta DVP32ES311T. The software used to program this PLC is called ISPSoft, and Delta have a separate software used for communication with the PLC called COMMGR. The instructions below are for downloading this software and then using it to connect to the PLC tool.

1. Follow this link - https://downloadcenter.deltaww.com/en-US/DownloadCenter
2. In the keywords box search for "ISPSOFT V3.17"
3. Download the option at the top of the list
4. Then in the keywords box search for "COMMGR V1.14"
5. Again, download the option at the top of the list
6. Install both pieces of software
7. Connect to PLC with an ethernet cable and ensure that your network adapter is set to an IPv4 address set out like this 192.168.1.xxx, with xxx replaced with your specific address. The tool is 192.168.1.5, so make sure not to use that address.
8. Open ISPSoft
9. Click on the drop downs at the top 'PLC' -> 'Transfer' -> 'Upload'.
10. Name the project and choose where you would like to save it
11. On the next screen, click Transfer and it will upload the existing configuration on the PLC training tool
12. From here you should be ready to begin using the tool

Initial Configuration

From the upload above, there should be a program called "Mapping" where all the inputs and outputs are mapped to/from the microcontrollers and the external switches. This mapping is as shown on table 1 below.

Table 1 – PLC I/O Mapping

PLC I/O Address	Global Variable Name
X00	MC1_OUTPUT_1
X01	MC1_OUTPUT_2
X02	MC2_OUTPUT_1
X03	MC2_OUTPUT_2

X04	MC3_OUTPUT_1
X05	MC3_OUTPUT_2
X06	GREEN_PB
X07	RED_PB
X08	SEL_SW
Y00	MC1_INPUT_1
Y01	MC1_INPUT_2
Y02	MC1_INPUT_3
Y03	MC2_INPUT_1
Y04	MC2_INPUT_2
Y05	MC2_INPUT_3
Y06	MC3_INPUT_1
Y07	MC3_INPUT_2
Y10	MC3_INPUT_3
D28000	MC1_ANALOG_OUT
D28001	MC2_ANALOG_OUT
D28002	MC3_ANALOG_OUT
D28004	MC2_ANALOG_IN
D28005	MC3_ANALOG_IN

This program should already exist on the PLC so there should be no need for any changes to this mapping. This will allow you to simply use the tag names on the right when you are writing your program. This program is in cyclic task 0, which should be currently set to active.

There should also be a number of example programs in cyclic task 1. This task is inactive, and the programs will not operate unless they are shifted to an active cyclic task like the one the Mapping program is in. These inactive programs are designed as examples for the user to look over, should they need to see one way the below scenarios can be completed. If the user wants to see one of these programs run, they will need to move it from task 1 to task 0. This is done by following the below steps.

1. Clicking the drop down on “Tasks” on the left menu in ISPSoft.
2. Double click on “Cyclic Task (1)”. This will open the task manager and you should be able to see that Task 1’s Active box is unchecked.
3. In the assigned POUs window select the program you wish to run.
4. Click the arrow pointing towards the unassigned POU window.
5. In the “Task Type” window, select Cyclic Task 0. This task should have the Active box checked.

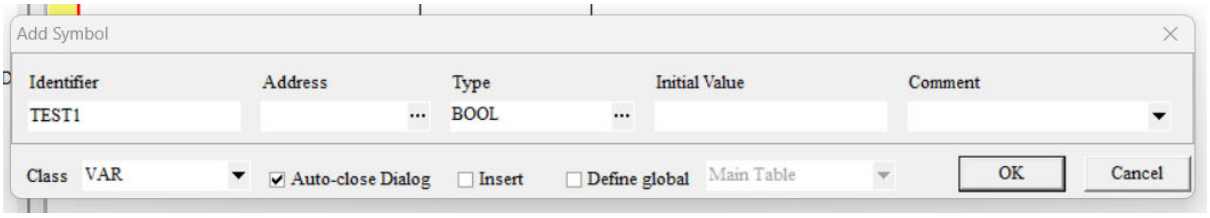
6. Select the program in the unassigned POUs.
7. Press on the arrow pointing towards the assigned POUs window.
8. Click ok
9. Download to the PLC and this program will now be running.

Programming

To begin building a program, follow the below steps.

