Arduino-Based Computer Interface for the PN-2320 Pneumatic-Control

A Proposed Project Presented to the

Department of Electrical and Computer Engineering

College of Engineering and Design

Silliman University

Dumaguete City

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in Computer Engineering

By

Sinco, Micah Jude Kabristante

Villamor, Alfonzo Miguel Maxino

Adviser: Engr. Johnson B. Diputado, MEng

Project Coordinator: Engr. Dennis V. Salboro, MEng, REE

March 2018

APPROVAL SHEET

This is to certify that this Final Project, entitled "Arduino-Based Computer Interface for the PN-2320 Pneumatic-Control", and submitted by Micah Jude K. Sinco and Alfonzo Miguel M. Villamor to fulfill part of the requirements for the degree of Bachelor of Science in Computer Engineering was successfully approved on _____.

Engr. Johnson B. Diputado, M Eng
Project Adviser

Engr. George Bernard E. Carlos, REE, M Eng
Member, Board of Panelist

Engr. Klint Ian V. Austero, MBA
Member, Board of Panelist

The Department of Computer and Electrical Engineering endorses acceptance of this Final Project as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering.

Engr. Hazel Joie A. Eparwa, REE, M Eng
Chairperson
Department of Electrical and Computer Engineering

The Final Project is hereby officially accepted as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering.

Engr. Dennis V. Salboro, REE, M Eng
OIC-Dean / Project Coordinator
College of Engineering and Design

ACKNOWLEDGEMENTS

This project would not have been possible without the knowledge imparted upon the by our adviser, Engr. Johnson Diputado. For guiding us throughout the entire process of this paper, we would like to thank our project coordinator, Engr. Dennis Salboro. For the well-constructed inputs for the betterment of this project, we would like to thank our panelists, Engr. George Bernard Carlos and Engr. Klint Ian Austero. For constantly lending us a helping hand on the operation of the electro-pneumatic trainer, we would like to express our deepest gratitude to Mr. Roger Alcancia and Engr. Jose Marlon Cornelio. Such help would not have been attained without the approval of the Mechanical Engineering chairperson, Engr. Louie Pierre Equio.

Furthermore, we would like to thank our parents for their moral and financial support in the creation of this project. Our classmates and friends were also of great help in motivating us to complete this project.

Lastly, we would like to thank the Almighty God for giving us the strength, ability, and opportunity to undertake this project and to preserve and complete it satisfactorily. Without His blessings, this achievement would not have been possible.

ABSTRACT

This paper presents a documented research project entitled “Arduino-Based Computer Interface for the PN2320 Pneumatic Control”. The purpose of this study is to be able to help the Mechanical Engineering students of the College of Engineering and Design of Silliman University in the field of pneumatics. The proponents are well aware that the current process in manipulating the electro-pneumatic trainer is time-consuming with errors that are bound to happen. To counter attack this problem, the proponents created a graphical user interface (GUI) application. The said application will act as an interpreter from widgets to lines of codes for the Arduino Uno microcontroller that would operate the electro-pneumatic trainer. Moreover, the researchers created a ULN2003 driver circuit which allows the Arduino Uno microcontroller to control the higher voltage solenoid valves of the PN2320 Pneumatic Control.

TABLE OF CONTENTS

CHAPTER 1: Introduction 1

1.1 Background of the Study 1

1.2 General Objective 3

1.3 Specific Objectives 3

1.5 Solution to the Problem 4

1.6 Significance of the Study 4

1.7 Scope and Limitations 5

1.8 Definition of Terms 6

CHAPTER 2: Review of Related Literature 7

2.1.1 Graphical User Interface 7

2.1.2 Microcontroller 9

2.1.3 Electro-Pneumatic Trainer 10

2.1.4 Solenoid Valves 11

2.1.5 Driver Circuit 12

2.2 Related Studies 13

2.3 Theoretical Framework 14

2.4 Conceptual Framework 15

CHAPTER 3: Methodology 17

3.1 Block Diagram 17

3.2 Materials and Equipment 18

3.2.1 Arduino Uno Microcontroller 18

3.2.2 ULN2003 Integrated Circuit 19

3.2.3 PN2320 20

3.2.4 PN2330 21

3.3 System Flow 22

3.4 Workflow 24

3.5 Time Table 26

CHAPTER 4: Presentation, Analysis and Interpretation of Result 27

4.1 Results from Experiment 27

4.2 Economic Analysis 30

CHAPTER 5: Summary, Conclusion and Reccomendation 32

5.1 Summary 32

5.2 Conclusion 33

5.3 Recommendation 33

REFERENCES 34

APPENDIX A: ULN 2003 Datasheet 37

APPENDIX B: Application Source Code 39

APPENDIX C: Arduino Source Code 54

APPENDIX D: Manual 57

APPENDIX E: Miscellaneous 68

LIST OF FIGURES AND TABLES

Figure 2.1. *Graphical User Interface*……..……..……..8

Figure 2.2. *Microcontroller*……..……..…….....9

Figure 2.3. *Electro-Pneumatic Trainer*……...……....…..10

Figure 2.4. *Off and On Solenoid Valves*……...…….…….12

Figure 2.5. *Conceptual Framework*……..…..16

Figure 3.1. *Block Diagram*……..…..….18

Figure 3.2. *Arduino Uno Microcontroller*……...…...…….….…..19

Figure 3.3. *ULN2003 IC*……...….....20

Figure 3.4. *PN2320*……..….....21

Figure 3.5. *PN2330*……...……....22

Figure 3.6. *System Flowchart*……....……....23

Figure 3.7. *Flowchart of Workflow*……...…....25

Figure 3.8. *Time Table*…….…….……..26

Figure 4.1. *Project Circuit*…….…….28

Figure 4.2. *Schematic Diagram*……..……...…….....28

Figure 4.3. *Graphical User Interface Application*…….....…...….29

Figure 4.4. *Complete Installation*……...……30

Table 4.1. *Economic Data*……...……..…....31

CHAPTER 1

INTRODUCTION

This chapter does not only contain the essential information that this project holds but also the vital goals that it must achieve. This chapter contains the background of the study, general objective, specific objectives, statement of the problem, solution to the problem, scope and limitations, and definition of terms.

1.1 Background of the Study

The fundamentals behind pneumatics and hydraulics are similar but with one difference, pneumatics use gas to transmit power rather than liquid. Although the most commonly used gas is compressed air, other good alternatives for special applications are nitrogen and other inert gases. A compressor pumps the gas into a receiver that can store a large volume of compressed air to be later used by the equipments or instruments (National Fluid Power Association, 2015). Mechanical engineering students of the College of Engineering and Design of Silliman University in Dumaguete City study pneumatics using an electro-pneumatic trainer. A common laboratory equipment used by many schools around the world. The trainer's pneumatic components are controlled electronically, hence it is referred to as electro-pneumatic: electro meaning electric related and pneumatic meaning gas or air operated. The trainer is operated by a student through a microcontroller. Dahl

(2014), explains that a microcontroller sends signals based on its reaction to signals it receives. The signal it receives is transmitted from a sensor and the signal that is outputted from it is used to operate devices. Writing a code written in a programming language like C and uploading it to the microcontroller is the way to control it.

Although it is essential for mechanical engineering students to learn how to code in order for them to use a microcontroller to operate the electro-pneumatic trainer. Writing a code that makes a microcontroller instructs the trainer to perform a specific action is rather time consuming. And even though a code has been fully written down, errors are bound to happen when it is being executed from time to time. The researchers of this project aim to make the operation of the electro-pneumatic trainer more efficient and quicker by designing a graphical user interface (GUI) application.

Rather than coding in a command line, a graphical user interface (GUI) allows interaction with electronic devices through visual indicators or icons (Computer Hope, 2017). It helps students operate the trainer more efficiently and faster since the GUI application acts as a sort of interpreter between them and programming language that the microcontroller understands and requires in order for it to instruct the trainer. In order for a student to operate the trainer in a specific way, a press of a single button out of many on the layout of the application is only needed. Once a button is pressed, a specific prewritten code related to the purpose of the button is uploaded automatically onto the microcontroller. The microcontroller processes the code and executes the instruction.

1.2 General Objective

Implement a graphical user interface (GUI) application that will act as an interpreter from simple widgets to lines of codes for a microcontroller that would operate the electro-pneumatic trainer.

1.3 Specific Objectives

- Construct a ULN2003 driver circuit that allows the Arduino microcontroller to control the higher voltage solenoid valves of the PN2320 Pneumatic Control.
- Layout a graphical user interface using Qt that will have button widgets for simple use and a text editor where users can write their own codes.
- Develop source code using Qt for GUI application that automatically uploads written code to microcontroller when widgets are triggered.

1.4 Statement of the Problem

It is currently the digital age but some laboratory equipment in the mechanical engineering department are still operated manually by the user. One example of this is the electro-pneumatic trainer. The mechanical engineering students learn how to operate the trainer electronically using a microcontroller. But operating the trainer through the

microcontroller and observing the output can take time since the students have to develop a written code just for the trainer to perform a single action. In addition, the written code might not work since errors are bound to happen from time to time which will then result to a more time-consuming laboratory experiment.

1.5 Solution to the Problem

As stated in the problem, mechanical engineering students cannot quickly observe the different operations of the electro-pneumatic trainer when controlling it with a microcontroller. To counter this problem, a graphical user interface application is developed. The GUI application allows immediate observation of the behaviors of the trainer with a press of a button on the layout of the user interface. And the application will likely not have any bugs or errors since it will be tested several times to ensure a fully functioning GUI application.

1.6 Significance of the Study

Designing a GUI application for the electro-pneumatic machine helps the student operate the trainer faster and with more efficiency. Operating the machine takes no time since the application automatically uploads a written code to the microcontroller. Due to this, students will be able to quickly observe the actions the electro-pneumatic trainer performs.

Through this application, students will also be able to learn more on how to program a microcontroller. The application displays the written code for the microcontroller when a button is pressed. This allows the students to see how a microcontroller is programmed to operate the trainer in a specific way.

1.7 Scope and Limitations

The focus of this project is the development of a graphical user interface application for the electro-pneumatic trainer in the mechanical engineering department of the College of Engineering and Design in Silliman University. The GUI application will control the behaviour of the solenoid valves on the trainer. Codes for the Arduino microcontroller are automatically uploaded when button widgets are triggered, or lines of codes are executed on the text editor of the layout. The microcontroller will be able to control the higher voltage loads of the trainer using a ULN2003 driver circuit.

The trainer will not properly function if the solenoid valves are not connected correctly to the proper ports on the driver circuit. The valves have to be connected properly since codes uploaded to the microcontroller when button widgets are triggered are fixed unlike the text editor wherein port outputs can be interchanged since the users will be writing the codes themselves.

1.8 Definition of Terms

Graphical User Interface (GUI). In reference to the site Computer Hope (2017), in 1981, the GUI was created by Alan Kay, Douglas Engelbart and another group of researchers. The GUI allowed users to interact with electronic devices through graphical icons.

Electro-Pneumatic Trainer. Experts in SAP Engineers & Consultants (2008) elaborated that the trainer helps you learn pneumatic control and the components used in these kind of control systems.

Pneumatics. In reference to the National Fluid Power Association (2015), unlike hydraulics, power is transmitted by gas in pneumatics. Pneumatics uses many kinds of gases, but compressed air is used constantly.

C. In reference to Bailey (n.d.), C is a small programming language that has only 32 keywords and is a general-purpose language used in many domains to write programs.

Solenoid Valve. The experts at Fluid Controls Ltd (n.d.), explained that the solenoid valve is opened and closed using an electromagnetic solenoid coil. If normally closed, the valve is opened by an electromagnetic force created when electricity passes through the coil.

CHAPTER 2

REVIEW OF RELATED LITERATURE

This chapter contains the important terms, concepts, and principles that are used throughout this project. The definitions included in this chapter are explanations as to the relation of these entries to the project: graphical user interface, microcontroller, electro-pneumatic trainer, solenoid valves, and driver circuit. Also, this chapter holds related studies, and as well as the theoretical and conceptual framework.

2.1.1 Graphical User Interface

Harding (1989) has written that a Graphical User Interface, as shown in Figure 2.1, is a visual operating display that the monitor presents on the monitor to the computer operator. Moreover, Bonsiepe (1968) said that a GUI is a specification for the look and feel of the computer system. Jansen later on discussed that the user issues commands via the GUI to computer applications and it usually has three major components in which are: windowing system, imaging model, and application program interface (Hayes and Baran, 1989). The windowing system builds the windows, menus, and dialog boxes that appear on the screen. The imaging model defines the fonts and graphics that appear on the screen. Finally, the application program interface is the means in which the user specifies how and what windows and graphics appear on the screen. The historical development of the GUI still impacts the three major GUI paradigms in the market today. Historically, all modern

GUI are off-shoots of the Apple Macintosh. This has led to a great deal of standardization and consistency among GUI design criteria. Therefore, most application's GUI adhere to one of the three major GUI paradigms, the Apple Macintosh, the IBM Systems Application Architecture (SAA), or the X-Windowing System. While none of these GUI designs are perfect, the overall design concepts are good enough to make radical departures counterproductive (Norman, 1988), unless there are significant performance enhancements (1998).

Fig. 2.1. *Graphical User Interface*

2.1.2 Microcontroller

A microcontroller, an example is shown below in Figure 2.2, is a computer with most of the necessary support chips onboard. All computers have several things in common, namely: a central processing unit (CPU) that ‘executes’ programs, some random-access memory (RAM) where it can store data that is variables, some read only memory (ROM) where programs to be executed can be stored, input and output (I/O) devices that enable communication to be established with the outside world i.e. connection to devices such as keyboard, mouse, monitors and other peripherals. (Calcutt, Cowan, and Parchizadeh, 2004)

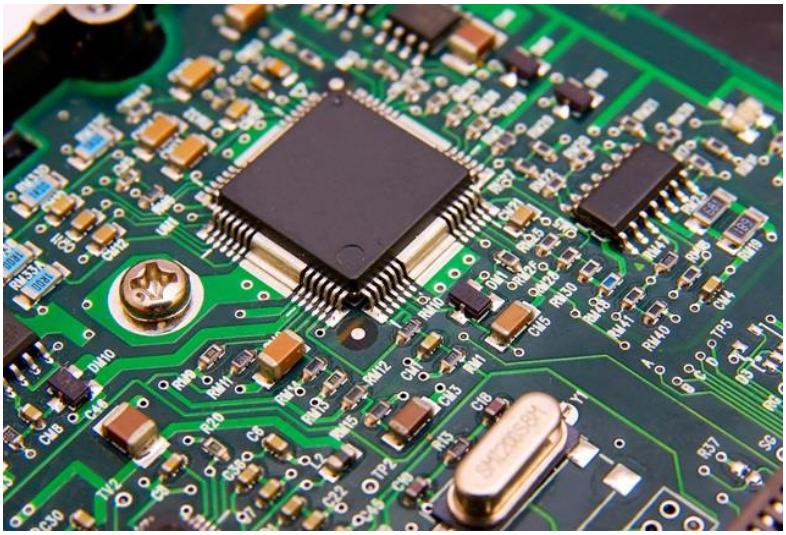


Fig. 2.2. *Microcontroller*

2.1.3 Electro-Pneumatic Trainer

As stated in TC Trainer Co., Inc, this integrated hands-on pneumatic training aid introduces students to basic principles of pneumatics and compressed air devices. This unit is designed to support a large number of student exercises that demonstrate a broad range of pneumatic principles. The associated student exercise manual takes the student step by step through each exercise to create a learning experience that provides a complete and practical introduction to pneumatic technology. The figure below, Figure 2.3, is the trainer to be used in this study.

