Holding the Temperature of a Cryostat Constant using the Arduino PID Libraries

Abstract

For this project, we created a device that held the temperature of a cryostat constant while it was lowered into liquid nitrogen. We used an Arduino board as it is designed to have easily understandable and effective hardware and software at an efficient price. We developed a program where our Arduino board converted a digital signal to an analog signal by using a solid-state relay and PID controller libraries. The PID (Proportional, Integral, Differential) controller allowed us to bring our current temperature read by a thermocouple to a set temperature by calculating the difference between them and inputting that difference into three respective PID equations. They each produce their own output that are all added together to produce a total output used to turn the relay on for a certain amount of time, then off again. When the relay was on, power flowed from a source to the heating tape coiled around the cryostat. The power flow stopped when the relay was off; this cycle slowly changed the cryostat's temperature. By determining the best constants for the PID equations, we were able to hold the temperature of the cryostat constant relative to our set temperatures while lowered into liquid nitrogen.

Introduction and Theory

Cryostats are devices meant to keep a constant low temperature, especially below 0℃. They use liquid cryogens, such as nitrogen and helium, to keep the temperature of certain samples cool[1]. While creating a device for maintaining the temperature of a cryostat can be done with analog hardware, creating a digital device is more efficient, uses less power, and is cost effective. Arduino is an electronics platform with easy-to-use hardware and software designed for anyone to do electronic projects[2]. Using the PID libraries and board from Arduino, one could

create a temperature controller that loops the difference between the temperature we inputted and the current temperature of the cryostat. This project's intention is to ease the labor of creating a temperature controller for a Cryostat, as the coding language for Arduino is easy enough to understand even for a beginner and the low cost of the hardware needed. For this experiment, we built and tested the efficiency of a digital temperature controller that was used to keep the temperature of a cryostat constant while in a dewar of liquid nitrogen using the Arduino PID libraries.

The acronym PID stands for Proportional, Integral, and Derivative, which are the three properties that are used for such a controller. The controller used a reference variable *r(t)*, in this case would be the set temperature, and found the difference between that and a process variable *y(t)*, the current temperature of the cryostat, and called that value the error *e(t)*[3]. The error *e(t)* is processed through the each of the three properties and inserted into the following equations:

$$P = K_p e(t) \tag{1}$$

where K_p is a constant directly proportional to the error *e(t)*,

$$I = K_i \int_0^t e(t)dt \tag{2}$$

where K_i is a constant directly proportional to the integral of the error *e(t)*, and finally

$$D = K_d \frac{d(e(t))}{dt} \tag{3}$$

where K_d is a constant directly proportional to the derivative of the error *e(t)*. However equations (2) and (3) differ for digital electronics; the new equations are

$$I = K_i \sum_{n=0}^{k} e(n) \tag{4}$$

and

$$D = K_d(e(t) - e(t-1)) \quad (5)$$

with respect to the integral and derivative properties of the PID controller[4]. Due to the mathematical definitions of integrals and derivatives, this did not make any significant difference. Each of the three PID properties affected the output of the PID cycle differently. K_P adjusted the output based on the current value of *e(t)* and brought *y(t)* closer to *r(t)* as K_P was proportional to the value of *P*, therefore, proportional to the output [3]. K_I adjusted the PID output based on past values of *e(t)*, meaning as more integrals were calculated for *I*, there was a greater effect on the output over time, which made *y(t)* more likely to reach *r(t)*[3]. K_D adjusted the PID output by predicting the future values of *e(t)*, so if *y(t)* approached *r(t)* too quickly, the derivative decreased the output[3]. By changing the constants, we could adjust the output of the PID cycle and the rate of temperature change. The PID controller took the sum of the values for *P*, *I*, and *D*, which we called the control variable *u(t)*[3],

$$u(t) = P + I + D \quad (6)$$

which we will explain how it was used in the next section. We used *u(t)* to minimize the error between *r(t)* and *y(t)* by creating a new *y(t)* that will be continuously looped back into the PID controller until *e(t)* is at a minimum.

