University of British Columbia Department of Electrical and Computer Engineering

Group 2 Final Report

Ana Bandari, Lehan Chandrabharthi, Nakul Dharan, Rex Ng
ELEC 291
Professor Saloome Motavas
June 25th, 2023

Group Member	**Student Number**	**Contribution to Project**
Ana Bandari	96047642	100
Lehan Chandrabharathi	53926895	100
Nakul Dharan	18351791	100
Rex Ng	13108162	100

Table of Contents

Design Decision Making

Our final implementation was developed from the continuous iteration and optimization of two key components of this project: the temperature sensing circuit and the control logic circuit. This allowed for a more thorough understanding of the functionality of our circuit, streamlined the prototyping process, and enabled effective debugging. To adhere to the counsel of our stakeholders, the deviations in prototypes come from how we chose to increase the temperature range, lower power consumption, and decrease the number of components. As these circuits have individually gone through several prototyping stages, we will go over each of these circuits in depth in the following two subsections.

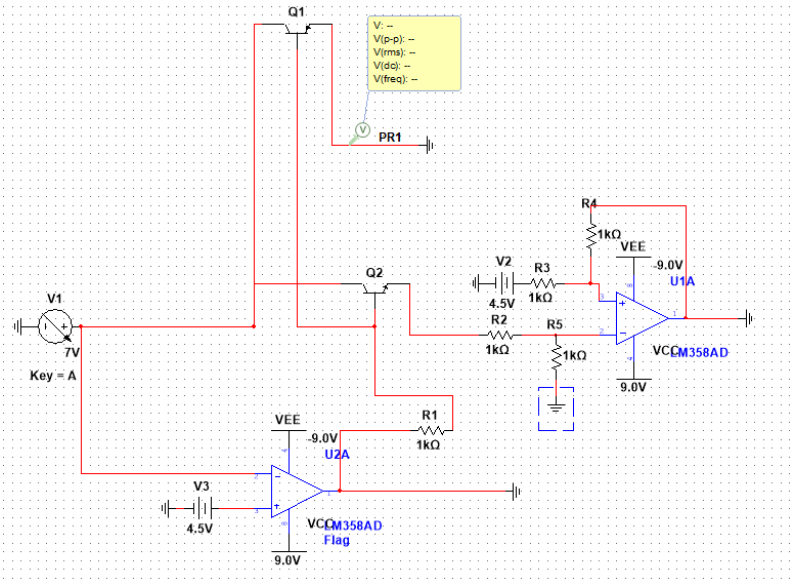
Part A - Temperature Measurement

The temperature measurement circuit utilizes a thermistor to measure temperatures within a defined range, determined by output voltages from a Wheatstone bridge. These output voltages are then inputted into the Arduino Uno microcontroller and mapped to temperature values for a range that ensure 0.1°C precision. It is important to note that the Arduino's input pins cannot exceed 5V, necessitating the implementation of circuit protection. To maintain the balancing act of having a larger temperature range, ensuring circuit protection, and prioritizing our stakeholder's values, we worked on different routes of optimization for this circuit that we will detail below:

Larger voltage range

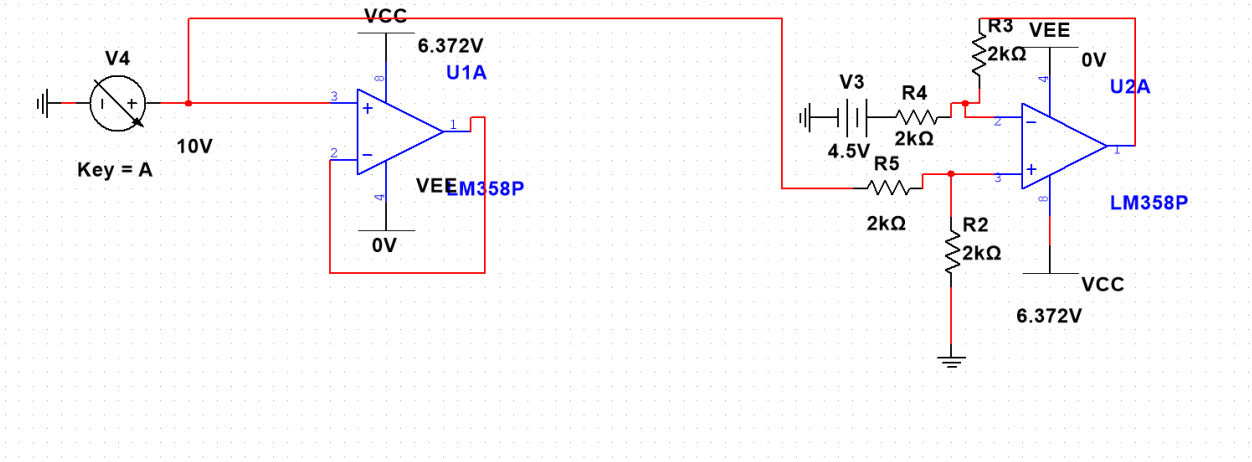
As the output voltage is directly correlated with the range of temperature values, one of the first optimizations utilized output voltages higher than 5V, particularly having a maximum output of 9V. This was troubling as the Arduino Uno microcontroller is only capable of voltages lower than 5V. However, using a comparator circuit that checks if the output voltage is above or below a 4.5V threshold ensures circuit protection and the possibility of an increased voltage range. When the value is above this threshold, we can set a flag in the code and feed the output into a voltage subtractor that

brings the maximum output voltage to 4.5V. We would then be able to map the extended voltage range to a broader temperature range.



