DEVELOPMENT OF A PYTHON BASED USER INTERFACE FOR REAL-TIME TEMPERATURE MEASUREMENT

BY

OLATOYE AFOLABI

WITH MATRICULATION NUMBER

EU130304-243

A PROJECT SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING, ELIZADE UNIVERSITY, ILARA-MOKIN, NIGERIA.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF ENGINEERING DEGREE IN ELECTRICAL AND ELECTRONIC ENGINEERING

SEMPTEMBER, 2018

CERTIFICATION

This is to certify that this project titled, "Development of a Python Based User Interface for Real-Time Temperature Measurement", was done and submitted by AFOLABI OLATOYE SAMAD (EU130304-243), under the supervision of Dr. Olugbenga K. Ogidan to the Department of Electrical and Electronics Engineering, Faculty of Engineering, Elizade University, Ilara-Mokin, Nigeria.

_____ _____

DR. O.K. OGIDAN Date

Project Supervisor

_____ _____

DR. O.K. OGIDAN Date

Head of Department

DEDICATION

This project work is dedicated to God Almighty my creator, my strong pillar, my source of knowledge and wisdom who has been with me throughout my undergraduate program. I also dedicate this work to my parents for their immeasurable love and support.

ACKNOWLEDGEMENT

My first appreciation goes to God, for giving me the strength, knowledge and opportunity to conduct this study and guiding me in making this project.

I am highly grateful to my supervisor Dr. Olugbenga K. Ogidan for his valuable support, guidance and encouragement during the project work. I am highly obliged to him for helping me to gain successful completion of this project.

My thanks and gratitude goes to all the teaching and non-teaching staffs of Elizade University, the knowledge imparted in me has, in one way or the other, contributed to the successful completion of this project.

Lastly, my appreciation goes to my parents for their unconditional love and belief in me and their contribution to my success in this university and to my project. Also, I would like to take this opportunity to sincerely thank all my course mates, friends and the experts who assisted me in the successful development of this project and monitoring of my progress towards a better understanding of everything the project entails.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project outlines the design and implementation of a software system to display data graphically in real-time. It is built using the python programming language and its vast array of packages for both Graphical User Interface development, plotting of graphs, database development and display of the data. The system utilizes the PyQt5 graphical user interface development tools, PySerial access tools and PyQtGraph plotting tools. The system interface is Windows-based but can also be ported to other operating systems. The software would be useful for the end user who wishes to log temperature data for various purposes including ovulation detection, vital signs measurement and the likes. At the end of the project, creation of a Graphical User Interface (GUI) is accomplished and a graph of the temperature data is plotted in it. The real-time saving of the data is also accomplished using comma separated values (CSV).

CHAPTER ONE

INTRODUCTION

1.1 Background of Study

This project involves the design and implementation of a software with a Graphical User Interface (GUI) written in the python programming language to be used to display temperature data in real-time.

There have been different software's used to display and read temperature data but most are written in other languages such as visual basic, Java and so forth. These languages have been around for a long time and their use is still essential.

In this project, I will be making use of the python programming language to design a software GUI which will display the temperature data in real-time and save it for future use. The hardware being used consist of an Arduino microcontroller which will be the device where the data will be collected from.

1.2 Statement of Problem

The methods of temperature display, analysis and storage available in the world currently are mostly analog and a few of the software being used to display and store this temperature data are outdated. While other software developed in the past have been built using programming languages such as visual basic, C, C#, C++, the software in this project is developed using the python programming language. The python programming language is a highly developed language with a vast array of packages and functionality. The other languages being used in the past still have their uses but the python programming language

has become one of the best languages for data analysis in both science and engineering. The software developed in this project will be able to do the following;

- Display data graphically
- Display data in real-time with minimum time lag
- Stores data in real-time with time stamps attached
- Displays data in different graphical formats
- Allows storage of data to be used elsewhere.

This project will lead to a new software program written in a popular programming language which allows for support on multiple operating systems (Windows, Linux). A software which is applicable for use both privately and freely.

1.3 Justification of The Work

A software program to display data in real-time written in the python programming language can be applied for different uses. Such applications can be found in the health sector, personal uses and engineering applications. The different reasons why this project is useful are highlighted below:

i. Display of data in real-time.
ii. Storage of data in real-time.
iii. Adjustment of software to display other types of data such as sine waves for engineering purposes.
iv. Programming language support on multiple operating systems.

1.4 Scope of Study

This project will involve the design and development of a software with a Graphical User Interface (GUI) to display data in real-time from an ovulation detection device.

1.5 Aim

The aim of this project is to develop a software program with a Graphical User Interface (GUI) to display temperature data collected from an Arduino microcontroller in real-time.

1.6 Objectives

1. To collect data from a temperature sensor circuit through an Arduino.
2. To display the temperature data graphically and in real-time.
3. To store the data.

1.7 Methodology

1. Design and implement a temperature sensor circuit.
2. Connect the Arduino UNO microcontroller to the temperature circuit.
3. Determine if the Arduino is receiving the temperature and display the data.
4. Develop a python script to collect the data from the Arduino and display graph of the temperature data using pyqtgraph package.
5. Design a Graphical User Interface using the PyQt5 designer.
6. Develop a python script to read the temperature data from the Arduino in real-time.
7. Integrate the python code into the GUI to display the data in the Graphical User Interface.
8. Develop a database to store the temperature data.

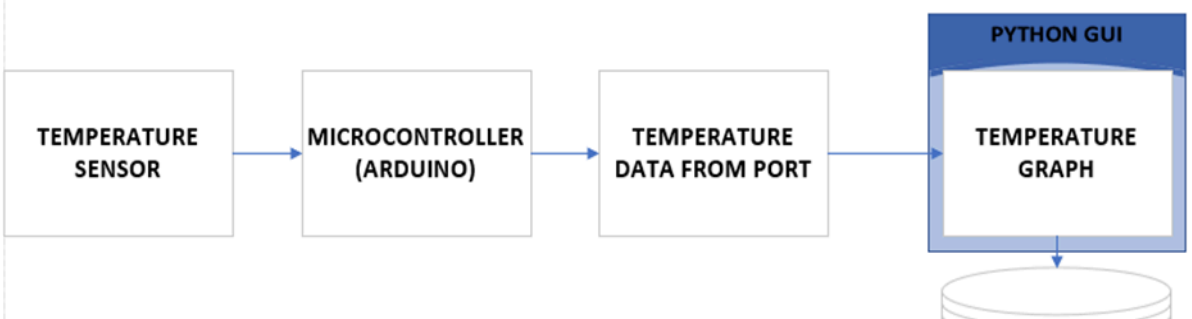

Figure 1.1: Block Diagram of System

1.8 Organization of Report

This section gives a brief overview of what is to be expected in the report. After the Chapter One offering an introduction into the project as well as the aim and objectives. Chapter Two will look at the review of literature, analysis of past projects similar to this one, pros of this project with respect to other projects as well as the limitations of this project. Chapter Three will delve in more detail into the methodology of this project which include an explanation on the tools used for this project. Chapter Four focuses on the implementation, testing as well as debugging of the system being developed. Chapter Five concludes the report.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

The concept of software with a Graphical User Interface for displaying data in real-time is not a new thing. There have been different implementations of such concepts with different purposes. In this chapter, these different projects will be discussed.

This chapter also gives an insight into various studies conducted by outstanding researchers, as well as explained terminologies with regards to ovulation detection as well as the software used with them.

2.1 Python

The history of the Python programming language dates back to the late 1980s. The python programming language was designed in the late 1980s and its implementation was started in December 1989 by Guido van Rossum in the Netherlands as a successor to the ABC programming language capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author (Tulchak et al, 2016). Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, C++ or Java. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

2.1.1 advantages of python

1. Clean syntax.
2. The normal distribution has a lot of useful modules including the module for developing GUI.
3. The use of python in interactive mode which is very useful for experimentation and solving simple problems.
4. The normal distribution is simple but at the same time has a powerful development environment.
5. Suitable for solving mathematical problems and as a means of working with complex numbers.

2.2 Real-Time Computing

Real-time computing (RTC) is a term for computing practices that have specific time constraints. Real-time computing has to be done in a time frame that is relatively imperceptible to the user. By contrast, other types of computing can be done on a delayed basis, for instance, where information is aggregated, kept and stored for later use (Stankovic, 1992).

One of the best ways to explain real-time computing is to use an example such as the "form load" command (Real-Time computing, 2010). Something like this is almost always done in real time. This way, when a user clicks on a command to open the program, the form opens up immediately. In optimal conditions, with the right bandwidth for Web-delivered systems, memory storage and powerful CPU operation, the form pops up in a split second. In other cases, there may be delays, but this still counts as real-time computing — it is computing that, when done on command, is programmed to happen almost immediately.

Real-time computing is a type of metric that developers and engineers must look at when they decide how a program will work. Another good example is a high-level computing task, such as a command-driven program that looks for discrepancies in text or numbers or generates complex calculations. Because of the sophistication of computer hardware today, many of these programs can be built for real-time computing, where the results come back almost as soon as the user hits the command button. The same is true for rendering complex graphics, ordering data or doing other high-level computation.

However, "real-time computing is not about speed". When people talk about real-time computing there are certain common misconceptions, especially that reacting in microseconds or nanoseconds makes the system real-time and taking five minutes means it is not real-time. But real-time means that the system reliably takes the same amount of time

every time it does a certain thing. Duration does not matter. If the system's highest priority is blinking a red LED, it always honours that priority, and no matter what else is happening, when the light needs to blink it leaves everything aside to do that and do it quickly.