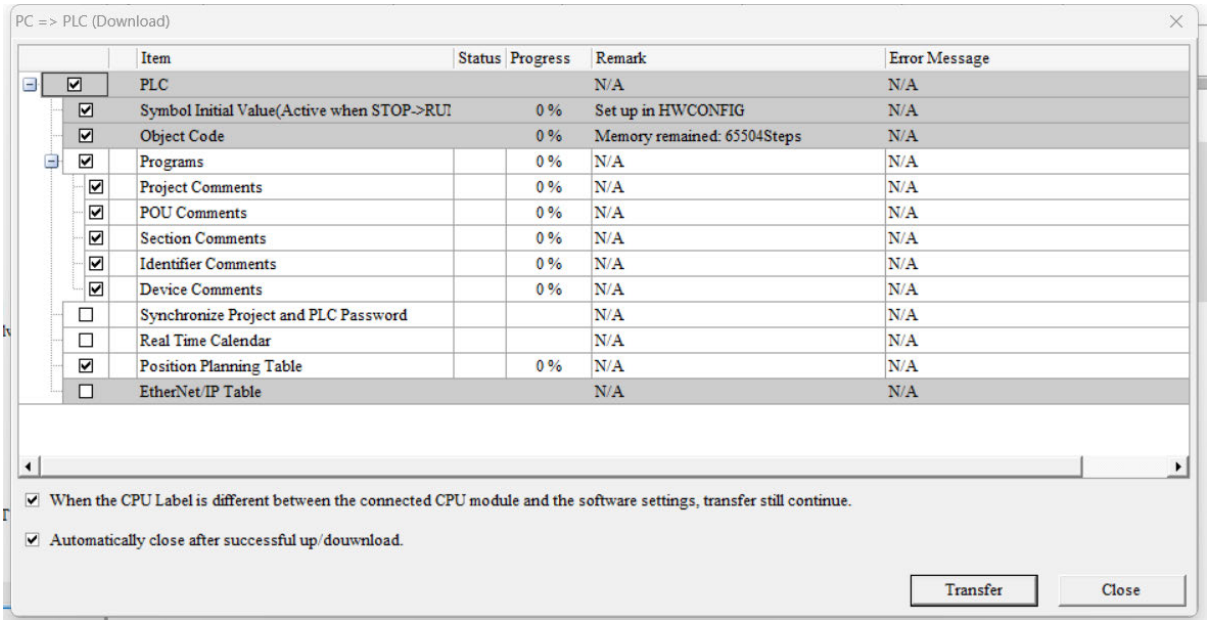
1. Right click on Programs on the left menu and then click on New
2. Type in the name you would like for your program, and select Cyclic Task 0 as this task is active and will allow your program to run.
3. Choose your desired programming language and then click ok.
4. If using structured text, you can simply type in your code line by line. If using ladder, you will need to select the item you'd like to drop into the network rung using the below buttons

5. Once one of the above is selected, you can go to the desired network and when hovering over the dotted line box you can click and it will drop the item. This dotted line box will then disappear and you can click anywhere on the line. Hovering over the line will indicate roughly where the item will go, eg. Left of previous, underneath previous, etc.
6. Click on the question marks on each item to assign them a variable name. If using the global variables from the Mapped program then they will already exist, but if you are creating variables that will only operate within this program then it will ask you to declare them as you type them in using the below box.



This will create a local variable, only usable within the program you are writing currently.

7. Once you've written the program you'd like to put onto the PLC for operation, click on the Compile menu at the top of the program, and then click Compile. This will check your program and if there are no errors it will compile into machine code for transfer.
   If there are errors, these will be shown at the bottom of the screen and you will need to fix any mistakes before attempting to compile again.
8. Click on PLC at the top of the program, then Transfer, and then Download. This will bring up the below window.



When you click transfer, it will inform you that the PLC will need to be stopped to make this transfer. Follow this through and once the transfer is completed it will change the PLC back into RUN and the transfer is complete.

9. Now you can click on PLC at the top of the program, then Online Mode. This will allow you to monitor your program running to see things changing state, changing values, etc.

Scenarios

DOL Motor

This program will simulate the normal signals to and from a Direct Online (DOL) Motor.

Setup

1. Turn microcontroller selector switch to 1.
2. Scroll until DOL Motor is shown on the HMI.
3. Press the select button and confirm that DOL Motor is the current selection.

The microcontroller is programmed to receive a run signal on MC1_INPUT_ONE, and after a short period of time it will return a running signal on MC1_OUTPUT_ONE. This is designed to mimic the behaviour of a contactor commonly used in a DOL motor arrangement.

Programming

Use the green pushbutton (GREEN_PB) to start the motor run signal, and use the red pushbutton (RED_PB) to stop it. Map both motor run, and motor running to the appropriate global variables.

Ensure network comments are left by clicking on the grey rectangular box just below the network label as seen below.

VSD Motor

This program will simulate the normal signals to and from a Variable Speed Drive (VSD) Motor.