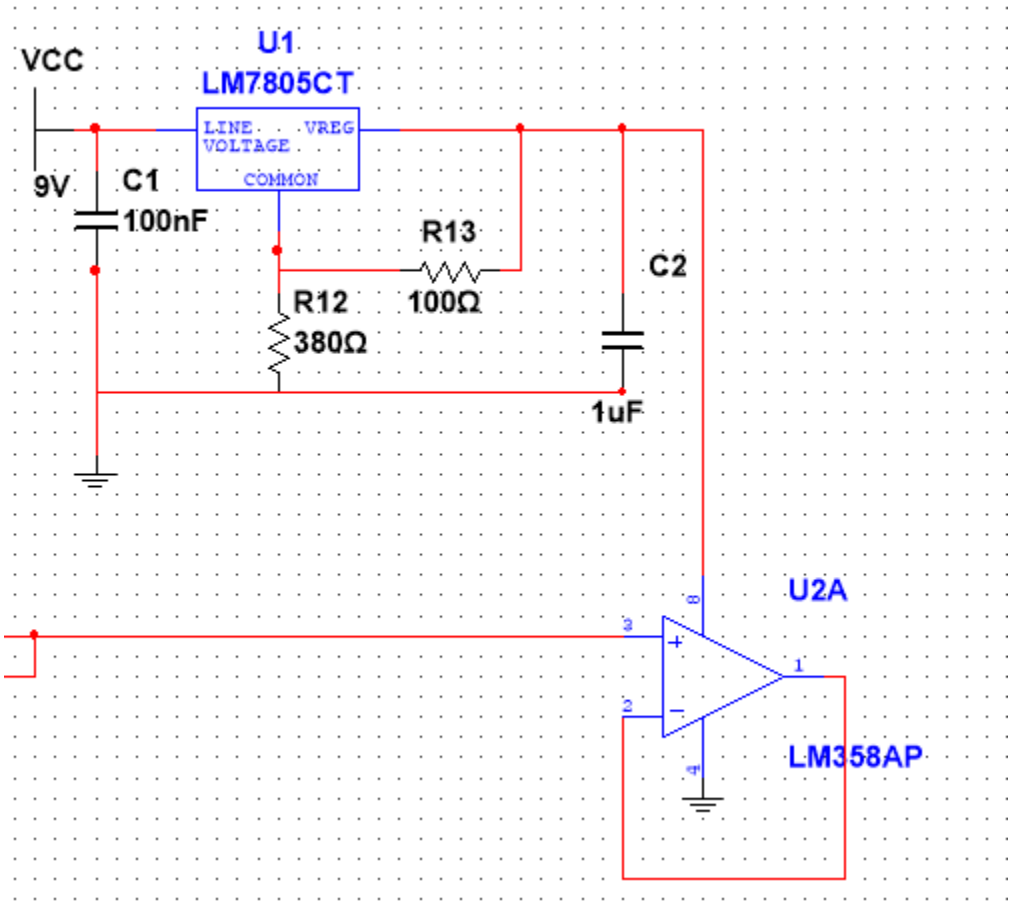
Saturated Op-Amp with Subtractor

Another approach that prioritized circuit protection utilized a voltage follower that saturated at voltages above a certain range. Voltages above a certain threshold would also be subtracted and outputted to the Arduino, to ensure circuit protection and increase the maximum voltage range. This acts as a simplified version of the previous design, in which we utilize a larger voltage range to map temperature values.



Saturated Op-Amp with Voltage Regulator

The final approach we chose to ensure circuit protection was to saturate an op-amp and use it in conjunction with a voltage regulator to ensure that the voltage powering the op-amp remains at 6V. The output of the op-amp feeds into the arduino and does not exceed 4.88V, as the output voltage of an op-amp cannot exceed 85% of its input voltage. However, the exact percentage is dependent on the specific op-amp.



Temperature Measurement WDM

Criterion	Weight	Larger voltage range	Saturated Op-Amp with Subtractor	Saturated Op-Amp with Voltage Regulator
Power Consumption	36%	6	8	9
Number of components	32%	7.5	10	10
Temperature range	32%	8	6	6
Total	100%	7.12	8	8.36