2.2.1 examples of real-time computing applications

- Videoconference applications
- VoIP (voice over Internet Protocol)
- Online gaming
- Community storage solutions
- Some e-commerce transactions
- Chatting
- IM (instant messaging)

The concept of gathering data in real-time is something that is popular in the field of IOT (Internet of Things). The Internet of Things (IoT) is the network of objects, such as some vehicles, mobile devices, and buildings that have electronic components, software, and network connectivity that enable them to collect data, run commands, and be controlled through the Internet.

In a study where it was proposed to use an Arduino microcontroller to monitor bus locations in real-time (Ibrahim et al, 2017). This was proposed by Mohammad Y. M. Ibrahim and Lukman Audah. In this study, the system has been separated into two parts, which are the hardware and the software. The hardware parts are the Arduino Uno and the Global Positioning System (GPS), while Google Earth and GpsGate are the software parts. The GPS continuously takes input data from the satellite and stores the latitude and longitude values in the Arduino Uno.

If they wanted to track the vehicle, they would need to send the longitude and latitude as a message to the Google Earth software to convert these into maps for navigation. Once the Arduino Uno is activated, it takes the last received latitude and longitude positions' values from GpsGate and sends a message to Google Earth. Once the message has been sent to Google Earth, the current location will be shown, and navigation will be activated automatically. Then after several successful simulations, the results will be shown in real time on a map. The scope of the project was limited only from Taman University to the UTHM campus.

The next study looked at applies the concept of real-time computing in the same way this project wishes to apply it in that it is displaying temperature in real-time. In this paper titled Ovulation Detection Mechanism – A microcomputer-based approach (Ogidan et al, 2011), they present a real-time microcomputer-based logger for measuring basal body temperature (BBT). The components used in the study were an NTC thermistor for sensing temperature, an analog to digital converter (ADC), a transceiver and a microcomputer. The key difference aside from a few components between this project and the one currently being discussed is the software program being used, while the one being discussed makes use of visual basic in developing the Graphical User Interface that was used, this project however uses the python programming language. Below is a brief comparism of the python programming language to visual basic.

Table 2.1: Table Comparing Python and Visual Basic

PYTHON	VISUAL BASIC
Cross Platform	Huge Community
Good introduction to data types	Capable Language

Has extensive libraries for scientific computing, data mining etc.	Sibling to C#
Can be used in many domains	Flexible
Easier to get started	English-like syntax
Clear Syntax	Easy to deal with win32API
Good Documentation	Very simple and efficient in terms of lines code

Another study to look at applying the concept of real-time computing is the paper authored by Alinafe et al titled Real time, web-based Temperature Monitoring System for Cold Chain Management in Malawi. The difference with this study is the added functionality of the internet. In this retrospect, we can consider the system being built to be an IOT implementation. In the paper, the authors grouped their design into three stages, the Physical layer (Layer 1), the Network layer (Layer 2) and the Application layer (Layer 3). What this particular project emphasizes aside from the development of a real time web-based system is the use of open source technologies in developing this project. When a product is open source refers to the source code of the product in terms of software being made available to the public which allows for modification and development of new variations of the product. This project makes use of open source technologies in the same way the above currently discussed project does. We can conclude that real-time computing is an important technological concept. This project applies the concept of real-time computing in the gathering of temperature data, storage of the data and display of the data.

Other papers reviewed include a paper titled Design and Development of Low Cost PC Based Real Time Temperature and Humidity Monitoring System authored by Nungleppam Monoranjan Singh and Kanak Chandra Sarma. The paper presents the design

and development of a low-cost Data Acquisition System (DAS) using PIC12F675 microcontroller for real time temperature and humidity monitoring (Singh et al, 2012). In this paper, the authors suggest a system to monitor temperature and humidity in real-time, the firmware used was written in Basic while the application program used was developed with visual Basic 6 to display the data.

The works covered Table 2.2 below all make use of real-time monitoring or control for different purposes. In [1], it is being used for control of microfluidic PCR chip. It can be seen that each makes use of the concept of real-time with none of them making use of software created using a programming language such as python.

2.3 Other Literature Review Works

Table 2.2: Literature Review Works

S/N	Paper	Problem Statement	Author/Location	Methodology	Result	Instrumentation
1	MCU based real-time temperature control system for universal microfluidic PCR chip	Dawoon Han · You-Cheol Jang · Sung-Nam Oh · Rohit Chand · Ki-Tae Lim · Kab-Il Kim · Yong-Sang Kim	Development of a Microcontroller Unit based real-time temperature control system for universal microfluidic Polymerase chain reaction (PCR) chip.	PCR is applied in a variety of ways such as genotyping, taxonomy, forensic investigation. The microfluidic PCR chip was made using photolithographic techniques.	An effective and automatic way to monitor and control the heating efficiency of the ITO micro heater on the proposed on-chip PCR system.	Microfluidic PCR chip, LM35Z temperature sensor, Microcontroller Unit.

Contd

2	Design and Development of Low Cost PC Based Real Time Temperature and Humidity Monitoring System (2012)	Nungleppam Monoranjan Singh and Kanak Chandra Sarma	Designing and developing a low-cost Data Acquisition System (DAS) using PIC12F675 microcontroller for real time temperature and humidity monitoring.	The Data Acquisition System consists of the PIC2F675 microcontroller, the MAX232 driver and a power supply. Pin 2 of the microcontroller is used to send the serial data gotten from the temperature and humidity sensor through the MAX232 driver IC to the PC. A Zener diode was used to prevent over voltage. The firmware used was written using Oshonsoft PIC Simulator IDE. An application program was written in visual basic 6.	A graph is plotted and the data is stored into the hard drive of the PC into comma separated value (.csv) format.	PIC2F675 microcontroller, 5v regulated power supply, MAX232 Driver IC, PC, LM35 temperature sensor, TPS00715 humidity sensor, OpAmp OP07, Zener diode, Oshonsoft PIC Simulator IDE

Contd

3	Real-Time, Web-based Temperature Monitoring System for Cold Chain Management in Malawi (2017)	Alinafe Kaliwo, Jonathan Pinifolo, Chomora Mikeka	Developing a low cost, real-time, web-based temperature monitoring system to monitor both fixed and mobile cold chain.	This project involves three layers, the Physical layer, the network/communication layer and the application layer.	The RWTMS has demonstrated a huge potential in tackling issues that arise due to poor monitoring solutions. The solution allows users to log in at anytime, anywhere to check the condition of their assets. The system is also able to prompt the user when an abnormal scenario has occurred.	Atmel programmable chips of designation ATMEGA 328/P, DS18B20 temperature sensors, solar panel, GSM module, cloud server.

Contd

4	Scheduling of Air Conditioner Based on Real Time Price And Real-Time Temperature (2015)	Zeeshan Haider, Faisal Mehmood, Xiaohong Guan, Fellow, IEEE, Jiang Wu, Member, IEEE, Yang Liu, Pervez Bhan	Development of a smart scheduling device of an air conditioner which demonstrates the general architecture along with practical implementation of the hardware that could be used to schedule the air condition based on real time price and temperature in order to reduce the cost of electricity bill devoid of compromising user comfort.	The proposed architecture of the overall system comprises of 3 parts, Data collection part, Scheduling part and control part. The system collects outdoor temperature as well as dynamic price in real time. Scheduling part serves as brain of proposed system and is responsible for taking decisions based on input parameters. It determines an optimal choice of energy consumption scheduling which takes into consideration the user comforts.	Hardware design of overall system.	Temperature sensors, AMI, raspberry pi, python script, server, database

CHAPTER THREE

METHODOLOGY

3.0 Introduction

This chapter provides a detailed description of the methods and components as well as a detailed description of how the project is being designed and implemented. It also looks at how the components being used interact in order to make the system function.

3.1 Framework Design

The framework being used for this project is the Software Development Life Cycle (SDLC). The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. Software development life cycle is the most important element in software development. It depicts the necessary phases in software development (Leau et al, 2012). The life cycle defines a methodology for improving the quality of software and the overall development process.

There are different models under the SDLC which include the waterfall model, the V-shaped model, the incrementing model. In this project, we will be making use of the waterfall model. A brief explanation of the waterfall model is shown below;

3.1.1 waterfall model

The Waterfall Model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases (Balaji et al, 2012). This means that any phase in the development process begins only if the previous phase is complete.