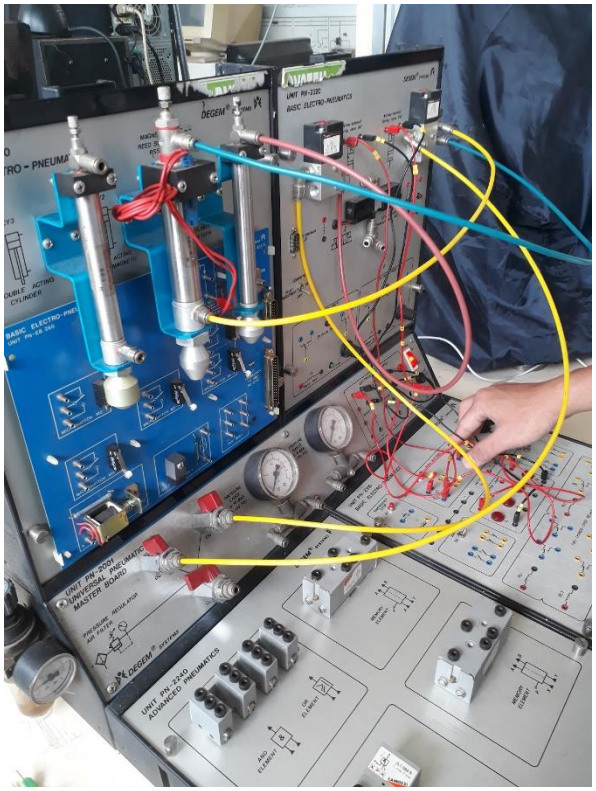


Fig. 2.3. *Electro-Pneumatic Trainer*

2.1.4 Solenoid Valves

In reference to Soft Robotics Toolkits' "*Introduction to Electro-pneumatics*", it was sited that pneumatic circuits can be controlled by electric circuits. The interface between these two circuits is a solenoid valve. Solenoid valves perform the same function as normal pneumatic valves but there are operated electrically. Inside the solenoid valve, there is a coil of wire through which an electric current is passed. It produces a magnetic field which attracts an iron armature. The movement of the armature operates the valve.

Solenoid Operation Off: When the electric current is not flowing, a spring pushes the iron armature out of the coil. A seal connected to the armature blocks port 1. Air can flow between ports 2 and 3. Solenoid Operation On: When current flows, the iron armature is attracted into the coil by a magnetic field. The spring pressure is overcome, and the seal moves to block port 3. Air can flow between port 1 and 2.

When the solenoid valve is on, an electric current will flow through the coil. When current flows through the coil, the iron armature is attracted by magnetism. The solenoid has control of the valve. Port 1 is connected to port 2 and air flows to inflate the soft robot.

When the solenoid valve is off, the coil is de-energized, and the spring has control of the valve. Port 2 is connected to port 3 and air flows out of the soft robot. Figure 2.4 shows a better understanding of how the solenoid valves operates through its schematics.

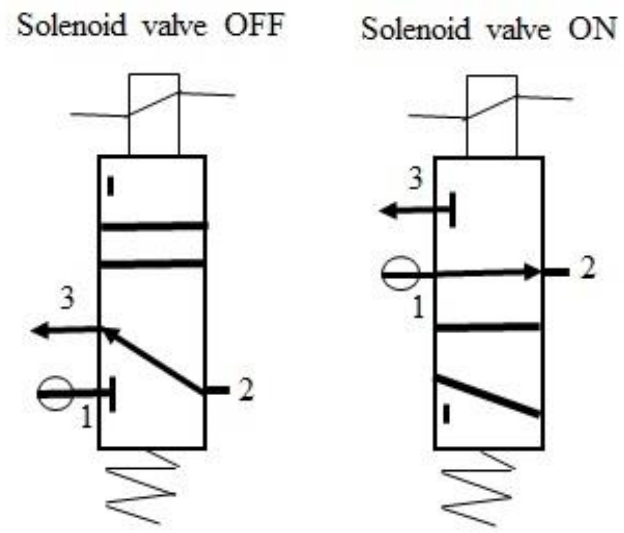


Fig. 2.4. Off and On *Solenoid Valves*

2.1.5 Driver Circuit

The driver circuit, with the use of the ULN2003, is a step-up circuit that will increase the amplitude of the output voltage of the microcontroller to 24V. Kushugra (n.d.) has written that the ULN2003 is a high voltage and high current Darlington array IC. It contains seven open collector darlington pairs with common emitters. A darlington pair is an arrangement of two bipolar transistors. ULN2003 belongs to the family of ULN200X series of ICs. Different versions of this family interface to different logic families. ULN2003 is for 5V TTL, CMOS logic devices. These ICs are used when driving a wide range of loads and are used as relay drivers, display drivers, line drivers etc.

2.2 Related Studies

Burhan, Azman and Othman, *"Electro pneumatic trainer embedded with programmable integrated circuit (PIC) microcontroller and graphical user interface platform for aviation industries training purposes"*. This project aimed to enhance the teaching and learning of the fundamentals of electro-pneumatics in the aviation industry. The researchers wanted to show that students learn better through simulations and hand-on experiences with real equipments or instruments and that the learning experience can be made easier and quicker by interfacing trainers, like the electro-pneumatic trainer they are using in this project, with a user interface. They used Visual Basic (VB) for their graphical user interface that allows the user to operate the trainer. A PIC 16F877 microcontroller is programmed to receive commands from the GUI and execute those commands by sending signals that operate the trainer.

Compared to the usual electro-pneumatic trainer, the researchers have noticed that the experience of operating the trainer is more efficient and safe with an embedded user interface. The students are no longer in danger when tampering with the 240V AC motor switches that power the air motor since it is electronically controlled by the user interface.

Shivajirao and Mane, *"Graphical User Interface for Universal Testing Machine Using Qt"*. This project was done by researchers in the Department of Electronics Engineering of Walachand College of Engineering in Sangli, Maharashtra, India. The project aimed to

implement a graphical user interface for a universal testing machine (UTM), a machine that tests the tensile and compressive strength of materials. Other than tensile and compressive strength, the machine can also test transverse, bend, shear and Brinell hardness of materials. The graphical user interface was designed using the software Qt. The GUI application allows the use of the machine to be easier. Aside from control features, the researchers also designed the GUI application the show data results and store these results on a database. The database used by the researchers is SQLite, an open source database browser on Mozilla Firefox.

2.3 Theoretical Framework

Many theories debate whether the use of the graphical user interface (GUI) is ideal or not. Sibay (2013) said that it can be considered as a language, a set of symbols, and a form of communication between human and machine. It consists of graphic, pictorial symbols that represent commands. Those pictorial symbols serve as a universal language that can be understood by almost anyone. They are laid out in the form of menus, tables, or buttons, in a way that facilitates working the software efficiently. Thanks to this, most people find it extremely easy to operate certain software or applications without any prior training or instruction. Many people self-instruct themselves on how to operate certain programs through trial and error, by experimenting with different buttons do and associating them with occurring actions. Through experience and continuous usage, it becomes easy for users to identify and predict what commands each icon does. The symbols in a GUI can be thought of as “visual metaphors” that take the place of text

commands. We “read” those symbols like we do for words. We call the left-pointing arrow in an Internet browser the “back” button rather than the “arrow” button, and the looking glass button is universally interpreted as “search” rather than “looking glass”.

However, Marcas (2015) stated that using a command line also has its fair share of advantages. In almost all cases, the command line is much faster and easier to use for an expert, not for a beginner. The command line can also be used remotely with greater ease than GUIs; remote GUI's are sluggish. When absolute precision is necessary, a mouse/GUIs are too inaccurate. Lastly, for complex tasks, the command line can be scripted to automate things, which in this case, the electro-pneumatic trainer.

2.4 Conceptual Framework

The four major steps that the study will undergo are: first, to program and create the graphical user interface which is essential for the users to easily manipulate. Such manipulation is made easy through designing a comprehensive, and visual-friendly layout. Second, to be able to convert the manipulation of the graphical user interface into codes which are directly inputted to the microcontroller, Arduino Uno. Third, to design a step-up circuit with the use of the ULN2003. This amplifies the 5V output of the microcontroller into 24V which is the required amount of voltage that is needed to drive the PN-2320. Lastly, the implementation of the above-mentioned steps take place. This is where the PN-2320’s solenoid valves opens or closes which then reflects to the movements of the actuators. Figure 2.5 shows how the conceptual framework of this study can be mapped.

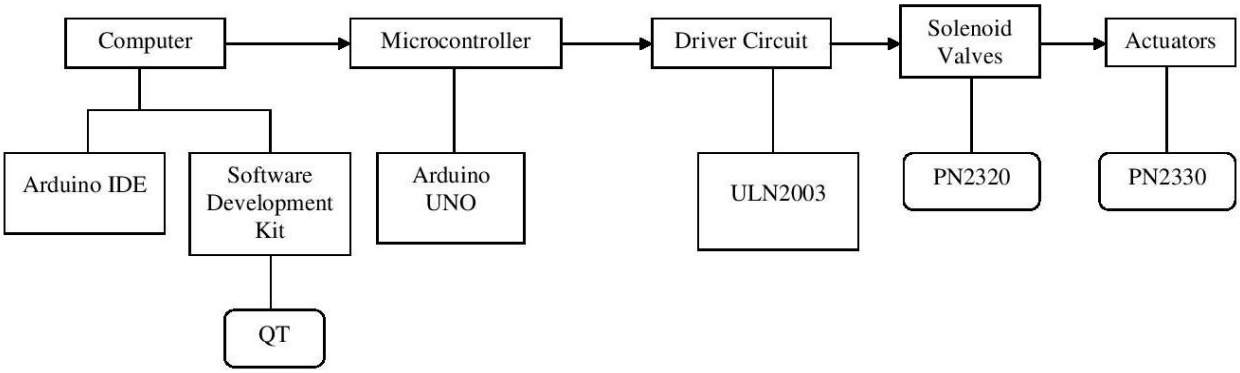


Fig. 2.5. *Conceptual Framework*

CHAPTER 3

METHODOLOGY

This chapter defines the steps taken to achieve a working project. It includes the block diagram of the project, materials and equipment, system flow, workflow, and a timetable showing how long each phase of the workflow takes time to do.

3.1 Block Diagram

The block diagram of this project, as shown in Figure 3.1, can be summed up into four phases. First, the computer allows the user to navigate the graphical user interface. The codes that correspond with the graphical user interface are then sent to the second phase of the block diagram which is the micro-controller. The micro-controller's output voltage is the amplified through the use of the driver circuit which is the 3rd phase. Lastly, the driver circuit is then connected to the electro-pneumatic trainer which consists of the solenoid valves, and the actuators.

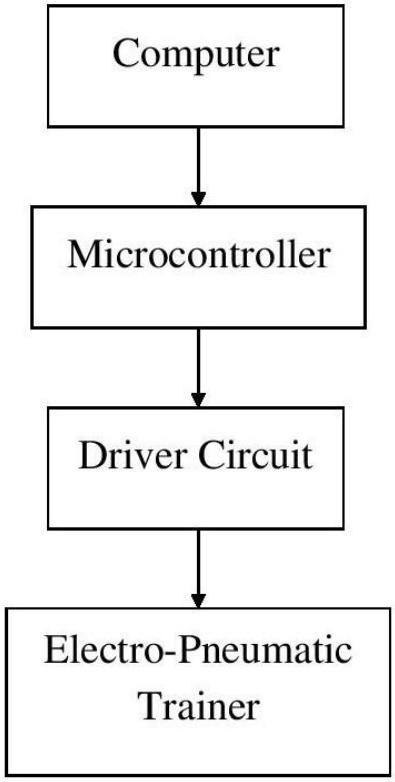


Fig. 3.1. *Block Diagram*

3.2 Materials and Equipment

3.2.1 Arduino Uno Microcontroller

Arduino Uno, as shown in Figure 3.2, is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery

to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again. "Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards. (Audino, n.d.)

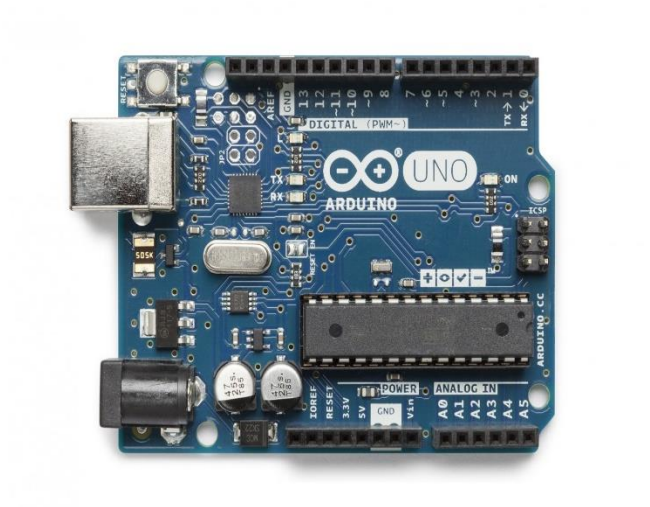


Fig. 3.2. *Arduino Uno Microcontroller*

3.2.2 ULN2003 Integrated Circuit

The ULN2003, as shown in Figure 3.3, is a monolithic IC consists of seven NPN darlington transistor pairs with high voltage and current capability. It is commonly used for applications such as relay drivers, motor, display drivers, led lamp drivers, logic

buffers, line drivers, hammer drivers and other high voltage current applications. It consists of common cathode clamp diodes for each NPN darlington pair which makes this driver IC useful for switching inductive loads. (George, 2004)

Fig. 3.3. *ULN2003 IC*

3.2.3 PN2320

In engineering terms, the solenoid describes transducer mechanisms used to convert energy into motion. Solenoid valves, shown in Figure 3.4, is the PN2320 which are to be used in this study, are controlled by the action of the solenoid and typically control the flow of water or air as a switch. If the solenoid is active (current is applied), it opens the valve. If the solenoid is inactive (current does not exist), the valve stays closed. The action of the pneumatic solenoid is controlled by the use of pneumatics. The opening or closing of a valve is referred to as “changing state.” (Burke, 2017)