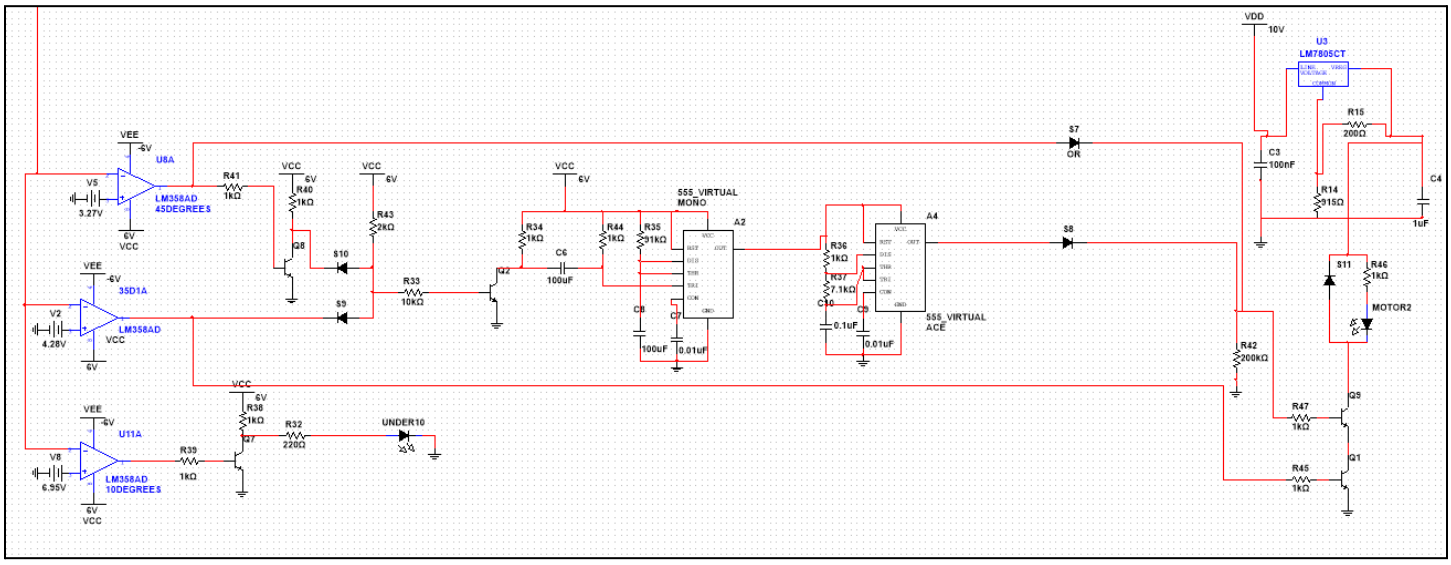
Justification

The designs above demonstrate that most critical optimizations addressing sustainability are related to adjusting the output voltage of the Wheatstone bridge. Bearing this in mind, it is essential to understand how each design scores based on respective criteria. In terms of power consumption, the design with the larger voltage range scores lower since it uses 9V

to power both op-amps. Conversely, the other two circuits—Saturated Op-Amp with Subtractor, and Saturated Op-Amp with Voltage Regulator—power the op-amps with around 6V. The number of components in these designs, from top to bottom, are 8, 6, and 6 respectively, where 6 is given a full score. The temperature range for the circuits incorporating subtractors or switches is broader since they can process voltages above 4.5V and relay them to the Arduino. However, a voltage loss may occur during this process. The first circuit outperforms the second because the usage of a voltage subtractor may result in a slight reduction of the voltage range.