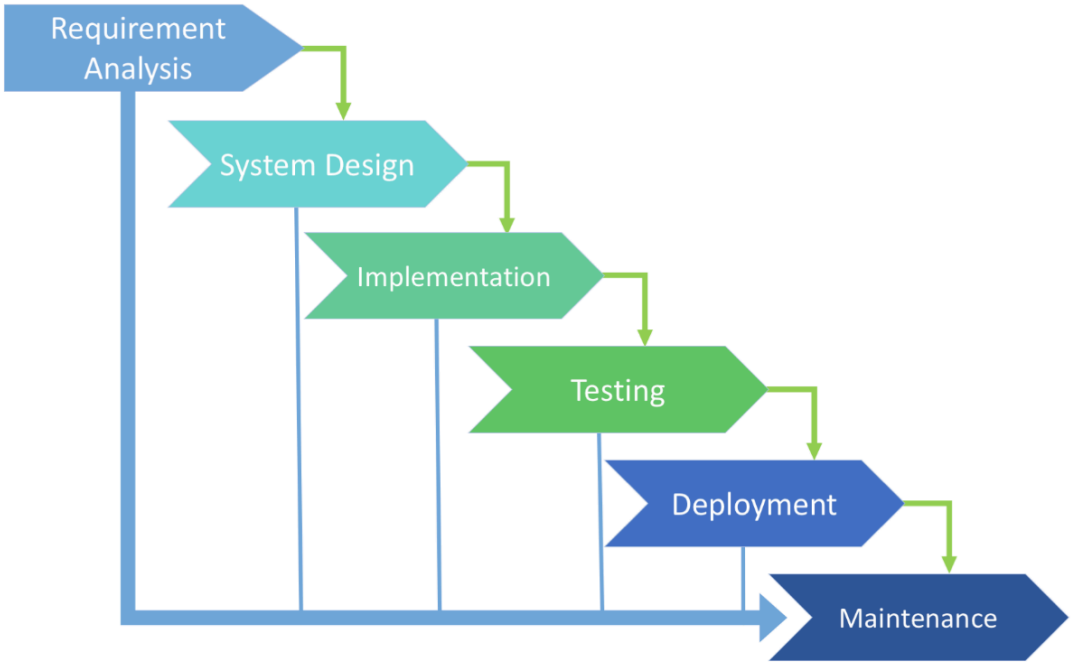

Figure 3.1: Diagram of the Waterfall Model (SDLC, 2008)

Table 3.1: Advantages and Disadvantages of the Waterfall Model (Mohamed Sami, 2012)

Advantages	Disadvantages
Easy to explain to users.	Assumes that the requirements of a system can be frozen.
Structures approach.	Very difficult to go back to any stage after it finished.
Stages and activities are well defined.	A little flexibility and adjusting scope is difficult and expensive.
Helps to plan and schedule the project	Costly and required more time, in addition to the detailed plan.
Verification at each stage ensures early detection of errors.	

3.2 Requirements Analysis

This stage involves identifying the system requirements and user requirements that is what the python user interface needs to accomplish as well as what will be needed to complete the system. These features, called requirements, must be quantifiable, relevant and detailed such as it must display the temperature data graphically.

3.2.1 user requirements

This is a result of identifying what a user can accomplish with the Python User Interface displaying data in real-time. The user can record daily temperature and store for later usage.

3.2.2 system requirements

This refers to the requirements necessary for the development of the Python Graphical User Interface. The following are the requirements identified;

- Temperature sensor circuit to gather temperature data.
- Arduino microcontroller.
- Laptop with Windows operating system to load the GUI.
- USB Serial port through which the data is collected.

3.3 System Design

In this section, the components used are discussed in detail as well as a flowchart showing the software process. This section looks at both the hardware and the software section. While this project focuses on the software, hardware is required to test the software to ensure the software functions as required. The components being discussed are the Arduino UNO, the python GUI packages (PyQt5, PyQtGraph, PySerial), temperature sensor circuit.

3.3.1 pycharm

This is the development environment used to develop the python program. It was selected because of its vast functionalities and add-ons for more functions. The python code is written in this program.

3.3.2 python packages

The different python packages being used are pyqt5 package, PyQtGraph, pyserial and NumPy.

Pyqt5

Qt is a set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development.

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.

PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications.

Pyqtgraph

PyQtGraph is an open-source graphics and user interface library for Python that provides functionality commonly required in engineering and science applications (Kelekar A, 2018). Its primary goals are

- To provide fast, interactive graphics for displaying data (plots, video, etc.)
- To provide tools to aid in rapid application development.

PyQtGraph makes heavy use of the Qt GUI platform (via PyQt or PySide) for its high-performance graphics and NumPy for heavy number crunching. In particular, pyqtgraph uses Qt's Graphics View framework which is a highly capable graphics system. It runs on Linux, Windows, and OSX.

Why pyqtgraph?

Below is a comparism between pyqtgraph and other GUI libraries in python i.e. Matplotlib and pyqwt5. From the differences, it can be seen that pyqtgraph possess the necessary requirements such as speed, data acquisition and analysis applications.

- Matplotlib: For plotting, pyqtgraph is not nearly as complete as Matplotlib, but runs much faster. Matplotlib is more aimed toward making publication-quality graphics, whereas pyqtgraph is intended for use in data acquisition and analysis applications. Matplotlib is more intuitive for MATLAB programmers; pyqtgraph is more intuitive for python/Qt programmers. Matplotlib does not include many of pyqtgraph's features such as image interaction, volumetric rendering, parameter trees, flowcharts, etc.
- pyqwt5: About as fast as pyqwt5, but not quite as complete for plotting functionality. Image handling in pyqtgraph is much more complete (again, no ROI widgets in qwt). Also, pyqtgraph is written in pure python, so it is more portable than pyqwt, which often lags behind pyqt in development (I originally used pyqwt, but decided it was too much trouble to rely on it as a dependency in my projects). Like matplotlib, pyqwt (to my knowledge) does not include many of pyqtgraph's features such as image interaction, volumetric rendering, parameter trees, flowcharts, etc.

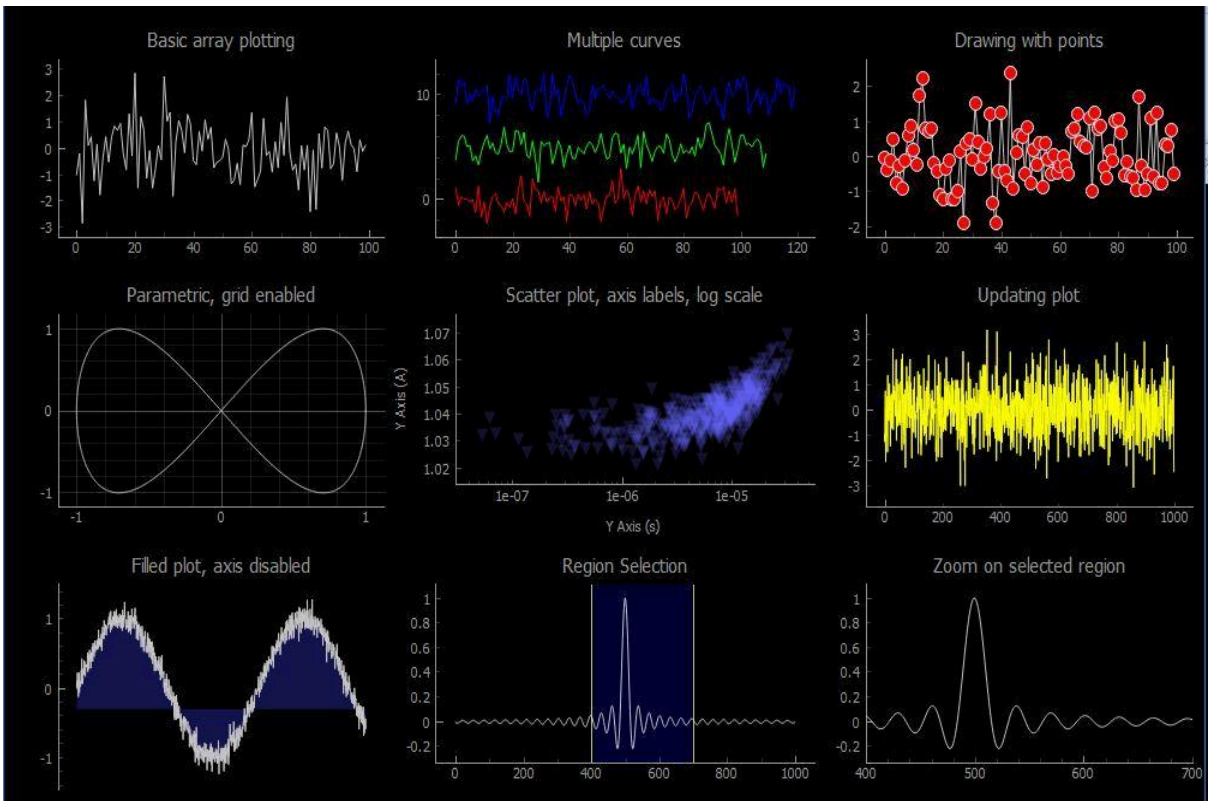

Figure 3.2: The Different Types of Graph Plotted Using PyQtGraph (pyqtgraph.examples)

Pyserial

The module comprises the access for the serial port. This module is what allows python to communicate with the serial port of the PC. From the serial port, commands are issued to the microcontroller and also through the serial port, temperature data is collected.

Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- NumPy can seamlessly and speedily integrate with a wide variety of databases.

3.3.3 system architecture

In this section, the flow of data from the Arduino to the GUI is shown in Fig 3.3 below as well as the flowchart shown in Fig 3.4. The temperature data is collected by the Arduino from the temperature sensor circuit. The temperature sensor circuit is not explained in detail as the focus is on the software process rather than the hardware process. The temperature data is sent to the PC through the serial port of the PC. The python GUI receives this data using the pyserial port mentioned above in the python packages section. The data is displayed on the GUI using the PyQtGraph package to display it graphically and the csv package to store the data.