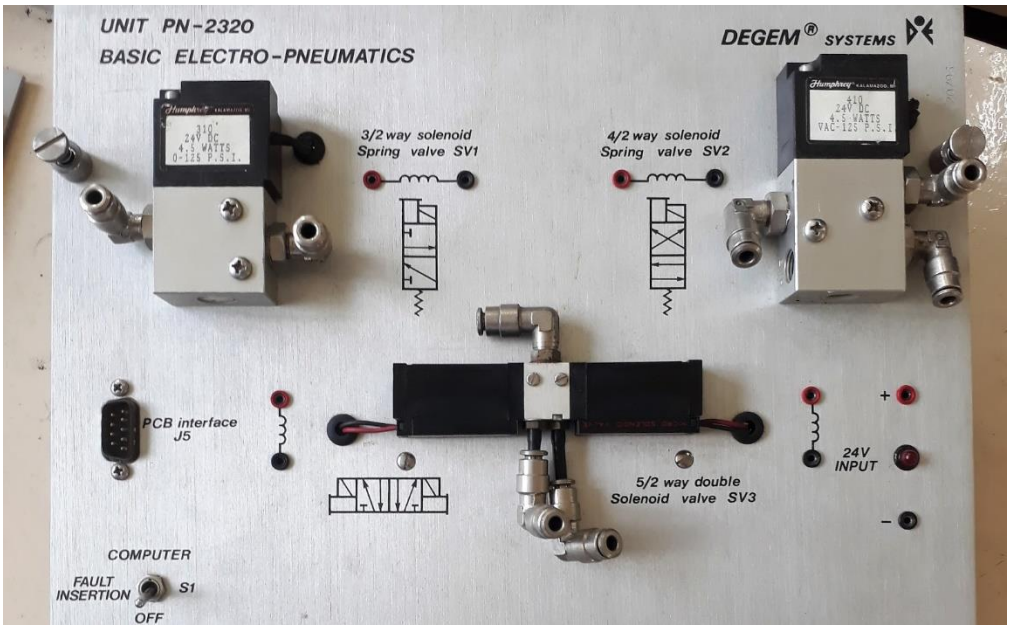


Fig. 3.4. *PN2320*

3.2.4 PN2330

As stated in Tech FAQ, pneumatic actuators, shown in Figure 3.5 is the PN2330 which are the actuators to be used in this study, are generally relatively simplistic and depend on their own ability to convert potential energy into kinetic energy. Pneumatic actuators usually consist of a cylinder or chamber in which regular air, a pressurized gas, or a mixture of the two, is contained and allowed to expand. As the gas expands, a pressure difference between the inside of the chamber and the natural atmospheric pressure causes the gas to build up energy. The gas is then allowed to leave the chamber in a controlled manner so that it is directed toward a piston, gear, or some other mechanical device. The piston is then used to perform the actual work to be done. Depending on how the gas is

directed toward the piston and how the actuator is designed, the piston can be driven in a straight line or in a circle.

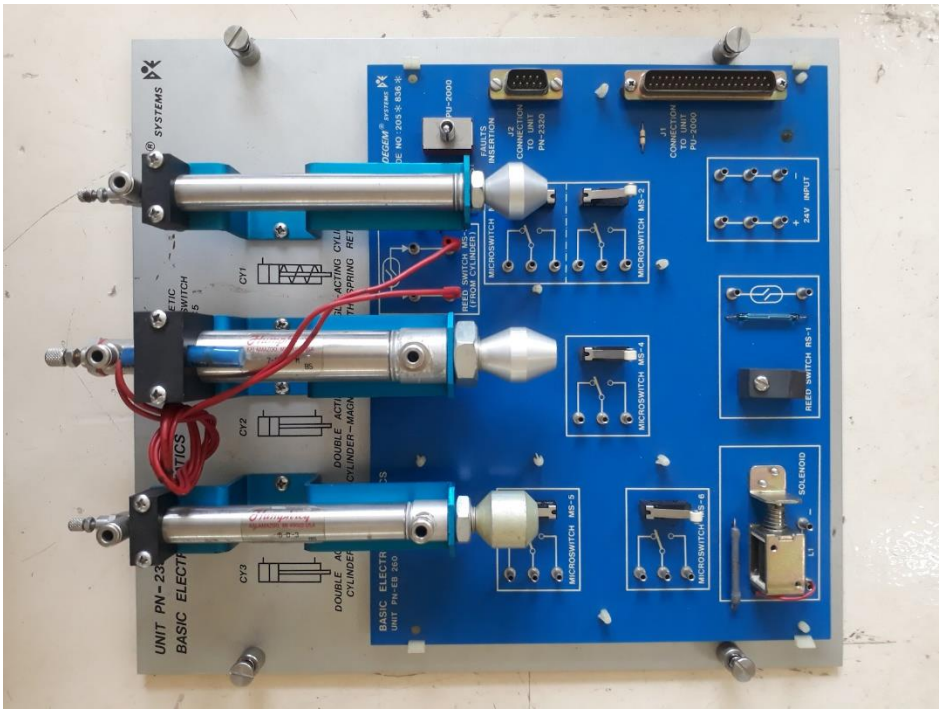


Fig. 3.5. *PN2330*

3.3 System Flow

The Figure 3.6 shows the system flow. When the user pushes down a button on the user interface, an Arduino code is uploaded to the Arduino microcontroller automatically. Using the code, the Arduino can control the operations of the trainer.

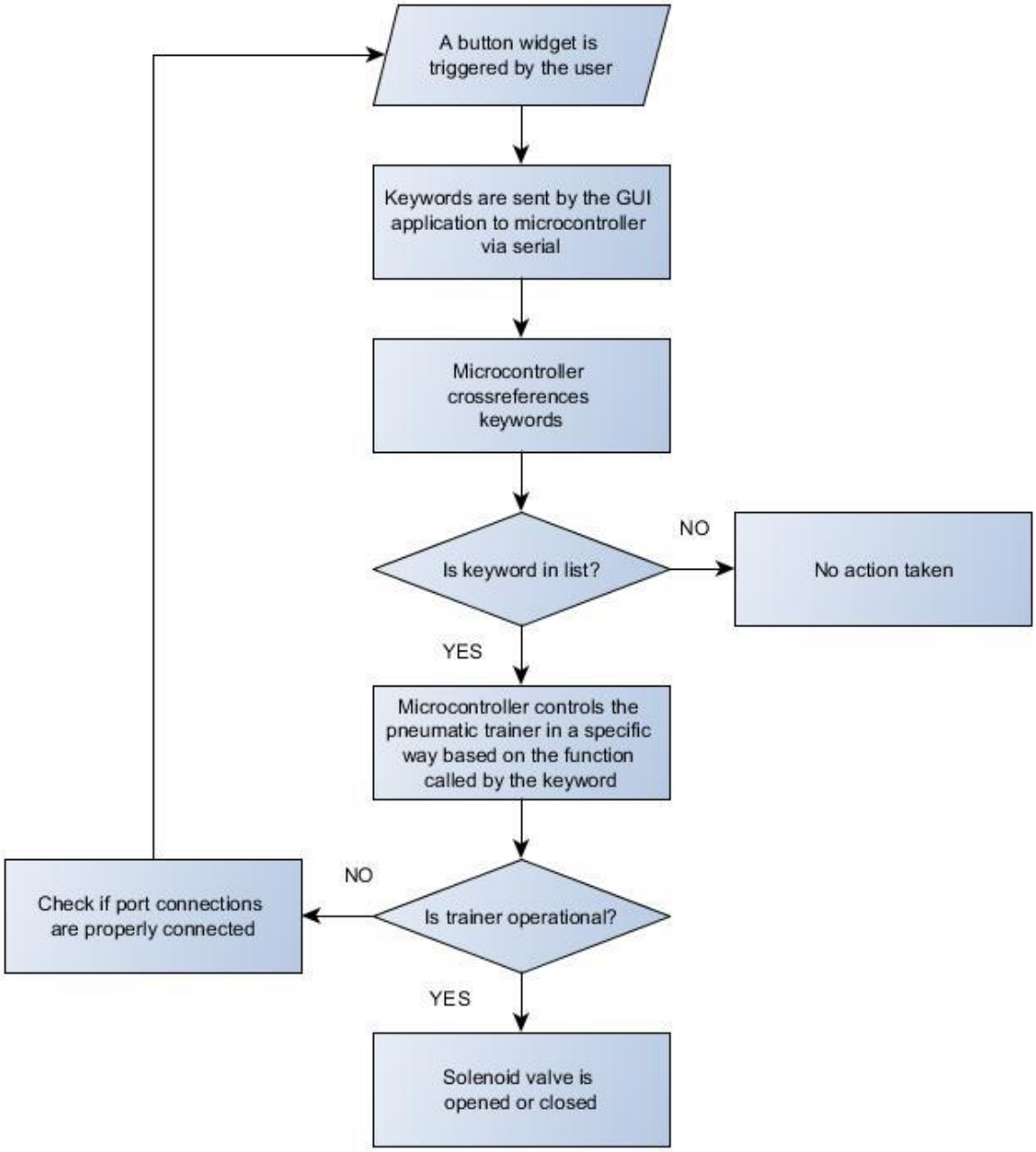


Figure 3.6. *System Flowchart*

3.4 Workflow

The workflow, as shown in Figure 3.7, shows the steps taken to achieve a working project. The first step taken is the procurement of the materials. When all materials have been collected, the researchers can start assembling the driver circuit and do preliminary tests to see if the circuit is working properly. When all hardware components of this project are fully functional, designing and development of the graphical user interface application can begin. Deployment is the last step taken, when the entire project is ready and fully functioning.

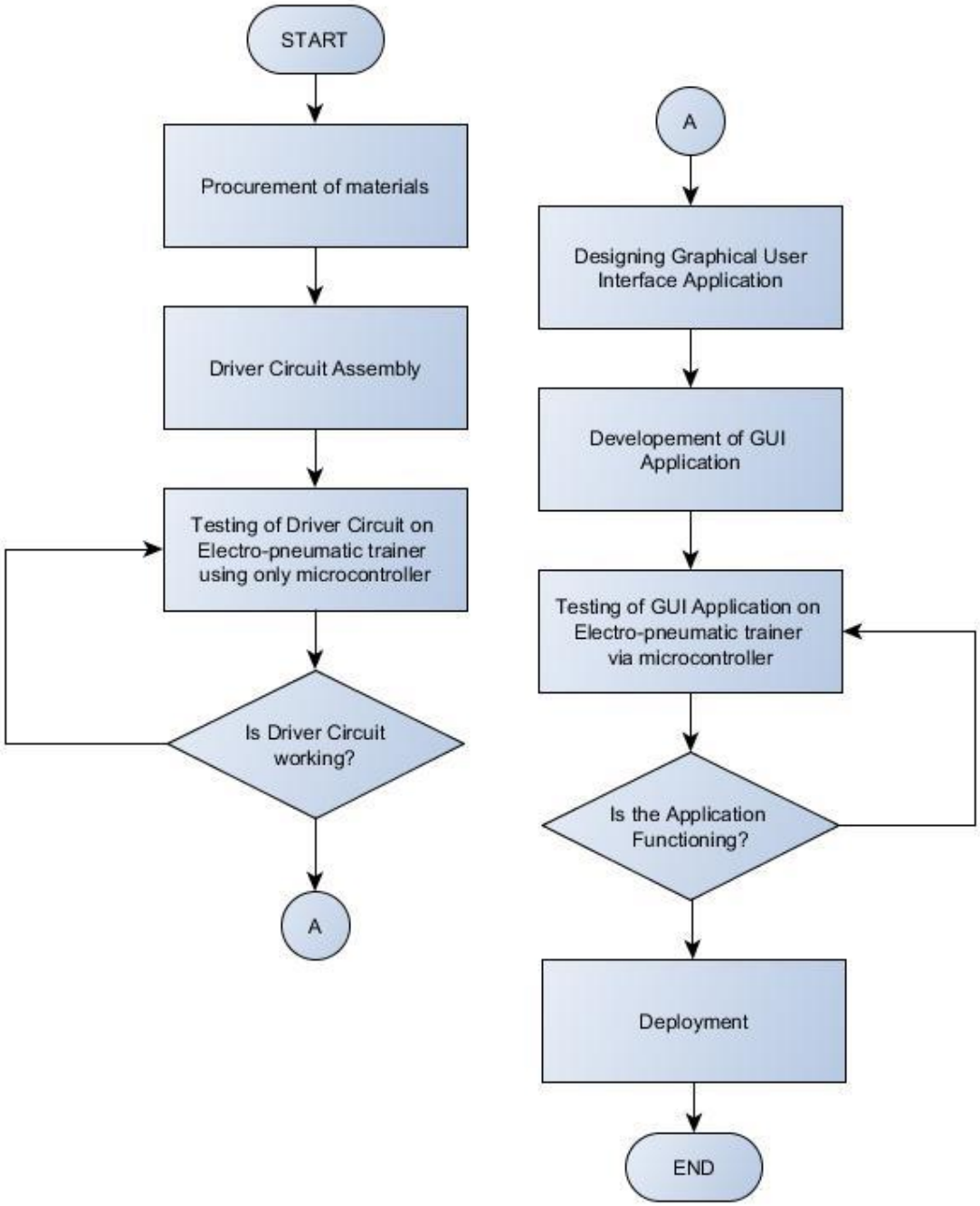


Figure 3.7. *Flowchart of Workflow*

3.5 Time Table

The time table, as shown below in Figure 3.8, shows the phases of the workflow and how long each one takes to finish. The driver circuit assembly and testing take around three weeks to complete. The designing, development and testing of the graphical user interface application needs eight weeks in total to have it fully functioning. And five weeks is needed for deployment and final inspections to ensure a working application that ideally has no bugs or errors.

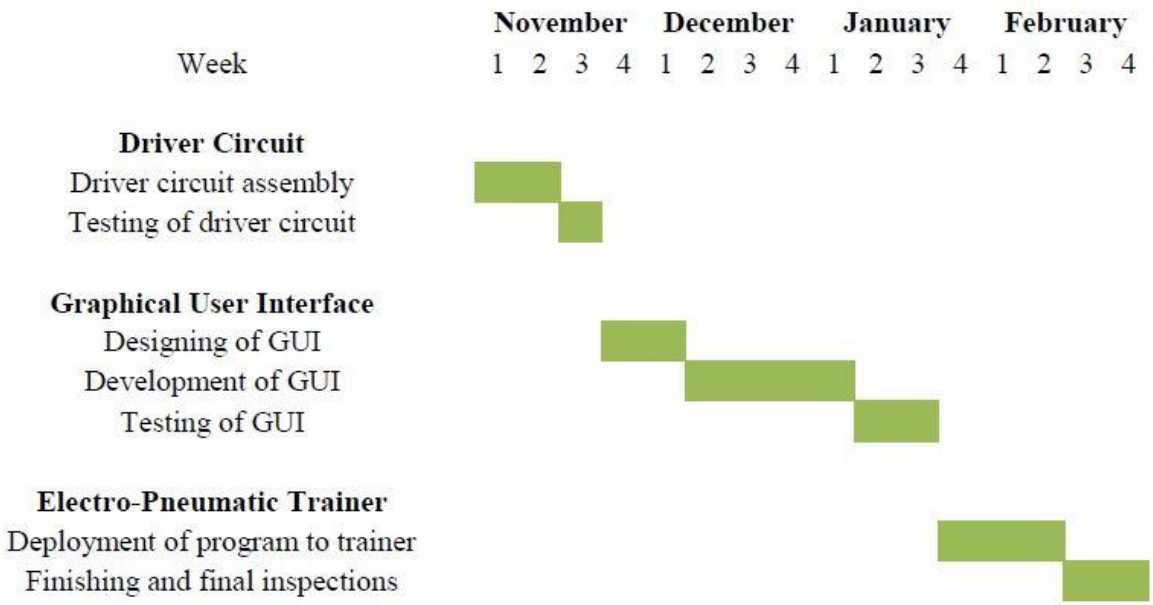


Figure 3.8. *Time Table*

CHAPTER 4

PRESENTATION, ANALYSIS, AND INTERPRETATION OF RESULTS

This chapter tackles the outcome of this project. It includes one of the most important aspects of this project which is the successful result from the experiment. The economic analysis is also being discussed in this chapter. Such analysis is needed in order to keep track of the monetary value of the system.