Setup

1. Turn microcontroller selector switch to 2.
2. Scroll until VSD Motor is shown on the HMI.
3. Press the select button and confirm that VSD Motor is the current selection.

The microcontroller is programmed to receive a run signal on MC2_INPUT_ONE, and after a short period of time it will return a running signal on MC2_OUTPUT_ONE. This is designed to mimic the behaviour of a VSD returning a running signal once it is up to speed. It is also expecting a Speed Setpoint analogue signal on MC2_ANALOG_IN and it will return a Current Speed value on MC2_ANALOG_OUT.

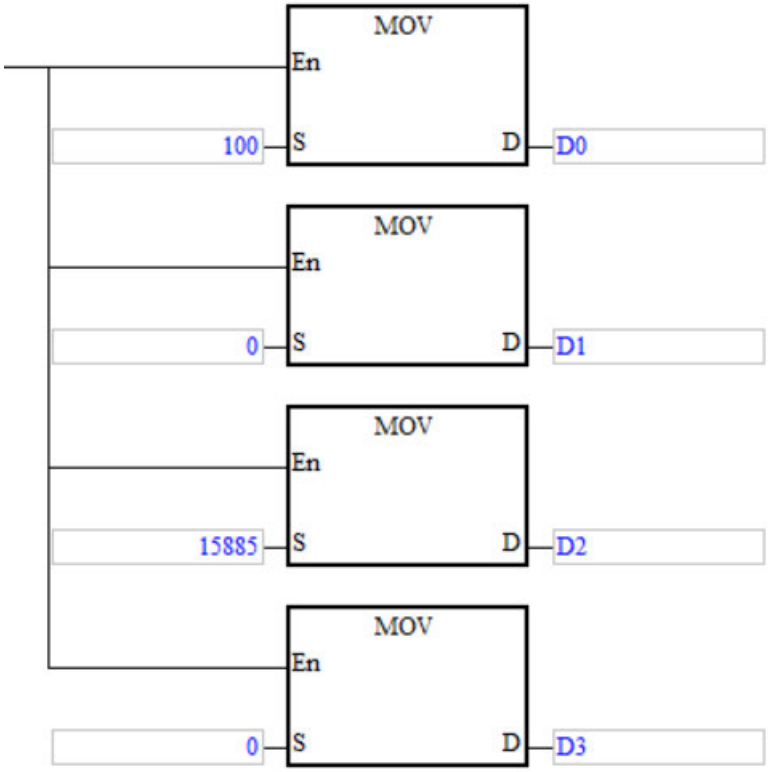
Programming

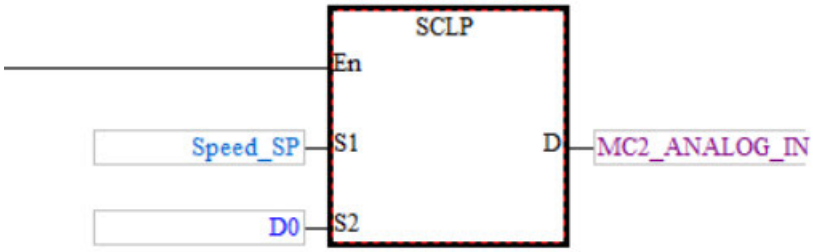
Use the green pushbutton (GREEN_PB) to start the motor run signal, and use the red pushbutton (RED_PB) to stop it. Map both motor run, and motor running to the appropriate global variables.

See the example VSD Motor Basic program for how to set up the analogue signal scaling on this PLC. Starting at an arbitrary address, such as D0 as the example program uses, move the scaling values in as per the image below. The below setup will allow for scaling FROM 0-100 to 0-15885 (15885 is roughly the maximum value on the analogue input/output signal in the PLC).

The SCLP function shown in the second image is what actually scales the value. Using the previously set up scaling values, this function allows the user to scale the value on S1 using the parameters addressed in S2, and outputs the new value on D. This allows the user to enter a speed setpoint in percentage form, and it will output the value in a usable value for the analogue output module.

Reversing the order of the scaling values allows for mapping the analogue signal coming back in from the microcontroller to the 0-100 percentage value. Again, this can be seen in the VSD Motor Basic example program, in networks 5 and 7.





Ensure network comments are left.

Discrete Valve

This program will simulate the normal signals to and from a Discrete (Open/Closed) Valve.

Setup

1. Turn microcontroller selector switch to 1.
2. Scroll until Discrete Valve is shown on the HMI.
3. Press the select button and confirm that Discrete Valve is the current selection.

