2016

# Understanding and Design of an Arduino-based PID Controller

Dinesh Bista
*Virginia Commonwealth University*

# Understanding and Design of an Arduino-based PID Controller

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science at Virginia Commonwealth University

By

Dinesh Bista

Masters of Science, Physics, Virginia Commonwealth University

Director: Dr. Jason Reed, Assistant Professor, Physics Department

Virginia Commonwealth University

Richmond, VA

Fall 2016

# Acknowledgment

I am thankful to my parents, sisters, and my girlfriend for their love and constant support. I wish to thank Dr. Jason Reed for his guidance throughout this project. I am grateful to my colleague Daniel Guest and my friend Vikash Chauhan for their help and some healthy discussions during my project. I would also like to express my gratitude to Dr. Marilyn Bishop and Dr. Shiv Narayan Khanna for their inspiration. Finally, I would like to express my appreciation to the entire VCU Physics Department for all their support and help.

# Table of Content

List of Tables…….…….....v

List of Figures……...vi

List of Abbreviations and symbols….....viii

Abstract…….....ix

I. Introduction…….…1

II. Background Literature Review

II.i: Literature Review of the PID Controller……..…….……...…4

II.ii: A Practical PID Controller ……..……..…8

II.iii: Discussion of some research papers on Arduino based PID controllers……..13

II.iv: Conclusion from the Literature Review ……19

III. Temperature Controller System with a PID Control

III.i: Description of the components in the system …….…..20

III.ii: Description of the System Design……..……26

IV. Observations and Conclusions

IV.i: Noise and Error …….....32

IV.ii: Setup, Procedure, and Observations……...36

1. On/Off Controller…….....37
2. P-Controller……..…….……...38
3. PI- Controller ……..…….43

IV.iii.Conclusion……..……..……..48

V. Appendix

Appendix 1A……..……..……50

Appendix 1B……..……..…51

Appendix 1C……..……..……52

Appendix 2 ……..……53

Bibliography …….....56

# List of Tables

Table 1.a: List of comparisons of response parameters of all designed P-Controllers……..…42

Table 1.b: List of comparisons of response parameters of P-and PI-Controllers……..…47

Table 1.c: List of comparisons of response parameters of PI-Controllers……..…..…47

Table A1C: Devices and Appliances used in building of our temperature controller system….52

# List of Figures


Fig 1.1: Schematic of PID controller with a feedback loop…….…….4

Fig 1.2: Measure of controller's performance…….....11

Fig 3.1: Duty Cycles ……..……...…….21

Fig 3.2: TMP36 analog temperature sensor ……...22

Fig 3.3: Working mechanism of H-bridges ……...23

Fig 3.4: Arduino UNO Board …….…...24

Fig 3.5: Schematic of the designed temperature controller system ……...26

Fig 3.6a: Temperature controller system……..……29

Fig 3.6b: Sample attached to the heatsink and the temperature sensor ……...29

Fig 3.6c: View of hot-side of TEC being attached to the sample ……..…….…….30

Fig 3.6d: H-bridges…….…….. …….……..30

Fig 3.6e: Operational amplifiers used in the system ……..……..……31

Fig 4.1: Plot of analog input voltage vs digital output of signal from TMP36 sensor error ……..…….....34

Fig 4.2: RMS measure of the signal from temperature sensor .....35

Fig 4.3: Plots of Temperature and Controller's Output vs Time of On/Off Controller ……..…….…….……..37

Fig 4.4: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$ =150 ……..…..…….39

Fig 4.5: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$=1500 …….…….……40

Fig 4.6: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$= 4300 ……..41

Fig 4.7: Plots of Temperature and Controller's Output vs Time of PI-Controller with $K_p$=150 and $K_i$=1……..……44

Fig 4.8: Plots of Temperature and Controller's Output vs Time of PI- Controller with $K_p$=150 and $K_i$=3.....45

Fig 4.9: Plots of Temperature and Controller's Output vs Time of PI-Controller with $K_p$=1500 and $K_i$=0.1.....46

Fig A1: Arduino UNO.....50

Fig A2: Closer view of TEC being attached to the sample …..……..…..…..….....50

Fig B1: Working mechanism of Peltier pellets……..…...…….51

# List of Abbreviations and Symbols

| | |
|---|---|
| ADC : | Analog to Digital Converter |
| AWU : | Anti-Wind Up |
| D : | Derivative |
| DC : | Direct Current |
| DOF: | Degree Of Freedom |
| HSAFM: | High Speed Atomic Force Microscope |
| $K_p$: | Proportional Gain or Controller's Gain |
| $K_i$: | Integral gain |
| $K_d$: | Derivative gain or parameter |
| LED : | Light Emitting Diode |
| P : | Proportional parameter |
| PID: | Proportional Integral Derivative |
| PV: | Process Variable |
| PWM: | Pulse Width Modulation |
| RMS: | Root Mean Square |
| SLFC: | Self-Learning Fuzzy Controller |
| SSE: | Steady State Error |
| TEC: | Thermo-Electric Cooler |
| ZN: | Ziegler Nichols |

# Abstract

UNDERSTANDING AND DESIGN OF AN ARDUINO-BASED PID CONTROLLER

By

Dinesh Bista, MS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science at Virginia Commonwealth University,

Virginia Commonwealth University, 2016

Director: Dr. Jason Reed, Assistant Professor, Physics Department

This thesis presents research and design of a Proportional, Integral, and Derivative (PID) controller that uses a microcontroller (Arduino) platform. The research part discusses the structure of a PID algorithm with some motivating work already performed with the Arduino-based PID controller from various fields. An inexpensive Arduino-based PID controller designed in the laboratory to control the temperature, consists of hardware parts: Arduino UNO, thermoelectric cooler, and electronic components while the software portion includes C/C++ programming. The PID parameters for a particular controller are found manually. The role of different PID parameters is discussed with the subsequent comparison between different modes of PID controllers. The designed system can effectively measure the temperature with an error of $\pm$ 0.6℃ while a stable

temperature control with only slight deviation from the desired value (setpoint) is achieved. The designed system and concepts learned from the control system serve in pursuing inexpensive and precise ways to control physical parameters within a desired range in our laboratory.

# **I.** Introduction

Modern industry, scientific workstations, robotics, and regular activities have been greatly aided by the use of control systems. The extensive use of the control systems is evident in the cruise control of vehicles, the temperature control of baby incubators, the temperature and humidity control of cell incubators, the mobile control of robots, and many more applications in countless scientific research and industries. The PID algorithm is a simple process, which is easy to understand conceptually and implement practically. The advantageous cost/benefit ratio provided by the PID controllers makes them the most frequently used control tools in industry.[1] The PID controllers have been extensively used since 1980s for the control engineering practice. The PID controller has been suggested as the second most important control decision and communication instrument of the 20$^{th}$ century only behind the "Microprocessors."[2] The cheaper price and the compatibility in interfacing with the advanced computing programs like MATLAB and LabVIEW make Arduino the preferred microcontroller for teaching.

The research in our laboratory focuses on the study of human cells: cancer cells, especially of skin cancer cell, melanoma (cell lines M238, M229, and M249), and other cells which are part of human immune systems (T-cell and Mast cell). We observe the rate of change in mass of the cancer cells and the change in absolute mass of the immune cells under the influence of drugs. The cells are placed inside the cell incubator, Steri-Cult $CO_2$ Incubator. The optimum conditions for temperature, % of $CO_2$, and the humidity for the cells in this incubator are 37℃, 5%, and above 60% respectively. The interferometric phase imaging camera, Phasics SID4BIO, which gives the

[1] Vilanova, R., & Visioli, Antonio. (2012). *PID control in the third millennium lessons learned and new approaches* (Advances in industrial control). London; New York: Springer.

[2] Rhinehart, R. (2000). The century's greatest contributions to control practice. *ISA Transactions, 39*(1), 3-13.

interferometric phase images of the cells and color camera, Basler acA645-100gc, which helps to focus on the cells and normalize the color image, are placed inside the cell incubator. The setpoint temperature (desired temperature) for the cell incubator is 37℃. The upper limit is very sensitive, as passing beyond 37.25℃ is not desirable. However, the lower limit is a bit flexible, about 36℃ . Hence, the temperature range of 36.50℃ ± 0.50℃ is preferable for the cell incubator. We are using a separate temperature controller, product of Omega platinum series instead of using the in-built temperature controller system of the incubator. This separate temperature controller system gives temperature reading with an accuracy up to two decimal places. This extra care is essential because of the heat generated inside the incubator (system) due to the following reasons: the motion of small motors adjusting the focus position at each imaging location (basically three motors to vary the objectives in x, y and z directions) and the working of cameras.

The phase imaging camera and the color camera used in the cell incubator become heated. The heating of the cameras contributes to the heating of the incubator in the short term when the experiment is carried out. However, in the long run the performance of the cameras degrades leading to the breakdown of cameras. Hence, they need to be cooled separately. Furthermore, we are trying to learn using a new and faster technique to control the focus of microscope by a laser beam reflection off of the sample in our laboratory. The applications of the control systems in the live cell-interferometry and live cell-imaging has significantly motivated me to learn and design a control system. The project work is split into following three parts:

The first part is the introductory part that discusses the background research of the project. This part starts with the description of PID parameters and the characteristics of a practical PID controller. The research analyzes the Arduino-based PID controller over the last few years. The

use of PID controller in different arenas is discussed with basic focus on the interfacing and optimization of a PID controller to get the better performances. The second part deals with the designing of our system, an Arduino-based temperature controller system using the PID algorithm. The codes written in C programming serve as the software portion, while the hardware portion includes Arduino UNO, electronic components, and solid state devices. The tuning of the PID controller is done manually, which is also called a hit and trial method. In the final part, the observations of different kinds of controllers are made through the experimental data and plots. The characteristics of various modes of controllers are observed. The logical interpretations justifying the experimental observations are presented. Finally conclusions are drawn based on the experimental observations with some discussions on the possible applications of the designed controller in our laboratory.

# **II.** Background Literature Review

## **II.i.** Literature Review of PID controller

There are various types of controllers used in industry, laboratory, and routine applications. Some of the commonly used controllers are on/off, PID, fuzzy, and neural. The latter two are a bit complex, and use more sophisticated concepts like artificial intelligence. The controllers can also be differentiated as "feedforward" and "feedback" controllers. The feedforward controller works by giving a result based on the anticipation of the next step, while the feedback controller works by giving an observed result that changes the processing value of the later step. In this project, the entire focus is on the feedback control system that uses the PID algorithm. PID stands for proportional integral and derivative.

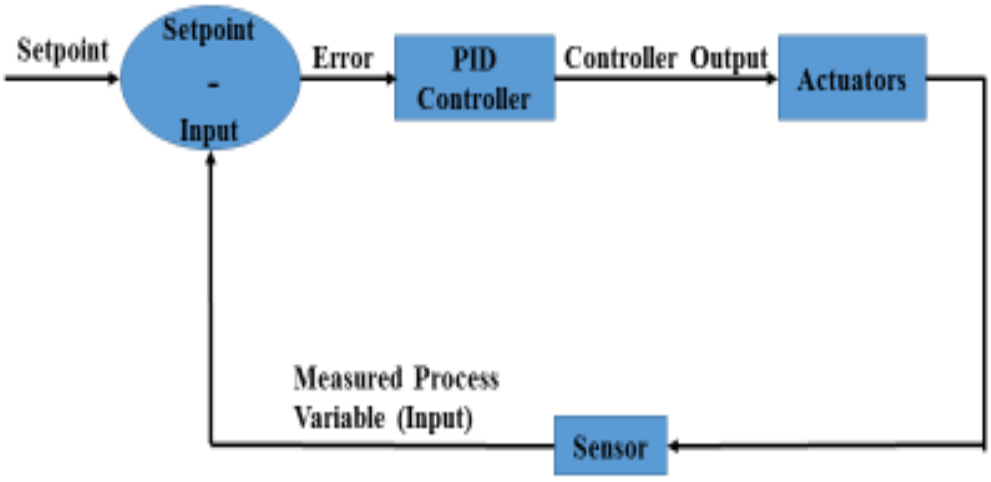


**Fig 1.1 Schematic of a PID controller with a feedback loop: The sensor block measures the process variable (PV) of a given process, which is compared to the setpoint (SP) to find the error. Observed error is used to find the controller output, which in turn is sent into the Actuators block.[3]**

[3] Peacock, F. (2008). How the PID Algorithm works and why it works. Retrieved from www.PID-Tuning.com

The PID controller uses a feedback loop, which controls a parameter of the system called the process variable (PV), which is supposed to match the desired output, the setpoint (SP). A PV is any physical quantity of the system that can be measured using the sensor and can be controlled. The PID controller uses an "error," the difference between SP and PV, in each loop to control the system. It is more like a machine that measures the error in each loop and gives the necessary signals to an actuator, a device that works according to the received signals from the controller, to minimize the error. Each term of a PID algorithm has its own characteristics and roles, which are described in detail as followed:

**P-Term**: This takes into account the present error only. The P-Term makes an effort in proportion to how far the PV is from the SP at the present time. However, approaching closer to the SP, the error becomes so small that the controller cannot trigger the PV enough to catch the SP, which implies that there always exists a steady state error (SSE), which appears as an offset from the setpoint in the system. However, a larger value of the proportional term can trigger the PV to the setpoint, but it makes the system unstable with oscillations and overshoots. Thus the controller's response in such case behaves more like the response coming from an on/off controller. Thus, the P-controller alone is not sufficient for the most control designs. Therefore, the P-Term is generally accompanied by an "I-term," which makes the controller become a PI-controller. Mathematically, the output of the P-controller is given by the equation:

$$\text{P-Term} = K_p . \, e\,(t) \qquad \ldots\ldots (1)$$

Where $K_p$ = Proportional gain and e (t) = error at the present time "t"

**I-Term:** This is the most important term in the PID controller. It takes into account all the errors present in the system from the starting point to that particular point of time in the process. It looks at the history of the error until that specific point. Mathematically, the I-Term is represented by the equation:

$$\text{I-Term} = \int_0^{\tau} K_i . e(t) dt \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

Where $K_i =$ integral gain and $\tau =$ total time of operation of the controller

The contribution from the I-Term tries to balance the difference in the time spent by the PV on the both sides of the setpoint, *i.e.* below and above the setpoint. For example, if PV spends 10 seconds running at 98% of the setpoint value, the I-Term will try to push the PV over to 102% for the other 10 seconds to compensate. Hence, there is an overshoot. For any sensitive system, the overshoot can seriously damage the whole structure of the system. However, the PI controller is still better than PID controller for most control systems due to its simplicity in operation. However, it is important to make sure that the integral windup has been completely detached from the controller. The integral windup refers to the saturated values of the integral term in either direction.

**D-Term**: This takes the derivative of a PV of the system at every point. Thus, it predicts the future of the operation of the PID loop (this loop is an operation of the PID controller for each iteration, as shown in Fig 1.1). The purpose of this term is to check how the process variable moves without overshooting the setpoint. The D-Term acts on the PI terms by counteracting them. As the PV approaches the setpoint, the PV settles with the set point with a small or no overshoot.