4.1 Results from Experiment

After several steps taken to conduct the experiment, the proponents are aware that it is possible to control the electro-pneumatic trainer through a graphical user interface (GUI) application. By using the ULN2003 driver circuit, as shown in Figure 4.1, with its corresponding schematic diagram, as shown in Figure 4.2, the Arduino Uno being controlled by the GUI is able to manipulate the valves that require higher loads. Instructions on how to set up the driver circuit for the pneumatic control can be found on the manual on Appendix D.

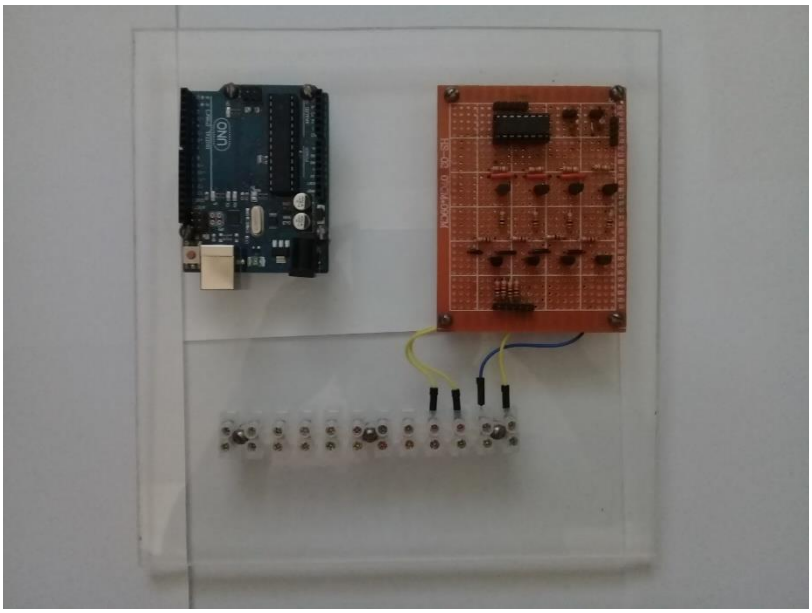


Figure 4.1. *Project Circuit*

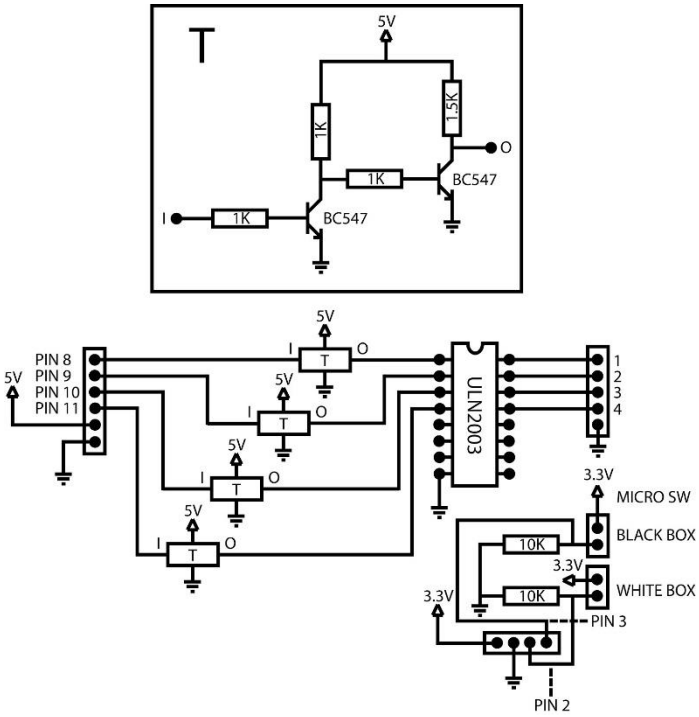


Figure 4.2. *Schematic Diagram*

The simple layout of the GUI, as shown in Figure 4.3, makes it easy for students to use. With a click of a button, the application will automatically send data through serial and control the microcontroller. For example, the Direct Control Push Buttons send data that controls the microcontroller to open a valve when a button is pressed down but sends data that prompts the microcontroller to close a valve when a button is released. Detailed descriptions of the other functions of the GUI can be seen on the manual on Appendix D.

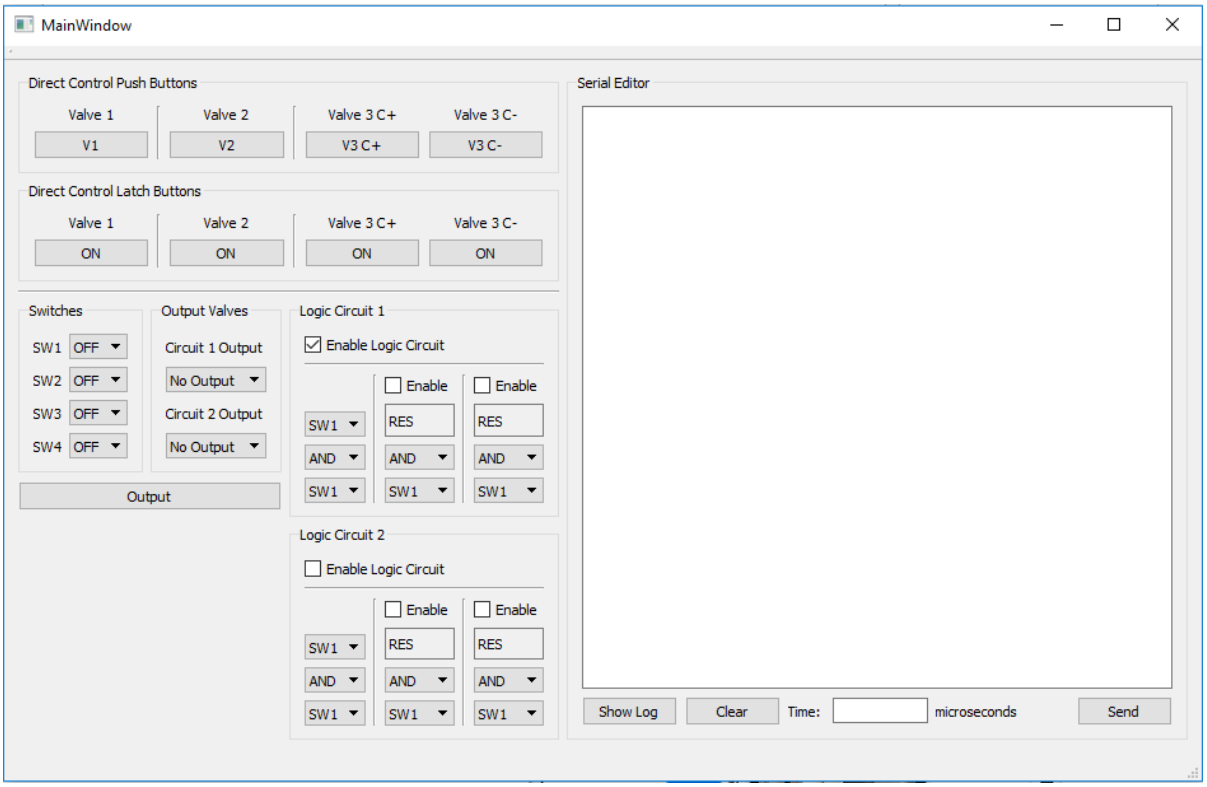


Figure 4.3. *Graphical User Interface Application*

The complete installation of the ULN2003 driver circuit, graphical user interface application and the electro-pneumatic trainer can be seen on figure 4.4 below. Steps on how to set-up the tubing for the solenoid valves and how to connect the wiring of the

PN2320 pneumatic control to the driver circuit can read on the manual attached on Appendix D.

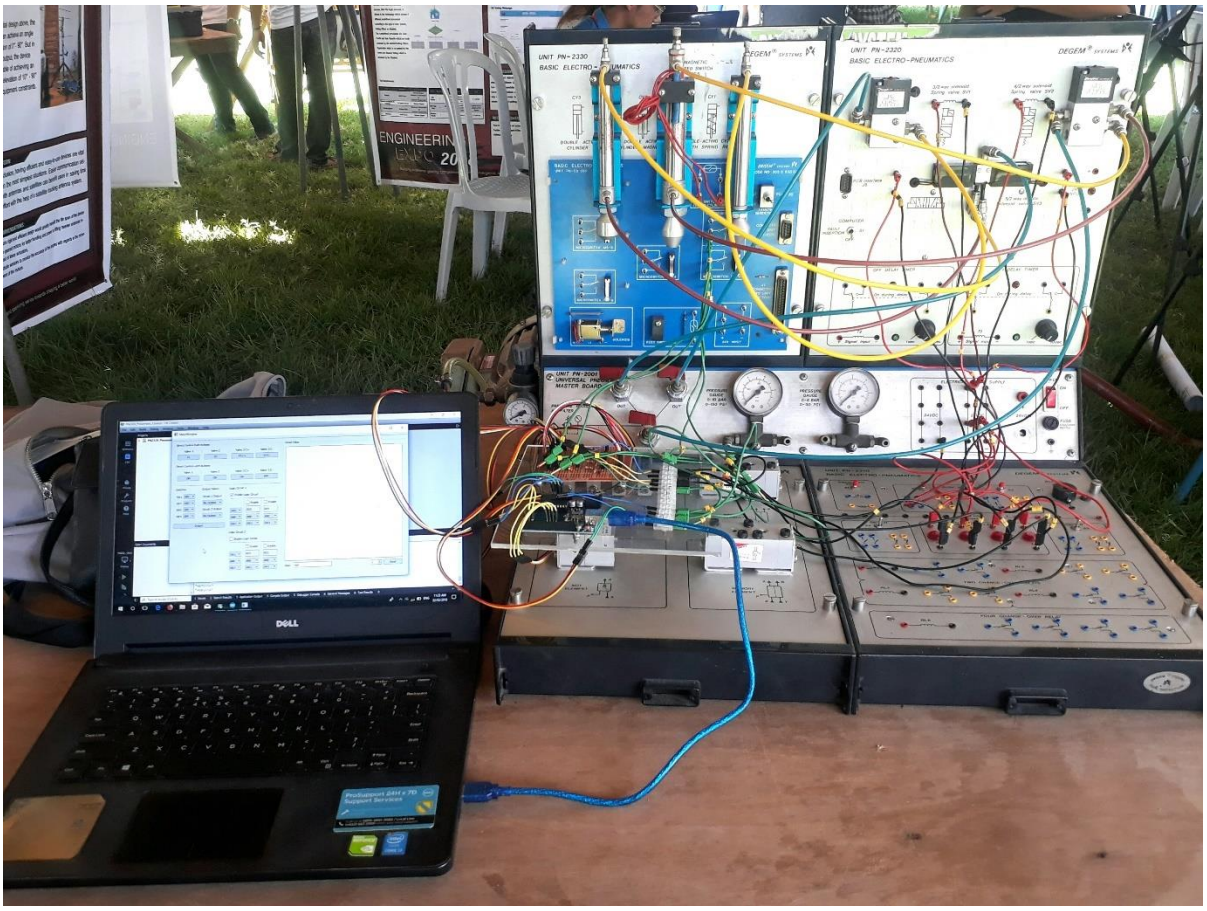


Figure 4.4. Complete Installation

4.2 Economic Analysis

The table below, as shown in Table 4.1, is the economic analysis of this project. In order to keep track of the project's monetary value, the economic analysis is needed. All of the materials needed for the creation of this system's hardware is being considered in the said table.

Name of Item	Quantity	Unit	Cost per Item (PH pesos)	Total (PH pesos)
BC547 NPN Transistor	8	Pieces	4.00	32.00
1 K ¼ W Resistor	12	Pieces	0.50	6.00
1.5 K ¼ W Resistor	4	Pieces	0.50	2.00
10 K ¼ W Resistor	10	Pieces	0.50	5.00
16-Pin IC Socket	1	Piece	3.00	3.00
Perfboard (9cm by 7cm)	1	Piece	18.00	18.00
Terminal Blocks	1	Strip	30.00	30.00
Male Pin Header	1	Strip	10.00	10.00
Female Pin Header	19	Pieces	8.00	152.00
Jumper	19	Pieces	8.00	152.00
Arduino Uno	1	Piece	500.00	500.00
Acrylic Glass	1	Piece	150.00	150.00
			Overall Total	**Php 1060.00**

Table 4.1. *Economic Data*

CHAPTER 5

SUMMARY, CONCLUSION AND RECOMMENDATION

The final chapter tackles the summary, conclusion and recommendation of this project. This is where the totality of the project is being summarized, and where the conclusion is able to meet the general objectives and the specific objectives. Furthermore, the recommendation is an aid for future researchers to improve this project.

5.1 Summary

As stated in the Chapter 1, the Mechanical Engineering students of Silliman University are required to study pneumatics using an electro-pneumatic trainer. However, operating the trainer through the microcontroller and observing its output is time consuming since the students have to develop a written code just for the trainer to perform a single action. To address this problem, the proponents created a graphical user interface application in order for the students to manipulate the electro-pneumatic trainer in an efficient and easier way. The graphical user interface (GUI) application allows immediate observation of the behaviours of the trainer with a press of a button on the layout of the user interface. Such functional GUI application would not have been possible without the construction of th ULN2003 driver circuit. The said circuit enables the Arduino Uno to control the higher voltage solenoid valves of the PN2320 Pneumatic Control.

5.2 Conclusion

The proponents conclude that it is possible to obtain the general objective stated in Chapter 1 which is to implement a graphical user interface (GUI) application that will act as an interpreter from simple widgets to lines of codes for a microcontroller that would operate the electro-pneumatic trainer. The GUI application contains button widgets for simple use and a text editor where users can write their own codes. Furthermore, the ULN2003 driver circuit is efficient in allowing the Arduino Uno microcontroller to manipulate the higher voltage solenoid valves of the PN2320 Pneumatic Control.