The microcontroller is programmed to receive an open signal on MC1_INPUT_ONE, and after a short period of time it will return an open signal on MC1_OUTPUT_ONE. This is designed to mimic the behaviour of a valve being told to open, and then indicating that it is open.

Programming

Use the selector switch (SEL_SW) to send the open signal. Map both valve open, and valve is open to the appropriate global variables.

Ensure network comments are left.

Analogue Valve

This program will simulate the normal signals to and from an Analogue (Positional) Valve.

Setup

1. Turn microcontroller selector switch to 2.
2. Scroll until Analogue Valve is shown on the HMI.
3. Press the select button and confirm that Analogue Valve is the current selection.

The microcontroller is programmed to receive a Speed Setpoint analogue signal on MC2_ANALOG_IN and it will return a Current Speed value on MC2_ANALOG_OUT. This is to mimic an analogue valve being given an analogue signal and providing feedback of its actual position.

Programming

See the example Analogue Valve Basic program for how to set up the analogue signal scaling on this PLC. Starting at an arbitrary address, such as D0 as the example program uses, move the scaling values in as per the image below. The below setup will allow for scaling FROM 0-100 to 0-15885 (15885 is roughly the maximum value on the analogue input/output signal in the PLC).

The SCLP function shown in the second image is what actually scales the value. Using the previously set up scaling values, this function allows the user to scale the value on S1 using the parameters addressed in S2, and outputs the new value on D. This allows the user to enter a position setpoint in percentage form, and it will output the value in a usable value for the analogue output module.

Reversing the order of the scaling values allows for mapping the analogue signal coming back in from the microcontroller to the 0-100 percentage value. Again, this can be seen in the Analogue Valve Basic example program, in networks 2 and 4.

Vessel – VSD Motor, Discrete Valve

Use some of what has been completed above, and combine to program a VSD Pump filling a vessel, with a discrete valve being used to drain the vessel.

Setup

1. Turn microcontroller selector switch to 1.
2. Scroll until Discrete Valve is shown on the HMI.
3. Press the select button and confirm that Discrete Valve is the current selection.
4. Turn microcontroller selector switch to 2.
5. Scroll until VSD Motor is shown on the HMI.
6. Press the select button and confirm that VSD Motor is the current selection.
7. Turn microcontroller selector switch to 3.
8. Scroll until Vessel is shown on the HMI.
9. Press the select button and confirm that Vessel is the current selection.

Microcontroller 1 is programmed to receive an open signal on MC1_INPUT_ONE, and after a short period of time it will return an open signal on MC1_OUTPUT_ONE. This is designed to mimic the behaviour of a valve being told to open, and then indicating that it is open.

Microcontroller 2 is programmed to receive a run signal on MC2_INPUT_ONE, and after a short period of time it will return a running signal on MC2_OUTPUT_ONE. This is designed to mimic the behaviour of a VSD returning a running signal once it is up to speed. It is also expecting a Speed Setpoint analogue signal on MC2_ANALOG_IN and it will return a Current Speed value on MC2_ANALOG_OUT.

Microcontroller 3 is programmed to receive a Pump Running signal on MC3_INPUT_ONE, and a Valve is Open signal on MC3_INPUT_TWO. It is also expecting to receive a Pump Speed signal on MC3_ANALOG_IN. This is to allow for the level of the tank to go up and down faster and slower based on the speed of the pump. It will return a Tank Level value on MC3_ANALOG_OUT.

Programming

Use the methods learned previously to combine these three devices into a usable program where you can input a Level setpoint, and the pump will vary its speed to try and maintain that level. The discrete valve is the only means of draining the vessel and it is slower than the pump is filling at full speed. See the example program for hints and tips on how to achieve this – PID tuning may be the most appropriate option.

Appendix G – User Feedback Form

TO BE COMPLETED PRIOR TO TOOL USE

How would you rate your ability with PLC programming currently?

☐ Advanced

☐ Intermediate

☐ Beginner

How often do you perform PLC programming

☐ Regularly

☐ Occasionally

☐ Rarely

☐ Never

How confident are you writing PLC programs in each language?

Ladder

☐ Very confident

☐ Somewhat confident

☐ Not confident

Structured Text

☐ Very confident

☐ Somewhat confident

☐ Not confident

If you’d like to, comment below what would you like to achieve from the use of this tool

TO BE COMPLETED AFTER TOOL USE

Overall, how would you rate your experience with this training tool?

☐ Positive

☐ Neutral

☐ Negative

Were the instructions clear & easy to follow? And if not, why not?