Mathematically, the D-Term is given by the equation:

$$\text{D-Term} = K_d \cdot \frac{d\,e(t)}{dt} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

where, $K_d$ represents the differential gain. However, the D-Term introduces noise and the wearing down of the equipment.[4]

The D-term looks at the fast and the short-term changes in the process variable, like noise that appears as spikes with significant variations. The D-term assumes that there is too much change going around and should be compensated by bringing about drastic undesirable changes to the system. It is better to use the filter to reduce the noise, but over-filtration may also remove the necessary signals and the essential information from the system. Furthermore, the D-Term can destabilize the system if used without proper care. When the dampening action is too high, higher proportional and integral gains are needed for the compensation. Furthermore, getting rid of an offset quickly cannot be achieved due to the derivative's dampening effect. Thus, the user has to set the higher values of the P and I terms for compensation. This way it is responsible for the wearing away of the equipment working as actuators.

It is important to make sure that if the PI-action alone is sufficient for the control designs, we do not use the D-term. If the D-term is required for a system, adding it should be done with a significant amount of care and appropriate use of the filters. The D-term is more desirable generally under the following specific conditions:

I. When the actuators are only one dimensional, i.e. the system can be heated, but cannot be cooled. In such systems, once there is an overshoot, coming back to the setpoint at

[4]Welander, P. (2010, February 1). Understanding Derivative in PID Control. Retrieved from http://www.controleng.com/search/search-singledisplay/understanding-derivative-in-pid-control/4ea87c406e.html

the user's will is very difficult.

II. It is necessary for a slow process, where the overshoot is strictly undesirable, like brewing beverages done at a particular temperature.

## **II.ii.** A Practical PID Controller

A practical PID controller[5] should be able to accomplish its objectives with the following characteristics:

i) Sample time: The controller should have a PID loop that runs through the system in each specific time interval, called the sample time, rather than at any random time interval. Hence, the execution of the controller's effort can be observed and calculated every sample time-interval. The user should be careful while writing the codes when changing the sample time of the PID loop.

ii) Derivative-Kicks: A good PID controller should have a characteristic that enables the user to change the setpoint. Changing the setpoint allows the user to attain different conditions of the system while working. But the change in setpoint changes the error, and that error gets amplified due to its differentiation. A simple mathematical formula that can help to get rid of this problem is as follows:

$$\text{Output (u)} = K_p.\, e\,(t) + \int_0^{\tau} K_i.\, e(t)dt + K_d \cdot \frac{d\, e(t)}{dt} \ldots\ldots\ldots\ldots (4)$$

[5]Bret. (2011, April). Improving the Beginner's PID – Introduction « Project Blog. Retrieved from http://brettbeauregard.com/blog/2011/04/improving-the-beginner-pid-introduction/

$$\text{D-Term} = K_d \cdot \frac{de(t)}{dt} = K_d \cdot \frac{d(SP-PV)}{dt} \ldots\ldots\ldots\ldots\ldots (5)$$

As the setpoint (SP) is constant for a certain operating period,

$$\text{D-Term} = - K_d \cdot \frac{d(PV)}{dt} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

Equation (6) eliminates the SP from the D-Term, so there will not be such a problem while changing the SP. Now the controller output can be written as:

$$u = K_p . e(t) + \int_0^\tau K_i . e(t)dt - K_d \cdot \frac{d(PV)}{dt} \ldots\ldots\ldots\ldots (7)$$

iii) On the-fly Tuning Changes: If the system is not too sensitive to the changes, the tuning parameters can be changed while the PID loop is in operation. Using the PID controller in such a way makes it more flexible and its tuning becomes much easier. However, for a sensitive system, it is always better to tune the PID loop using a transfer function from MATLAB Simulink (The transfer function of a linearly time invariant system is defined as the ratio of the output to the input in the frequency domain.[6]) While changing the PID parameters, due to the integration (summation) of errors in the integral term, an unpredictable output is observed from the controller.

The following change in the I-Term helps in getting a smooth controller output.

$$\text{I-term+} = K_i \cdot e(t) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

[6]Aziz, M. M.(2010). Transfer Function and Block Diagrams. Retrieved from ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ea619_2s12/transfer_function_block_diagram.pdf

This change will make the controller look at the change in $K_i$ only at that specific point, neglecting its past. Finally, the controller output becomes

$$u = K_p.e(t) + K_i \cdot e(t) - K_d \cdot \frac{dPV}{dt} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (9)$$

iv) No Integral and Total Windup: The PID controller cannot exceed certain higher and lower limits. The higher and lower limits of the integral terms must be set in the program such that beyond those limits, the PID loop is programmed to saturate to its corresponding limiting values. Likewise, the total output of the PID controller cannot exceed the limits in either direction. Again, the PID loop is programmed to saturate at the corresponding limiting values of controller output in either direction. This phenomenon is called total anti-windup. It is one of the most important characteristics of a robust PID controller.

For practical purposes, the controller's performance[7] can be analyzed through the following controller's response parameters:

i) Rising Time: The time taken by the controller to trigger the PV to 90% of the setpoint from the 10% value is called the rising time for an underdamped system. [8] A smaller rising time is one of the characteristics of a good controller.

[7] National Instruments. (2011, March 29). PID Theory Explained. Retrieved from http://www.ni.com/white-paper/3782/en/

[8] Levine, W. (1996). *The Control handbook* (Electrical engineering handbook series). Boca Raton, Fl. : [New York, NY]: CRC Press ; IEEE Press.

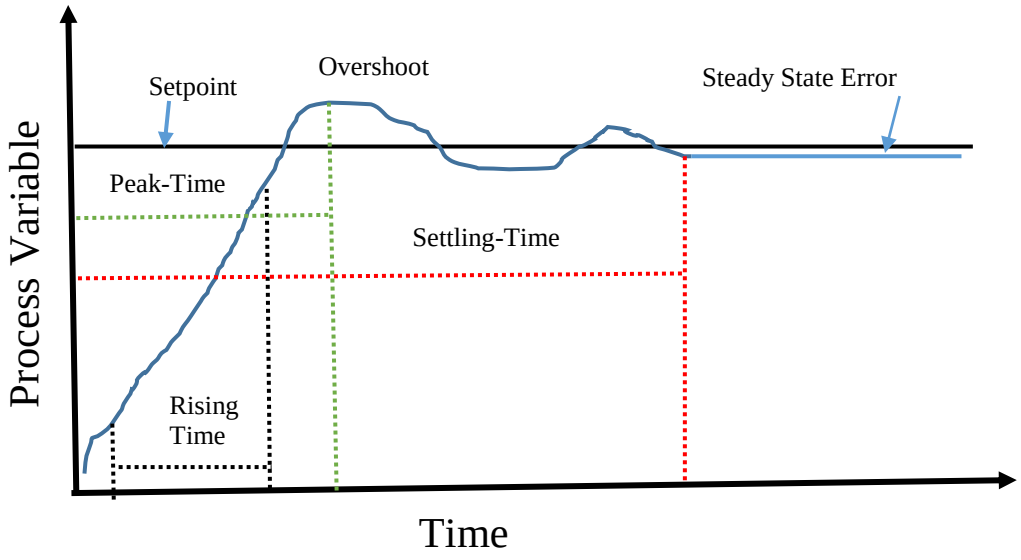


**Fig 2.1 Measures of controller's performance: The vertical axes could represent any physical quantity; the solid curve gives the measure of Process Variable.**

ii) Peak-time: The peak-time can be defined as the time required for the response to reach the maximum value of the process variable for the first time.

iii) Steady State Error (SSE): After the controller's effort, the PV settles to a value closer to the SP. The difference between SP and the settled PV at this particular point is called the steady state error. The smaller the value of the steady state error, the better the controller behaves. The I-Term contributes to the reduction and even elimination of the SSE.

iv) Settling time: The time taken by the controller for the PV to settle to the steady state error is called the settling time. The smaller settling time also implies a better controller in terms of the response rate. Generally, the PID controller has a smaller settling time than the PI controller.

v) Overshoot: While controlling the PV, the controller happens to push the PV beyond the setpoint, the phenomenon is called the overshoot. The percentage overshoot represents the amount

that the response overshoots its steady-state (or final) value at the peak time, which is expressed as a percentage of the steady-state value. It is given by equation (10)

$$\% \text{ overshoot} = \frac{overshoot\ value - settling\ value\ of\ the\ PV}{settling\ value\ of\ the\ PV} \times 100\ \% \quad .....(10)$$

The PI controller is generally regarded as giving a larger overshoot than the PID controller.

While designing a controller system, the aim is to achieve the following properties:

a. Reduction or omission of the steady state error.

b. Reduction of the overshoot (if any).

c. No oscillation (dampening of oscillation is preferred).

d. Small settling time.

## **II.iii.** Discussion of some research papers on the Arduino-based PID controllers

A discussion of some of the exciting works with the Arduino-based PID controllers is made in this section. The Arduino is an open source device which is cheap (price ranges from approximately $24 for an Arduino-UNO to almost $44 for the Arduino-Mega) and is user friendly, making it popular among hobbyists. Moreover, it can be programmed using the simple programming language "C/C++," and can be interfaced easily with other advanced computing environments like MATLAB and LABVIEW (Arduino is discussed in detail in section III.i.5). The range of applications of the Arduino-based PID controller varies from a simple temperature controller to a robot controller. Five different research papers that present different applications of the Arduino-based PID controller and use varying interfaces and tuning methods are discussed below:

**"Design and implementation of a PID controller-based baby incubator"** [9]

In this research paper, the author uses a PID algorithm with Arduino to control the temperature and humidity of a baby incubator. The temperature of the baby incubator is measured, and displayed a using programmed Arduino, and controlled using Pulse Width Modulation (abbreviated as PWM, which is discussed in detail in section III.i.1) through the PID loop. The author emphasizes the use of the PID controller over the simple on/off controller for the following two reasons: fast response, which is important for controlling a sensitive system, and for lower power consumption. The on/off controller consumes higher power as it has to regularly feed the power supply to the heater and fan to switch the system on and off.

[9]Theopaga, A. K., Rizal, A., & Susanto, E. (2014). Design and implementation of PID control based baby incubator. *Journal of Theoretical & Applied Information Technology*, *70*(1).

The experimental part and the tuning portion of this research can be summarized into the following steps:

i.) The temperature of the baby and the incubator environment are measured and displayed separately using an analog Read command (a command used to read the analog values of sensors using the analog pins of the Arduino) on the Liquid Crystal Display (LCD) screen through the coding uploaded in the Arduino.

ii.) The input reading is sent to the controller and compared to the setpoint.

iii.) If the temperature reading is less than the setpoint, the controller orders the heater to turn on through PWM using a "digitalWrite" command. (This command assigns output values, i.e. 0-255 PWM values, to the actuators through the digital pins of the Arduino.) But if the temperature reading is less than the setpoint, the controller instructs the heater to turn off and the fan to turn on again through the PWM signals.

iv.) The most challenging part of this research is tuning the PID controller. The author uses the transfer function of the plant (process) in parallel form. The transfer function is defined as the ratio of the output of the plant to its input in the frequency domain. The frequency domain is obtained by taking the Laplace transform of the transfer function in the time domain. The parallel form implies that all the parameters, proportional, integral, and derivative, are parallel to each other. The author uses the Zeigler-Nichols (ZN) first tuning method. This method uses plant response for the unit step input in the open loop system, which produces an S-curve (S-shaped curve) for time (t) vs output (u). The tangent line drawn at the point of inflection gives the following two parameters: delay time (L) and the time constant (T). The delay time (L) is the distance between the intersection point of the tangent line to the time-axis and the origin. The time constant (T) is

the distance between the perpendicular lines drawn through the tangent line at which the output starts giving the constant value to the time axis, to the origin.

v.) The values of delay time (L) and the time constant (T) obtained from the response curve are used to determine the values of PID parameters $K_P$, $K_I$, $and\ K_d$ using the Zeigler-Nichols first tuning table. The tuned values of $K_P$ , $K_I$, $and\ K_d$ are used in the coding using the PID library of the Arduino. With these parameters, the controller controls the system to get the required setpoint temperature. The rise time, the settling time, and the steady state error are calculated. These observations show that the PID controller is very well tuned and the required SP for the temperature controller system has been attained.

**"Real Time DC Motor Speed Control using a PID Controller in LABVIEW"** [10]

This research paper discusses the control of DC motors by using Arduino-based PID controller. The Arduino is interfaced with LabVIEW. The DC motors are very important tools for industrial, scientific research, and experimental applications. The control of DC motors to a precise accuracy is important for position control of sensors and in many other scientific applications. The authors are using a very low cost technique to control the speed of the DC motors. An Arduino is used as a very low cost data acquisition device. It is interfaced with a Graphic User Interface (GUI) of LabVIEW called LabVIEW Interface for Arduino (LIFA) so that the users can set their target speed for the motor on the screen of LabVIEW. The comparison between the open loop control and closed loop feedback control (PID control) performance is also made. The tachometer is used as a sensor to measure the speed of DC motors. The control process is summarized below:

[10]Vikhe, P., Punjabi, N., & Kadu, C. (2014). Real Time DC Motor Speed Control using PID Controller in LabVIEW. *IJAREEIE, 03*(09), 12162-12167. doi:10.15662/ijareeie.2014.0309046

a) The tachometer is attached to the shaft of the DC motors and is connected to the analog input pins of the Arduino. The tachometer measures the speed of the DC motor. The PWM values from the digital output pins (PWM pins) of the Arduino are given to the motor to control its speed. These are all done through the coding in the Arduino.

b) The Arduino UNO is interfaced with LABVIEW through a serial communication and the authors are using LIFA as the GUI. The accessory files should be installed to connect LabVIEW and Arduino. Then the programmed Arduino is controlled through the front panel of the LabVIEW screen. The users can change the setpoint speed of the DC motors.

c) The open loop control performance is observed just by setting up a setpoint and controlling the speed of the motor without any feedback loops. The performance is found to be very poor.

d) The closed feedback control with the PID loop is implemented to see the performance of the control system. The speed of the motor is compared with the setpoint speed. The difference is called the error "I" and the PID loop will try to minimize this error in every lap until the speed of the motor is close enough to the desired setpoint (SP). The PID controller in the LABVIEW control system palette can be helpful in tuning the PID parameters. The PID gains for the motor speed controls can be also tuned by a hit and trial method.

In this way, this research paper presents a very simple and low cost method that is still effective enough to control the speed of the DC motors.

**"Implementation of PID control to reduce wobbling in a line following robot"** [11]

The authors are using a PID controller with Arduino UNO to control the motion of the robot in this research paper. The PID is tuned manually using a trial and error method. Two motors on two

[11]A. N. (2013). Implementation of PID control to reduce wobbling in a line following robot. *International Journal of Research in Engineering and Technology*, *02*(10), 531-535. doi:10.15623/ijret.2013.0210083

wheels are used as actuators while Infrared (IR) sensors are used to find the position of the robot at any instant. The functioning of the line sensors and designing of the PID algorithm are interesting parts of this paper.

The IR sensors are attached to the lower portion of the robot. The line that the robot should be following is 3 cm wide and is black in color, while the background is chosen to be white. The white and black surfaces reflect different amounts of light. The transmitters of the sensors transmit light. On the basis of light being reflected back, the sensors find out whether the robot is on the target line, the black line, or on the surrounding region given by the white background. The sensors are employed in three different regions of the body of the robot: left, center and right. Hence, this arrangement of the sensors gives detailed information about the position of the robot.