5.3 Recommendation

The proponents recommend to further enhance this project by being able to manipulate the PN2320 Pneumatic Control through a smart phone with a better graphical user interface application. In addition to this, the proponents recommend using Arduino Mega instead of Arduino Uno because it has six interrupt pins which are used for the micro-switches.

REFERENCES

Hayes, Frank and Baran, Nick. (1989). A Guide to GUIs. *Byte*, 4, 250-257.

Calcutt, D., Cowan, F., and Parchizadeh, H. (2004). *8051 Microcontrollers.* Oxford, UK: Elsevier

Bonsiepe, Gui. (1968). Interpretations of Human User Interface. *Visible Language.* 24, 262-285.

Harding, Bruce A. (1989). Windows & Icons & Mice, Oh My! The Changing Face of Computing. *Frontiers in Education Conference,* 337-342.

Jansen, B. J. (1998). The Graphical User Interface: An Introduction . *SIGCHI Bulletin*. 30(2), 22-26.

Arduino. (n.d.). Retrieved September 9, 2017 from https://store.arduino.cc/usa/arduino-uno-rev3

TC Trainers Co., Inc. (n.d.). Pneumatic Control. Retrieved September 9, 2017 from http://www.tctrainers.com/pneumatic_trainer.html

Sibay, Somaia. (2013). The Graphical User Interface. *CCTP74: Media Theory and Digital Culture.* Retrieved September 5, 2017 from https://blogs.commons.georgetown.edu/cctp-748-spring2013/2013/02/20/the-graphical-user-interface/

Marcas, Neal. (2015). What are the advantages and disadvantages of Command Line Interface?. *Quora*. Retrieved September 5, 2017 from https://www.quora.com/What-are-the-advantages-and-disadvantages-of-Command-Line-Interface

National Fluid Power Association. (2015). What is pneumatics?. Retrieved September 5, 2017 from http://www.nfpa.com/fluidpower/whatispneumatics.aspx

Dahl, O. (2014). What is a microcontroller?. Retrieved September 5, 2017 from https://www.build-electronic-circuits.com/what-is-a-microcontroller/

Computer Hope. (2017). GUI. Retrieved September 5, 2017 from https://www.computerhope.com/jargon/g/gui.htm

Burhan I, Azman A and Othman R. (2016). Electro pneumatic trainer embedded with programmable integrated circuit (PIC) microcontroller and graphical user interface platform for aviation industries training purposes. Retrieved September 18, 2017 from http://iopscience.iop.org/article/10.1088/1757-899X/152/1/012065/pdf

Halterman R. (2017). Fundamentals of C++ Programming. Retrieved September 18, 2017 from http://python.cs.southern.edu/cppbook/progcpp.pdf

Bailey, T. (n.d.). An Introduction to C Programming Language and Software Design. Retrieved September 18, 2017 from http://www-personal.acfr.usyd.edu.au/tbailey/ctext/ctext.pdf

Shivajirao G, Mane Y. (2017). Graphical User Interface for Universal Testing Machine Using Qt. Retrieved September 18, 2017 from https://www.irjet.net/archives/V4/i7/IRJET-V4I7359.pdf

SAP Engineers & Consultants. (2008). Electro Pneumatic Trainer. Retrieved September 18, 2017 from http://www.sapengineers.com/electro-pneumatic-trainer.html

Soft Robotics Toolkit. (n.d.). Introduction to Electro-pneumatics. Retrieved September 22, 2017 from https://softroboticstoolkit.com/low-cost-ep-circuit/introduction

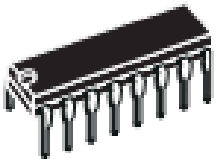
Kushugra. (n.d.). ULN 2003. *Engineers Garage.* Retrieved September 22, 2017 from https://www.engineersgarage.com/electronic-components/uln2003-datasheet

APPENDIX A

ULN 2003 DATASHEET

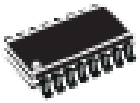
SEVEN DARLINGTON ARRAYS

- SEVEN DARLINGTONS PER PACKAGE
- OUTPUT CURRENT 500mA PER DRIVER (600mA PEAK)
- OUTPUT VOLTAGE 50V
- INTEGRATED SUPPRESSION DIODES FOR INDUCTIVE LOADS
- OUTPUTS CAN BE PARALLELED FOR HIGHER CURRENT
- TTL/CMOS/PMOS/DTL COMPATIBLE INPUTS
- INPUTS PINNED OPPOSITE OUTPUTS TO SIMPLIFY LAYOUT



DIP16

ORDERING NUMBERS: ULN2001A/2A/3A/4A



SO16

ORDERING NUMBERS: ULN2001D/2D/3D/4D

DESCRIPTION

The ULN2001A, ULN2002A, ULN2003 and ULN2004A are high voltage, high current darlington arrays each containing seven open collector darlington pairs with common emitters. Each channel rated at 500mA and can withstand peak currents of 600mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

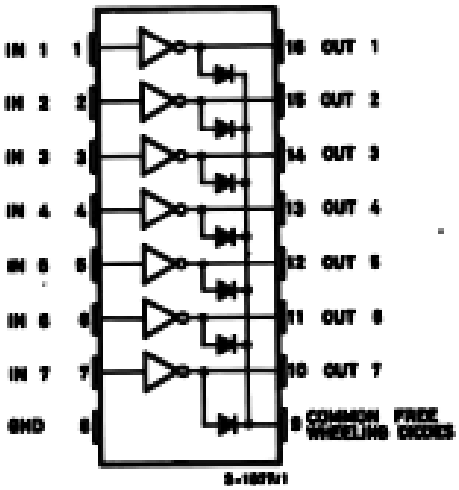
The four versions interface to all common logic families :

ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14-25V PMOS
ULN2003A	5V TTL, CMOS
ULN2004A	6–15V CMOS, PMOS

These versatile devices are useful for driving a wide range of loads including solenoids, relays DC motors, LED displays filament lamps, thermal printheads and high power buffers.

The ULN2001A/2002A/2003A and 2004A are supplied in 16 pin plastic DIP packages with a copper leadframe to reduce thermal resistance. They are available also in small outline package (SO-16) as ULN2001D/2002D/2003D/2004D.

PIN CONNECTION



ELECTRICAL CHARACTERISTICS (T_{amb} = 25°C unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit	Fig.
I_{CEX}	Output Leakage Current	V_{CE} = 50V T_{amb} = 70°C, V_{CE} = 50V T_{amb} = 70°C for ULN2002A V_{CE} = 50V, V_i = 6V for ULN2004A V_{CE} = 50V, V_i = 1V			50 100 500 500	µA µA µA µA	1a 1a 1b 1b
$V_{CE(sat)}$	Collector-emitter Saturation Voltage	I_C = 100mA, I_B = 250µA I_C = 200 mA, I_B = 350µA I_C = 350mA, I_B = 500µA		0.9 1.1 1.3	1.1 1.3 1.6	V V V	2 2 2
$I_{i(on)}$	Input Current	for ULN2002A, V_i = 17V for ULN2003A, V_i = 3.85V for ULN2004A, V_i = 5V V_i = 12V		0.82 0.93 0.35 1	1.25 1.35 0.5 1.45	mA mA mA mA	3 3 3 3
$I_{i(off)}$	Input Current	T_{amb} = 70°C, I_C = 500µA	50	65		µA	4
$V_{i(on)}$	Input Voltage	V_{CE} = 2V for ULN2002A I_C = 300mA for ULN2003A I_C = 200mA I_C = 250mA I_C = 300mA for ULN2004A I_C = 125mA I_C = 200mA I_C = 275mA I_C = 350mA			 13 2.4 2.7 3 5 6 7 8	V	5
h_{FE}	DC Forward Current Gain	for ULN2001A V_{CE} = 2V, I_C = 350mA	 1000				 2
C_i	Input Capacitance			15	25	pF	
t_{PLH}	Turn-on Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
t_{PHL}	Turn-off Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
I_R	Clamp Diode Leakage Current	V_R = 50V T_{amb} = 70°C, V_R = 50V			50 100	µA µA	6 6
V_F	Clamp Diode Forward Voltage	I_F = 350mA		1.7	2	V	7

APPENDIX B

APPLICATION SOURCE CODE

main.cpp 1

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <QTextDocument>
#include <QTextBlock>
#include <QFile>
#include <QTextStream>
#include <QDateTime>
#include "arduino.h"
#include "logicgatesimulation.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void readText();
    void readSerialData();
    void getSwitchState();
    int getCircuit1Result(int SW1, int SW2, int SW3, int SW4);
    int getCircuit2Result(int SW1, int SW2, int SW3, int SW4);
    void outputCircuitResults(int output1, int output2);
    void writeFile(QString filename, QString value);
    void readFile(QString filename);

private slots:
    void on_directButton1_clicked();
    void on_directButton2_clicked();
    void on_directButton3Pos_clicked();
    void on_directButton3Neg_clicked();
    void on_send_clicked();
    void on_outputButton_clicked();
    void on_pushButton1_pressed();
    void on_pushButton1_released();
    void on_pushButton2_pressed();
    void on_pushButton2_released();
    void on_pushButton3Pos_pressed();
    void on_pushButton3Pos_released();
    void on_pushButton3Neg_pressed();
    void on_pushButton3Neg_released();
    void on_logButton_clicked();

    void on_pushButton_clicked();

private:
```

```
    Ui::MainWindow *ui;
    arduino *microcontroller;
    logicGateSimulation *gate;
    QString mFilename;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    mFilename = "log.txt";

    microcontroller = new arduino;
    gate = new logicGateSimulation;

    microcontroller->setArduino();

    connect(microcontroller->microcontroller, &QSerialPort::readyRead, this,
&MainWindow::readSerialData);
}

MainWindow::~MainWindow()
{
    microcontroller->closeArduino();
    delete ui;
}

void MainWindow::readText()
{
    QTextDocument *doc = ui->plainTextEdit->document();
    int line = doc->lineCount();
    //qDebug() << s << line;

    for(int i = 0; i < line; i++) {
        QTextBlock tb = doc->findBlockByLineNumber(i);
        QString text = tb.text();
        text = text.replace(" ", "");

        if(text != "") {
            qDebug() << text;
            microcontroller->updateArduino(text);
            Sleep(uint(200));
        }
    }
}

void MainWindow::readSerialData()
{
    QByteArray serialData = microcontroller->microcontroller->readAll();
    QString temp = QString::fromStdString(serialData.toStdString());
    qDebug() << temp;
    ui->timeValue->setText(temp);
    writeFile(mFilename, temp);
}
```

```
void MainWindow::getSwitchState()
{
    int SW1 = 0, SW2 = 0, SW3 = 0, SW4 = 0;
    int result1 = 0, result2 = 0;

    if (ui->comboBoxSW1->currentText() == "ON")
        SW1 = 1;
    if (ui->comboBoxSW2->currentText() == "ON")
        SW2 = 1;
    if (ui->comboBoxSW3->currentText() == "ON")
        SW3 = 1;
    if (ui->comboBoxSW4->currentText() == "ON")
        SW4 = 1;

    if(ui->checkBoxCircuit1->isChecked())
        result1 = getCircuit1Result(SW1, SW2, SW3, SW4);
    if(ui->checkBoxCircuit2->isChecked())
        result2 = getCircuit2Result(SW1, SW2, SW3, SW4);

    outputCircuitResults(result1, result2);
}

int MainWindow::getCircuit1Result(int SW1, int SW2, int SW3, int SW4)
{
    int input1=0, input2=0, result=0;
    qDebug() << SW1 << SW2 << SW3 << SW4;

    if(ui->comboBoxUpper1->currentText() == "SW1")
        input1 = SW1;
    else if(ui->comboBoxUpper1->currentText() == "SW2")
        input1 = SW2;
    else if(ui->comboBoxUpper1->currentText() == "SW3")
        input1 = SW3;
    else if(ui->comboBoxUpper1->currentText() == "SW4")
        input1 = SW4;

    if(ui->comboBoxLower1->currentText() == "SW1")
        input2 = SW1;
    else if(ui->comboBoxLower1->currentText() == "SW2")
        input2 = SW2;
    else if(ui->comboBoxLower1->currentText() == "SW3")
        input2 = SW3;
    else if(ui->comboBoxLower1->currentText() == "SW4")
        input2 = SW4;

    if(ui->comboBoxGate1->currentText() == "AND")
        result = gate->logicGateAND(input1, input2);
    else if(ui->comboBoxGate1->currentText() == "OR")
        result = gate->logicGateOR(input1, input2);

    if(ui->checkBoxCon1->isChecked()) {
        if(ui->comboBoxLower2->currentText() == "SW1")
```

```
            input2 = SW1;
        else if(ui->comboBoxLower2->currentText() == "SW2")
            input2 = SW2;
        else if(ui->comboBoxLower2->currentText() == "SW3")
            input2 = SW3;
        else if(ui->comboBoxLower2->currentText() == "SW4")
            input2 = SW4;

        if(ui->comboBoxGate2->currentText() == "AND")
            result = gate->logicGateAND(result, input2);
        else if(ui->comboBoxGate2->currentText() == "OR")
            result = gate->logicGateOR(result, input2);
    }

    if(ui->checkBoxCon2->isChecked()) {
        if(ui->comboBoxLower3->currentText() == "SW1")
            input2 = SW1;
        else if(ui->comboBoxLower3->currentText() == "SW2")
            input2 = SW2;
        else if(ui->comboBoxLower3->currentText() == "SW3")
            input2 = SW3;
        else if(ui->comboBoxLower3->currentText() == "SW4")
            input2 = SW4;

        if(ui->comboBoxGate3->currentText() == "AND")
            result = gate->logicGateAND(result, input2);
        else if(ui->comboBoxGate3->currentText() == "OR")
            result = gate->logicGateOR(result, input2);
    }

    qDebug() << "result 1 is" << result;
    return result;
}

int MainWindow::getCircuit2Result(int SW1, int SW2, int SW3, int SW4)
{
    int input1=0, input2=0, result=0;

    if(ui->comboBoxUpper1_2->currentText() == "SW1")
        input1 = SW1;
    else if(ui->comboBoxUpper1_2->currentText() == "SW2")
        input1 = SW2;
    else if(ui->comboBoxUpper1_2->currentText() == "SW3")
        input1 = SW3;
    else if(ui->comboBoxUpper1_2->currentText() == "SW4")
        input1 = SW4;

    if(ui->comboBoxLower1_2->currentText() == "SW1")
        input2 = SW1;
    else if(ui->comboBoxLower1_2->currentText() == "SW2")
        input2 = SW2;
    else if(ui->comboBoxLower1_2->currentText() == "SW3")
        input2 = SW3;
```