Yes/No

_____
_____
_____
_____

Would you describe your knowledge and familiarity with writing PLC programs as any different after using the tool?

_____
_____
_____
_____

Was there anything you found particularly difficult whilst using the tool?

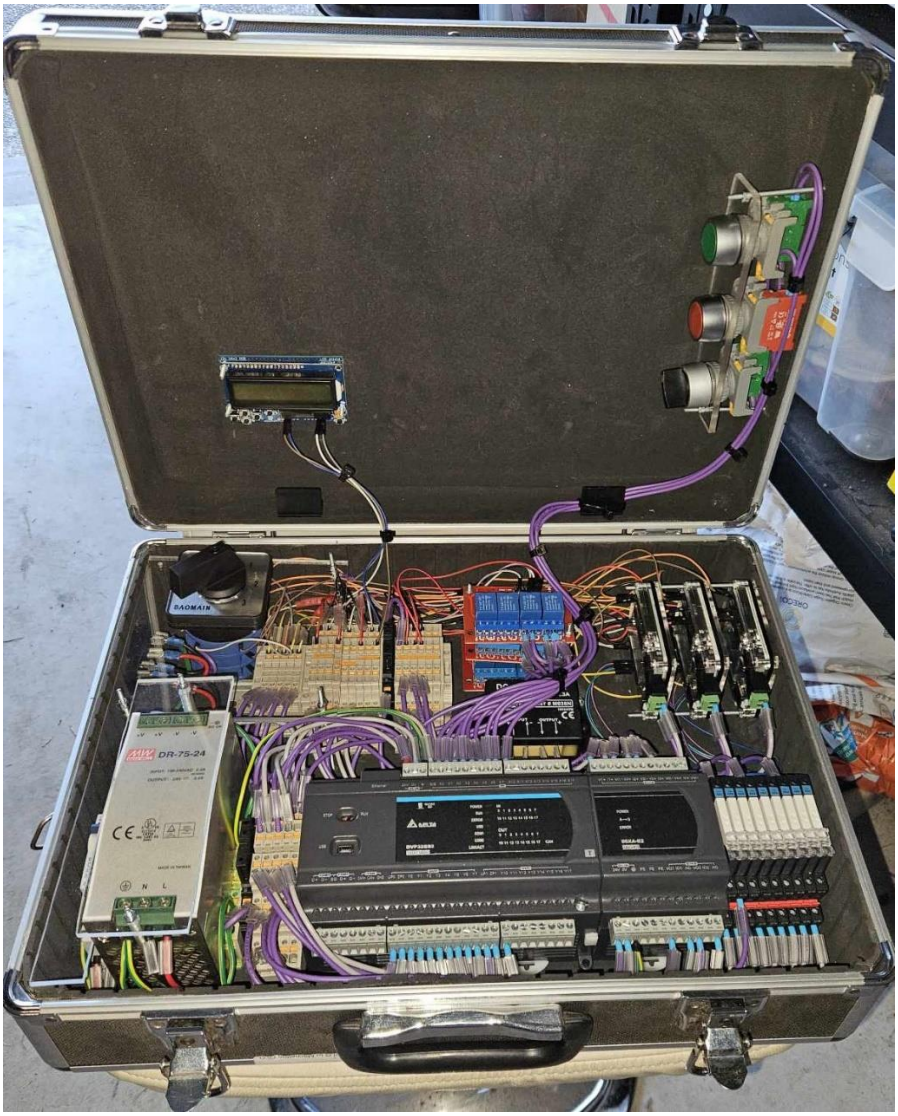
_____
_____
_____
_____
_____
_____

How did you find the operation of the microcontrollers simulating industrial devices? Did they behave in ways that you'd expect? If not, why not?