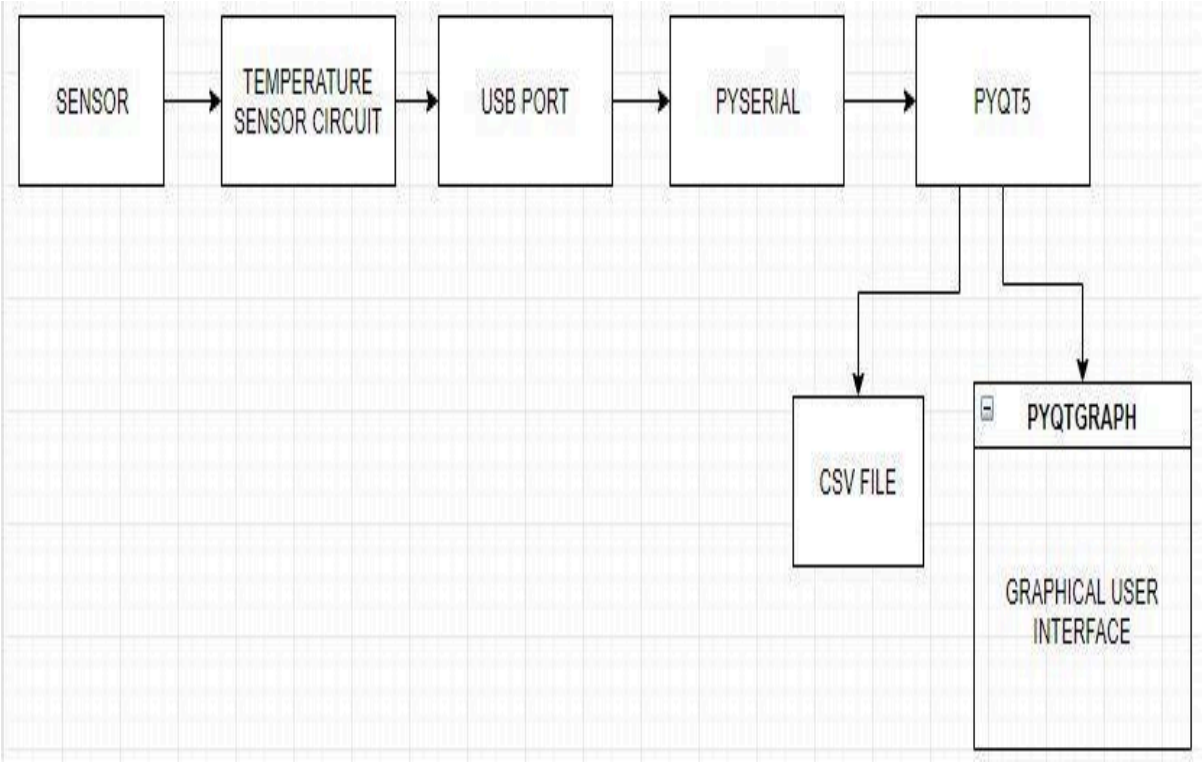

Figure 3.3: System Architecture Diagram

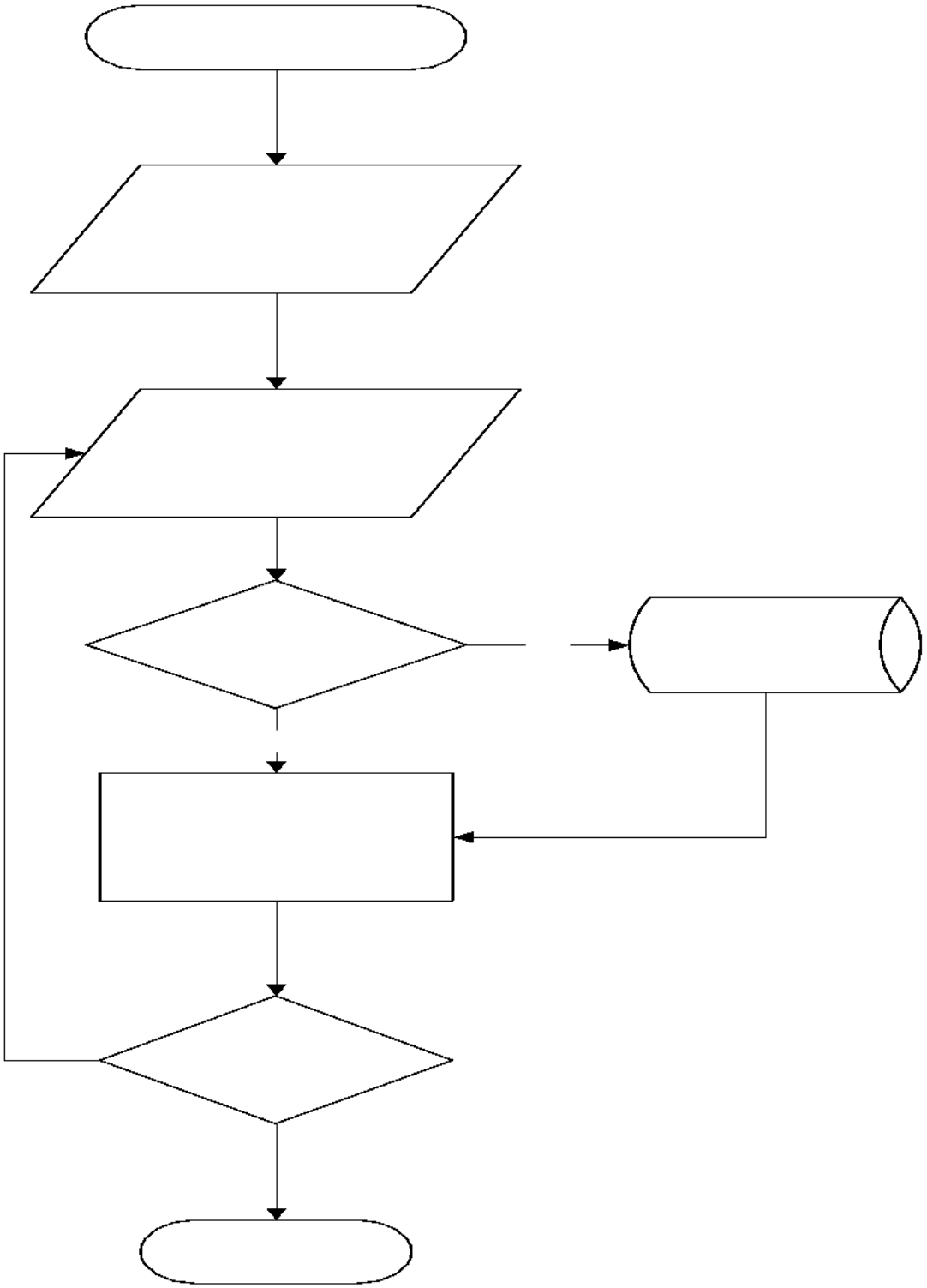

Figure 3.4: Flowchart for Python GUI

CHAPTER FOUR

IMPLEMENTATION AND TESTING

4.0 Introduction

This chapter provides a detailed description of the implementation and testing of the design of a graphical user interface using python to display data in real-time. It also looks at how the components discussed in chapter three interact in order to make the system function. It also shows the display achieved at the end of this project.

4.1 Hardware Implementation

While this project focuses mainly on the design of the software used to display data. This section covers the design of the hardware used to collect temperature data in real-time to determine the efficiency and workability of the project. An overview of the components and how they were connected is listed below:

- The Arduino is connected to the breadboard to provide power to the circuit.
- A thermistor is used to collect temperature data from the room and LEDs are used to display which temperature range is currently active.
- Resistors are placed at strategic points on the breadboard to avoid damage to the other instruments.
- Wires are used to make connections in the circuit and from the Arduino to the breadboard.

The different hardware components used are discussed below with their respective values also noted.

4.1.1 resistors

The resistors used in this project include three 220ohm resistors connected from the cathode legs of the LEDs to the ground of the Arduino. A 10k ohm resistor is used to connect the thermistor to Arduino.

4.1.2 thermistors

A thermistor is an element with an electrical resistance that changes in response to temperature. They are widely used in a variety of electronic applications, most often as temperature sensors. Additional uses of thermistors include current limiters, current protectors, and heating elements. The thermistor is used as a temperature sensor in this implementation because it gives more accurate readings.

4.1.3 arduino uno

As discussed in the previous chapter, an Arduino is a microcontroller. It is simple to use and easier to set up than other microcontrollers out there which is why it is being used in this project.