```
    else if(ui->comboBoxLower1_2->currentText() == "SW4")
        input2 = SW4;

    if(ui->comboBoxGate1_2->currentText() == "AND")
        result = gate->logicGateAND(input1, input2);
    else if(ui->comboBoxGate1_2->currentText() == "OR")
        result = gate->logicGateOR(input1, input2);

    if(ui->checkBoxCon1_2->isChecked()) {
        if(ui->comboBoxLower2_2->currentText() == "SW1")
            input2 = SW1;
        else if(ui->comboBoxLower2_2->currentText() == "SW2")
            input2 = SW2;
        else if(ui->comboBoxLower2_2->currentText() == "SW3")
            input2 = SW3;
        else if(ui->comboBoxLower2_2->currentText() == "SW4")
            input2 = SW4;

        if(ui->comboBoxGate2_2->currentText() == "AND")
            result = gate->logicGateAND(result, input2);
        else if(ui->comboBoxGate2_2->currentText() == "OR")
            result = gate->logicGateOR(result, input2);
    }

    if(ui->checkBoxCon2_2->isChecked()) {
        if(ui->comboBoxLower3_2->currentText() == "SW1")
            input2 = SW1;
        else if(ui->comboBoxLower3_2->currentText() == "SW2")
            input2 = SW2;
        else if(ui->comboBoxLower3_2->currentText() == "SW3")
            input2 = SW3;
        else if(ui->comboBoxLower3_2->currentText() == "SW4")
            input2 = SW4;

        if(ui->comboBoxGate3_2->currentText() == "AND")
            result = gate->logicGateAND(result, input2);
        else if(ui->comboBoxGate3_2->currentText() == "OR")
            result = gate->logicGateOR(result, input2);
    }

    qDebug() << "result 2 is" << result;
    return result;
}

void MainWindow::outputCircuitResults(int output1, int output2)
{
    QString value1, value2;

    if (output1 == 1) {
        if (ui->comboBoxOutput1->currentText() == "Valve 1")
            value1 = "valve1,1";
        else if (ui->comboBoxOutput1->currentText() == "Valve 2")
            value1 = "valve2,1";
```

```
        else if (ui->comboBoxOutput1->currentText() == "Valve 3 C+")
            value1 = "valve3Pos,1";
        else if (ui->comboBoxOutput1->currentText() == "Valve 3 C-")
            value1 = "valve3Neg,1";
    } else {
        if (ui->comboBoxOutput1->currentText() == "Valve 1")
            value1 = "valve1,0";
        else if (ui->comboBoxOutput1->currentText() == "Valve 2")
            value1 = "valve2,0";
        else if (ui->comboBoxOutput1->currentText() == "Valve 3 C+")
            value1 = "valve3Pos,0";
        else if (ui->comboBoxOutput1->currentText() == "Valve 3 C-")
            value1 = "valve3Neg,0";
    } microcontroller->updateArduino(value1);

    if (output2 == 1) {
        if (ui->comboBoxOutput2->currentText() == "Valve 1")
            value2 = "valve1,1";
        else if (ui->comboBoxOutput2->currentText() == "Valve 2")
            value2 = "valve2,1";
        else if (ui->comboBoxOutput2->currentText() == "Valve 3 C+")
            value2 = "valve3Pos,1";
        else if (ui->comboBoxOutput2->currentText() == "Valve 3 C-")
            value2 = "valve3Neg,1";
    } else {
        if (ui->comboBoxOutput2->currentText() == "Valve 1")
            value2 = "valve1,0";
        else if (ui->comboBoxOutput2->currentText() == "Valve 2")
            value2 = "valve2,0";
        else if (ui->comboBoxOutput2->currentText() == "Valve 3 C+")
            value2 = "valve3Pos,0";
        else if (ui->comboBoxOutput2->currentText() == "Valve 3 C-")
            value2 = "valve3Neg,0";
    } microcontroller->updateArduino(value2);
}

void MainWindow::writeFile(QString filename, QString value)
{
    QFile file(filename);

    if(!file.open(QIODevice::WriteOnly | QIODevice::Append))
        qDebug() << "could not write to file";

    QTime time = QTime::currentTime();
    QString timeTxt = time.toString("hh : mm : ss");

    QDate date = QDate::currentDate();
    QString dateTxt = date.toString("ddd, MMM d yyyy");

    value = value.replace("\r\n", "");

    QTextStream out(&file);
    out << "\r" << dateTxt << "\t\t" << timeTxt << "\t\t" << value << "\r" <<
```

```
endl;
    file.flush();
    file.close();
}

void MainWindow::readFile(QString filename)
{
    QFile file(filename);

    if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
        qDebug() << "could not read file";

    QTextStream in(&file);
    ui->plainTextEdit->setPlainText(in.readAll());
    file.close();
}

void MainWindow::on_directButton1_clicked()
{
    if (ui->directButton1->text() == "ON") {
        ui->directButton1->setText("OFF");
        QString value = "valve1,1";
        microcontroller->updateArduino(value);
    } else if (ui->directButton1->text() == "OFF") {
        ui->directButton1->setText("ON");
        QString value = "valve1,0";
        microcontroller->updateArduino(value);
    }
}

void MainWindow::on_directButton2_clicked()
{
    if (ui->directButton2->text() == "ON") {
        ui->directButton2->setText("OFF");
        QString value = "valve2,1";
        microcontroller->updateArduino(value);
    } else if (ui->directButton2->text() == "OFF") {
        ui->directButton2->setText("ON");
        QString value = "valve2,0";
        microcontroller->updateArduino(value);
    }
}

void MainWindow::on_directButton3Pos_clicked()
{
    if (ui->directButton3Pos->text() == "ON") {
        if(ui->directButton3Neg->text() == "OFF")
            on_directButton3Neg_clicked();

        ui->directButton3Pos->setText("OFF");
        QString value = "valve3Pos,1";
        microcontroller->updateArduino(value);
    } else if (ui->directButton3Pos->text() == "OFF") {
```

```
        ui->directButton3Pos->setText("ON");
        QString value = "valve3Pos,0";
        microcontroller->updateArduino(value);
    }
}

void MainWindow::on_directButton3Neg_clicked()
{
    if (ui->directButton3Neg->text() == "ON") {
        if(ui->directButton3Pos->text() == "OFF")
            on_directButton3Pos_clicked();

        ui->directButton3Neg->setText("OFF");
        QString value = "valve3Neg,1";
        microcontroller->updateArduino(value);
    } else if (ui->directButton3Neg->text() == "OFF") {
        ui->directButton3Neg->setText("ON");
        QString value = "valve3Neg,0";
        microcontroller->updateArduino(value);
    }
}

void MainWindow::on_send_clicked()
{
    readText();
}

void MainWindow::on_outputButton_clicked()
{
    getSwitchState();
}

void MainWindow::on_pushButton1_pressed()
{
    QString value = "valve1,1";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton1_released()
{
    QString value = "valve1,0";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton2_pressed()
{
    QString value = "valve2,1";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton2_released()
{
    QString value = "valve2,0";
```

```
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton3Pos_pressed()
{
    QString value = "valve3Pos,1";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton3Pos_released()
{
    QString value = "valve3Pos,0";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton3Neg_pressed()
{
    QString value = "valve3Neg,1";
    microcontroller->updateArduino(value);
}

void MainWindow::on_pushButton3Neg_released()
{
    QString value = "valve3Neg,0";
    microcontroller->updateArduino(value);
}

void MainWindow::on_logButton_clicked()
{
    readFile(mFilename);
}

void MainWindow::on_pushButton_clicked()
{
    ui->plainTextEdit->clear();
}
```

```
#ifndef LOGICGATESIMULATION_H
#define LOGICGATESIMULATION_H

#include <QWidget>

namespace Ui {
class logicGateSimulation;
}

class logicGateSimulation : public QWidget
{
    Q_OBJECT

public:
    explicit logicGateSimulation(QWidget *parent = 0);
    ~logicGateSimulation();
    int logicGateAND(int input1, int input2);
    int logicGateOR(int input1, int input2);

private:
    Ui::logicGateSimulation *ui;
};

#endif // LOGICGATESIMULATION_H
```

```
#ifndef ARDUINO_H
#define ARDUINO_H

#include <QWidget>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QtWidgets>

namespace Ui {
class arduino;
}

class arduino : public QWidget
{
    Q_OBJECT

public:
    explicit arduino(QWidget *parent = 0);
    ~arduino();
    void setArduino();
    void closeArduino();
    void updateArduino(QString);
    QSerialPort *microcontroller;

private:
    Ui::arduino *ui;
    static const quint16 arduinoVendorId = 9025;
    static const quint16 arduinoProductId = 67;
    QString arduinoPortName;
    bool arduinoIsAvailable;
};

#endif // ARDUINO_H
```

```
#include "arduino.h"
#include "ui_arduino.h"

arduino::arduino(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::arduino)
{
    ui->setupUi(this);
}

arduino::~arduino()
{
    delete ui;
}

void arduino::setArduino()
{
    microcontroller = new QSerialPort;
    arduinoPortName = "";
    arduinoIsAvailable = false;

    foreach (const QSerialPortInfo &serialPortInfo,
QSerialPortInfo::availablePorts()) {
        if (serialPortInfo.hasVendorIdentifier() &&
serialPortInfo.hasProductIdentifier()) {
            if (serialPortInfo.vendorIdentifier() == arduinoVendorId) {
                if (serialPortInfo.productIdentifier() == arduinoProductId) {
                    arduinoPortName = serialPortInfo.portName();
                    arduinoIsAvailable = true;
                }
            }
        }
    }

    if (arduinoIsAvailable) {
        microcontroller->setPortName(arduinoPortName);
        microcontroller->open(QSerialPort::ReadWrite);
        microcontroller->setBaudRate(QSerialPort::Baud115200);
        microcontroller->setDataBits(QSerialPort::Data8);
        microcontroller->setParity(QSerialPort::NoParity);
        microcontroller->setStopBits(QSerialPort::OneStop);
        microcontroller->setFlowControl(QSerialPort::NoFlowControl);
        qDebug() << "Connected";
    } else {
        qDebug() << "Not Connected";
    }
}

void arduino::closeArduino()
{
    if (microcontroller->isOpen()) {
        microcontroller->close();
        qDebug() << "Arduino Closed";
```

```
    }
}

void arduino::updateArduino(QString command)
{
    if (microcontroller->isWritable()) {
        microcontroller->write(command.toStdString().c_str());
    } else {
        qDebug() << "Could Not Write";
    }
}
```

APPENDIX C

ARDUINO SOURCE CODE

```
int valve1 = 8;
int valve2 = 9;
int valve3Pos = 10;
int valve3Neg = 11;

char val_byte = 0;
String val_str = "";
int val_num = 0;

volatile unsigned long time1;
volatile unsigned long time2;

volatile boolean valid1 = false;
volatile boolean valid2 = false;

void setup() {
  // put your setup code here, to run once:
  pinMode (valve1, OUTPUT);
  pinMode (valve2, OUTPUT);
  pinMode (valve3Pos, OUTPUT);
  pinMode (valve3Neg, OUTPUT);

 attachInterrupt(0,interrupt1,CHANGE);
 attachInterrupt(1,interrupt2,CHANGE);

  Serial.begin (115200);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (valid1 && valid2) {
      Serial.println(time2 - time1);
      noInterrupts();
      valid1 = valid2 = false;
      interrupts();
  }

  if (Serial.available() > 0) {
    val_byte = Serial.read();
    //val_num = Serial.parseInt();

    if (val_byte != ',') {
      val_str += val_byte;
    } else {
      val_num = Serial.parseInt();

      if (val_str == "delay") {
        delayTime(val_num);
      } else {
```

```
        switching(val_str, val_num);
      }

      val_str = "";
      val_num = 0;
    }
  }
}

void interrupt1() {
    time1 = millis();
    valid1 = true;
}

void interrupt2() {
  if (!valid2 && valid1) {
    time2 = millis();
    valid2 = true;
  }
}

void switching(String str, int num) {
  if (num == 1) {
    switchOn(str);
  } else if (num == 0) {
    switchOff(str);
  } else {
    //Serial.print("Invalid Input");
  }
}

void switchOn(String str) {
  if (str == "valve1") {
    digitalWrite(valve1, HIGH);
  } else if (str == "valve2") {
    digitalWrite(valve2, HIGH);
  } else if (str == "valve3Pos") {
    digitalWrite(valve3Pos, HIGH);
  } else if (str == "valve3Neg") {
    digitalWrite(valve3Neg, HIGH);
  } else {
    //do nothing
  }
}

void switchOff(String str) {
  if (str == "valve1") {
    digitalWrite(valve1, LOW);
  } else if (str == "valve2") {
```

```
    digitalWrite(valve2, LOW);
  } else if (str == "valve3Pos") {
    digitalWrite(valve3Pos, LOW);
  } else if (str == "valve3Neg") {
    digitalWrite(valve3Neg, LOW);
  } else {
    //do nothing
  }
}