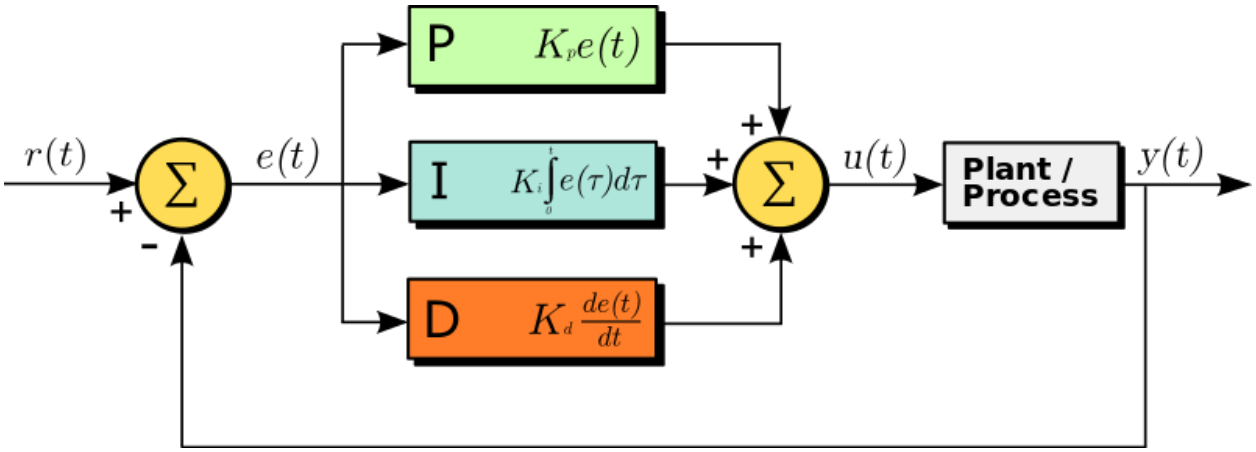

Fig. 1: Block Diagram of PID Cycle[5]

Many used this method of minimizing the error between a set value and a current value to create a temperature controller, but mostly for home thermostats or cooking utilities. We decided to use this method for a more scientific purpose and created a temperature controller to keep a cryostat at a constant temperature.

The Experiment

In this experiment, we used the Arduino web editor available through any web browser to create and upload the program that would send a signal to the Arduino board to start the PID cycle. The program that we wanted to use was originally created for purely analog hardware, but we modified it to account for digital hardware we plan to use[6]. The code was modified by inserting a program known as the "PIDRelayOutput" method in order to convert an analog signal to a digital one[7], and by inserting a command to "attach interrupts" so that the Arduino board could read the rotary encoder and we could change the temperature by turning a knob[8]. The Arduino board was connected to a breadboard with a rotary encoder that was designed to adjust the set temperature *r(t)* of the cryostat and an LCD screen that displayed the current temperature *y(t)* of the cryostat. With a mix of digital and analog equipment, we needed a way to send the digital signal to our analog equipment without the use of D/A converters.

Digital electronics differ from analog electronics as they only have two outputs: one and zero. In order for this experiment to work with Arduino, we needed to encode the digital signal from a laptop to an analog signal for the power to increase the temperature. We accomplished this task by using the digital signal from the uploaded program as a makeshift voltage to create a pulse width modulation (PWM) with a solid state relay (SSR) attached to a variac that would limit the voltage from the power supply that provided the power to increase the temperature. For this program, another measure we added to ensure that *y(t)* never exceeded *r(t)* was to extend the

period of the PWM to thirty seconds. The program measured the amount of time that it started in milliseconds, so we used that to create the period and the amount of time that the SSR stayed on. If the difference between the current amount of time and a second value that continuously increased by thirty seconds was greater than *u(t)*, the SSR would turn on for T_o seconds. After T_o seconds passed, that difference would be greater than *u(t)*, which caused the SSR to turn off until the difference was less again after (30 - T_o) seconds. This created an analog PWM signal by pulsing a certain amount of power at a time that gradually changed the current temperature to *r(t)*. The SSR was also connected to a variac, which was used to limit the amount of voltage from the power supply that would run through the SSR to increase the temperature. The SSR acted like a digital switch that turned the power from the variac on and off to create a PWM. By increasing *r(t)*, the efficiency of the PWM also increased as well, which also increased the amount of power going through the SSR, which causes an increase in temperature.

The voltage from the power supply, which was limited by the SSR, was converted to power that was sent to some heating tape that was coiled around the cryostat. The temperature of the heating tape changed depending on the "on-time" of the PWM. The heat from the heating tape also changed the temperature of the cryostat that was lowered into the dewar of liquid nitrogen. The cold finger on the end of the cryostat did not need to be fully dipped into the liquid nitrogen, but it had to be at a lower temperature than the contents of the cryostat. The temperature was read by the thermocouple inside of the cryostat, which read the temperature as an analog signal. The thermocouple was connected to a thermocouple-to-digital converter capable of reading temperatures below 0℃, which sent the reading back to the Arduino board where it was looped back into the PID control unit as value *y(t)* as well as displayed on the LCD

screen and recorded on a laptop. The loop continued until *e(t)* was at a point where it held the temperature of the cryostat constant.