The information gathered from the sensors about the position of the robot is the measured position, the process variable (PV) which is then compared with the desired position (setpoint) of the robot. Their difference is defined as the error value (e). The output drive (TURN) from the controller is given by the standard PID equation:

$$\text{TURN} = K_P * \text{e} + K_I * \int e\, dt + K_d * \frac{de}{dt} \quad \ldots\ldots\ldots\ldots\ldots (11)$$

Where, $K_P$, $K_I$ and $K_d$ are proportional, integral and the differential gains respectively.

The TURN value is given to the actuator as a PWM from the Arduino UNO, which in turn controls the direction of the robot. Likewise, the desired speed can be obtained using the digital output of Arduino to the motors, the actuators. The tuning is done manually, starting with only the value of $K_P$. It is increased until an oscillation is achieved. Then the values of $K_I$ are introduced and

increased until there is the least possible steady state error. Furthermore, the optimized value of $K_d$ would ward off any overshoots. Finally, the path of the robot with PID control is compared to the motion of the robot without PID control. The wobbling of the robot is found to be significantly reduced with the introduction of the PID controller while following the given path.

**"Fast Steering Mirror Control Embedded Self-Learning Fuzzy Controller for Free Space Optical Communication"** [12]

The research paper discusses the control applied to wireless communication. The paper proposes the acquisition, tracking, and pointing system for free space laser communication to improve the misalignment issues in Free Space Optical Communication. The system detects the image of the laser beam using a Charge Coupled Device (CCD) sensor array while the central detection is employed to find the location of the image. A Self-Learning Fuzzy Logic Controller (SLFC) is used to derive the fast-steering mirror mechanism to point the laser beam on the receiver for the coarse and the fine tracking. The SLFC controls the two DC motors for controlling the fast-steering mirror mechanism. The adaptive fuzzy logic has been employed through the Arduino Mega 2560 board. The controller gains obtained using MATLAB & Simulink for the given systems have been used for the experimental setup. The results show the proposed methodology for stabilizing the fuzzy logic controller. The PID controller, tuned using Ziegler Nichols (ZN) method, controls the plant while the fuzzy controller is learning the plant behavior. The fuzzy controller then gradually starts controlling the system through its output, until it has taken total control of the system. In this situation, the PID controller becomes idle.

[12]Alvi, B. A., Asif, M., Siddiqui, F. A., Safwan, M., & Bhatti, J. A. (2014). Fast Steering Mirror Control Using Embedded Self-Learning Fuzzy Controller for Free Space Optical Communication. *Wireless Personal Communications*, *76*(3), 643-656

Since the role of the PID controller is the topic of interest, this study focuses on the working of the PID controllers. The PID controller plays a significant role even when advanced controllers like fuzzy or neural controls are employed to control the system. The PID controller is responsible for controlling the system initially until SLFC learns the plant (process) behavior.

## **II.iv.** Conclusion from the Literature Review

The use of PID controller in diverse applications is studied with the realization of the interplay of the Arduino interfaced with technical computing language and the basic understanding about the signals, the latter of which gives an idea about the measurement and control of the signals. The interface of the Arduino with a high level computing language like MATLAB, LABVIEW, and some other specifically developed program, like NARMA, is observed. Such interfacing enables users to apply the PID algorithm to yield the optimum control of the signal and make the controller more versatile and compatible. The overwhelming role played by the PID controllers while using the sophisticated controllers, like fuzzy and neural controllers, is recognized. The fuzzy controller uses the PID controller in the beginning to control the system until the fuzzy controller finds the pattern of the system. The significance of the widespread applications of the PID controller from different regimes is understood from the literature.

# **III.** Temperature Controller System with a PID Control

The temperature controller system is designed to understand the mechanism of an Arduino-based PID controller. The designed system is used for studying the roles of the PID parameters and their effects on the controller's response parameters in different PID-modes. The different modes of a PID controller are P-controller, PI-controller, and PID-controller mode. The implementation of derivative control is difficult and should be encompassed only when it is a necessary, as discussed in section II.i. Hence, we are not using the derivative term in our algorithm because we are simply trying to control the temperature of the designed system to keep it closer to the setpoint. The P-controller mode can be achieved from the PI-controller mode by simply assigning a zero value to the integral gain (I-value).

## **III.i.** Description of the components in the system

The control designs generally consist of microcontrollers, actuators, and sensors. The Arduino UNO and thermo-electric coolers (TECs) are chosen as microcontroller and actuators respectively. TECs are easy to operate, inexpensive, and are two directional, and so they can control the temperature of the sample (heating and cooling) based on the required temperature conditions. TMP36 analog temperature sensor is the preferred temperature sensor because of its low cost, self-contained nature, its wide range of operation, and compatibility for interfacing with the sample. The other electronic components used in the system design are h-bridges, non-inverting operational amplifiers, and separate power supplies for the operational amplifier, Arduino UNO, and TECs. We have extensively used a technique known as pulse width modulation (PWM). All the components including PWM used in the system design are discussed in detail below.

1. Pulse Width Modulation (PWM):

PWM is a technique to get an average analog output signal from a digital input. The concept of PWM can be understood by the following example of an LED. If an LED is turning on and off at a moderate rate, then the turning on and off phenomenon of the LED can be observed. However, if the rate of switching is fast enough, one can observe a dimmer LED instead of the turning on and off phenomenon. A PWM signal is represented by a square wave of a given duty cycle and frequency. A duty cycle expressed in percentage specifies the time of an input signal being "on" over a given period of time. For example, 50 % duty cycle means that the signal is on for half of the time period as shown in Fig 3.1.

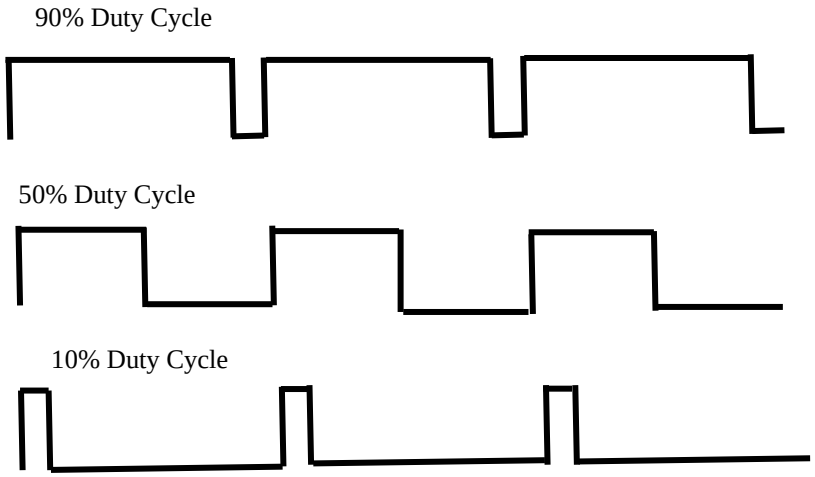


**Fig 3.1 Duty Cycles: 90%, 50%, and 10% duty cycle represent that the signal is turned on for nine tenths, half, and one tenth of the whole time period respectively.[13]**

[13] Dominguez, B., Barba, F., & Castrejon, C. (2014, December 8). DC Motor Driver by EMG and EOG | imditesmprojects. Retrieved from https://imditesmprojects.wordpress.com/dc-motor-driver-by-emg-and-eog/

The frequency of a signal represents how many times the signal runs in a given time period. In other words, it represents how fast the PWM switches between the higher and lower states. The frequency of the signal should not be too high or too low for the load to respond. This technique avoids the wearing out of components that normally occurs when the rheostat (a mechanical switching) is used to control the voltage.

2. Analog Temperature Sensor (TMP36):

The TMP36 temperature sensor is a silicon based temperature sensor with three pins, the ground pin, the analog voltage pin, and the positive voltage pin with a range of 2.7 to 5V as shown in Fig 3.2. The TMP36 temperature sensor is robust, precise in measurement, and compatible with the sample. It has a very wide range of measuring capability from - 40℃ to 150℃. The change in temperature around the sensor provides the proportional voltage across the diode of the sensor. The precise amplification of the generated voltage provides an analog signal.

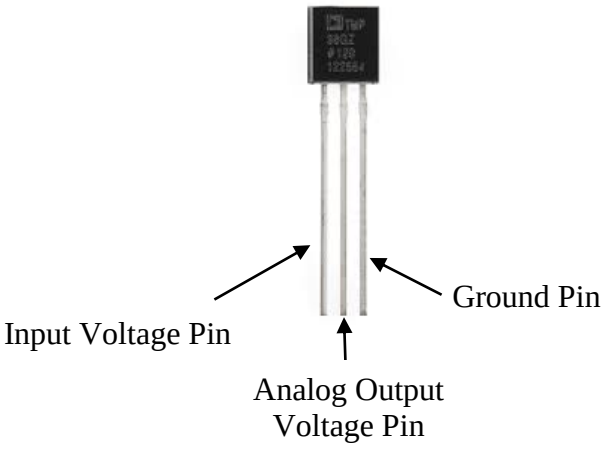


**Fig 3.2 TMP36 analog temperature sensor: The central pin labelled as analog voltage output gives the measure of analog output while the black block is the body that senses the change in temperature of the sample attached.[14]**

[14]Fried, L. (2012, July 29). Overview | TMP36 Temperature Sensor | Adafruit Learning System. Retrieved from https://learn.adafruit.com/tmp36-temperature-sensor/overview

3. Thermo-electric cooler (TEC):

An array of p-type and n-type semiconductors soldered between two ceramic plates electrically in series and thermally in parallel forms a TEC. It is a solid state device which works according to the Peltier effect. The Peltier effect is a phenomenon in which the heat flux is generated between the junctions of two dissimilar materials when connected to an external DC power supply. The working mechanism of TEC is explained in Appendix A2.

4. H-bridge:

An h-bridge is an electronic circuit causing the current to flow in either direction across the load. Fig 3.4 shows the h-bridge with four solid-state switches. When the $A_1$ and $A_4$ switches are closed while keeping $A_2$ and $A_3$ open, the current passes through the motor (load) in the clockwise direction. On the other hand, the current flows counter-clockwise when $A_1$ and $A_4$ are open while $A_2$ and $A_3$ are closed. Both switch pairs ($A_1$ and $A_2$) and ($A_3$ and $A_4$) should not be closed at the same time because that leads to short circuit.

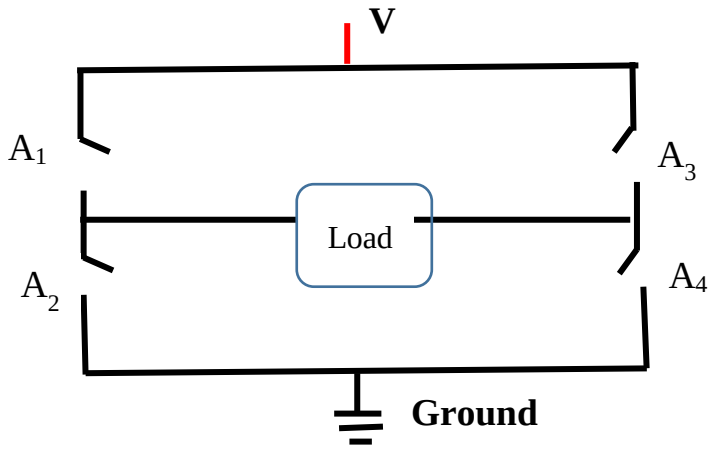


**Fig 3.3 Working mechanism of H-bridges**: When switches $A_1$ and $A_4$ are closed, the current flows in clockwise direction across the load. The current flows counter-clockwise through the load when the switches $A_2$ and $A_3$ are closed.[15]

[15]Hunt, O. (n.d.). HVLabs.com. Retrieved from http://www.hvlabs.com/hbridge.html

5. Arduino UNO: An Arduino UNO is a microcontroller which operates by reading the sensor values, making logical decisions, and sending the necessary signals to the actuators.[16] A brief description of Arduino UNO shown in Fig 3.5 is as follows:

   i.) It has a USB power plug and a separate power plug. The separate power plug takes the power supply through the external voltage source.

   ii.) It has five analog input pins that measure the signals from the sensors.

   iii.) It has a USB power plug and a separate power plug. The separate power plug takes the power supply through the external voltage source.

   iv.) It has twelve digital pins for digital input/output (2-13), among them five pins (3, 6, 9, 10 and 11) are PWM pins.

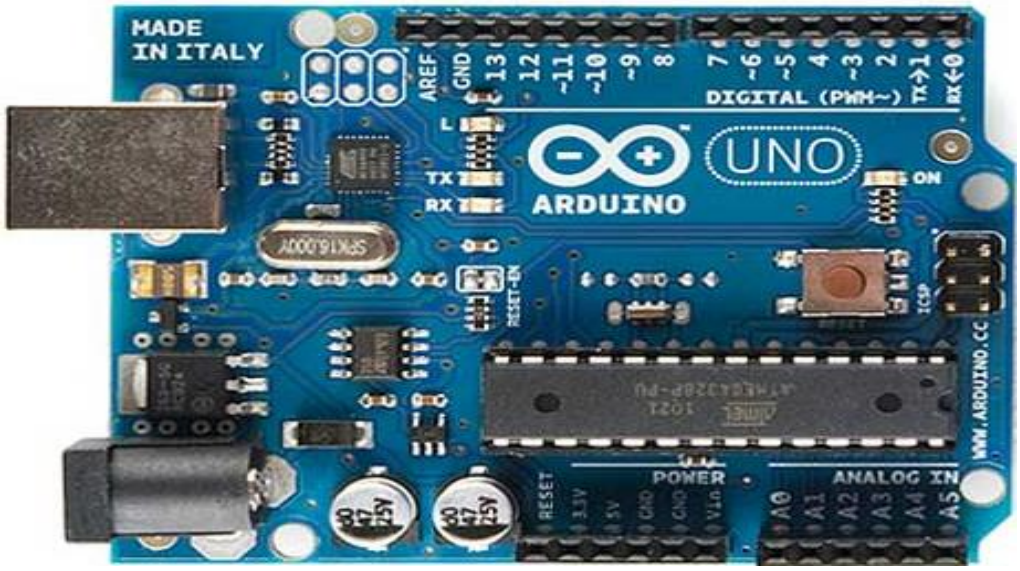


**Fig 3.4 Arduino UNO Board: Arduino UNO board showing 16 digital pins, 6 analog pins, and other various parts.[17]**

[16]Premeaux, E., & Evans, B. (2011). Saving the world. In *Arduino Projects to Save the World, Spider temp*. Berkeley, CA: Apress.

[17]Roberge, J. K., & Lundberg, K. H. (2007). Background and Objectives. In *Operational Amplifiers: Theory and Practice* [1.81] (2nd ed., pp. 1-11). Retrieved from http://web.mit.edu/klund/www/books/opamps181.pdf

v.) Digital pins 0 and 1 are labeled RX and TX respectively, which are serial in and serial out pins.

vi.) There are three ground pins, one input voltage pin, one 5 V pin, one 3.3 V pin, one reset push-button, and one Analog Reference (AREF) pin.

vii.) Atmega328 is a microprocessor used with an In Circuit Serial Programmer (ICSP).

In our experimental setup, the code written in a C programming language (Arduino software) is uploaded to the Arduino UNO instead of using the PID library.

In our project, Arduino UNO is basically performing the following three tasks:

a) Reading the analog signals (values) from the temperature sensor.

b) After comparing the received signal with the desired signal, sending digital output as PWM signals to the actuators.

c) Printing the data as text-files, the process related to data acquisition.

6. Operational Amplifier:

The operational amplifier amplifies the input signal. The magnitude of the output voltage from the amplifier is determined by the resistors connected across the negative feedback loop.