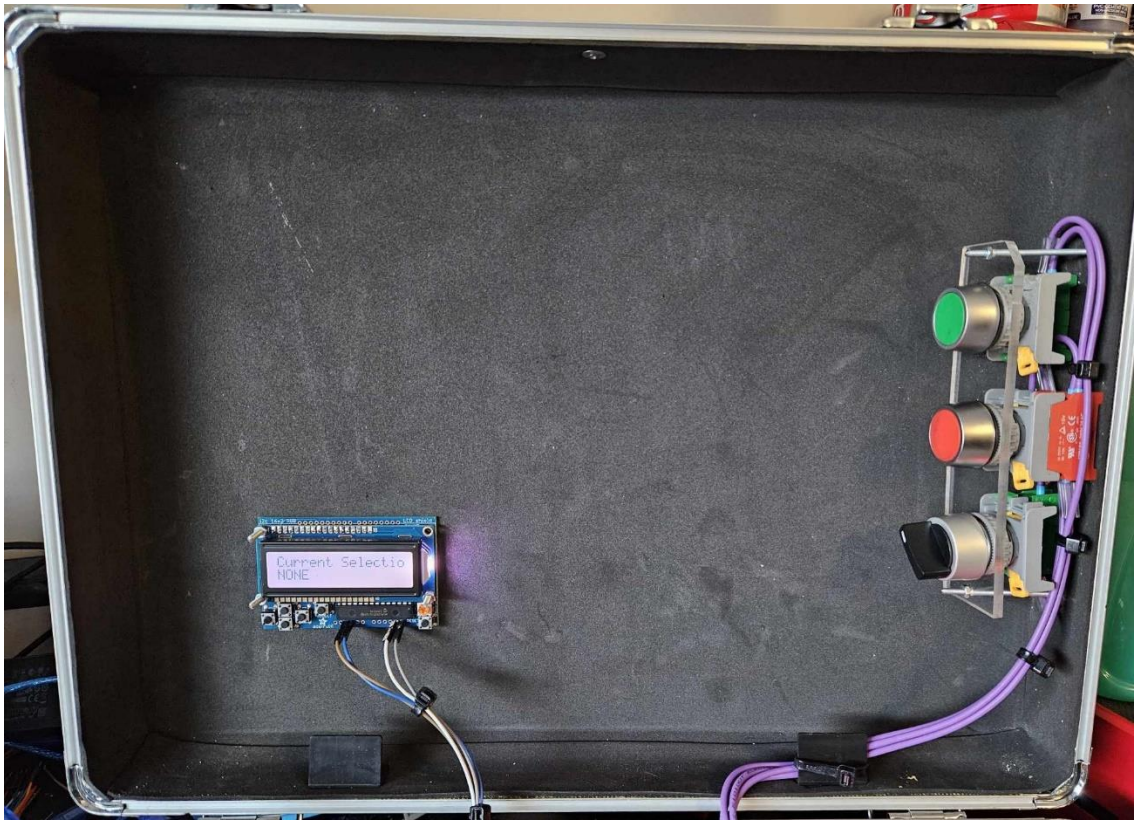
Any further comments?