For the data collection, it was simply a matter of determining which values of K_P, K_I, and K_D were the most efficient at keeping the temperature constant while the cryostat was inside the dewar of liquid nitrogen. The Arduino web editor had a window to the side that would print out values if you put the command to print them out in the software. The program printed out a new row into the side window every second that contained the time in seconds, *r(t)*, *y(t)*, *u(t)*, and either a 1 or a 0 depending if the SSR was on or not in that instance of time. After setting a value for r(t) using the rotary encoder, we waited as the PID-PWM cycle automatically caused the temperature *y(t)* to approach *r(t)*. After keeping y(t) at an approximately constant temperature, whether or not *y(t)* was approximately constant at r(t), we changed the value r(t) to a new temperature and repeated the cycle of waiting until *y(t)* was held constant. This process of changing *r(t)* and waiting repeated about 2 or 3 times per run. The rows were then pasted onto a spreadsheet for analysis later. Afterwards, depending on whether the average temperatures of *y(t)* were over or under the set temperatures of *r(t)* during the run, we would adjust the hardware and software parameters.

If the current temperature overshot the set temperature by a significant amount (1℃ or greater), then we either decreased K_I, increased K_D, or decreased the voltage through the variac. If the current temperature undershot the set temperature by a significant amount (same error values as above), we did the opposite where we either increased K_I, decreased K_D, or increased the voltage through the variac. Since the resolution of the thermocouple hardware was 0.25℃, limiting the oscillations to 1℃ or less was not unreasonably high as a goal. After that, we performed another run; we performed runs of the temperature controller until we found which

PID constants and voltage value kept the value of *y(t)* held constant at approximately *r(t)* through trial-and-error experiments.

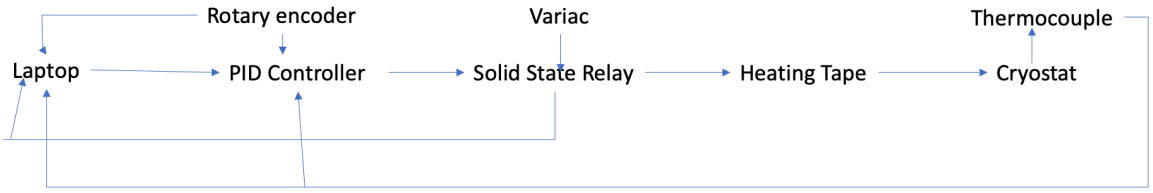

Fig. 2: Basic Schematic for Experiment Design

Data and Results

We performed approximately twenty-two runs where we measured the temperature of the cryostat as it was lowered into the liquid nitrogen every second. Out of all the runs, the final run had *y(t)* approach *r(t)* the closest with minimum oscillations. The constants that gave us this result were $K_P = 200$, $K_I = 2$, and $K_D = 420$, with the variac set to only provide approximately 104 V from the power source, which was about 85% of the maximum voltage. These values were found through a series of trial and error where we continuously adjusted the constants within the code and the voltage from the variac to find the closest possible temperatures to *r(t)*.

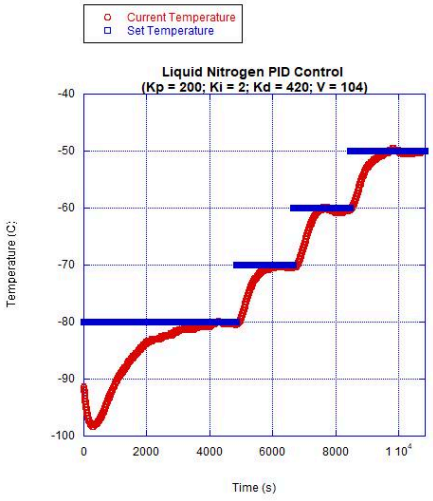

Fig. 3: Final Results for the Cryostat Temperature

Though a small oscillation was present, the current temperature *y(t)* had minimum error to the set temperature r(t). This insignificant error shows that our PID temperature controller is efficient in keeping the temperature constant while it is lowered in liquid nitrogen.

Furthermore, we showed that there was a direct correlation between the value of the set temperature *r(t)* and the PWM produced by the SSR. The program was also made to record when the SSR was turned on every second; when it was on, a 1 was produced, when it was off, a 0 was produced. We took these 1's and 0's and put them into a linear graph over time to create a visual representation of the duty cycle.