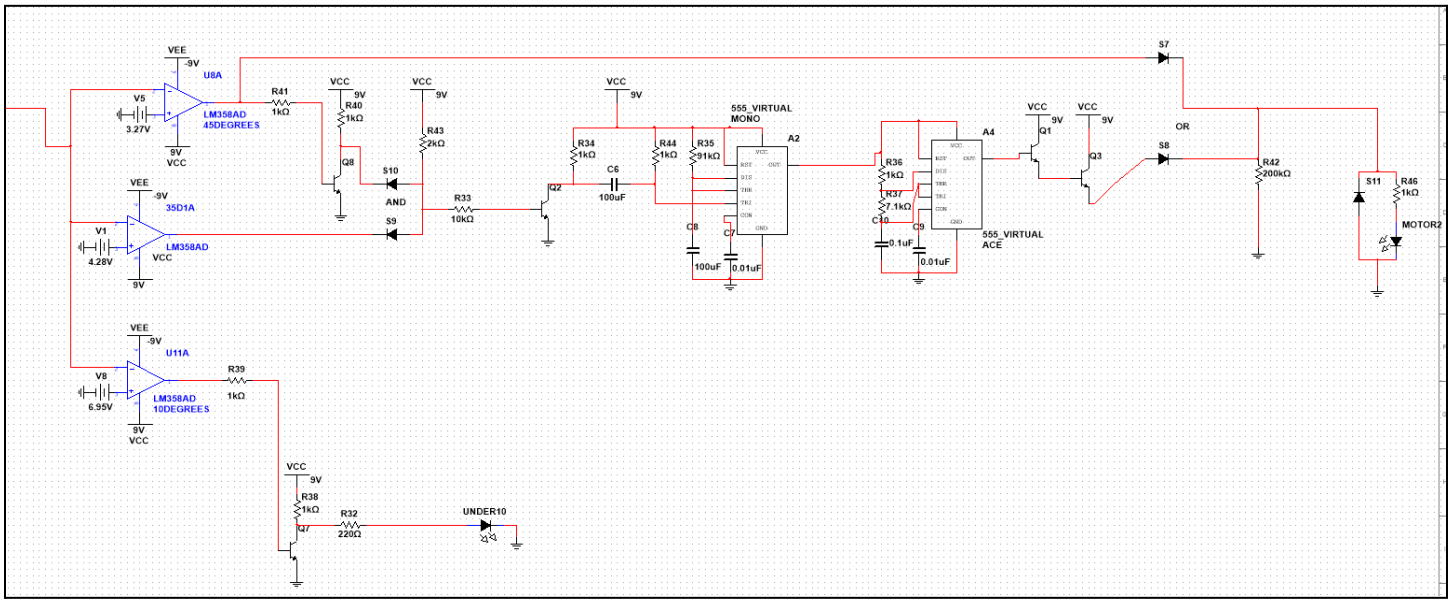
Part B - Control logic

Diode-based digital logic



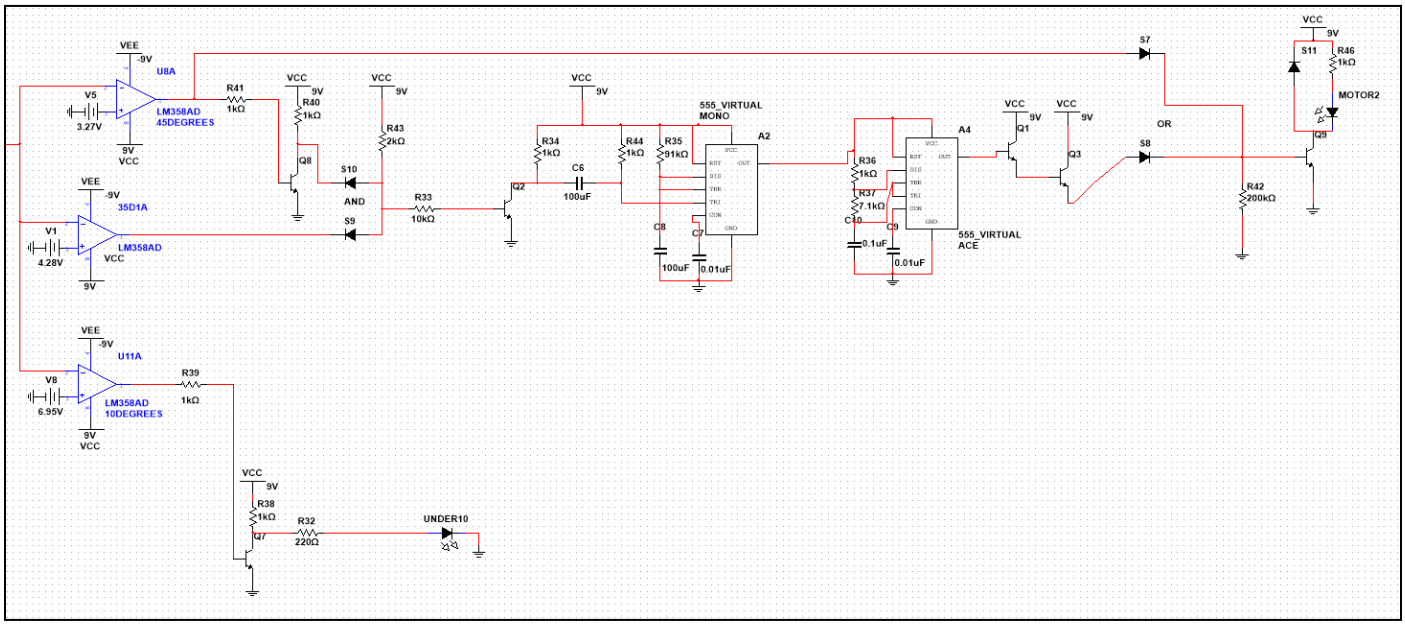
Motor Control through VCC

Our initial design involved powering the motor directly from the logic circuit's 9V supply line. This approach, despite its simplicity and reduced component count, encountered issues in balancing the operational needs of the motor and the control logic. The motor's high current demand, when sourced from the shared VCC, led to erratic motor behavior and instability in the control circuit. We also noticed that this led to an increase in the overall component count, negating the initial advantage of simplicity.