void delayTime(int num) {
  delay(num);
}
```

APPENDIX D

MANUAL

Arduino-Based Computer Interface

for the PN-2320 Pneumatic-Control Manual

Prepared by:

Sinco, Micah Jude Kabristante

Villamor, Alfonzo Miguel Maxino

HARDWARE

Plastic Tubing Connection

1. Connect valve openings labeled with white circles to air pressure source as shown in figure 1.1.

2. Connect valve openings labeled with colors other than white, as shown in figure 1.1, to the actuator openings with the same color labeling as shown in figure 1.2.

3. Check if tubings as securely connected.

Wiring Connection (Pneumatic-Control)

1. Connect all positive inputs of all solenoid valves to the positive output of the 24V power supply.

2. Connect negative input of solenoid valve 1 labeled 1, as shown in figure 1.2, to the pin labeled 1 on the driver circuit as shown on figure 1.3 on the farthest right of the figure.

3. Do the same for the other three negative inputs labeled with numbers.

4. Connect ground pin of the driver circuit, as shown in figure 1.3 on the farthest right, to the ground input of the 24V power supply.

5. Connect micro-switches labeled with black and white boxes, as shown in figure 1.2, to the pins on the driver circuit with the same color labels as shown in figure 1.3.

Wiring Connection (Arduino)

1. Connect all pins labeled with voltage inputs, as shown in figure 1.3, to the respective voltage input pins on the Arduino.

2. Connect all pins labeled with ground inputs, as shown in figure 1.3, to the ground input pins on the Arduino.

3. Connect pin 2 and pin 3 on the driver circuit, as shown in figure 1.3, to the respective pins on the Arduino with the same pin number.

4. Connect pins 8, 9, 10 and 11 on the driver circuit, as shown in figure 1.3, to the respective pins on the Arduino with the same pin number.

5. Check for errors in the connection.

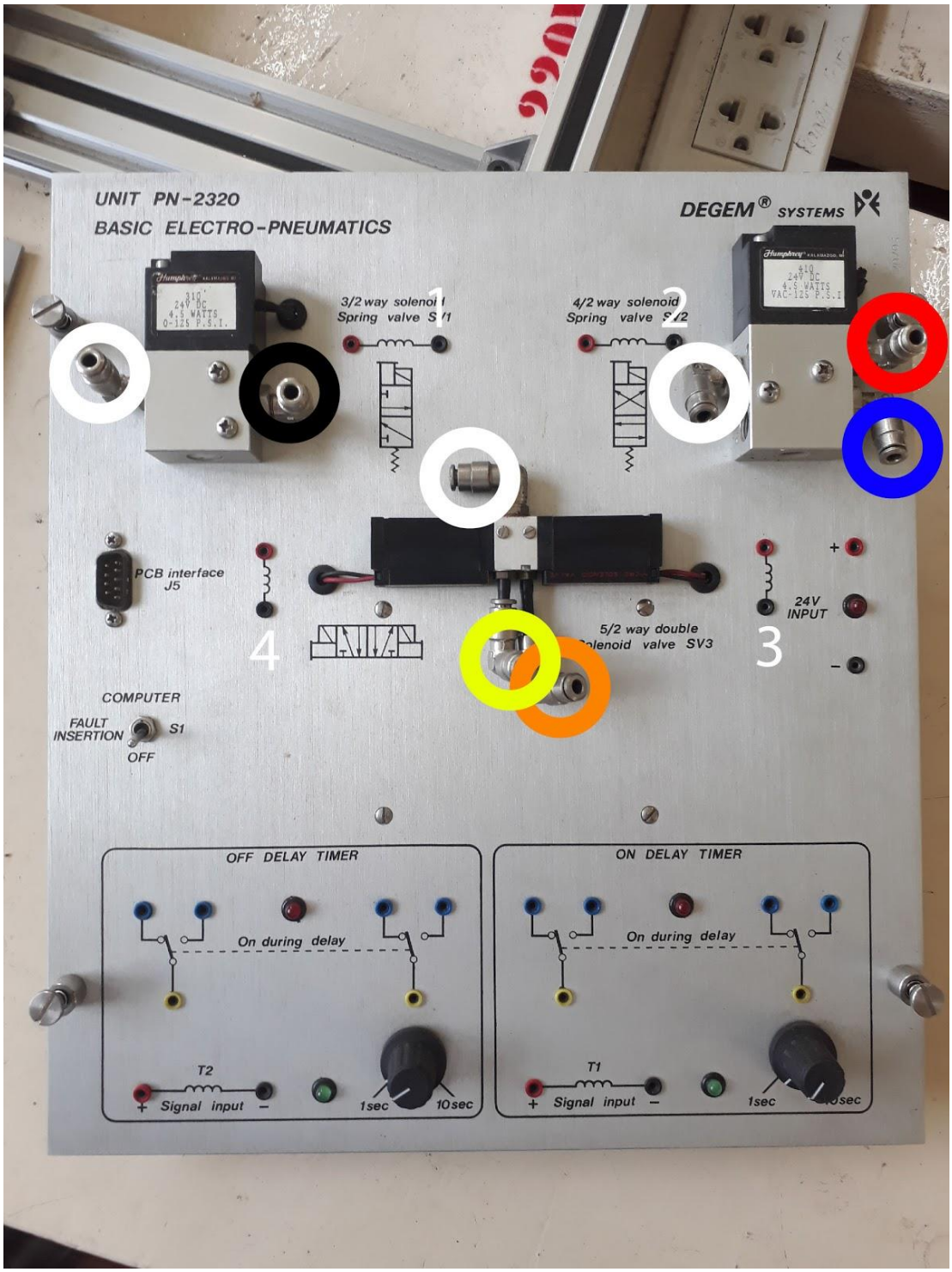


Figure 1.1. *Solenoid Valves*

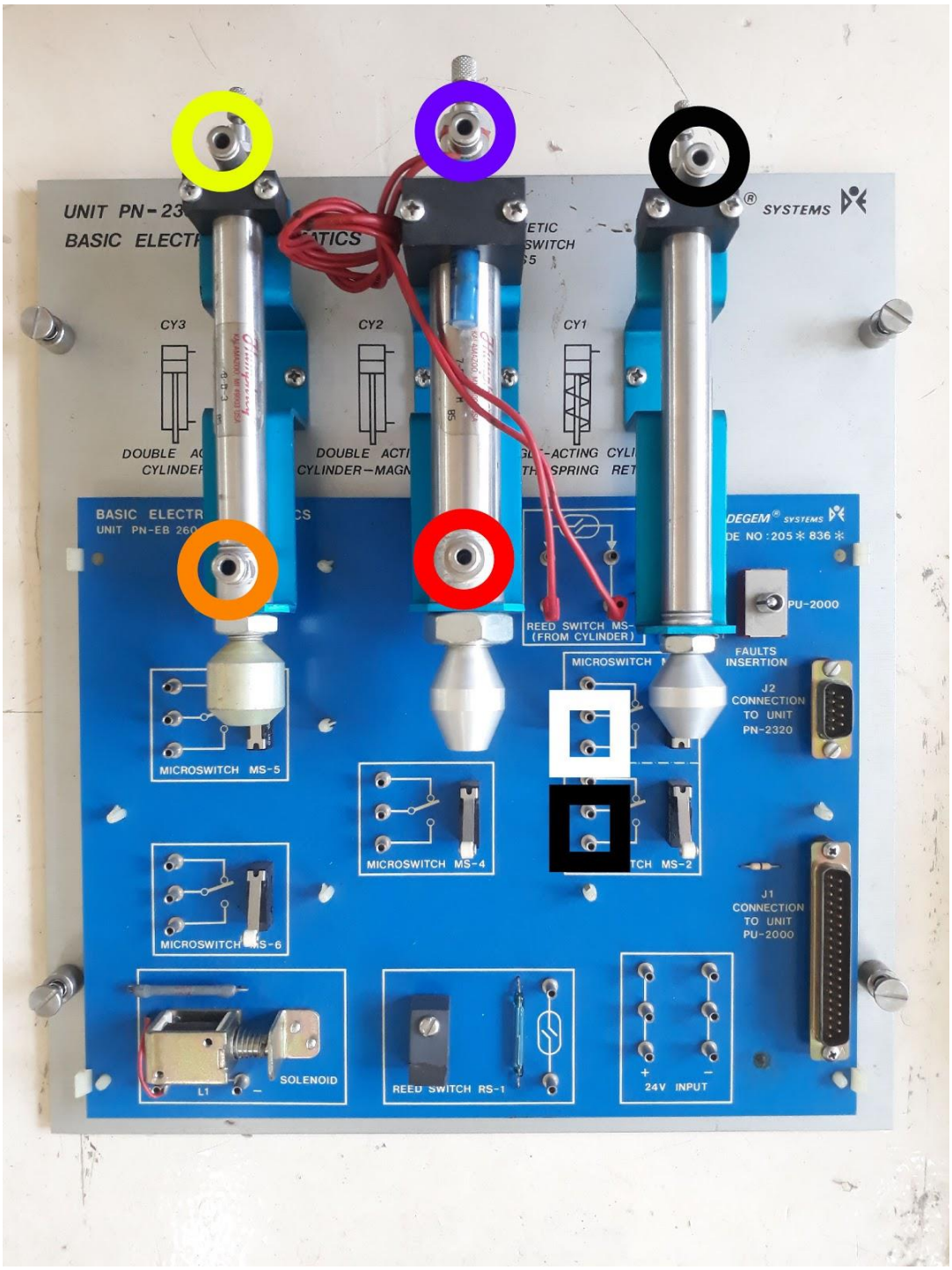


Figure 1.2. *Actuators*

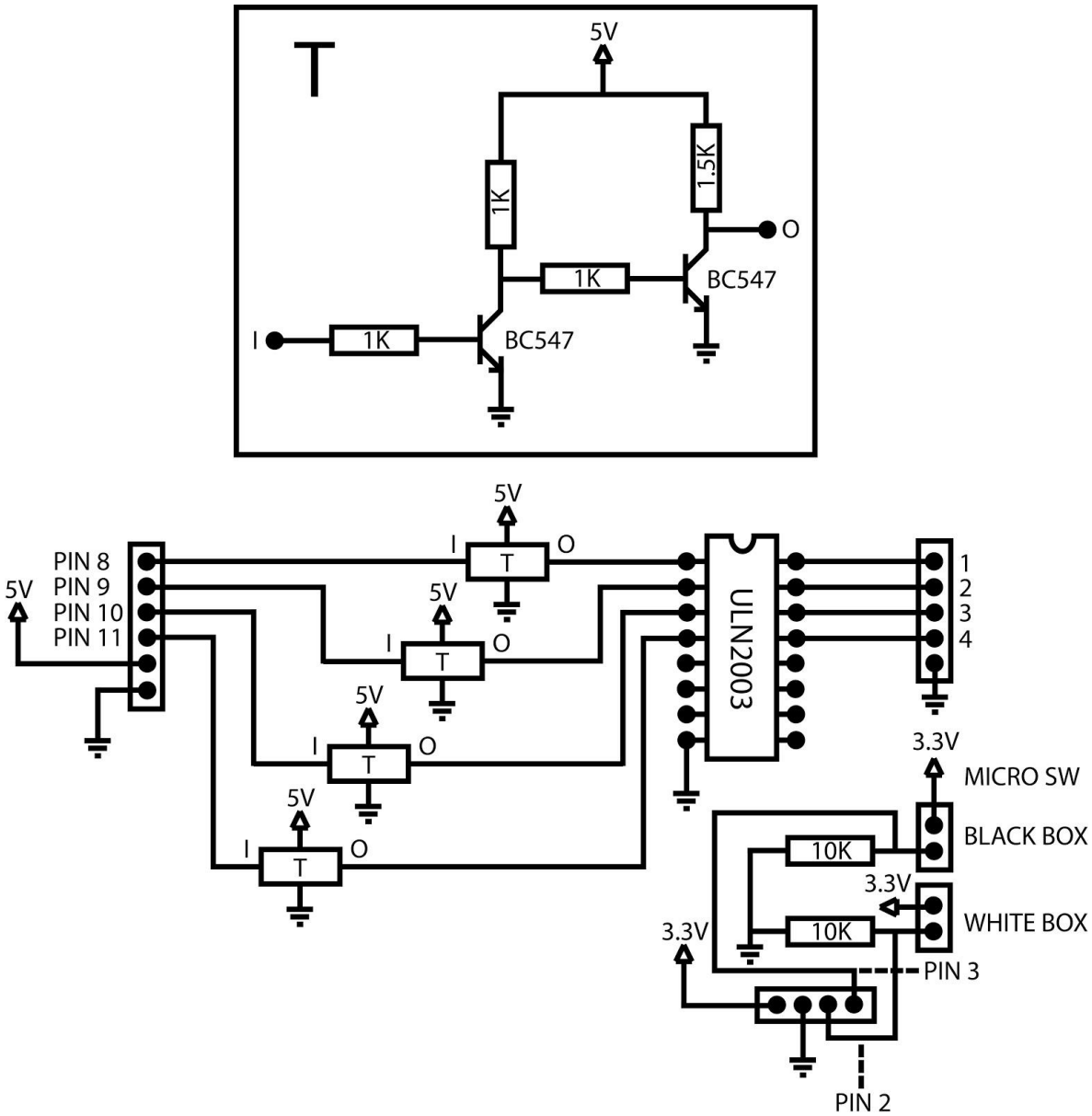


Figure 1.3. *Schematic Diagram*

USER INTERFACE

Figure 2.1 shows the user interface for the application that controls the PN2320 Pneumatic Control. The UI has several functions: Direct Control Push Buttons, Direct Control Latch Buttons, Logic Circuit Simulation and Serial Editor. All functions will simply result to the switching on and off of solenoid valves.

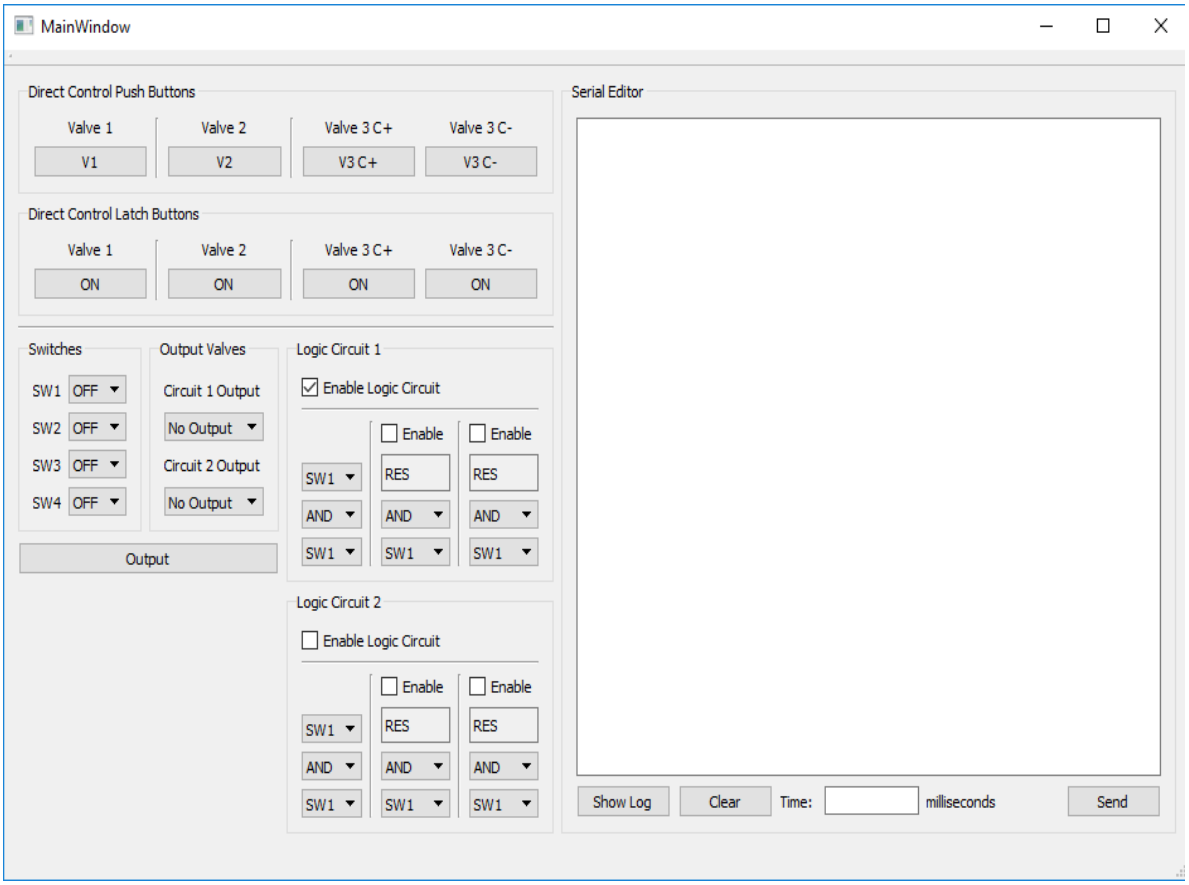


Figure 2.1. *Graphical User Interface*