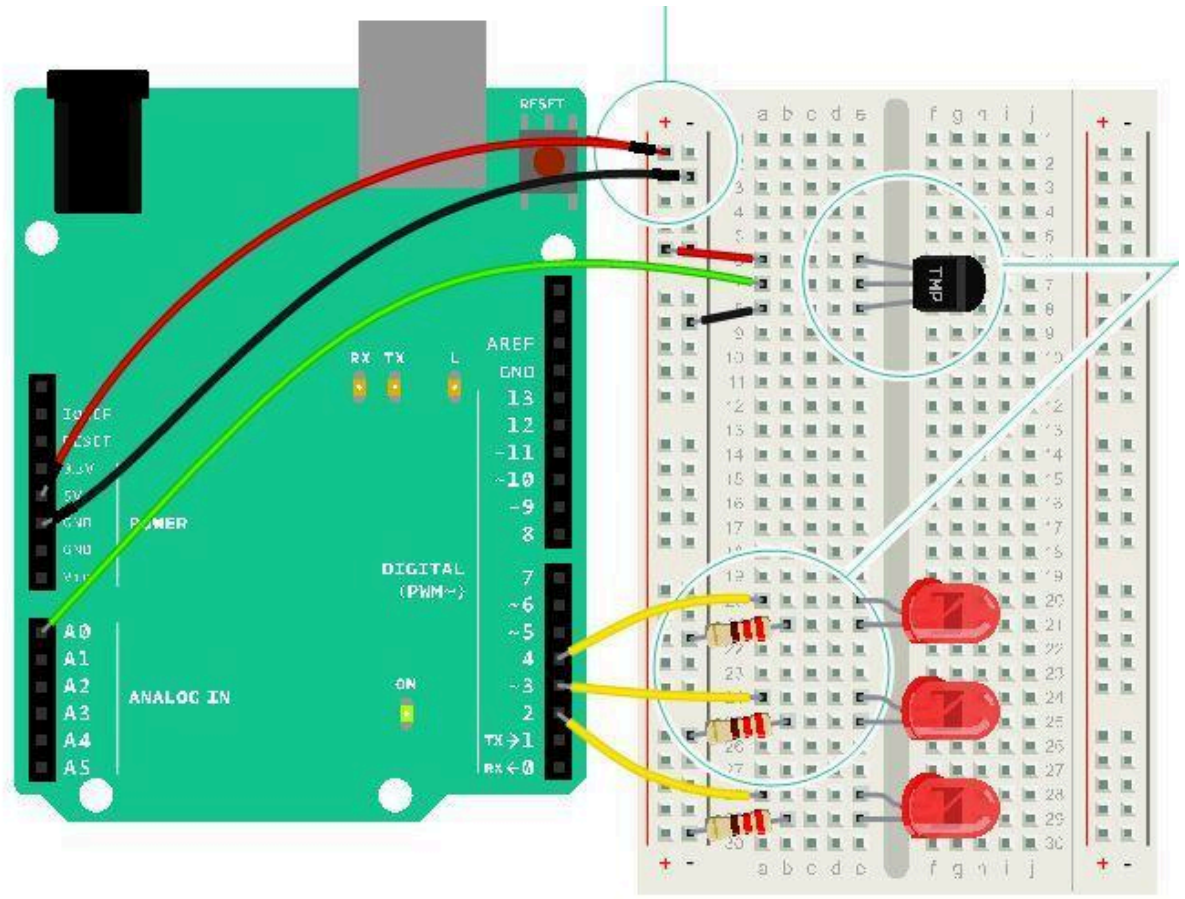

Figure 4.1: LED Circuit Connection (Fitzgerald & Shiloh, 2012)

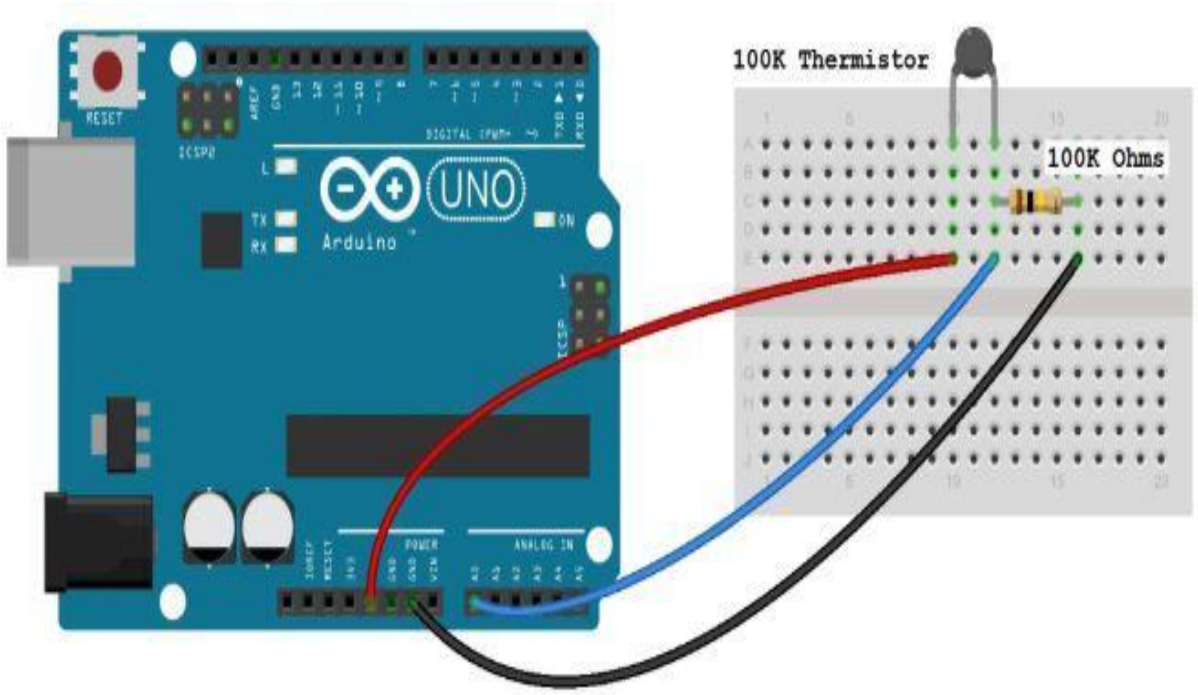

Figure 4.2: Thermistor Circuit Connection (Circuit Basics, 2016)

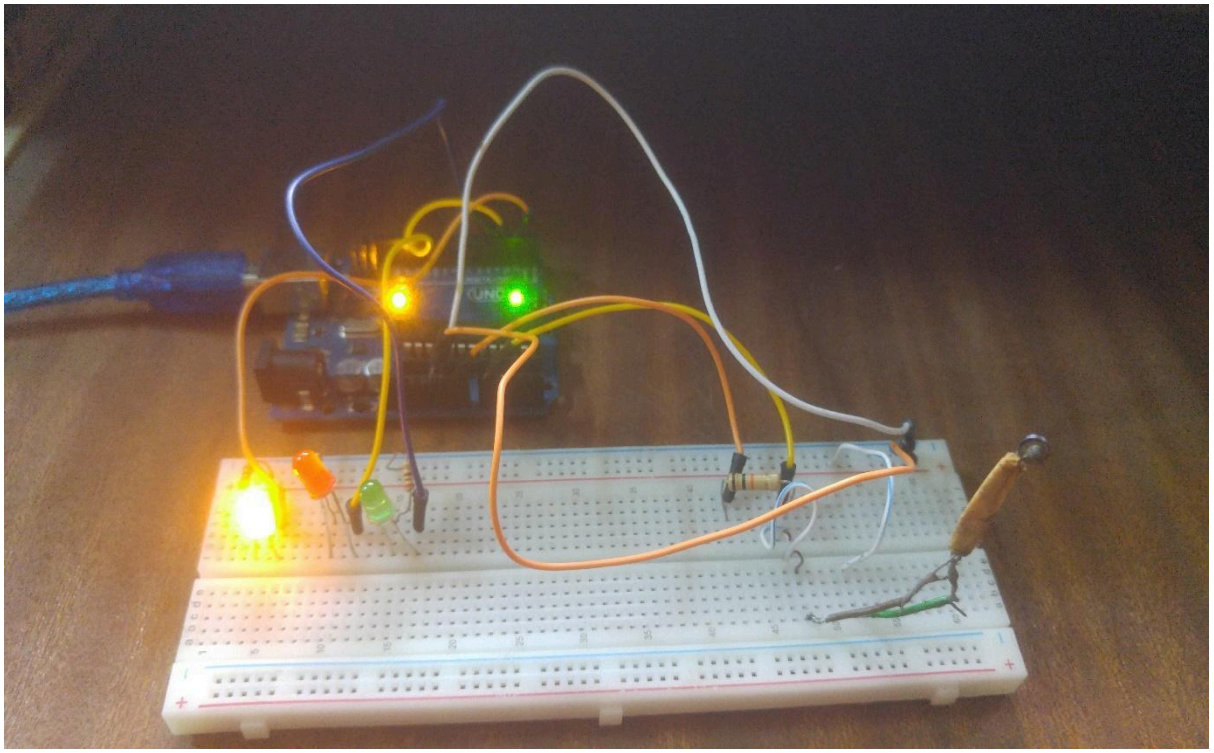

Figure 4.3: Final Hardware Implementation.

4.2 SOFTWARE IMPLEMENTATION

4.2.1 ARDUINO IDE

The IDE stands for Integrated Development Environment and it is the platform used to develop code highlighting instructions for the Arduino microcontroller. Arduino IDE makes use of its own Arduino syntax which is similar to that of the C programming language. A figure showing the Arduino IDE and the code is shown on the next page.

4.2.2 ARDUINO CODE

The code used in the Arduino is discussed below:

i. Two variables hot and cold were created to assign a constant value to the temperature when it is hot and when it is cold i.e. above or below a certain threshold.

ii. In the setup section, the pin specifications were made with each pin corresponding to an LED. This tells the Arduino where each pin is connected and where any signal is to be sent to on the Arduino.

iii. Also, in the setup section, the Serial.begin is initialised which tells the Arduino to begin operation.

iv. In the loop section, the code to indicate where Analog readings from the thermistor are to gotten from in this case this is the Analog port A0. The Analog readings are then converted to integer readings using the **int** statement. The other code in this section consists of the if statement to indicate at which temperature each LED should be ON.

Figure 4.4: The Arduino Integrated Development Environment.

```
void loop() {
  // put your main code here, to run repeatedly:
  int sensor = analogRead(A0);
  float tempC = (sensor * 0.0833835);
  //Serial.print("Temperature: ");
  Serial.println(tempC);
  //Serial.print(" ");
  if (tempC < cold) {
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    //Serial.print(" It's cold.");
  }
  else if (tempC >= hot) {
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    // Serial.print(" It's hot.");
  }
  else if (cold < tempC <= hot){
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    // Serial.print(" Its warm.");
  }
  delay(1000);
}
```

```
Done uploading.

Sketch uses 3634 bytes (11%) of program storage space. Maximum is 32256 bytes.
Global variables use 200 bytes (9%) of dynamic memory, leaving 1848 bytes for local variables. Maximum is 2048 bytes.
```

Figure 4.5: The Arduino Code.