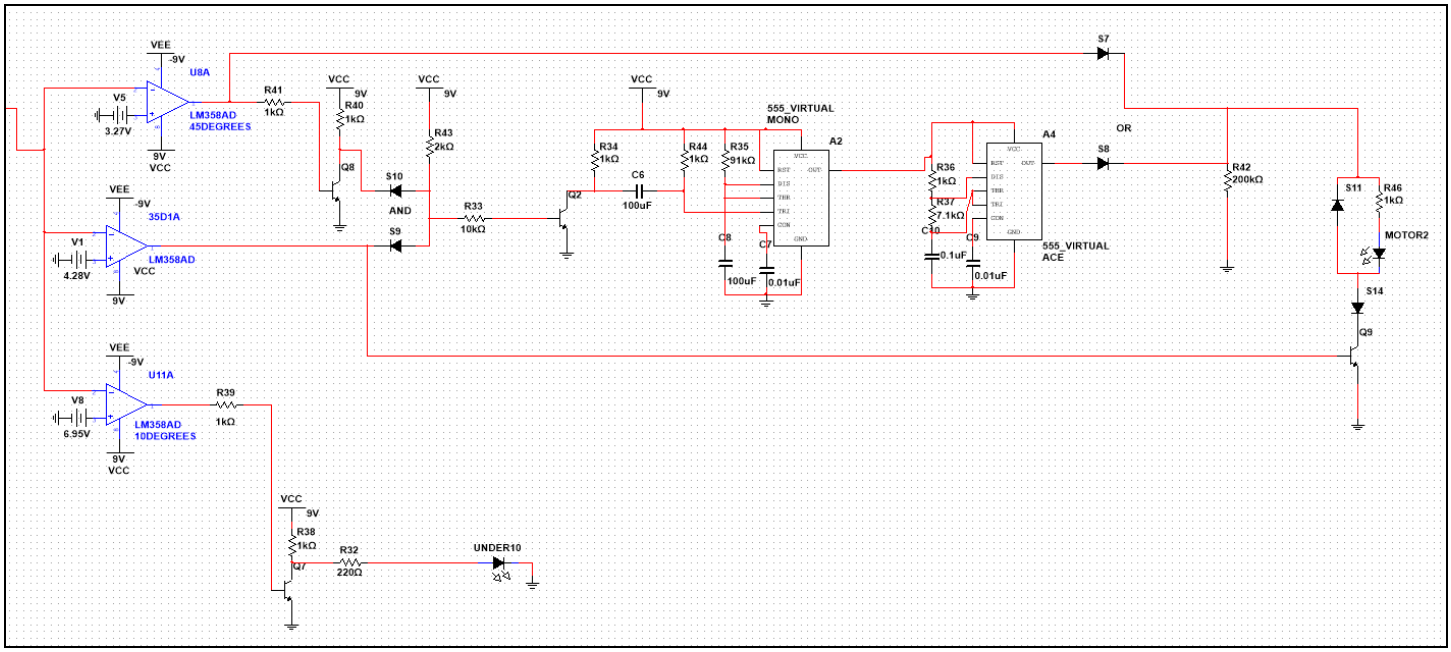
Motor Control through GRD

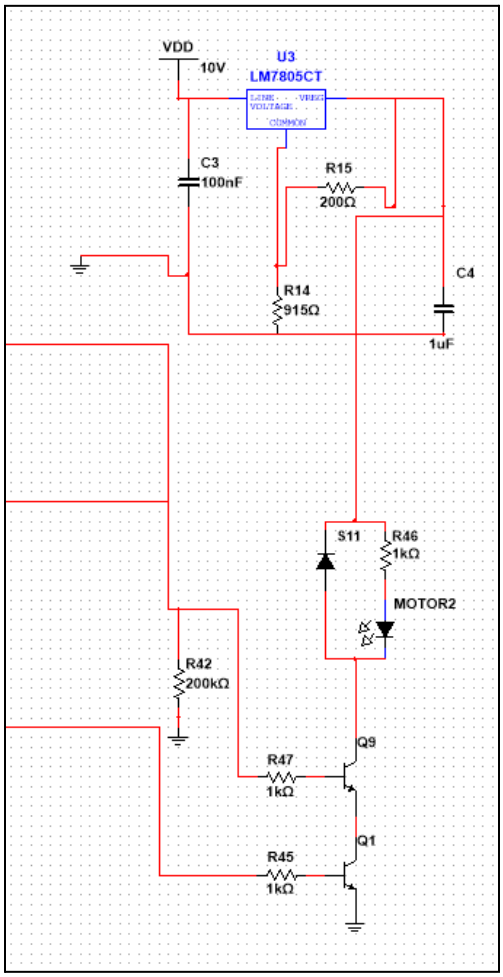
In our second design iteration, we shifted the control focus to the motor's ground connection. Through rigorous experimentation with various configurations involving NPNs, Darlington pairs, and PNPs with inverters, we found controlling the motor's connection to the ground yielded more predictable outcomes compared to controlling the connection to VCC. A combination of inverters and AND gates allowed us to manage an NPN, which effectively governed the motor's ground connection, turning it on and off as required.



Motor Control through GRD and Separate VCC

The third and final iteration involved connecting the motor to its dedicated 10V supply rail, isolated from the logic circuit's power supply. This modification resolved issues observed in the second design, where the motor's high current demand at startup and during operation caused erratic behavior in the logic circuit. With a voltage regulator ensuring the motor received a consistent 7V at full speed, the motor operation became stable and reliable. Furthermore, we identified that the motor would never need to turn on again once a certain voltage level (4.28V) was reached. This understanding allowed us to use the output of the middle op-amp (compared with 4.28V) to directly control the NPN, thus reducing component count and simplifying the design.





Control Logic WDM

Criterion	Weight	Motor Control through VCC	Motor Control through GRD	Motor Control through GRD and Separate VCC
Power Consumption	36%	10	10	8.9
Number of components	32%	6	8.32	10
Temperature range	32%	N/A	N/A	N/A
Total	100%	8.72	9.4624	10

Justification

For power consumption, the first and second options were both powered by 9V VCC whereas the final option was powered by 10V VCC; ergo, this 1V increase is captured by the lower score of the final option: $0.\overline{8} = 1 - \frac{1V}{9V}$.