In order to open the gates of the MOSFETs completely, approximately 10V is required although the PWM signal from the Arduino UNO can supply a maximum output of only 5V. The complete opening of the MOSFET gates can be achieved by using the operational amplifier with gain of 2 (using two resistors each of 5K-Ohm). If the gates are not opened completely, the following two problems may occur:

(i) MOSFETs becomes very hot and can easily get damaged.

(ii) A sufficient amount of current cannot flow through the actuators.

## **III.ii.** Description of the System Design

The schematic of the designed temperature controller system is shown above in Fig 3.5.

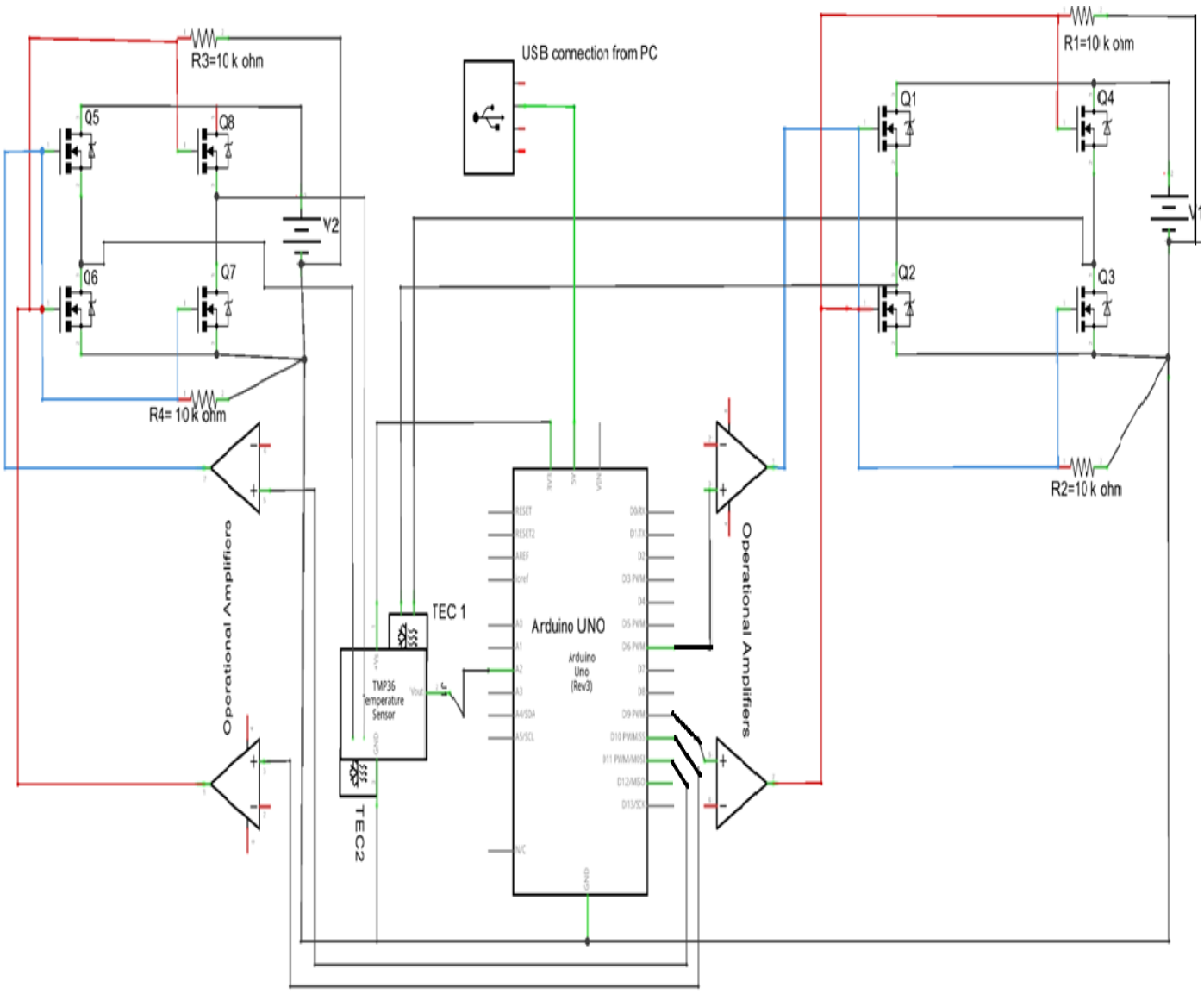


**Fig 3.5: Schematic of the designed temperature controller system**

In the schematic, the temperature sensor TMP36 is sandwiched between two TECs (TEC1 and TEC2). However, the TMP36 sensor is inserted into the sample and then sandwiched between the TECs in the real experiment. The N-channel MOSFETs used in making the h-bridges are labelled as Qi, where i=1 to 8 (integers only). Two h-bridges shown are formed by these eight MOSFETs as shown in the schematic on the either side of the Arduino-UNO. The gates of the MOSFETs are grounded through the 10 $k\Omega$ resistors (plated through-hole resistors). A pair of TECs work as loads for these two h-bridges. The dual power source used for the non-inverting operational amplifier is not shown in the schematic. The working mechanism of the temperature controller system can be summarized with the help of the shown schematic as follows:

i) With the connection as shown in schematic, the code (as given in Appendix-2) for the particular controller mode is uploaded to the Arduino through personal computer (PC).

ii) The Arduino reads the temperature of the sample using the TMP36-temperature sensor, which is connected to the Arduino through analog-pin A2 as shown in the schematic.

iii) The received temperature value of the sample is compared with the setpoint temperature and the necessary digital outputs are given through four digital output PWM signals (PWM pins- 3, 9, 10, and 11). These PWM signals are sent to the non-inverting amplifiers of gain 2. The amplified outputs from the amplifiers are supplied to the common point of two gates (a switch) of the diagonal MOSFETs as shown in the schematic.

iv) The TECs working as loads for the h-bridges receive the necessary PWM signals and can heat and cool the attached sample to maintain the setpoint temperature. The received PWM signals control the amount and the direction of the current across the TECs. The amount of the current varies due to the different values of duty cycles (PWM

signals) provided as the controller's output as given by the equation (9). The direction of the current flow can be changed by sending the PWM signals through the different switches.

Fig 3.6a shows the whole temperature control system where individual parts are shown by the succeeding pictures. A rectangular aluminum bar in Fig 3.6b shows the sample of our designed system with a hole drilled in the center. The TMP36 is placed inside of the mentioned hole while the pins of the temperature sensor are connected to the Arduino. It is assumed that the temperature at the center represents the average temperature of the whole sample. We have chosen aluminum as the sample because of its higher thermal conductivity and easier availability in our lab. Fig 3.6c shows the hot sides of a pair of TECs being attached to the surfaces of the rectangular sample. The heat sinks with screws help the TECs to remain attached to the sample. The left pin of the TMP36 in Fig 3.2 is connected to both the AREF pin and the 3.3 V pin of the Arduino, while the central and right pins are connected to the analog input pin and the ground pin of the Arduino UNO respectively. Fig3.6d shows a common point connecting the gates of two MOSFETs to form one switch. The PID parameters are found manually for each type of controller.

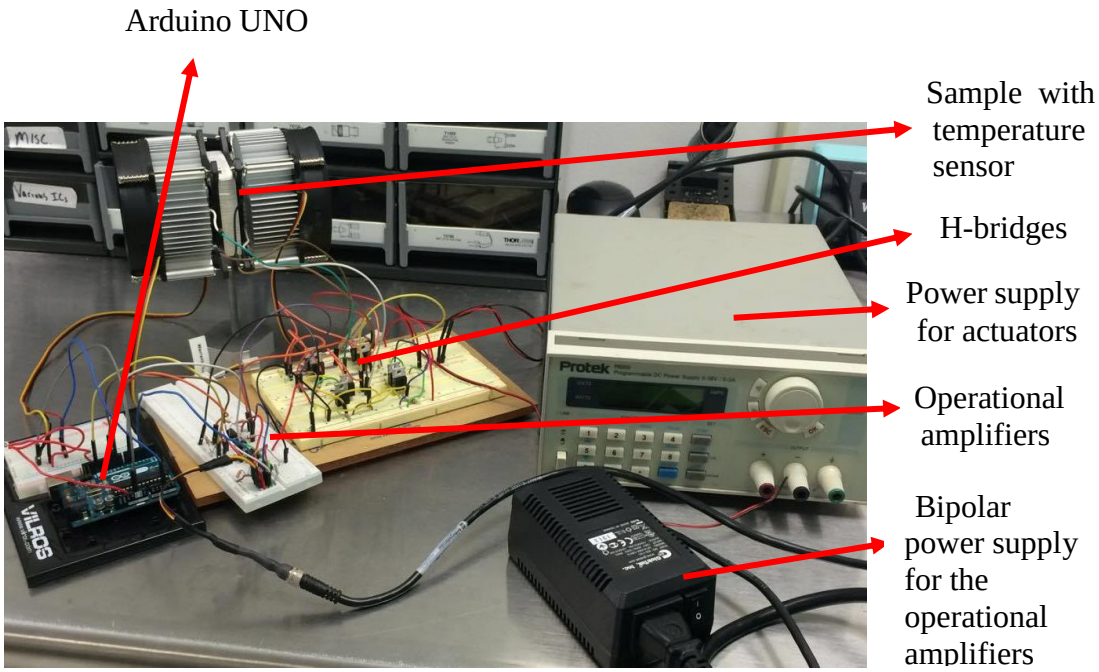


**Fig 3.6a: Temperature controller system**

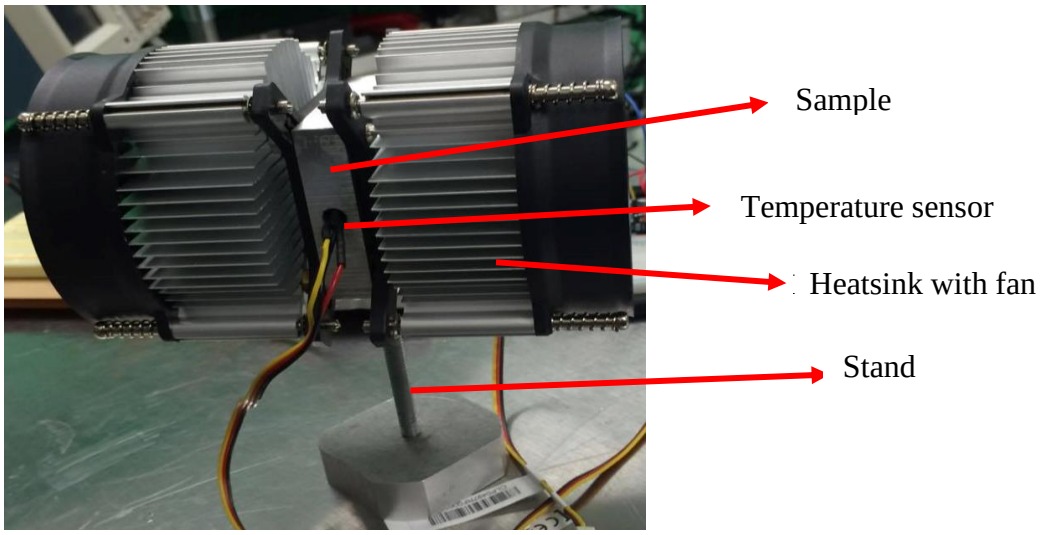


**Fig 3.6b: Sample attached to the heatsink and the temperature sensor**

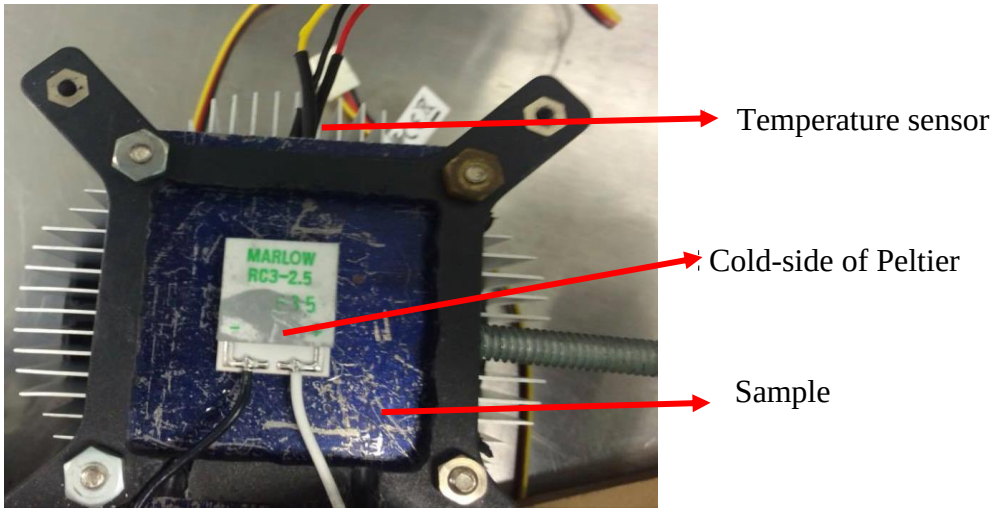


**Fig 3.6c: View of hot-side of TEC being attached to the sample**

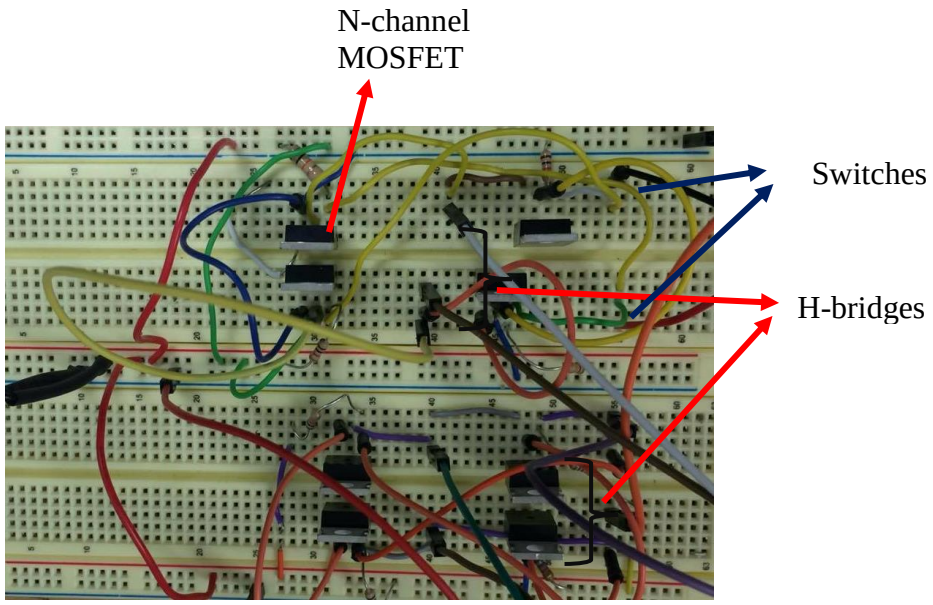


**Fig 3.6d H-bridges: Four N-channel MOSFETS are connected in specific way by jumpers to build one h-bridge; two h-bridges are made on the breadboard**

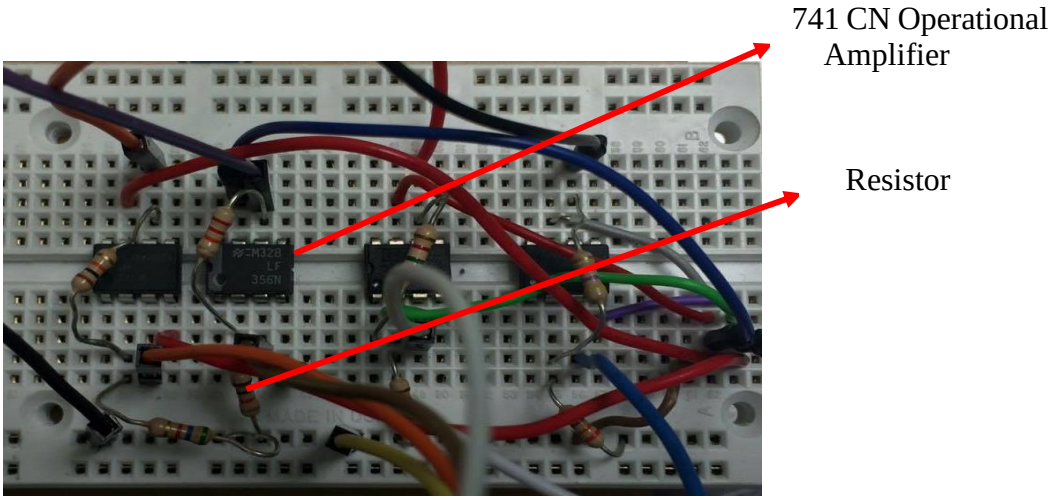


**Fig 3.6e Operational amplifiers used in the system: Four digital outputs from Arduino serve as input for four separate 741 CN operational amplifiers**

# **IV.** Observations and Conclusions

The designed system is run in on/off, P, and PI-controller modes to observe the controller's response parameters. The designed temperature controller can both heat and cool the sample to get the desired constant temperature. However, we are only using the temperature controller system in one direction (heating) to study the PID parameters and the controller's response parameters. At first, some discussion on the noise and errors in our temperature controller system is made. Then different modes of controller is designed and their response parameters are observed.