Appendix H – PLC Training Tool Photos

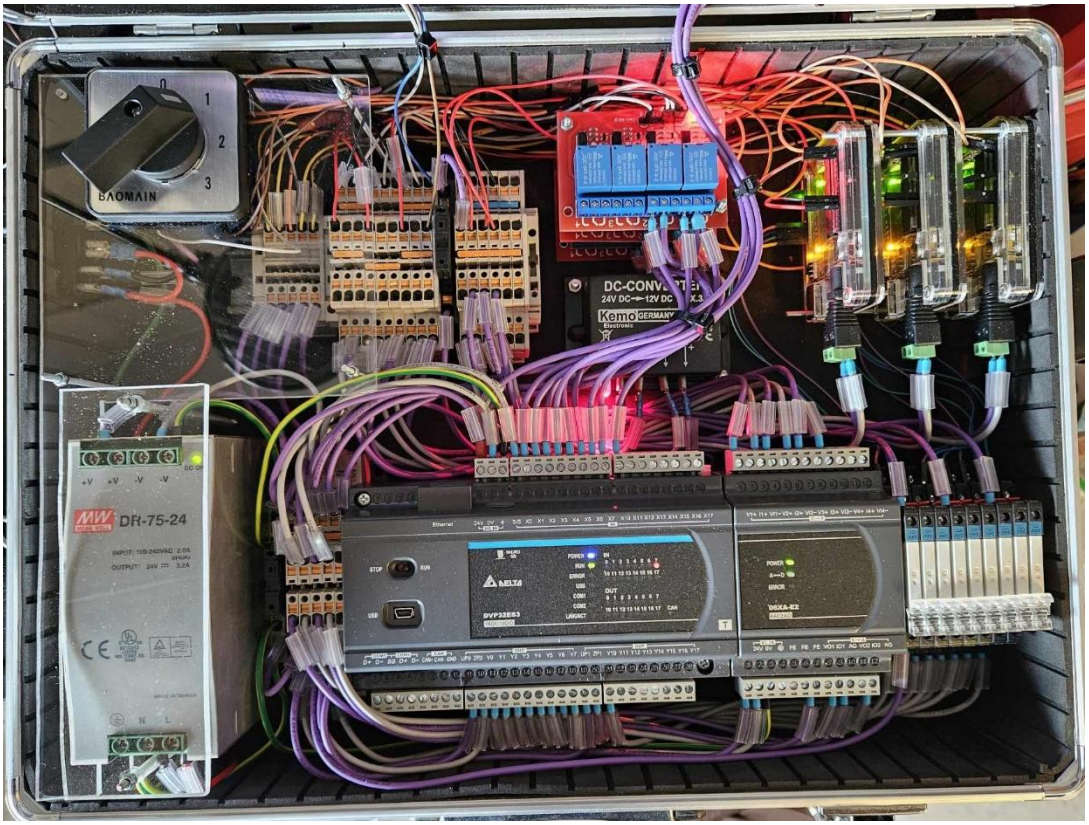
Exterior Case of PLC Training Tool



Interior of PLC Training Tool



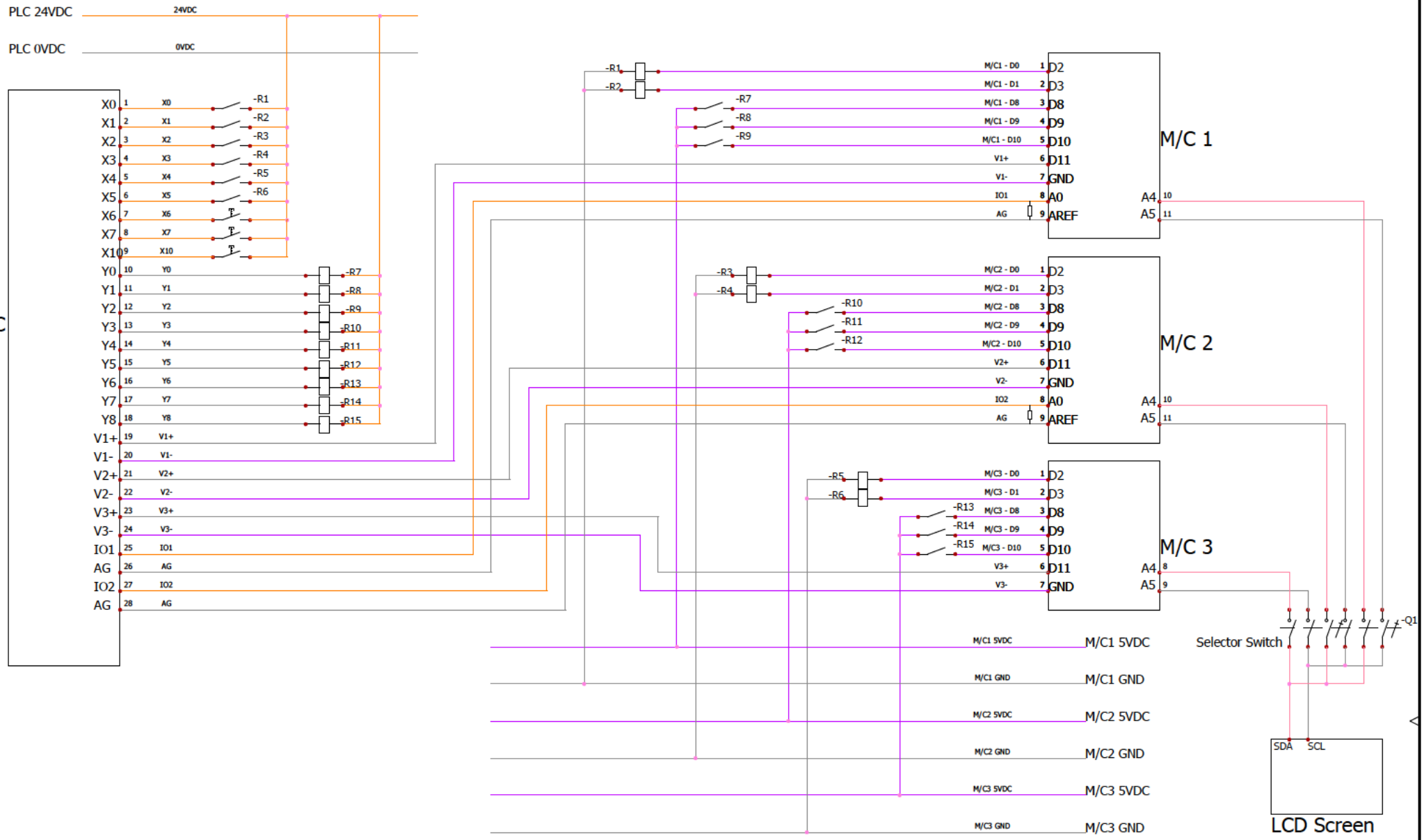
Upper Interior of PLC Training Tool



Lower Interior of PLC Training Tool

Appendix I – Electrical Schematics

Please see the following pages for the electrical schematics for this tool.