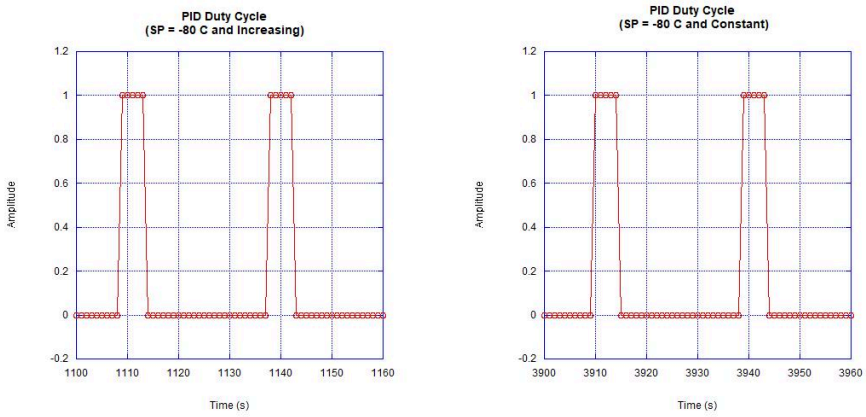

Fig 4: PWM at *r(t)* = -80℃

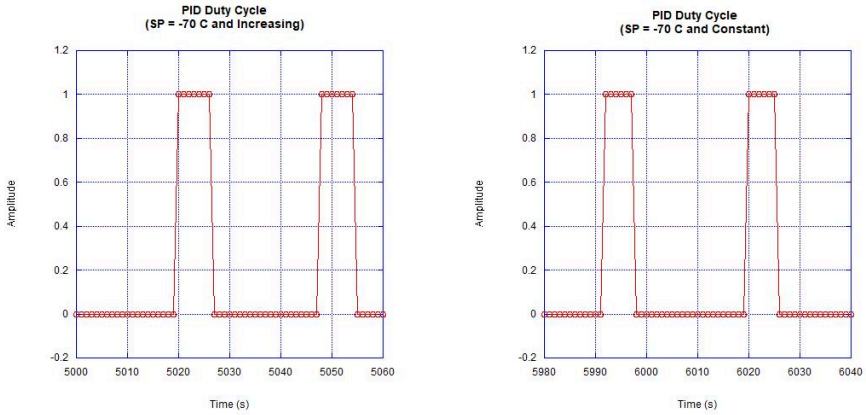

Fig 5: PWM at *r(t)* = -70℃

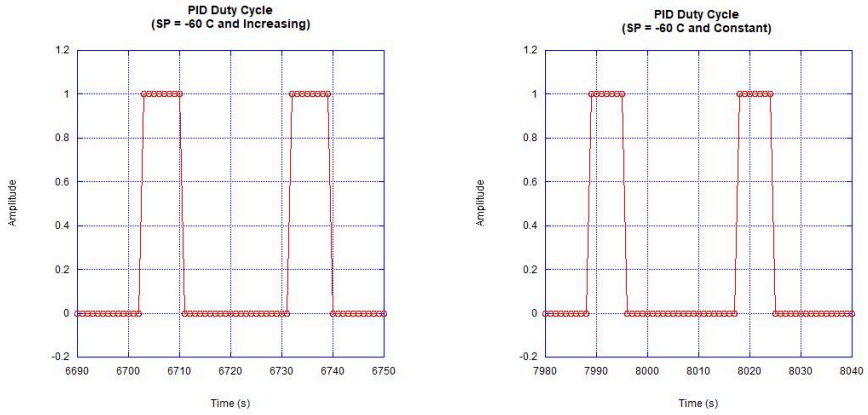

Fig 6: PWM at *r(t)* = -60℃

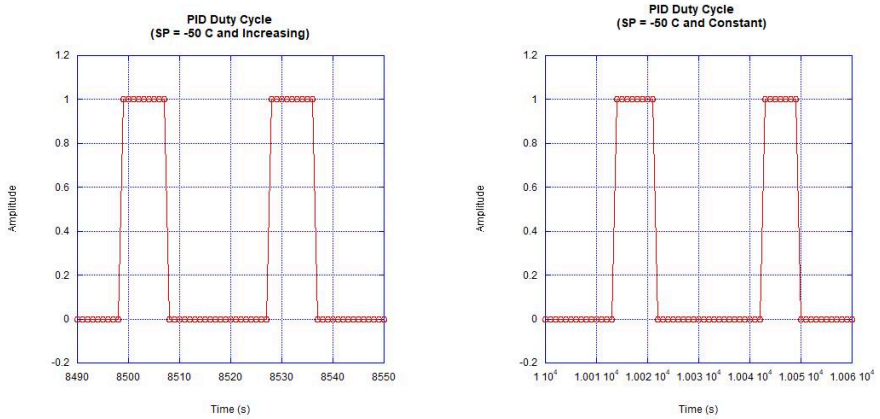

Fig 7: PWM at *r(t)* = -50℃

Though the period was set at a constant thirty seconds for all temperatures, the crest of the amplitude got wider when the temperature increased. As shown in the figures above, the on-time of the PWM increased whenever the set temperature *r(t)* was increased and decreased when *r(t)* decreased as well. Furthermore, the efficiency was shown to be larger as *y(t)* increased to *r(t)*, but when it got closer and was held relatively constant to *r(t)*, the efficiency decreased to prevent *y(t)* from exceeding *r(t)*. Due to this observation, We proved that there was a direct correlation between the set temperature and the PWM produced by the signal sent by the program and the SSR.

Conclusions

Using the electronic hardware and PID libraries from Arduino, we were able to keep the temperature of a cryostat relatively constant while it was lowered into liquid nitrogen. Through a series of trial and error experiments, we found the PID constants and exact voltage that kept the oscillations of the cryostat's temperature at a minimum as the current temperature *y(t)* approached the set temperature *r(t)*. Furthermore, we proved that there was a direct correlation between *r(t)* and the PWM produced by the SSR that was used to change the temperature. The efficiency of the PWM increased as the value of *r(t)* increased and vice versa. In retrospect, we could have placed a resistor into the cryostat to have further limited the power from the heating tape to the cryostat and could have prevented the temperature from overshooting. We could have also used a different variac as we believed that the one we used was malfunctioning due to age. We noticed close to the end of the experiment that the voltage going through it would oscillate when set to a single percentage of the maximum voltage, so that may have caused the temperature to undershoot slightly in the final run. One thing that we weren't able to accomplish was determining the resistance of a superconductor using the cryostat and a 4-contact probe, so maybe someone else can perform that experiment with the program I'm leaving