## **IV.i.** Noise and Error:

The noise consists of any random fluctuations in the signal or any undesired random disturbance of the signals. The noise in the temperature controller system is coming due to the digitization of ADC and the random fluctuations of signals from the sensor (TMP36), which are discussed below:

a) Digitizing error and noise: The analog to digital converter (ADC) converts an analog input signal to a digital output. The analog signal is continuous in amplitude and time while the digital signal is discrete in both the amplitude and the time. The process of discretizing the signal amplitude and the time are called the digitization (quantization) and the sampling respectively.

The digitization brings about an inherent error associated with the ADC called the digitizing error. This phenomenon is the limitation of ADC being not able to convert an analog signal to a precise digital value. A range of analog signals produces the same digital output value. This range, also known as step-size, is the minimum change in input analog voltage that can be resolved by the ADC. The step size (Q) depends on the input voltage range and the resolution of the ADC which is given by the equation (12),

$$Q = \frac{Input\ voltage\ range}{Resolution} \quad \text{……..…..…..…..} \ (12)$$

The resolution of ADC (Arduino UNO) = $2^{10}$ (with 10 bit of precision)

We are using analogReference (EXTERNAL) command to use a 3.3V pin of Arduino as a powering voltage for the TMP36 temperature sensor. But the actual voltage reading coming out from the 3.3V pin is found to be only 3.2V. Hence, the input voltage range for ADC = 3.2V.

$$Q = \frac{3.20 \pm 0.03\ V}{2^{10}} = 3.14\ mV \text{...……} \ (13)$$

Using the 3.3 V pin instead of the 5V lowers the step size (Q), which helps us to measure the temperature more precisely. We are interested in the temperature change corresponding to the voltage change of the sensor. A 10 mV change corresponds to the temperature change by 1℃ for TMP36-temperature sensor. Hence, in terms of temperature, the step-size of the ADC is given by equation (14)

$$Q \approx 0.31℃ \text{……..……..….} \ (14)$$

Hence, the step-size corresponding to the temperature is approximately 0.31℃. The input analog signal coming from TMP36 being measured using Arduino and its digital output values are shown in Fig 14.1. The difference between the input and output gives the digitizing error. The digitizing error lies within $\pm$ Q/2 as shown in Fig 14.1. Any value of error is equally likely to occur in a uniform distribution between $\pm$ Q/2. Hence, the RMS (root mean squared) values of digitizing

errors in $\pm\frac{Q}{2}$ interval gives the digitizing noise and can be represented by the formula of equation (15).[18]

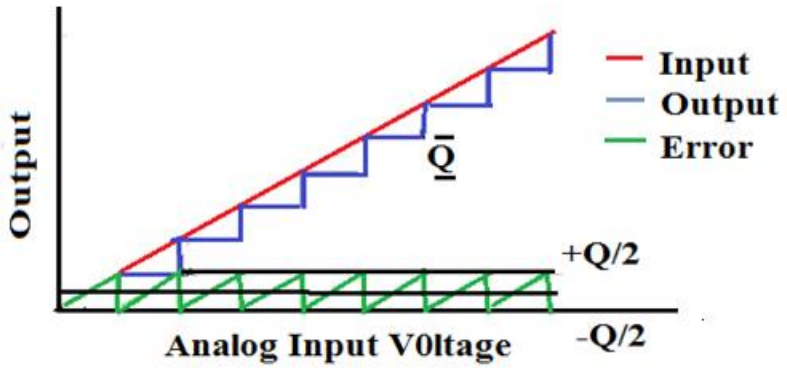


**Fig 4.1: Plot of analog input voltage vs digital output of signal from TMP36 sensor error. The green saw wave like curve gives the digitizing error that ranges between $\pm\frac{Q}{2}$.**

$$\text{The digitizing noise } (\sigma_d) = \frac{Q}{\sqrt{12}} \quad \ldots\ldots\ldots\ldots\ldots\ldots (15)$$

Hence, $\sigma_d = \frac{0.31℃}{\sqrt{12}} \approx 0.09℃ (0.89\ mV)$

b. Sensor noise: The RMS value (ac) of the sensor signal gives the average sensor noise in the system. The measurement is carried out when the TMP36 sensor is giving a stable temperature reading. The body of the sensor is placed inside a plastic wrapper and inserted into the normal water in a beaker for few minutes until the constant temperature reading is observed in the serial monitor. This isolating phenomenon of the temperature sensor just ensures that the RMS values being measured are not associated with the temperature change of the sensor and other interactions of the sensor with the surroundings. Sensor noise ($\sigma_s$) is

[18] Gupta, S., & Pathak, A. (2012, January). ADC Guide, Part 1- The Ideal ADC. Retrieved from http://www.cypress.com/file/113946/download

found by measuring the RMS volts reading from the sensor signal (in ac coupling mode of the oscilloscope) which is shown in Fig 4.2. The bandwidth of the oscilloscope used is 70MHz. The vertical scale is fixed to 10 mV as the temperature change of 1 degree Celsius corresponds to 10 mV in our experiment.

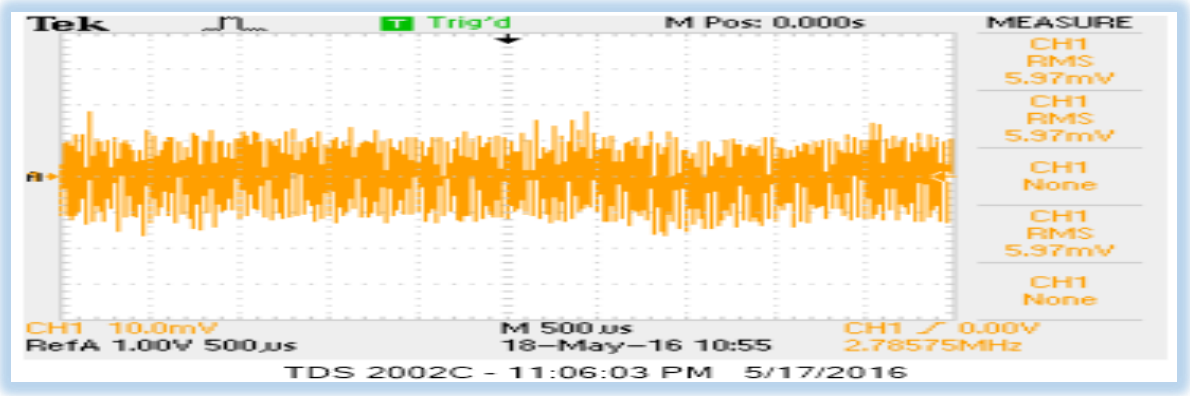


**Fig 4.2 RMS measure of the signal from temperature sensor: The snapshot of Tektronix oscilloscope screen is taken while measuring the RMS value of the isolated temperature sensor.**

The RMS value of the sensor signal is measured to be approximately 5.97 mV, i.e. $\sigma_s = 5.97$ mV, that corresponds to the temperature change = $\frac{5.97 \text{ mV}\,℃}{10 \text{ mV}} \approx 0.6℃$

Hence, the total noise in the system due to the digitization and the sensor in terms of volts and ℃ are estimated as follows:

$$\sigma = \sqrt{(\sigma_d)^2 + (\sigma_s)^2} = \sqrt{(0.09)^2 + (0.6)^2} = 0.603℃ \approx 0.6\ ℃$$

The sensor noise contributes almost to the total noise.

**Calibration of TMP36:** The accuracy of the temperature reading from the TMP36 temperature sensor is found to be $\pm$ 0.35℃. It is calibrated with the help of Digi-Sense calibrated high accuracy digital thermometer from our laboratory, which gives temperature reading up to two decimal places.

## **IV.ii.** Setup, Procedure and Observations:

The same following procedures are followed from uploading a code in Arduino to interpreting the results for each controller mode:

i) Uploading code: The code written in C language, according to our purpose and on the basis of the employed controller-mode, is uploaded in Arduino UNO through laptop/PC. The codes for different controllers' modes are given in Appendix-2.

ii) Ambient temperature and choosing the setpoint: The (ambient) initial temperature of the system is the room temperature which ranges approximately from 19℃ to 22℃. The setpoint is chosen to be 36.00℃, regarding the setpoint temperature for the mammalian (human) cell incubator is 37.00℃. (Generally, human cell incubator is also kept at 36.50℃ to get the optimum temperature with tolerance of $36.50℃ \pm 0.5℃$)

iii) Data acquisition and plots: The temperature (PV) and the controller's output (PWM) are printed and saved into the PC using "CoolTerm", an open source serial communication for Arduino, every 100ms for approximately 18 minutes. The PID controller is also updated every 100ms. It is always advisable to use interrupts to make sure that the task is done exactly after the certain time after using "delay" command. The data acquired as text-files through "CoolTerm" are imported in MATLAB. The average over distinct blocks of 10 values are taken. The plots of the temperature (PV) and the controller's output (PWM) against the time are plotted in the same window in MATLAB. This helps us to show how the controller is responding to the changes in the PV (indirectly to the error).

The following section discusses the observation and their conclusion based on the observation of on/off controller, P-Controller, and PI-Controller respectively. The setpoint temperature is

taken as 36.00℃ while the ambient temperature of the sample is found between from 19.00℃ to 22.00℃.

1. On/Off-Controller:

   The On/Off controller feeds 100% duty-cycle of the PWM signal, i.e., 255 to the TECs if the temperature is below the setpoint, otherwise it supplies 0 % duty cycle of PWM. The controller keeps the TECs either completely turned on or completely turned off. The observed temperature and corresponding PWM values against the time are plotted in Fig 4.3.

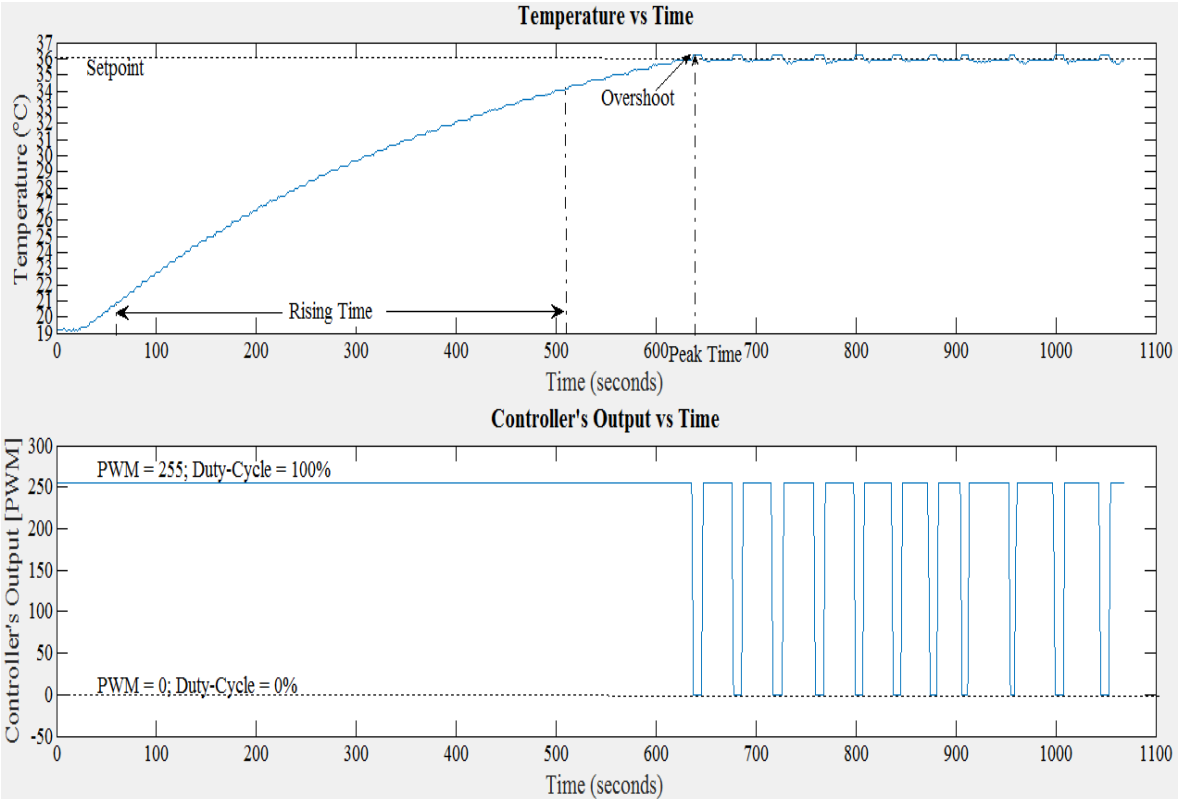


**Fig 4.3: Plots of Temperature and Controller's Output vs Time of On/Off Controller**

The on/off controller's response parameters are discussed below:

i) Rising Time:

90% value of the temperature change from initial value to the setpoint value = (36.00℃ - 19.09℃) x 0.9 = 15.22℃ ±0.6 ℃

The value of PV with this 90% change value = 19.09℃ + 15.22℃= 34.31℃ ± 0.6℃, which is found in approximately 514 seconds.

10% value of the temperature change from initial value to the setpoint value = (36. 00℃ - 20.00℃) x 0.1 = 1.69℃ ± 0.6 ℃

The value of PV with the 10% value change = 19.09℃ +1.69℃ = 20.78℃ ± 0.6 ℃

, which is found in approximately 59 seconds.