The Direct Control Push Buttons, as shown in figure 2.2, send signals to the microcontroller when the buttons are either pressed or released. When buttons are pressed, the application sends a keyword via serial to trigger the microcontroller to open the valve. On the other hand, the application sends a keyword to trigger the microcontroller to close the valve.

Pressed: valve opens

Released: valve closes

Figure 2.2. *Direct Control Push Buttons*

The Direct Control Latch Buttons, as shown in figure 2.3, operate similarly to the push buttons but has a slight difference. The buttons toggle ON or OFF when clicked or pressed.

Toggled ON: valve opens

Toggled OFF: valve closes

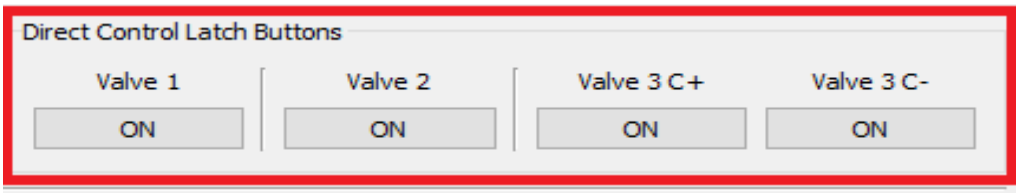


Figure 2.3. *Direct Control Latch Buttons*

Figure 2.4 shows the Logic Circuit Simulation function of the application. The SWITCHES group contains four drop-down boxes that control the state of four switches. The OUTPUT VALVES group contains two drop-down boxes which lets the user select which valve to use as output for the two logic circuits. The LOGIC CIRCUIT groups contain several drop-down boxes and checkboxes that lets the application simulate a AND or OR gate connection.

Procedure:

1. Enable a logic circuit by checking the “Enable Logic Circuit” checkbox.
2. Enable 2nd or 3rd logic circuit connection if needed.
3. Set switches to simulate on Logic Circuit group box.
4. Set logic gate for simulation on Logic Circuit group box.
5. Set output valve for either circuit 1 or 2 on Output Valves group box.
6. Set switch states on Switches group box.
7. Click Output

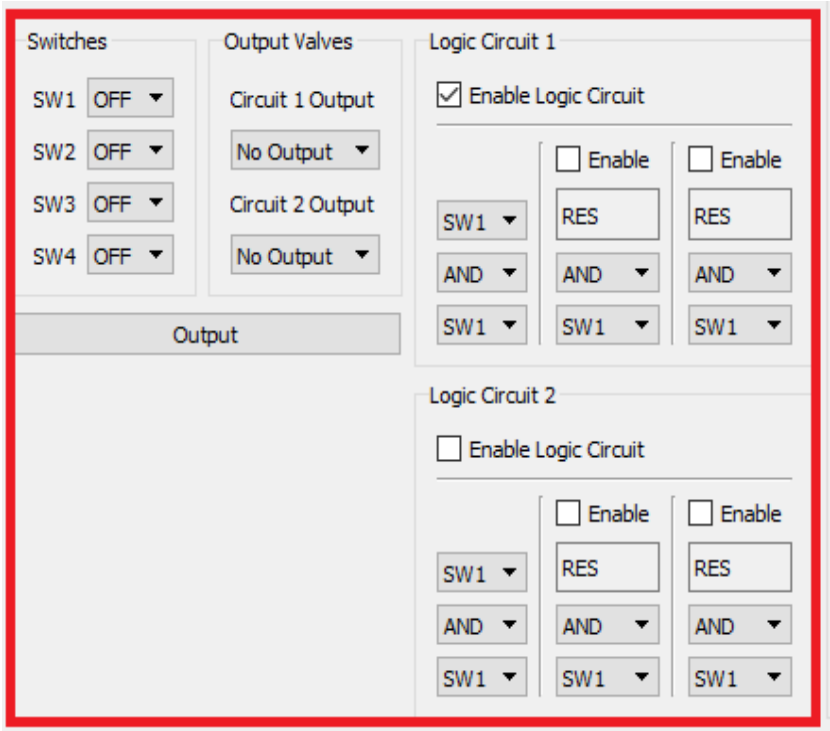


Figure 2.4. *Logic Gate Simulator*

The Serial Editor, as shown in figure 2.5, allows the user to customly send instructions to the microcontroller via serial. Keywords are typed onto the text editor and are read and sent to the microcontroller when SEND button is clicked.

From the bottom left is the SHOW LOG button which displays the logged data for the duration of an actuator to reach from point A to point B. Next to it is the CLEAR button which clears the text editor. And lastly is the display box which shows the duration of an actuator to move from point A to point B in milliseconds.

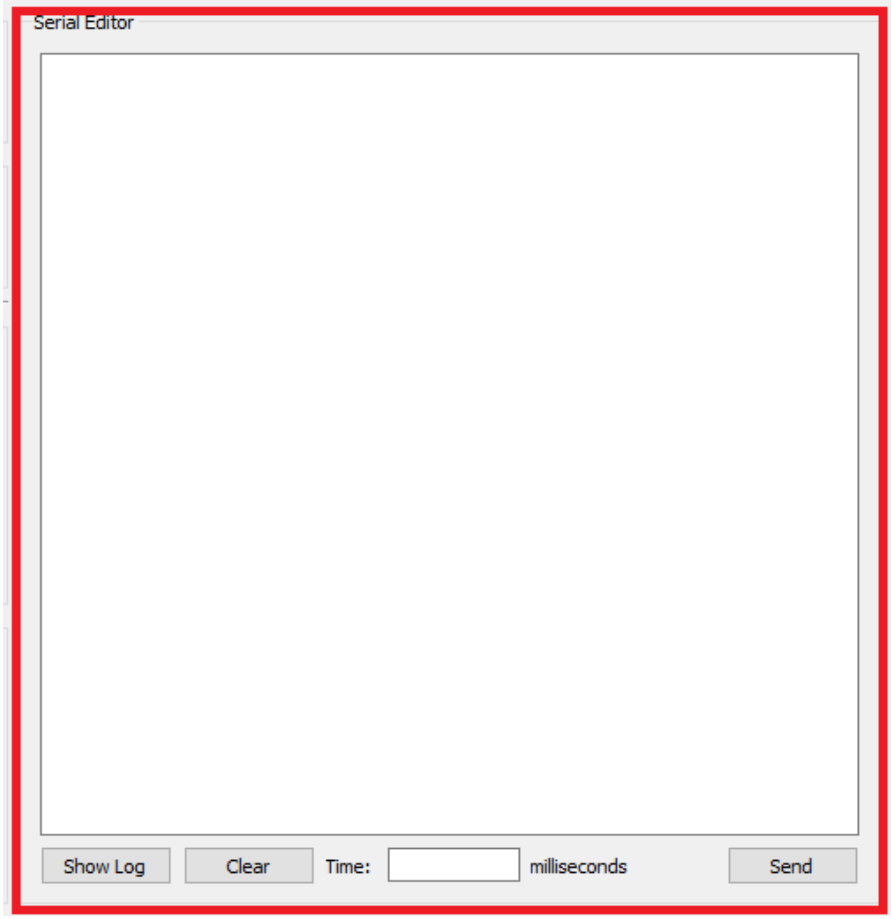
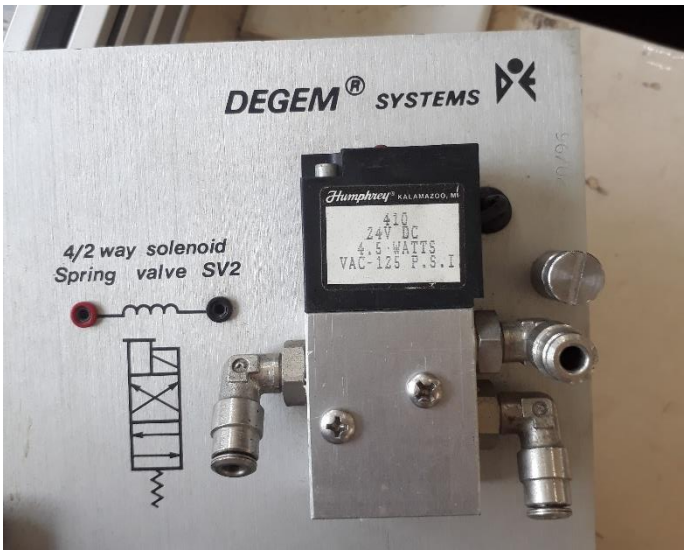


Figure 2.5. *Serial Editor*

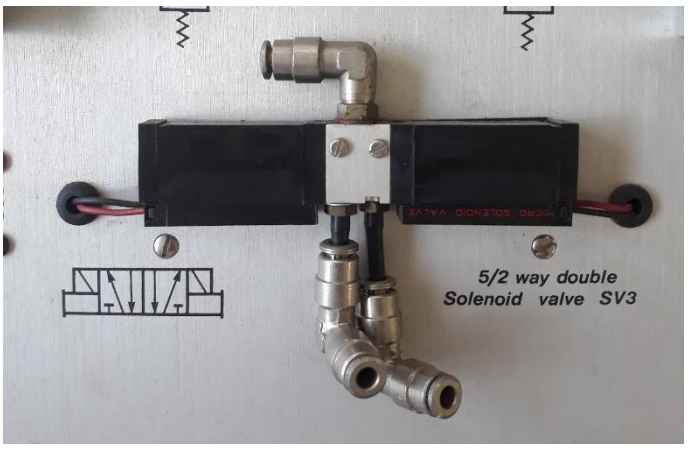
APPENDIX E

MISCELLANEOUS

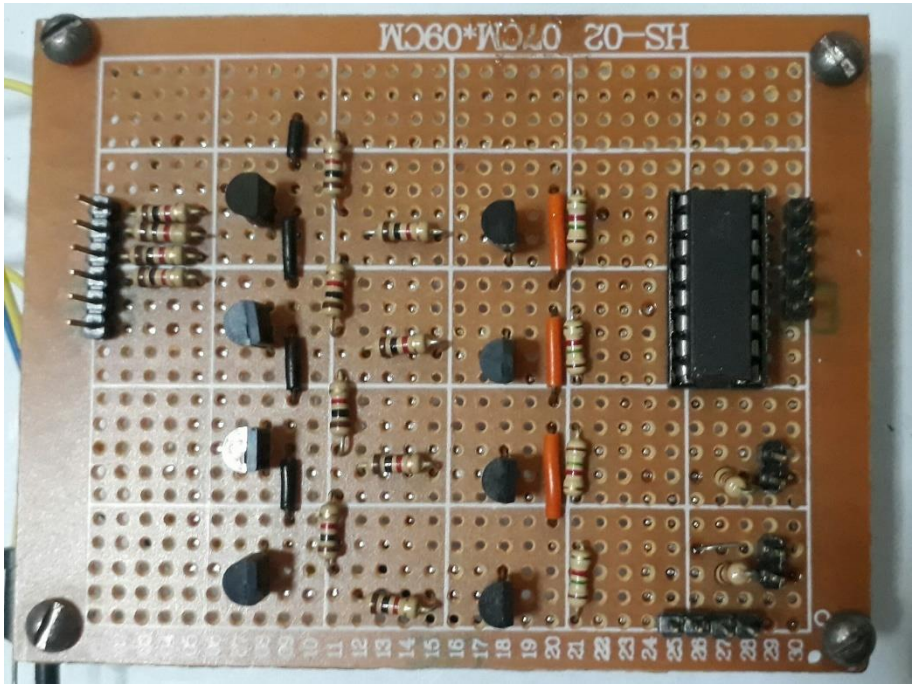
3/2 Way Solenoid



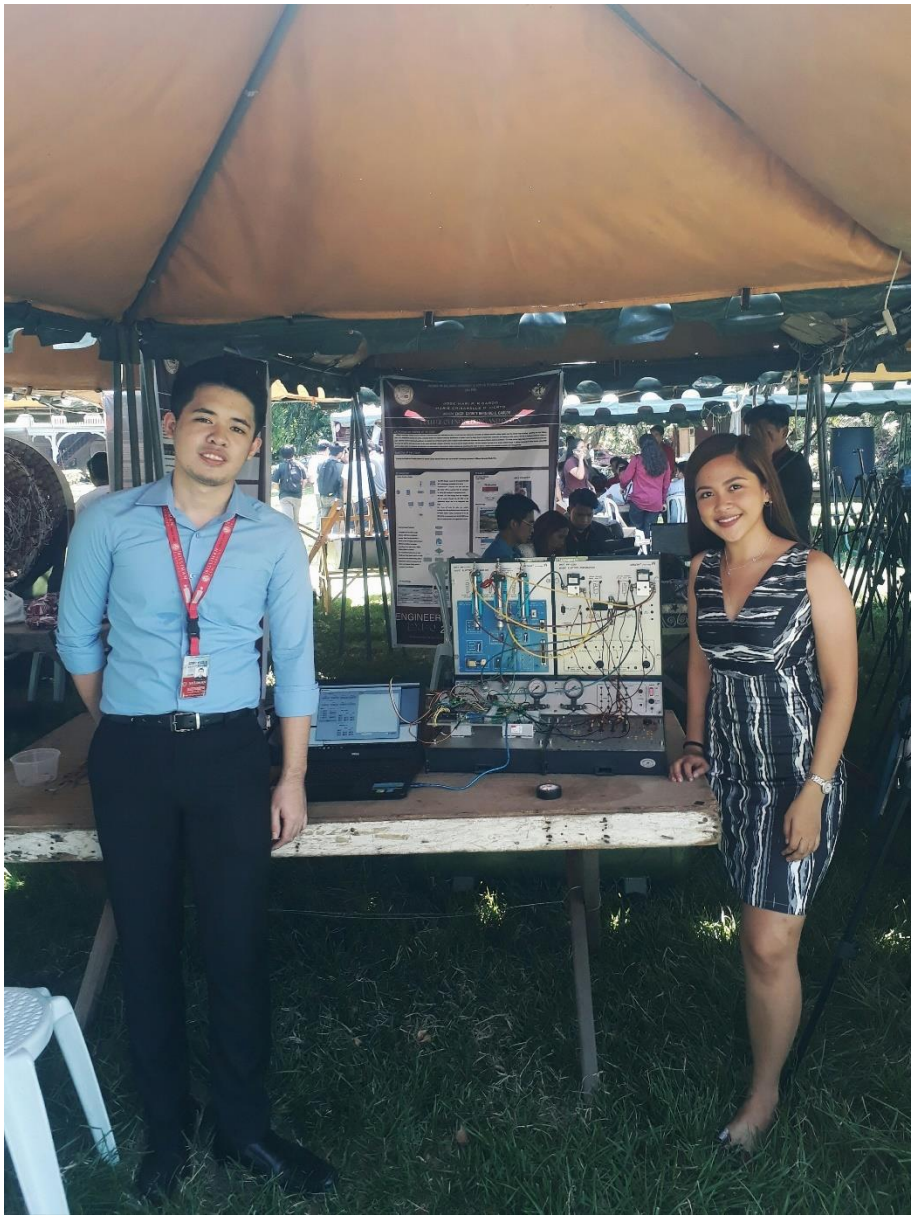
4/2 Way Solenoid



5/2 Way Double Solenoid



Driver Circuit Close-up



The Proponents