behind. Cryostats are designed to preserve certain tissue samples by keeping them at a temperature below 0℃, meaning they can be used for other biological experiments around campus. Studies that involve preserving microorganisms such as bacteria and viruses can be performed much more efficiently with a cryostat whose temperature can be controlled. We believed that this experiment would be beneficial to others because with a functioning cryostat whose temperature can be controlled and kept constant, many other students can study tissues that need to be preserved and conduct their own experiments.

References

1. Mitchinson, S. (2022, August). *What is a cryostat?* Oxford Instruments. Retrieved February 7, 2023, from https://andor.oxinst.com/learning/view/article/what-is-a-cryostat
2. (2018, February). *What is Arduino?* Arduino. Retrieved February 7, 2023, from https://www.arduino.cc/en/Guide/Introduction
3. *PID Controller Explained.* PID Explained. Retrieved February 7, 2020, from https://pidexplained.com/pid-controller-explained/
4. R. S. Guerra. A Digital PID Controller Using RTAI. Federal University of Rio Grande do Sul. Retrieved March 6, 2022, from http://rodrigoguerra.com/wp-content/uploads/2017/12/a05_guerra.pdf
5. Urquizo, A. (2011). PID controller overview [PNG]. Wikimedia. https://commons.wikimedia.org/wiki/File:PID_en.svg
6. Das, D. (2021, November). *Arduino PID Temperature Controller using MAX6675 K-Thermocouple to Digital Converter IC*. Circuit Digest. Retrieved February 7, 2023, from https://circuitdigest.com/microcontroller-projects/arduino-pid-temperature-controller
7. Arduino Playground. Retrieved February 7, 2023, from https://playground.arduino.cc/Code/PIDLibraryRelayOutputExample/
8. How Rotary Encoder Works and Interface It with Arduino. Last Minute Engineers. Retrieved February 7, 2023, from https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/

Acknowledgements

I'd like to thank Dr. Kevin Riggs for helping me understand how to set up the hardware for this project. I'd also like to thank Dr. Holley Lynch for giving us helpful suggestions for how to fix the problems that arose during the course of this project. I'm also thankful to my fellow seniors Grayson Taber, Lean Lategan, Xavier Inosencio, and Ren Pupo for their help as well. Finally, I'd like to thank the Stetson Physics Department for funding this research.