Hence, Rising Time = (514 – 59) seconds = 455 seconds

ii) Peak Time: The peak time of this system is found to be 637s, when the temperature of the system reaches 36.25℃ + 0.6℃.

iii) Settling value & settling time: The system undergoes oscillation and does not settle.

iv) Overshoot: Ample overshoots are observed.

2. P-Controller:

Rewriting the equation of P-Controller's ouput from equation (1), section II.i.

$$\text{Controller Output (P-Term)} = K_p . \text{e (t)}$$

The controller's output is directly proportional to the proportional gain and the error. We would try to observe the controller's behavior for different values of proportional gains. We have used following three values of the proportional gains in our study of the P-Controllers.

I) P-Controller with $K_p$ = 150

The ambient temperature of the system is found to be 20.00℃. The plot of temperature (Process Variable) and the controller's output (PWM) against the time of the P-Controller with Kp= 150, is shown in Fig 4.4 below:

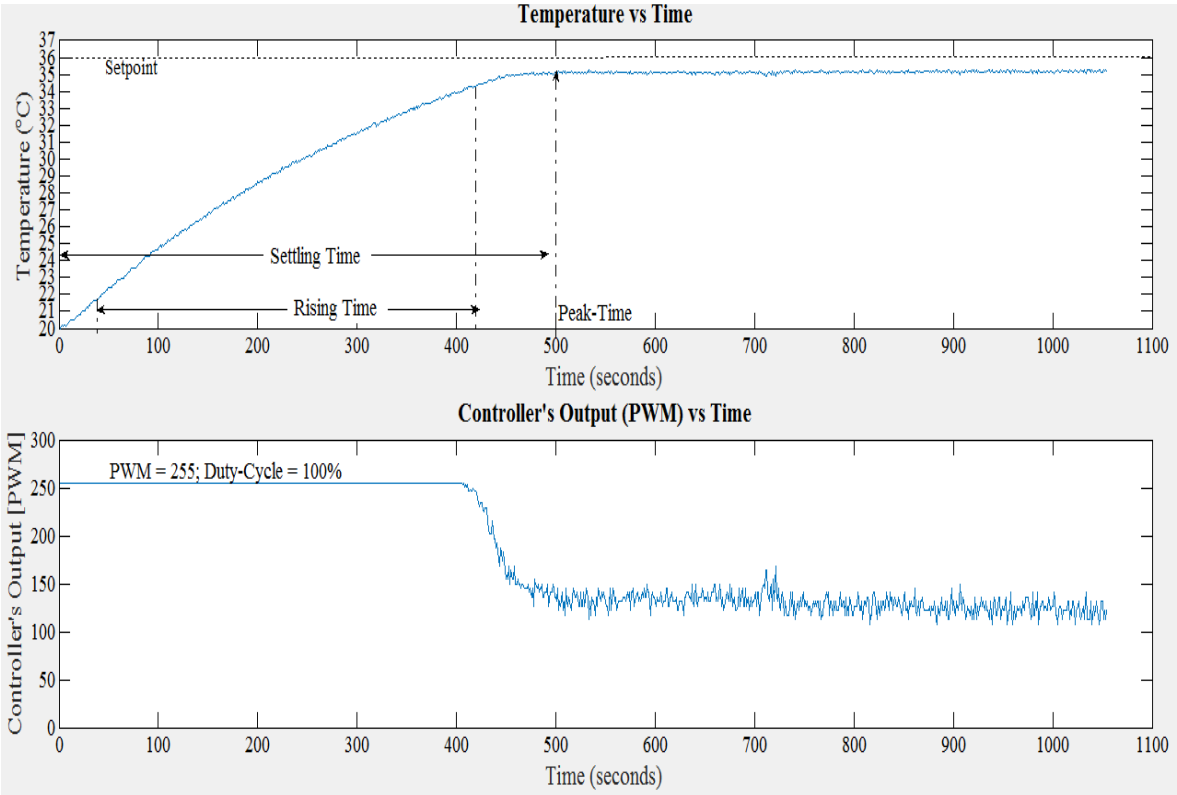


**Fig 4.4: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$ =150**

The controller's response parameters are discussed below:

i) Rising Time: The rising time of the system is calculated to be 387 seconds.

ii) Peak Time: The peak time is 501s, when the temperature reaches 35.14℃ ± 0.6℃.

iii) Settling value & settling time: The system settles in approximately 501s, when the temperature of the sample reaches 35.14℃ ± 0.6℃.

iv) Steady State Error (SSE) = 36.00 ℃ - 35.14℃= 0.86℃ ± 0.6℃.

v) Overshoot: There are no overshoots in the system.

II) P-Controller with Kp=1500

The ambient temperature of the system is found to be 21.93℃ ± 0.6℃. The plots of temperature and the controller's output vs time of the P-Controller with $K_p$ = 1500, is shown in Fig 4.5 below:

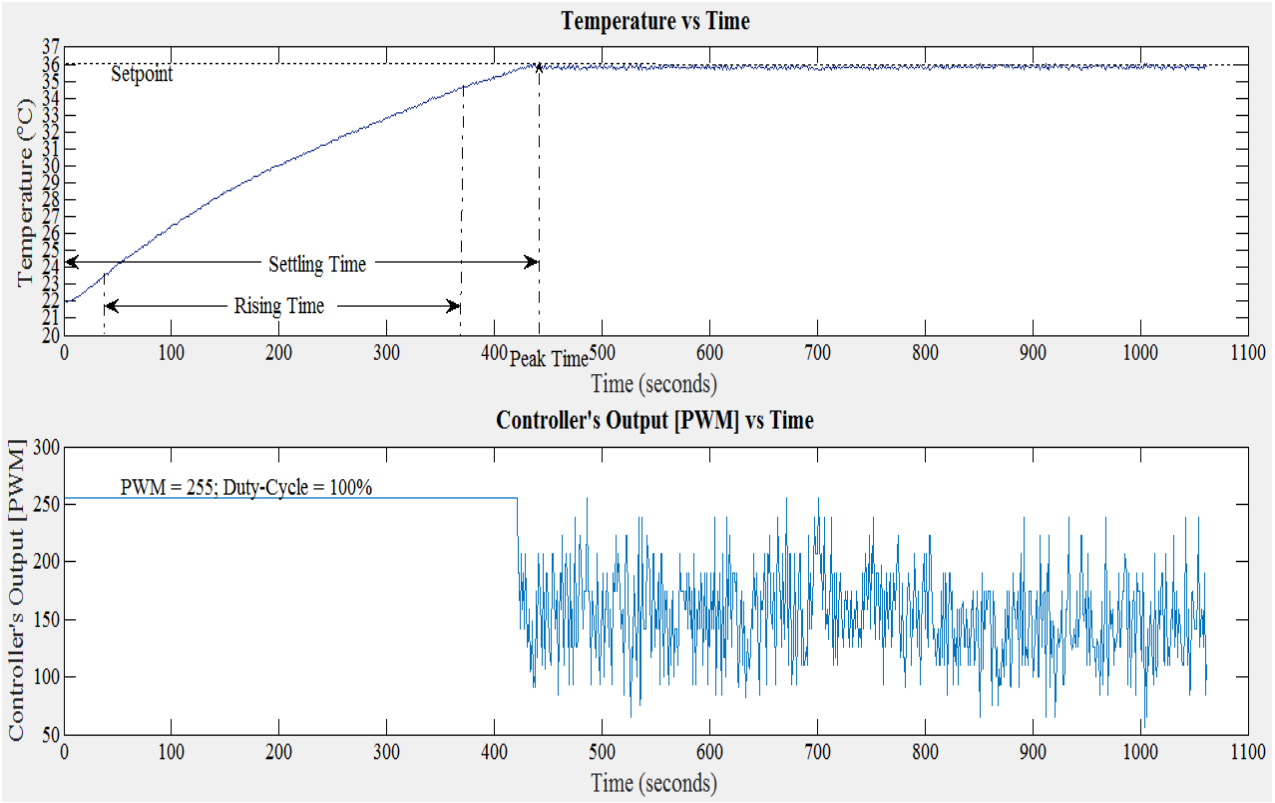


**Fig 4.5: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$=1500**

The response parameters of the P-Controller with $K_p$ =1500, are discussed below:

i) Rising Time: The rising time of the system in this mode is calculated to be 335 seconds.

ii) Peak Time: The peak time of the system is found to be at approximately 441s, when the temperature of the sample reaches 35.83℃ ± 0.6℃.

iii) Settling value & settling time: The system settles in approximately 441seconds, when the settling value of the system is found to be 35.83℃ ± 0.6℃.

iv) SSE = 36.00℃ - 35.83℃ = 0.17℃ ± 0.6℃.

v) Overshoot: No overshoots are observed in the system.

III) P-Controller with Kp=4300

The ambient temperature obtained in this configuration is found to be 21.50℃ ± 0.6℃. The plots of temperature and the controller's output (PWM) vs time of the P-Controller with Kp = 4300, is shown in Fig 4.6 below:

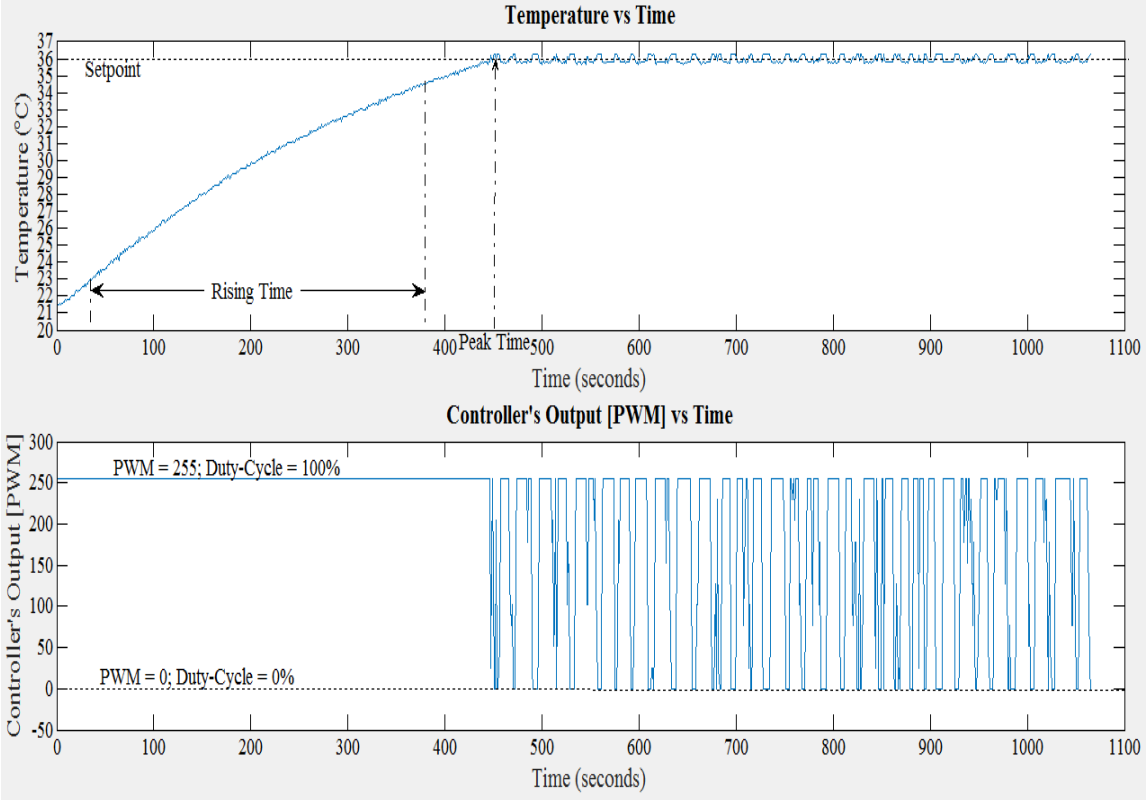


**Fig 4.6: Plots of Temperature and Controller's Output vs Time of P-Controller with $K_p$= 4300**

The controller's response parameters of P-Controller with $K_p$= 4300, are discussed below:

i) Rising Time: The rising time is calculated to be 346 seconds.

ii) Peak Time: The peak time is 451seconds, when the temperature of the system reaches 36.25℃ ± 0.6℃.

iii) Settling value, SSE, & settling time: The process variable of the system oscillates along the setpoint and the system does not settle.

iv) Overshoot: Ample overshoots are observed.

Table 4.a lists the response parameters of all the designed P-Controllers.

**Table 4.a: List of comparisons of response parameters of all designed P-Controllers**

| Parameters | $K_p$ = 150 | $K_p$ = 1500 | $K_p$ = 4300 |
|---|---|---|---|
| Settling value | The system settles to the value 35.14℃ ± 0.6℃. This temperature is below the setpoint. | The system settles to the temperature of 35.83℃ ±0.6℃ temperature (little below the setpoint). | The system does not settle. The PV (temperature) oscillates about the setpoint. |
| Settling Time | 501s | 441s | Cannot be determined |
| SSE | 0.86℃ ± 0.6℃. | 0.17℃ ± 0.6℃. | Cannot be determined |
| Overshoots | No overshoots. | No overshoots. | Ample overshoots are observed. |
| Rising time | 387 seconds | 335 seconds | 346 seconds |

**Conclusion based on the observations of the P-Controllers**:

a) The SSE exists for this mode of controller.

b) The system becomes unstable with oscillations for larger values of proportional gains.

c) When the proportional gain is increased, the rising time,settling, and the SSE of the system is generally found to decrease. Since the initial temperatures are different for theses P-controllers, thus the values of settling times and rising times are not relatively comparative. However, when we analyzed the data, we observed that there is only slight decrease in settling time and rising time going from $K_p$=150 to $K_p$=1500. However, the rising time for the P-Controllers with $K_p$ = 1500 and $K_p$ = 4300 are found to be almost same.

3. PI-Controller:

The output of the PI-controller is given below using equation (2) from section II.i as:

Controller Output = P-Term + I-Term

Controller Output = $K_p \times$ e (t) + $\int_0^\tau K_i . e(t) dt$

We have already studied the bahaviors of the P-Controllers with the values of " $K_p$" as 150, 1500, and 4300. Now we are designing the PI-controllers by setting varying values of integral gains ( $K_i$), while taking the $K_p$ values 150 and 1500 respectively.