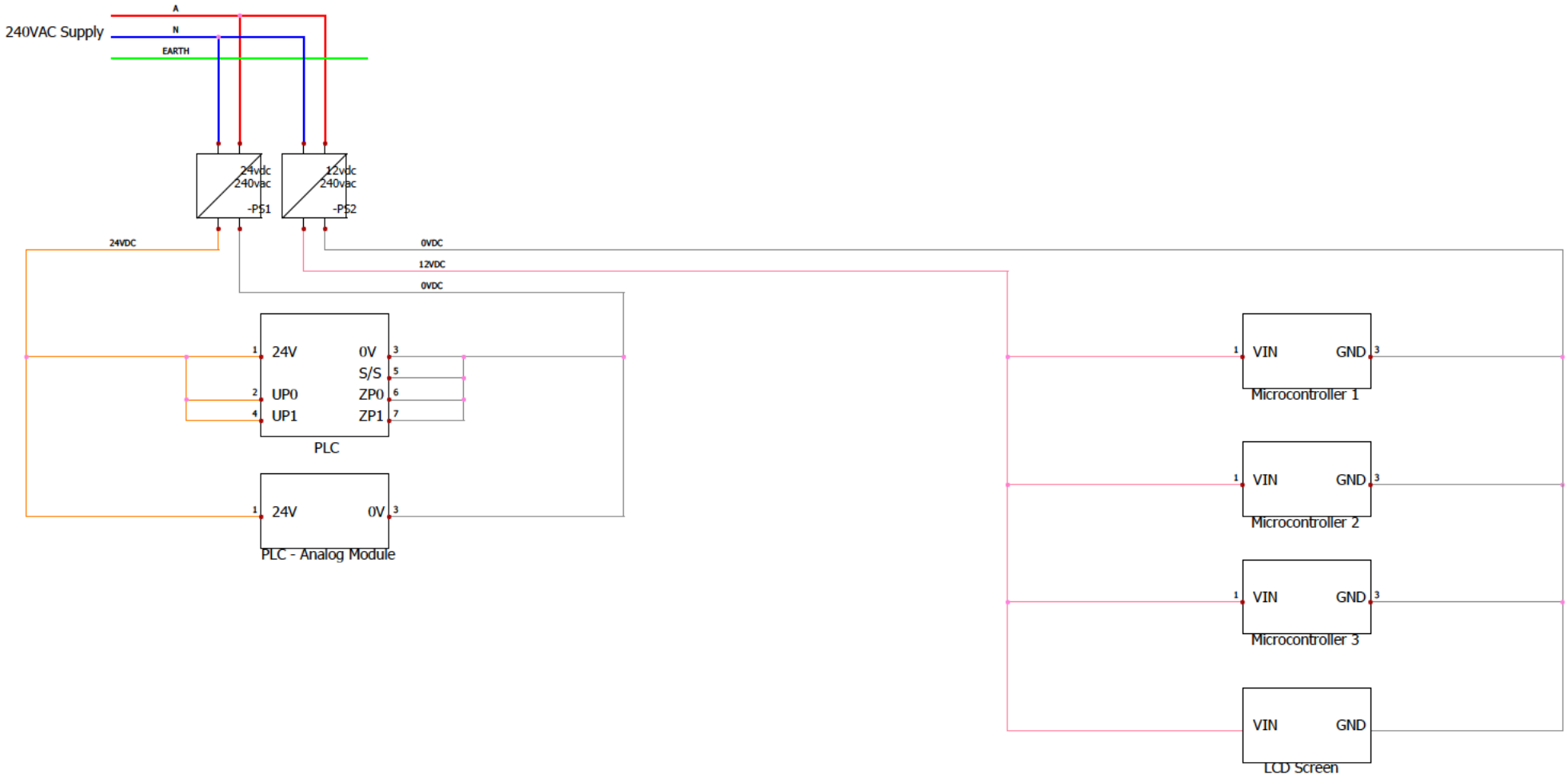
SOLIDWORKS Electrical

Schematics

				REVISION
				0
0	24/05/2023	shane		
REV.	DATE	NAME	CHANGES	SCHEME
User data 1			User data 2	01

CONTRACT:

LOCATION: +L1 Main electrical closet



SOLIDWORKS Electrical

Schematics

REV.	DATE	NAME	CHANGES
0	24/05/2023	shane	

CONTRACT:

LOCATION: +L1 Main electrical closet

User data 1

User data 2

REVISION 0

SCHEME 02