For the number of components, the first iteration has a Darlington pair of BJTs feeding into an OR gate which outputs to the positive terminal of the motor while its negative terminal is grounded. The second iteration scraps the Darlington pair in favour of one BJT acting as a kill switch at the negative terminal of the motor whose positive terminal is connected to VCC. The final iteration employs a voltage regulator connected to VCC that outputs a limited 7V to the motor whose negative terminal is connected to the NPN BJT AND gate. The total number of pertinent components of all the iterations comprise of five BJTs and one voltage regulator (first iteration: 2 x BJTs; second iteration: 1 x BJT; third iteration: 1 x voltage regulator, 2 x BJTs) yielding the following scores, respectively:

First iteration: $1 - \frac{2}{6}components = 0.\overline{6}$

Second iteration: $1 - \frac{1}{6}components = 0.8\overline{3}$

Third iteration: $1 - \frac{3}{6}components = 0.5$

However, even though the first and second iterations did satisfy our requirements, they could not reliably reproduce the same results whereas the third and final iteration could. As such, our WDM will add a compensating reliability factor to reflect this observation:

First iteration: $0.\overline{6} \cdot 50\% = 0.\overline{3}$

Second iteration: $0.8\overline{3} \cdot 50\% = 0.41\overline{6}$

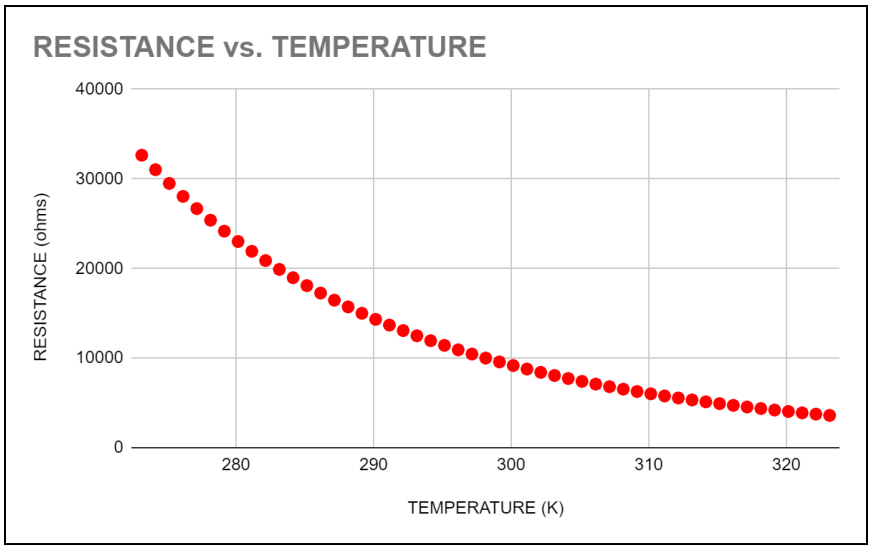
Third iteration: $0.5 \cdot 100\% = 0.5$

We then scaled these scores accordingly with respect to our WDM.

Lastly, as our control logic is separate from our temperature measurement circuit, the control logic WDM above is not applicable to temperature range.

Temperature Measurement

The research process of the NTC thermistor was initialized in assessing values displayed in the thermistor data sheet. From extracting information regarding the idealized behavior of the thermistor, we were able to understand expected behavior and use it as a guideline for our calibration.

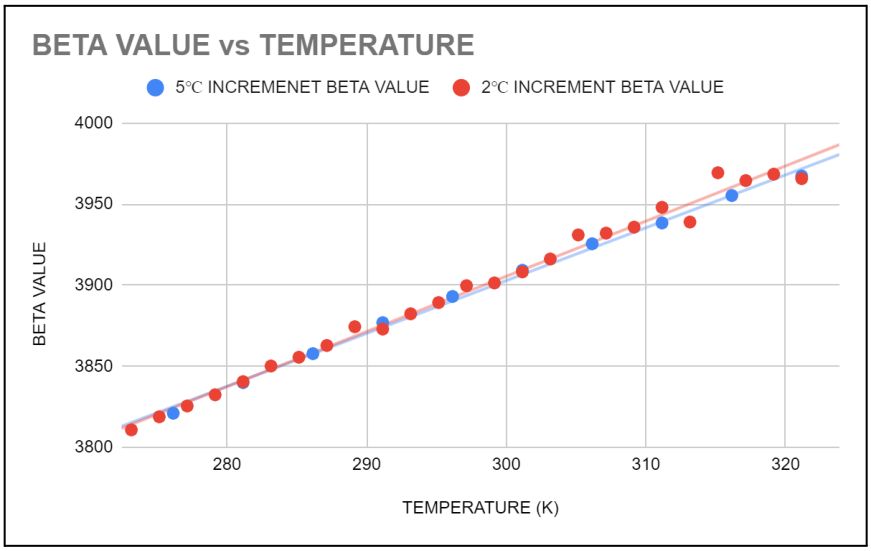


The beta value of the thermistor as provided by the manufacturer is defined for the range of 25-85℃. This value describes the behavior of the resistance in relation to temperature changes:

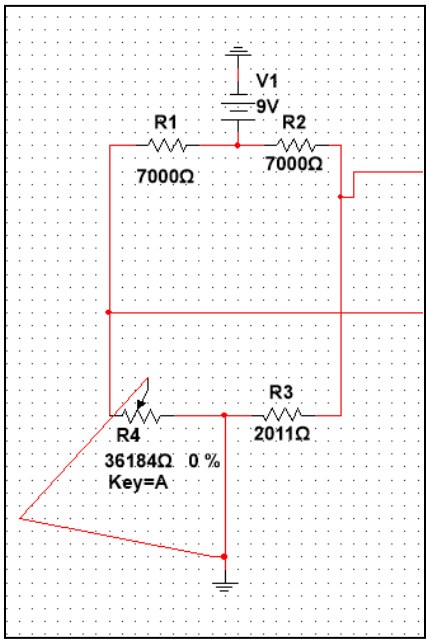
$$\beta = \frac{ln\left(\frac{R_{T1}}{R_{T2}}\right)}{\left(\frac{1}{T_1} - \frac{1}{T_2}\right)}$$

As beta is used for the approximation of resistance in a temperature range (T_1 to T_2), we realized that by reducing said range, the approximation would be more accurate for those temperatures. This meant that by subdividing our 0-50℃ range into smaller segments, and calculating the specific beta values for each segment, we could have a much more accurate representation of the resistance change across all temperatures, and thus a more accurate final reading.