I) PI-Controller with $K_p$=150 and $K_i$=1

The initial temperature of the system in this configuration is obtained to be 20.00℃ $\pm$ 0.6℃. The plots of temperature and the controller's output (PWM) vs time of the P-Controller with $K_p$=150 and $K_i$=1, is shown in Fig 4.7 below:

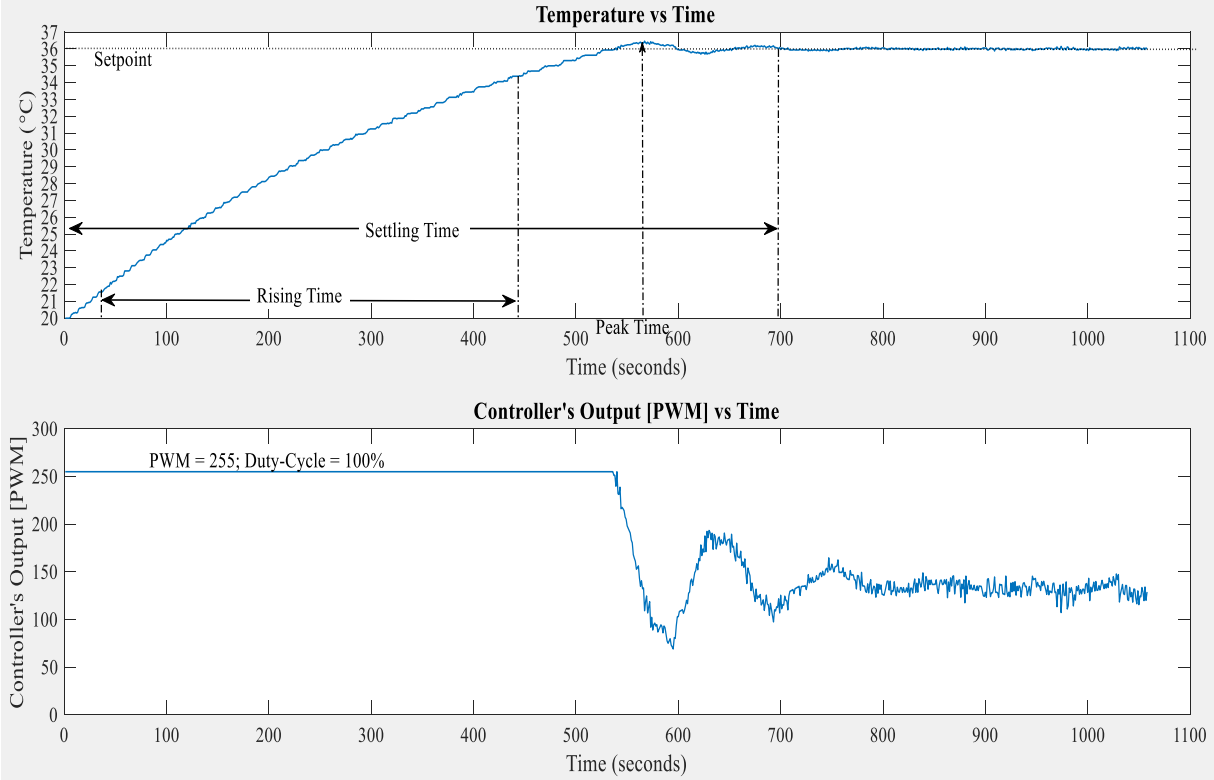


**Fig 4.7: Plots of Temperature and Controller's Output vs Time of PI-Controller with $K_p$=150 and $K_i$=1**

The PI-controller's ($K_p$=150 and $K_i$=1) response parameters are discussed below:

i) Rising Time: The rising time is calculated to be 409 seconds.

ii) Peak Time: The peak time for this system is found to be 567 seconds, when the temperature reaches 36.48℃.

iii) Settling value & settling time: The steady value with which the system settles is found to be 35.99℃ at 696 seconds.

iv) SSE = 36.00℃ - 35.99℃ = 0.001℃ ±0.6℃, which is so small such that it can be neglected. The SSE is eliminated by this controller. (0.001℃ is smaller by more than 200 times the step-size of ADC that is measuring the temperature.)

v) Overshoot and % overshoot = The overshoots are observed closer to 500 seconds as shown in Fig 4.7. The calculation of the overshoot % is given below:

% overshoot = (36.48℃ - 35.99℃)/ 35.99℃ x 100%=1.36 %

I) PI-Controller with $K_p$=150 and $K_i$=3

The ambient temperature for this configuration is obtained to be 20.69℃ ±0.6℃. The plots of temperature and the controller's output (PWM) vs time of the P-Controller with $K_p$ = 150 and $K_i$ = 3, is shown in Fig 4.8 below. The controller's response parameters are discussed below:

i) Rising Time: The rising time is calculated to be 306 seconds.

ii) Peak Time: It is found to be 419 seconds, when the temperature of the sample reaches 36.34℃ ±0.6℃.

iii) Settling value & settling time: The system oscillates and does not settle with a steady value.

iv) Overshoot: Ample overshoots are observed.

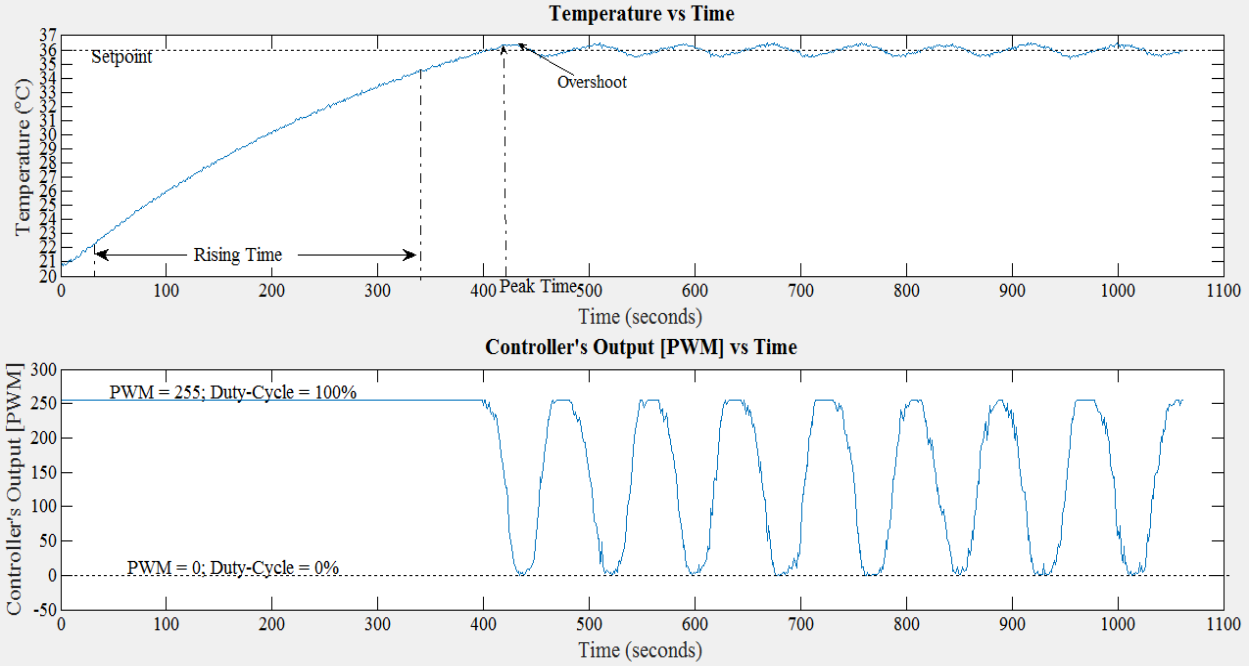


**Fig 4.8: Plots of Temperature and Controller's Output vs Time of PI-Controller with $K_p$=150 and $K_i$=3**

II) PI-Controller with $K_p$=1500 and $K_i$= 0.1

The ambient temperature for this configuration is obtained to be 21.87℃ ±0.6℃. The plots of temperature and the controller's output (PWM) vs time of the P-Controller with $K_p$= 1500 and $K_i$= 0.1, is shown in Fig 4.9 below. The controller's response parameters are discussed below:

i) Rising Time: The rising time is obtained to be 333 seconds.

ii) Peak Time: The peak time is found to be 447 seconds, when the temperature of the sample becomes 36.25℃ ±0.6℃.

iii) Settling value & settling time: There is plenty of oscillations in the system such that the system does not settle with a steady state value.

iv) Overshoot: Ample overshoots and oscillation are observed such that the system becomes unstable.

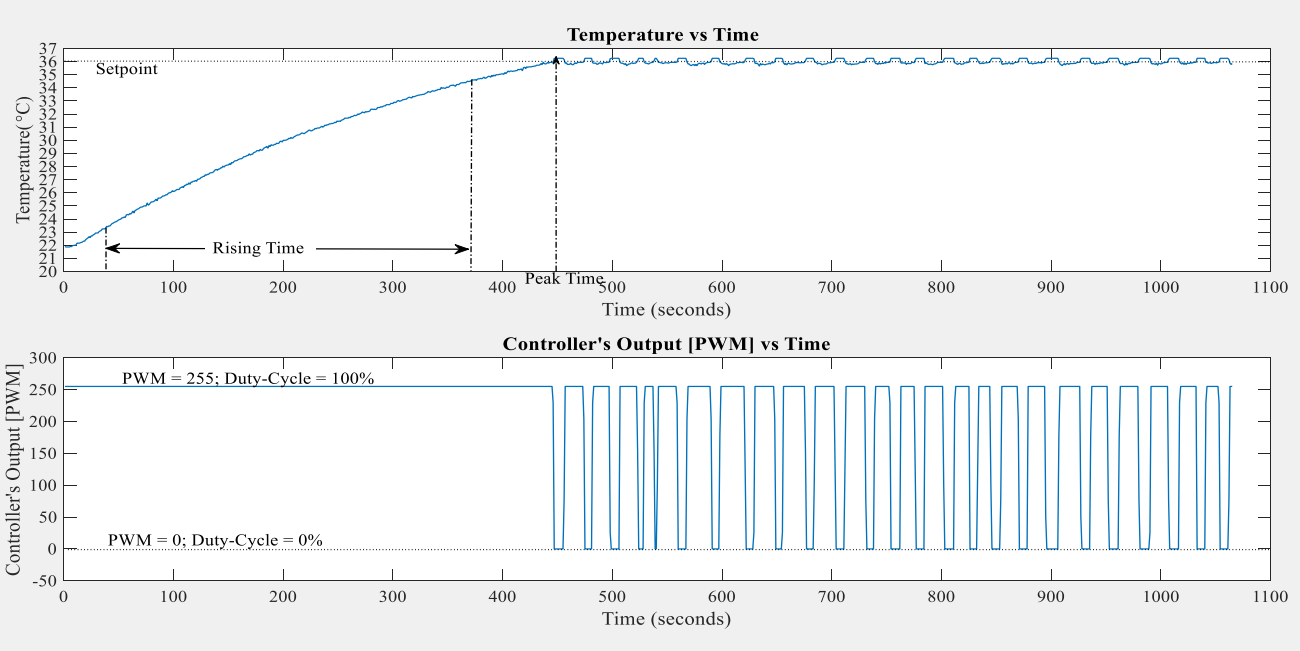


**Fig 4.9: Plots of Temperature and Controller's Output vs Time of PI-Controller with $K_p$=1500 and $K_i$=0.1**

**Table 1.b: List of comparisons of response parameters of P- and PI-Controllers**

| Parameters | $K_p$ = 150 | $K_p$ = 150 and $K_i$ = 1 |
|---|---|---|
| Rising time | 387s | 407s |
| Peak-time | 816s | 567s |
| Settling time | 501s | 696s |
| SSE | 0.86℃ ± 0.6℃ | 0.001℃ ± 0.6℃ (negligible) |
| Overshoots | No overshoots | Overshoots Observed (1.3%) |

**Table 1.c: List of comparisons of response parameters of PI-Controllers**

| Parameters | $K_p$ = 150; $K_i$ = 1 | $K_p$ = 150; $K_i$ = 3 |
|---|---|---|
| Rising time | 407 | 306s |
| Peak-time | 567s | 669s |
| Settling time | 696s | Cannot be determined |
| SSE | 0.001℃ ± 0.6℃ (negligible) | Cannot be determined |
| Overshoots | Few Overshoots just before settling (1.3% of overshoot) | Ample Overshoots with continuous oscillations. |

**Conclusions based on the observations of the PI-Controllers**:

a) SSE is reduced and eliminated.

b) Overshoots observed.

c) Increasing integral gain leads to the decrease in the rising time.

d) Increasing integral gain leads to the increase in overshoots.

e) Adding even a small fraction of an integral gain, i.e. $K_i$ = 0.1, in the P-Controller with $K_p$=1500, the process variable of the system oscillates. Thus, it leads the system to become unstable.

## **IV.iii.** Conclusion

We found that the marking of an exact temperature at any time becomes extremely difficult due to the digitizing error. This error depends on the step size of the ADC, which can be minimized by increasing the resolution of the ADC and decreasing the powering voltage range. However for our temperature control system, the digitizing error of approximately $\pm$0.16℃ is quite acceptable regarding the purpose.

The observed controller's response parameters demonstrated that the on/off controller responds faster but incurs oscillations with the overshoots. For the P-controllers with lower values of the proportional gains, the system settles at the temperature lower than the setpoint with SSE while the system is found to settle near the setpoint for the moderate value of the proportional gain. However, the larger values of the proportional gain leads the process variable to overshoot and oscillate, thereby leading the system to become unstable.

In PI-controller, increasing the integral gain while keeping the proportional gain constant, we observed that the controller responds faster but with the increments in the occurrence of overshoots too. The appropriate choice of the proportional and integral gains leads the system to settle with a very small steady state error values and smaller overshoots. The system becomes stable closer to

the setpoint. Among all the controllers that we designed for this project, we found that the PI-controller with $K_p = 150$; $K_i = 1$, gives a very stable temperature controller system with small overshoots. Hence, the PI-controller with appropriate gain values lead to a more stable and desirable control of our temperature controller system.

Hence, the contribution of my dissertation and the system that I designed are as follows:

1) Regarding the sensitivity of the temperature and the other parameters in the cell incubator in the laboratory, the system that I designed helped me to the basic understanding of the control systems. I learned the methods to build devices and systems in control designs that may help in the further development of better controllers for lower prices and to use them for other applications. The system that I designed can be used as the temperature controller system for the incubator which would cost around $45 to $85 (discussed in appendix 1C). It would be cheaper than the temperature controller, product of Omega platinum series, which we are using for the cell incubator (costs around $230). Although a different sets of actuators should be used to affect the temperature of the incubator. Furthermore, noise should be calculated precisely regarding the temperature sensitivity of the cells in the incubator.
2) The system that I designed may find applications in cooling of the cameras inside cell incubator.
3) The PI-controller is found to help in controlling the focus of microscope by a laser beam reflection off of the sample in our laboratory. This is the new technique for fast sampling of the images from the sample. The knowledge of the functioning of PI-controllers helps significantly to tune the PID parameters for the optimum focus of the sample.