4.3 Results

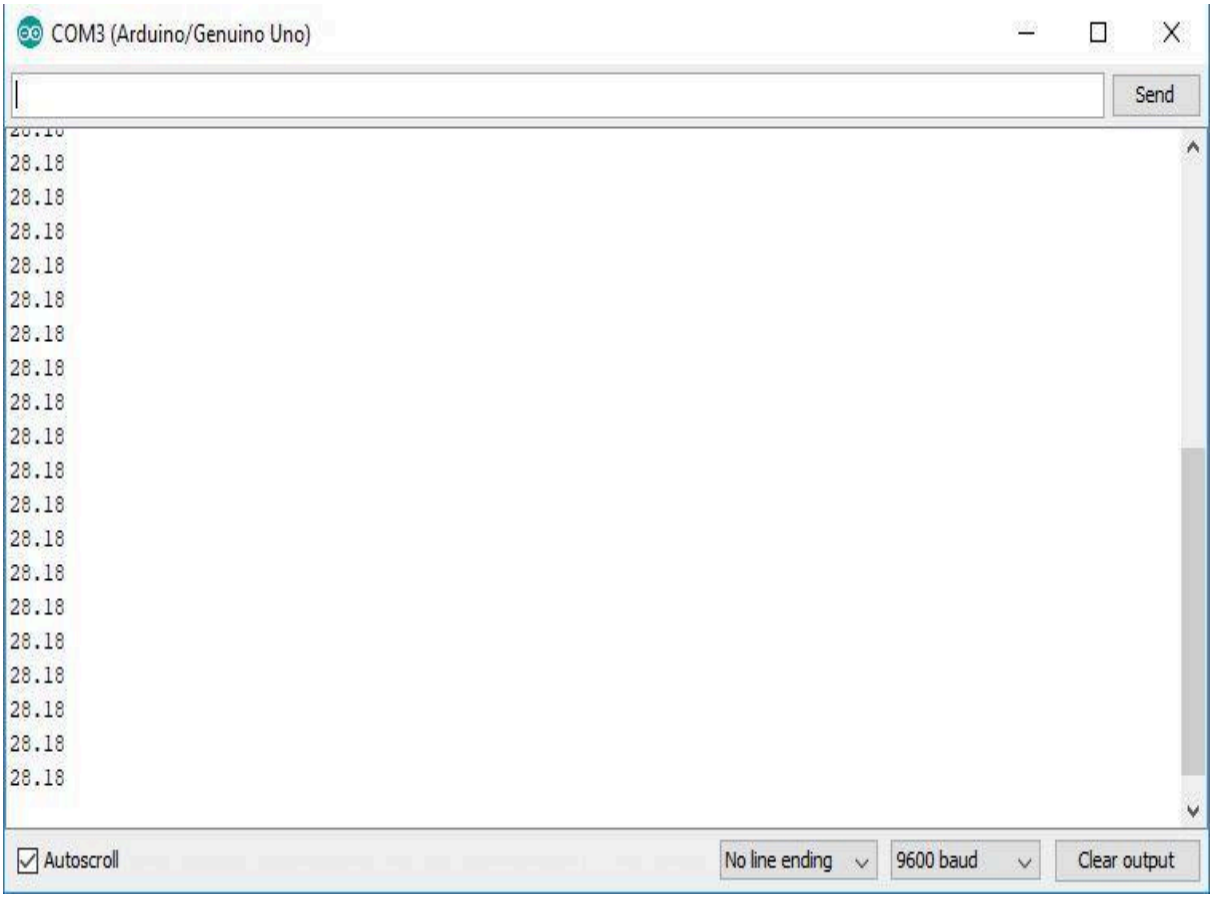

Figure 4.6: The Temperature Data Readings in the Arduino Serial Plotter.

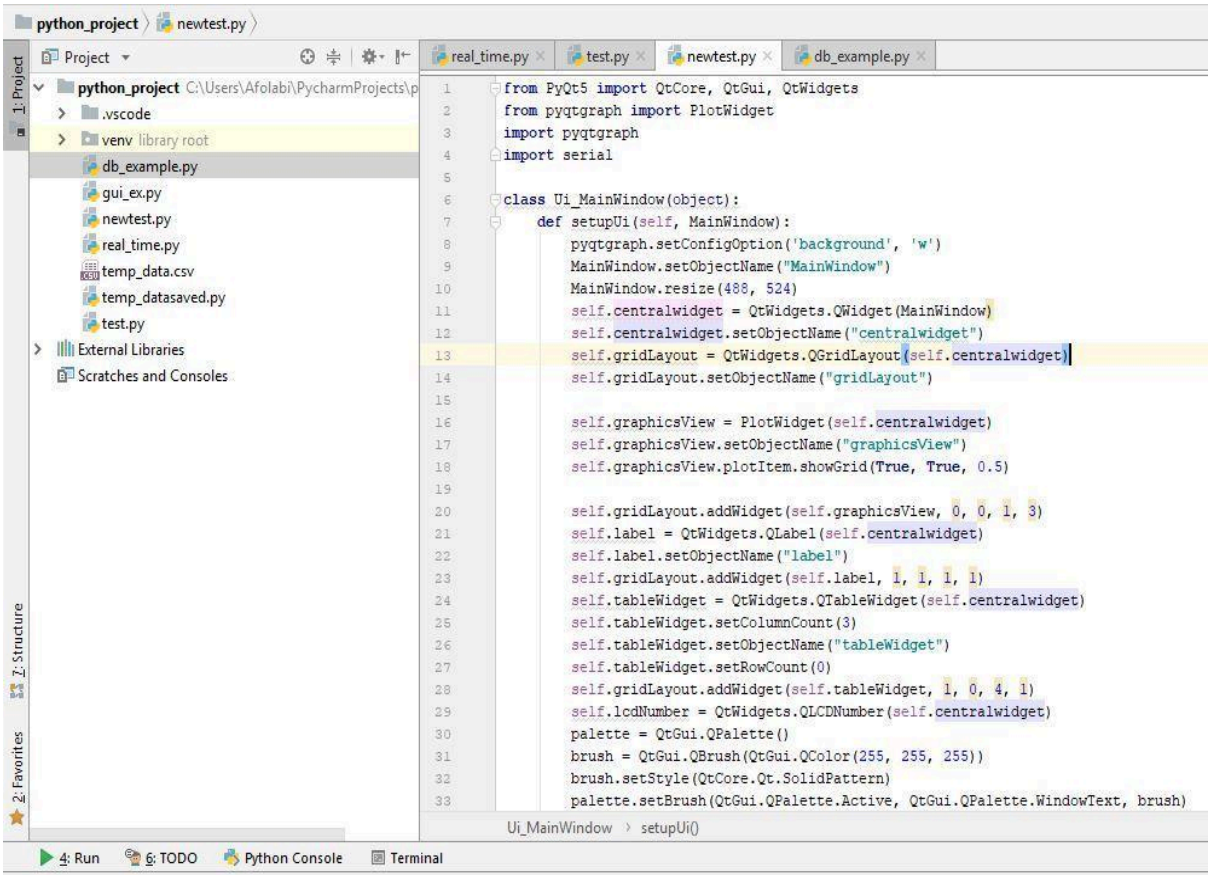

Figure 4.7: Python Code in PyCharm Development Environment.

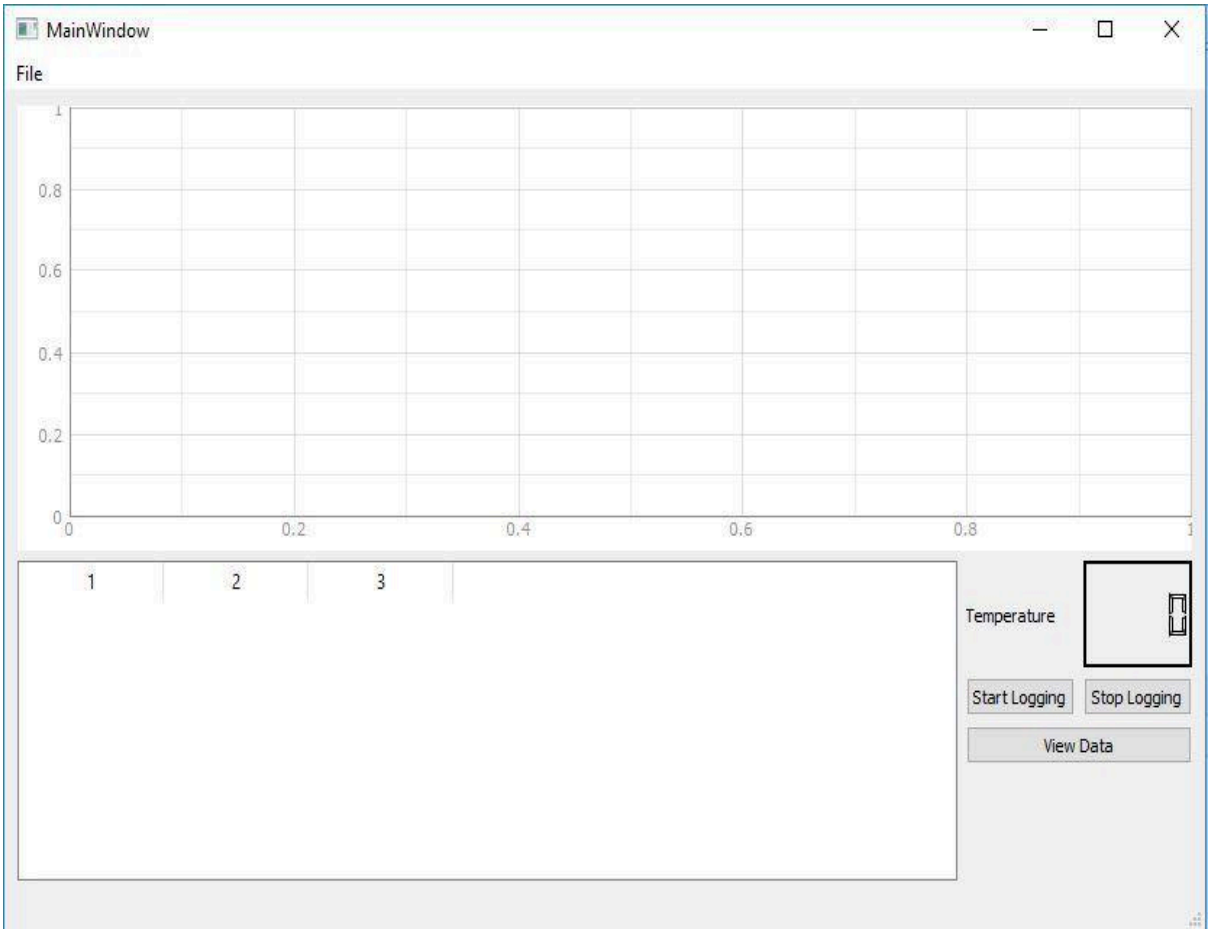

Figure 4.8: The Completed Python Program.