Our initial testing involved subdividing the 25-85℃ range into 5℃ increments. However, in our experimentation, we noticed the thermistor's precision waned at certain temperature ranges. In an effort to combat this, we employed a more granular subdivision of 2℃ increments, and plotted the beta values against temperature for both 5℃ and 2℃ intervals. The graph confirmed our observations as we can see in certain temperature ranges (285-295 K and 305-320 K) there are sharper beta variations represented by the 2℃ points that were flattened out in the 5℃ increments

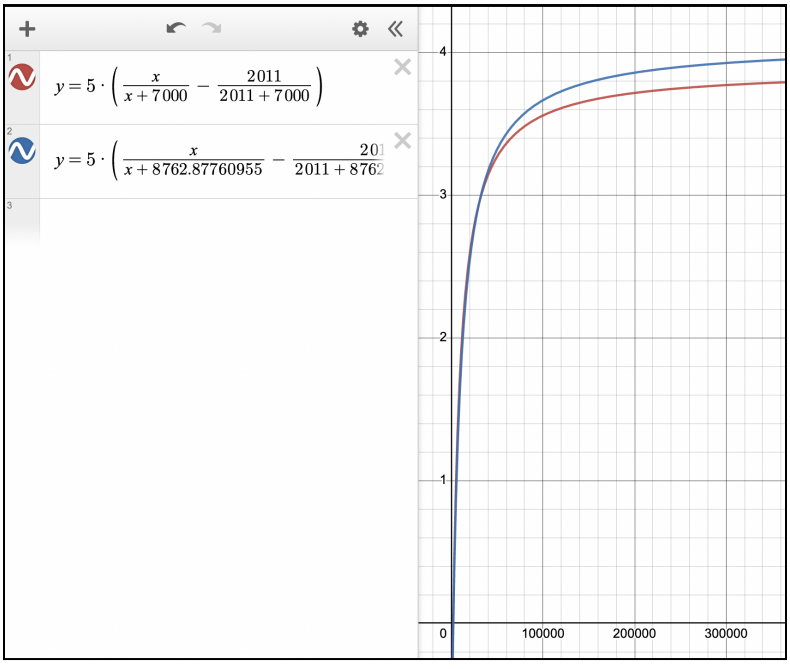

BETA VALUE vs TEMPERATURE
5℃ INCREMENET BETA VALUE
2℃ INCREMENT BETA VALUE
4000
3950
3900
3850
3800
BETA VALUE
280
290
300
310
320
TEMPERATURE (K)