# Appendix 1A

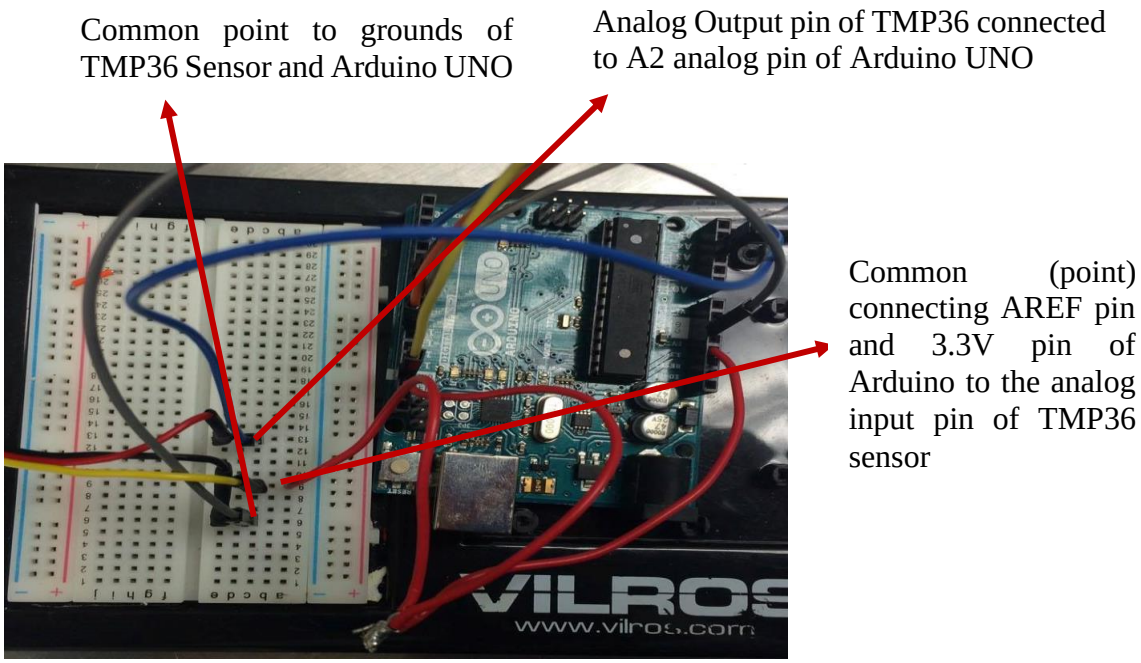


**Fig A1 Arduino UNO: Jumpers from temperature sensors and input channels of operational amplifiers are connected to the analog and digital pins of Arduino UNO.**

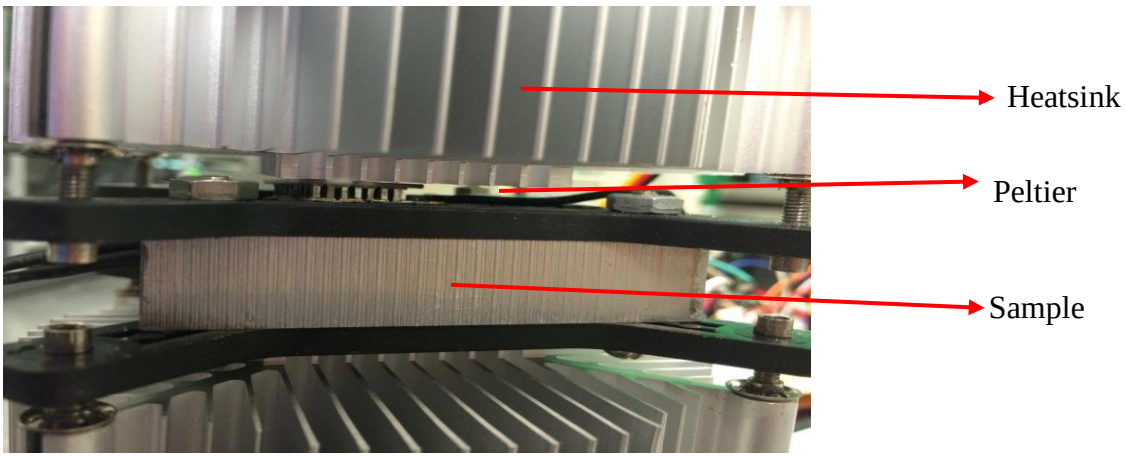


**Fig A2: Closer view of TEC being attached to the sample**

# Appendix 1B

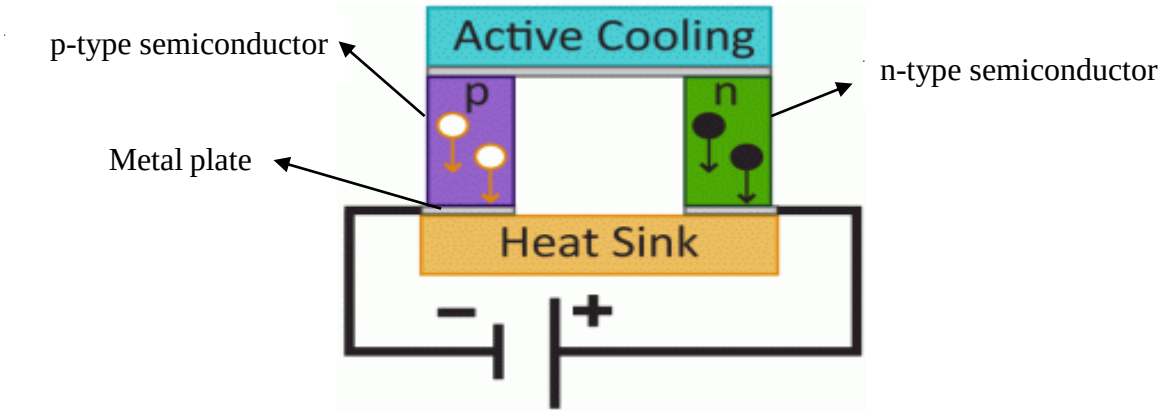


**Fig B1: Working mechanism of Peltier pellets**

The electrons are at greater energy levels in an order from n-type semiconductor, metal, and, p-type semiconductor respectively. The heat is carried along the elements by electron transport and released on the opposite side as the electrons move from a high- to low-energy state and the thermal energy from the surrounding is absorbed for electrons moving from low –to high energy state. When the electrons travel from metal to p-type semiconductor, the energy is released which heats the lower ceramic plate labelled as heat sink. Then, the electrons transferring from p-type semiconductor to metal absorbs heat from surrounding and cools the upper ceramic plate. Likewise, the electrons from metal plate transferring to the n-type semiconductor absorbs heat from the surrounding thereby cooling the upper plate. Finally, the transfer of electrons from the n-type semiconductor transfer to the metal heats the lower ceramic.

# Appendix 1C

Table A1C: Devices and appliances used in the building of our temperature controller system:

| Devices | Name | Number of quantity used |
|---|---|---|
| Thermoelectric Cooler | MARLOW RC3-2.5 | 2 |
| Temperature Sensor | TMP36 | 1 |
| N-MOSFET | FQP30N06L | 8 |
| Operational Amplifier | LM741 Op. Amp | 4 |
| Arduino-UNO | ATmega328 | 1 |
| Resistors | 5 kilo-ohm | 8 |

1) Power Source: Protek P6000 Programmable DC Power Supply is used for supplying power to the actuators through the h-bridge, and a dual power source is used for operational amplifier.

2) Jumping wires, Fret boards, USB cable, and Aluminum bar (Sample) are used for the system design. 0.1µF ceramic capacitor (connected across Voltage Input pin and ground pin of TMP36 temperature sensor to minimize the radio frequency interference.)The Schematic is drawn using Fritzing (open source software).

# Appendix 2

**1) Codes for ON/OFF Controller**

```
int sensorPin = A2; //initiate analog sensorPin 2
int Setpoint = 36.00 ; // setting setpoint value
int Output;//PWM values
float error;
void setup (){
  Serial.begin(9600);
  pinMode(3, OUTPUT); //Initiates digital pin 3
  pinMode(9, OUTPUT); //Initiates digital pin 9
  pinMode(10, OUTPUT); //Initiates digital pin 10
  pinMode(11, OUTPUT); //Initiates digital pin 11
  analogReference (EXTERNAL);//calling external analogReference to use 3.3V pin
  delay(10);}// delaying 10ms to call the 3.3V pin
void loop (){
   float reading = analogRead(sensorPin); // reading ADC values through sensor
   float voltage = reading *3.2 ;//converting ADC value to the voltage
   voltage /= 1024.00; //dividing the voltage into precision of 10 bits
   float Input = (voltage- 0.5) * 100 ;  //converting to the degrees with 10 mV/℃  with 500 mV offset
if (Input<Setpoint){
   analogWrite(3,0);  //turning off output through digital pin 3
   analogWrite(9,255); //Establishes output through digital pin 9
   analogWrite(10,255); //Establishes output through digital pin 10
   analogWrite(11,0); //turning off output through digital pin 11
   Output = 255; } // declaring the output
else{
   analogWrite(3,0); //turning off output through digital pin 3
   analogWrite(9,0); //turning off output through digital pin 9
   analogWrite(10,0); //turning off output through digital pin 10
   analogWrite(11,0);//turning off output through digital pin 11
   Output=0;}  //declaring the output
 Serial.print ("Temperature" );
 Serial.print (Input); //printing the output temperature (PV) in each loop
 Serial.print("Output");
 Serial.println(Output); //printing the controller's output(PWM) in each loop
 delay(100); } //delaying for 100ms
```

**2) Codes for P-Controller with $K_p = 150$**

```
int sensorPin = A2; //initiate analog sensorPin 2
float Kp = 150 ; // Proportional gain values
float Output, Input, error, Setpoint = 36.00; //  variables
void setup (){
 Serial.begin(9600);
  pinMode(3, OUTPUT); //Initiates digital pin 3
  pinMode(9, OUTPUT); //Initiates digital pin 9
  pinMode(10, OUTPUT); //Initiates digital pin 10
  pinMode(11, OUTPUT); //Initiates digital pin 11
 analogReference (EXTERNAL);//calling external analogReference to use 3.3V pin
 delay(10); //delaying for 10ms to call the 3.3V pin
  }
void loop (){
 float reading = analogRead(sensorPin); // reading ADC values through sensor
 float voltage = reading * 3.2;//converting ADC value to the voltage
 voltage /= 1024.00; //dividing the voltage into precision of 10 bits
 Input = (voltage - 0.5) * 100 ;  //converting from 10 mV/℃ with 500 mV offset
 error = Setpoint - Input; // calculate error
Output = Kp * error; // preparing the output variable
 if (Output > 255)
Output = 255; // keeping the output within the available output range
 if (Output < 0)
 Output = 0;
  analogWrite(3,0); // turning off output through digital pin 3
  analogWrite(9,Output); //Establishes output through digital pin 9
  analogWrite(10,Output);// Establishes output through digital pin 10
  analogWrite(11,0);   //turning off output through digital pin 11
   Serial.print(" TEMPERATURE ");
   Serial.print (Input); // printing the output temperature in each loop
   Serial.print("  PWM  ");
  Serial.println(Output); //printing the controller output (PWM) in each loop
  delay(100);//delaying for 100ms
```

**3) Codes for PI-Controller with $K_p = 150$ and $K_i = 1$**

```
int sensorPin = A2; //initiate analog sensorPin 2
float Kp = 150, Ki = 1; // PID gain values
float error, Output, Input, ITerm, Setpoint = 36.00; // Setting variables
void setup () {
 Serial.begin (9600);
 pinMode (3, OUTPUT); //Initiates digital pin 3
 pinMode (9, OUTPUT); //Initiates digital pin 9
  pinMode (10, OUTPUT); //Initiates digital pin 10
```

```
 pinMode (11, OUTPUT); //Initiates digital pin 11
 analogReference (EXTERNAL); //calling external analogReference to use 3.3v pin
 delay (10);
  }
 void loop () {
 float reading = analogRead (sensorPin); // reading ADC values through sensor
 float voltage = reading * 3.2;//converting ADC value to the voltage
 voltage /= 1024.00; //dividing the voltage into precision of 10 bits
 Input = (voltage - 0.5) * 100; //converting from 10 mV/℃ with 500 mV offset
 error = Setpoint - Input; // calculate error
 ITerm += (Ki *error); // add current error to running total of error
if (ITerm > 255) // check for integral windup and correct for upper limit
ITerm = 255;
if (ITerm < 0)//check for integral windup and correct for lower limit
 ITerm = 0;
 Output = Kp * error + ITerm; // preparing the output variable
 if (Output > 255); // keeping the output within the available output range
 Output = 255; //
 if (Output < 0)
 Output = 0;
  analogWrite(3, 0); //turning off output through digital pin 3
  analogWrite (9, Output); //Establishes output through digital pin 9
  analogWrite(10, Output); //Establishes output through digital pin 10
  analogWrite(11, 0);   //turning off output through digital pin 11
  Serial.print("TEMPERATURE");
  Serial.print(Input); printing the Temperature (PV) in each loop
  Serial.print(" PWM ");
  Serial.println(Output); //printing the controller's  output (PWM) in each loop
  delay(100);//delaying for 100ms
```

# Bibliography


Alvi, B. A., Asif, M., Siddiqui, F. A., Safwan, M., & Bhatti, J. A. (2014). Fast Steering Mirror Control Using Embedded Self-Learning Fuzzy Controller for Free Space Optical Communication. Wireless Personal Communications, *76*(3), 643-656.

Arduino-ArduinoBoardUno. (n.d.).

Retrieved from https://www.arduino.cc/en/Main/ArduinoBoardUno

Aziz, M. M. (2010). Transfer Function and Block Diagrams. Retrieved from ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ea619_2s12/transfer_function_block_diagram.pdf

Bret. (2011, April). Improving the Beginner's PID – Introduction « Project Blog. Retrieved from http://brettbeauregard.com/blog/2011/04/improving-the-beginner-pid-introduction/

Dominguez, B., Barba, F., & Castrejon, C. (2014, December 8). DC Motor Driver by EMG and EOG | imditesmprojects. Retrieved from https://imditesmprojects.wordpress.com/dc-motor-driver-by-emg-and-eog/

Fried, L. (2012, July 29). Overview | TMP36 Temperature Sensor | Adafruit Learning System. Retrieved from https://learn.adafruit.com/tmp36-temperature-sensor/overview

Gupta, S., & Pathak, A. (2012, January). ADC Guide, Part 1- The Ideal ADC. Retrieved from http://www.cypress.com/file/113946/download

Hunt, O. (n.d.). HVLabs.com. Retrieved from http://www.hvlabs.com/hbridge.html

Levine, W. (1996). *The Control handbook* (Electrical engineering handbook series). Boca Raton, Fl.: [New York, NY]: CRC Press; IEEE Press.

Nath, A.S., R, A.K., & Malik, T. (2013). "Implementation of PID Control to Reduce Wobbling in a Line Following Robot". International Journal of Research in Engineering and Technology,2(10).doi:10.15623/ijret http://esatjournals.net/ijret/2013v02/i10/IJRET20130210083

National Instruments. (2011, March 29). PID Theory Explained. Retrieved from http://www.ni.com/white-paper/3782/en/

Peacock, F. (2008). How the PID Algorithm works and why it works. Retrieved from www.PID-Tuning.com

Premeaux, E., & Evans, B. (2011). Saving the world. In *Arduino Projects to Save the World, Spider temp*. Berkeley, CA: Apress.

Rhinehart, R. (2000). The century's greatest contributions to control practice. *ISA Transactions, 39*(1), 3-13.

Roberge, J. K., & Lundberg, K. H. (2007). Background and Objectives. In *Operational Amplifiers: Theory and Practice* [1.81] (2nd ed., pp. 1-11). Retrieved from http://web.mit.edu/klund/www/books/opamps181.pdf

Sparks, T.D. (2013). How do thermoelectrics work? Retrieved from http://www.eng.utah.edu/~sparks/how-do-thermoelectrics-work.html

Theopaga, A. K., Rizal, A., & Susanto, E. (2014). Design and implementation of PID control based baby incubator *Journal of Theoretical & Applied Information Technology*, *70*(1).

Vikhe, P., Punjabi, N., & Kadu, C. (2014). Real Time DC Motor Speed Control using PID Controller in LabVIEW. *IJAREEIE*, *03*(09), 12162-12167. doi:10.15662/ijareeie.2014.0309046

Vilanova, R., & Visioli, Antonio. (2012). *PID control in the third millennium lessons learned and new approaches* (Advances in industrial control). London; New York: Springer.

Welander, P. (2010, February 1). Understanding Derivative in PID Control. Retrieved from http://www.controleng.com/search/search-singledisplay/understanding-derivative-in-pid-control/4ea87c406e.html