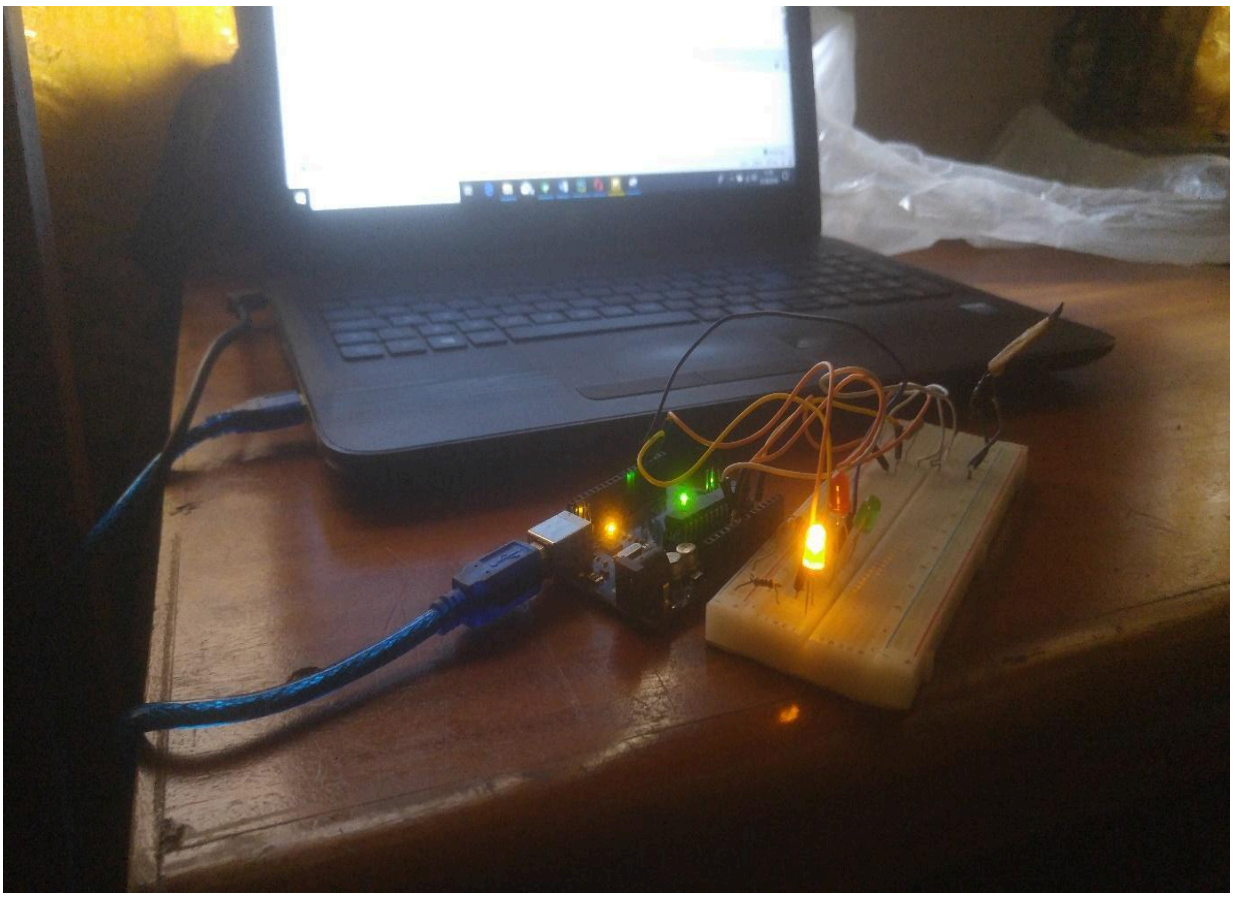

Figure 4.9: Hardware Connected to the USB Serial Port

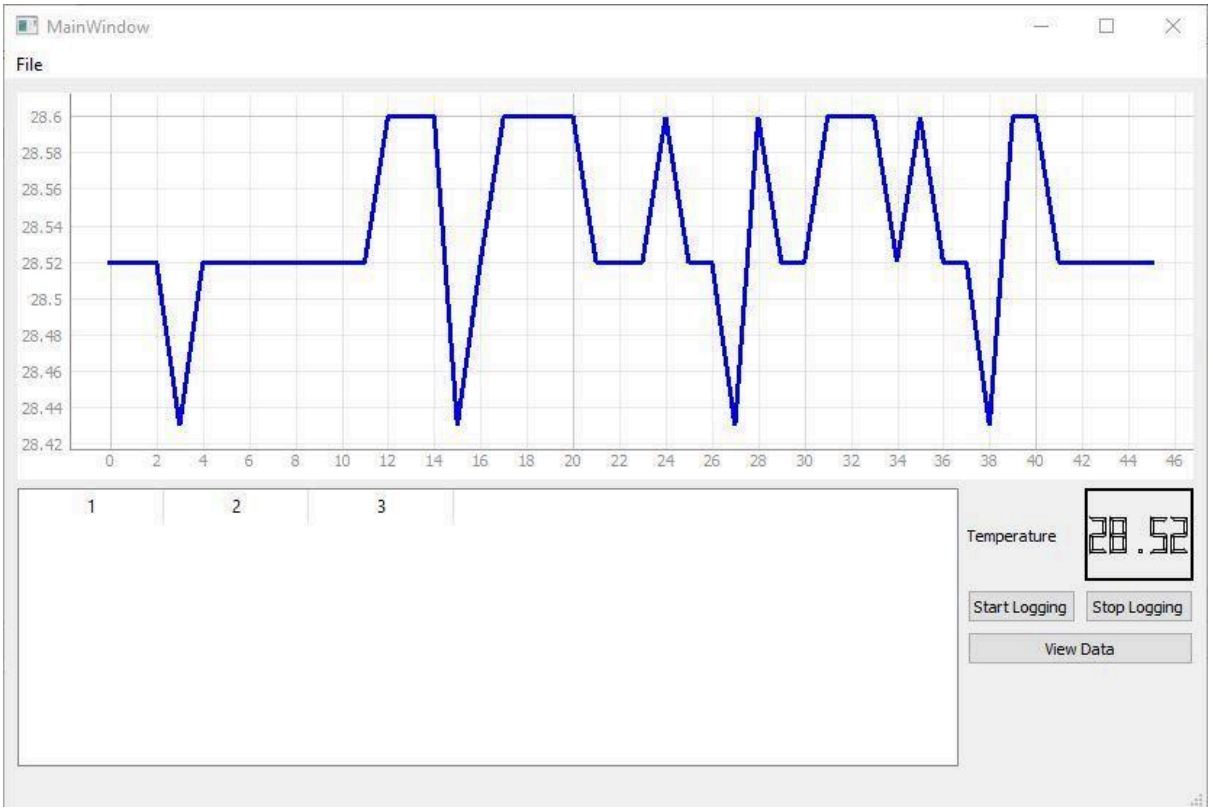

Figure 4.10: GUI Reading Data from the Arduino Microcontroller.

4.4 Results Analysis

In the previous figures shown under results, the temperature sensor circuit was connected to the USB serial port of the laptop. The python program was launched and in Fig 4.8, the python code can be seen. The readings of the temperature can be seen in Fig 4.6 in the Arduino serial plotter. These readings are sent to the python program through the USB serial port as shown in Fig 4.9 where it is connected to the hardware. In Fig 4.10, the graph is plotted in the python graphical user interface and the temperature is shown alongside it.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.0 Overview

This chapter contains a summary of the project, problems encountered during the course of finishing the project as well as recommendations for how the project can be improved.

5.1 Summary

This project entails the design of a Graphical User Interface to display temperature data in real-time using the python programming language. The project involves the use of particular python packages specifically the PyQt5, PyQtGraph and the PySerial packages. Each package has a certain use such as the PyQt5 package which provides the interface through which the user interface is displayed, the PyQtGraph package provides the tools required to display the data graphically while the PySerial package provides the means through which the temperature readings are read through the USB serial port of the PC.

5.2 Challenges

The challenges encountered during the course of this project are discussed below with the solutions used to deal with such problems:

I. Low number of examples to facilitate the building of the GUI faster. The python programming language is suitable for data analysis and evaluation but documentation on how to build a complete GUI is not available. The solution I made use of is going on websites where questions are posted and answered such as Quora and stack exchange. Then after collecting the answers, I was able to build a full-fledged GUI.

II. Problem while trying to get the temperature data to display graphically. The python program was able to collect temperature data and store it but it initially lacked the

ability to display the data graphically. This was solved by changing the package being used from Matplotlib to PyQtGraph. PyQtGraph made the integration of python code to the Arduino data more seamless with less errors when trying to display the data graphically.

5.3 Conclusion

In conclusion, this software system provides a way to view and monitor temperature data through a graphical display and provides a way to store temperature data. This project also showed practical applications of courses learned such as Object Oriented Programming.

5.4 Recommendations

- The addition of a networked interface can be included where remote access is provided to a health practitioner to view this data and also sent to a hospital database and automatically added to a patients file.
- The addition of other monitoring systems to collect more data such as heart rate monitoring system to provide more detailed medical information for health practitioners.
- The addition of a database is also suggested to provide access to the data over the internet.

REFERENCES

Alinafe Kaliwo, Jonathan Pinifolo, Chomora Mikeka (2017). Real-Time, Web-based Temperature Monitoring System for Cold Chain Management in Malawi.

Al-thobaiti, B. M., Abosolaiman, I. I., Alzahrani, M. H., Almalki, S. H., & Soliman, M. S. (2014). Design and implementation of a reliable wireless Real-Time home automation system based on Arduino uno single-board microcontroller. International journal of control, Automation and systems, 3(3), 11-15.

Arduino, S. A. (2015). Arduino. Arduino LLC.

Badamasi, Y. A. (2014, September). The working principle of an Arduino. In Electronics, computer and computation (icecco), 2014 11th international conference on (pp. 1-4). IEEE.

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1), 26-30.

Bokingkito, P. B., & Llantos, O. E. (2017). Design and Implementation of Real-Time Mobile-based Water Temperature Monitoring System. Procedia Computer Science, 124, 698-705.

Campagnola, L. (2016). PyQtGraph-scientific graphics and GUI library for python.

Duda, P. (2017). Development and application of the power plant real-time temperature and stress monitoring system. Science and Technology of Nuclear Installations, 2017.

Ege, Y., Kalender, O., Çıtak, H., Nazlıbilek, S., & Çoramık, M. (2015). New real time temperature monitoring and evaluation system.

Eva Rothgang, Wesley D. Gilson, Li Pan, Jörg Roland, Klaus J. Kirchberg, Frank Wacker, Joachim Hornegger, and Christine H. Lorenz (2012). An Integrated System for MR-Guided Thermal Ablations: From Planning to Real-Time Temperature Monitoring.

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). The Arduino projects book. Arduino.

Graphics, P. S. (2013). GUI Library for Python. Retrieved March, 4.

Ibrahim, M. Y., & Audah, L. (2017, September). Real-time bus location monitoring using Arduino. In AIP Conference Proceedings (Vol. 1883, No. 1, p. 020016). AIP Publishing.

Haider, Z., Mehmood, F., Guan, X., Wu, J., Liu, Y., & Bhan, P. (2015, May). Scheduling of air conditioner based on real time price and real-time temperature. In Control and Decision Conference (CCDC), 2015 27th Chinese (pp. 5134-5138). IEEE.

Kelekar, A. (2018). Towards Better Open-Source Development: Improving PyQtGraph's Feature-Development Process.

Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. In International Conference on Information and Network Technology (Vol. 37, No. 1, pp. 162-167).

Millman, K. J., & Aivazis, M. (2011). Python for scientists and engineers. Computing in Science & Engineering, 13(2), 9-12.

Model, W., Model, S., & Model, I. (1991). Object-oriented design.

Ogidan, O. K., Bamisaye, A. J., & Adetan, O. (2011). Ovulation detection mechanism—a microcomputer based approach. Journal of Biomedical Science and Engineering, 4(12), 769.

Pina-Martins, F., & Paulo, O. S. (2016). Ncbi mass sequence downloader–large dataset downloading made easy. SoftwareX, 5, 80-83.

Rajesh Daga, Mahendra Kumar Samal (2013). Real Time Monitoring of High Temperature Components.

Real-Time Computing (2010),

https://www.techopedia.com/definition/16991/real-time-computing-rtc. Accessed on 15/05/2018.

Repici, J. (2010). The comma separated value (CSV) file format. Creativyst Inc.

Shin, K. G., & Ramanathan, P. (1994). Real-time computing: A new discipline of computer science and engineering. Proceedings of the IEEE, 82(1), 6-24.

Singh, N. M., & Sarma, K. C. (2012a). Low cost PC based real time data logging system using PCs parallel port for slowly varying signals. arXiv preprint arXiv:1207.2739.

Singh, N. M., & Sarma, K. C. (2012b). Design and development of low cost pc based real time temperature and humidity monitoring system. arXiv preprint arXiv:1207.3433.

Software development Life cycle and methodologies (2012),

https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/. Accessed on 15/05/2018.

Stankovic, J. A. (1992). Real-time computing. Byte, pág, 155-162.

Tulchak, L. V., & Marchuk, A. O. (2016). History of python (Doctoral dissertation, ВНТУ).

APPENDIX

```
// Code for Project Implementation

from PyQt5 import QtCore, QtGui, QtWidgets

from pyqtgraph import PlotWidget

from time import sleep

import pyqtgraph

import serial


class Ui_MainWindow(object):

    def setupUi(self, MainWindow):

        pyqtgraph.setConfigOption('background', 'w')

        MainWindow.setObjectName("MainWindow")

        MainWindow.resize(488, 524)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)

        self.gridLayout.setObjectName("gridLayout")


        self.graphicsView = PlotWidget(self.centralwidget)

        self.graphicsView.setObjectName("graphicsView")
```

```
self.graphicsView.plotItem.showGrid(True, True, 0.5)



self.gridLayout.addWidget(self.graphicsView, 0, 0, 1, 3)

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setObjectName("label")

self.gridLayout.addWidget(self.label, 1, 1, 1, 1)

self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)

self.tableWidget.setColumnCount(3)

self.tableWidget.setObjectName("tableWidget")

self.tableWidget.setRowCount(0)

self.gridLayout.addWidget(self.tableWidget, 1, 0, 4, 1)

self.lcdNumber = QtWidgets.QLCDNumber(self.centralwidget)

palette = QtGui.QPalette()

brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))

brush.setStyle(QtCore.Qt.SolidPattern)

palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.WindowText, brush)

brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))

brush.setStyle(QtCore.Qt.SolidPattern)

palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)

brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
```

```
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Light, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Midlight, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Dark, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Mid, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Text, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.BrightText, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.ButtonText, brush)
```

```
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Shadow, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.AlternateBase, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.ToolTipBase, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.ToolTipText, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
```

```
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.WindowText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Light, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Midlight, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Dark, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Mid, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Text, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
```

```
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.BrightText, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.ButtonText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Shadow, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.AlternateBase, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.ToolTipBase, brush)
```

```
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.ToolTipText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.WindowText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Light, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Midlight, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Dark, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
```

```
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Mid, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Text, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.BrightText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.ButtonText, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Shadow, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
```

```
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.AlternateBase, brush)
brush = QtGui.QBrush(QtGui.QColor(255, 255, 220))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.ToolTipBase, brush)
brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
brush.setStyle(QtCore.Qt.SolidPattern)
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.ToolTipText, brush)
self.lcdNumber.setPalette(palette)
self.lcdNumber.setAutoFillBackground(False)
self.lcdNumber.setObjectName("lcdNumber")
self.gridLayout.addWidget(self.lcdNumber, 1, 2, 1, 1)
self.pushButton = QtWidgets.QPushButton(self.centralwidget)
self.pushButton.setObjectName("pushButton")

self.pushButton.clicked.connect(self.update)

self.gridLayout.addWidget(self.pushButton, 2, 1, 1, 1)
self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_2.setObjectName("pushButton_2")
```

```
# self.pushButton_2.clicked.disconnect(self.pushButton)


self.gridLayout.addWidget(self.pushButton_2, 2, 2, 1, 1)
self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_3.setObjectName("pushButton_3")
self.gridLayout.addWidget(self.pushButton_3, 3, 1, 1, 2)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 488, 21))
self.menubar.setObjectName("menubar")
self.menuQuit = QtWidgets.QMenu(self.menubar)
self.menuQuit.setObjectName("menuQuit")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
self.actionQuit = QtWidgets.QAction(MainWindow)
self.actionQuit.setObjectName("actionQuit")
self.menuQuit.addSeparator()
```

```
    self.menuQuit.addAction(self.actionQuit)
    self.menubar.addAction(self.menuQuit.menuAction())


    self.retranslateUi(MainWindow)
    QtCore.QMetaObject.connectSlotsByName(MainWindow)


def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
    self.label.setText(_translate("MainWindow", "Temperature"))
    self.pushButton.setText(_translate("MainWindow", "Start Logging"))
    self.pushButton_2.setText(_translate("MainWindow", "Stop Logging"))
    self.pushButton_3.setText(_translate("MainWindow", "View Data"))
    self.menuQuit.setTitle(_translate("MainWindow", "File"))
    self.actionQuit.setText(_translate("MainWindow", "Quit"))


def update(self):
    arduino_data = serial.Serial('COM3', 9600)
    temp_f = []
```

```
def update_data():
    while True:
        arduino_string = arduino_data.readline()
        temperature = float(arduino_string)
        temp_f.append(temperature)
        self.lcdNumber.display(temperature)
        if temperature > 50:
            pen = pyqtgraph.mkPen(color='r', width=3)
        else:
            pen = pyqtgraph.mkPen(color='b', width=3)
        self.graphicsView.plotItem.plot(temp_f, pen=pen, clear=True)
        QtGui.QGuiApplication.processEvents()
        # QtCore.QTimer.singleShot(0, self.update)


t = QtCore.QTimer()
t.timeout.connect(update_data)
t.start(50)
update_data()
```

```
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```