Final Design and Calculations

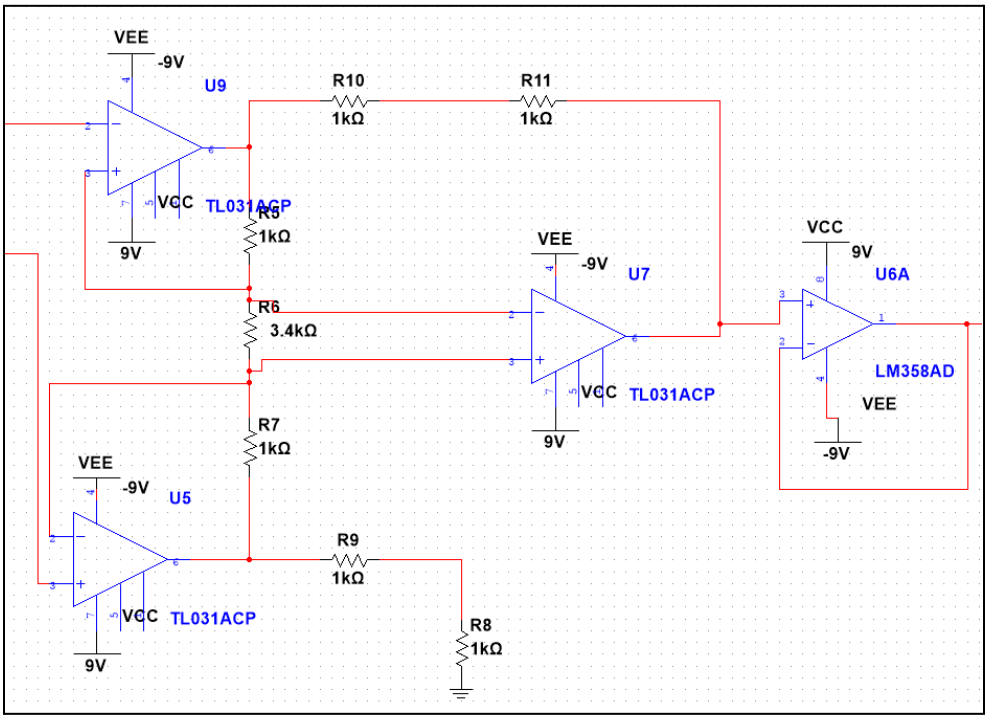


Wheatstone Bridge

We use a set of series resistors, R, in our Wheatstone bridge that were determined through analysis of the behaviors of the wheatstone bridge equation graph. The value 7000Ω seemed to work best to mitigate biasing as the geometric mean would bias higher resistances in a wider range of temperatures. The graph below demonstrates how the geometric mean (shown in blue) favours higher resistances in the range from -2°C to 66°C. Moreover, The offset resistor is the lowest resistance, being 2011Ω.



Graphical Representation of Wheatstone Bridge



Instrumentation Amplifier

The Wheatstone bridge output voltage falls in the range of 3.07365623976V - 0.0306649035382V from -2°C to 66°C respectively. The Arduino can only read a minimum of 4.9mV, which means a gain is required to meet the desired temperature range for this module. The following formula was used to calculate the gain of the instrumentation amplifier:

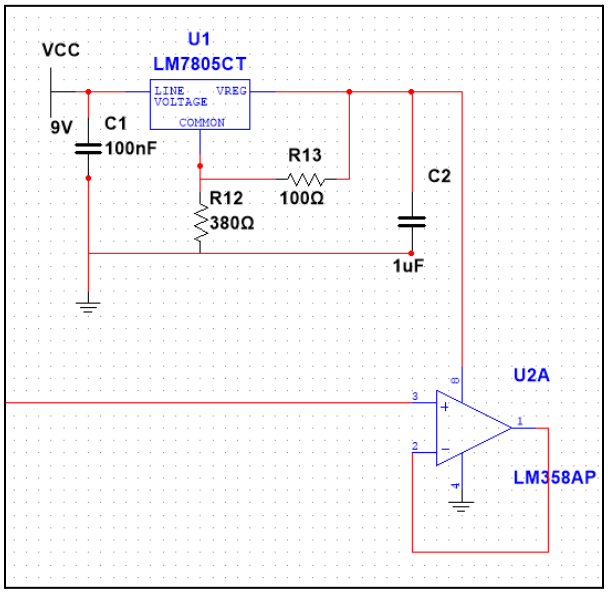
$$Let\ R_f = R = 1k\Omega$$

$$V_o = (1 + \frac{2R}{R_G}) \cdot (V_2 - V_1)$$

$$R_G = \frac{2 \cdot R}{\frac{V_o}{(V_2 - V_1)} - 1} = \frac{2 \cdot 1k\Omega}{\frac{4.83V}{3.04299133622V} - 1} = 3.4k\Omega$$

$$Gain = (1 + \frac{2 \cdot 1k\Omega}{3.4k\Omega}) = 1.58823529412$$

The Wheatstone bridge output voltage falls in the range of 4.88168932198V - 0.0487030820902V from -2°C to 66°C after receiving a gain of 1.588.



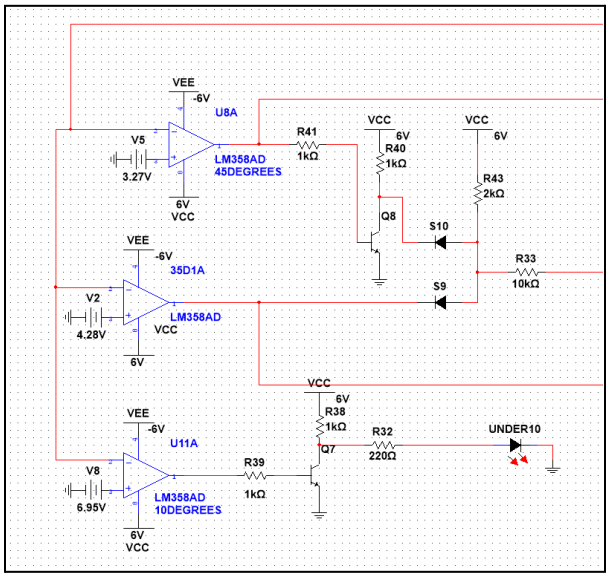
Arduino Input Saturator

The output of the instrumental amplifier is fed into the above Saturator for the purpose of circuit protection. To ensure that the output to the Arduino does not exceed either of the extremes of the 0-5V range, we must saturate the output voltage for both voltage extremes. By grounding -VCC, the lower limit of the output voltage is effectively grounded or at 0V. As determined earlier, 4.88V is enough to represent the temperature range from -2°C to 66°C. Therefore, we should saturate the op-amp to ensure the maximum voltage cap out at 4.9V. Due to a concept known as 'output voltage swing' +Vcc needs to be 1.4V higher than the desired output voltage of an op-amp.

OUTPUT						
V_O	Voltage output swing from rail	Positive rail (V+)	I_{OUT} = 50 µA	1.35	1.42	V
			I_{OUT} = 1 mA	1.4	1.48	V
			I_{OUT} = 5 mA(1)	1.5	1.61	V

Voltage Swing from LM358P data sheet

Thus to achieve an output voltage of 4.88V we must add the voltage output swing to determine the +Vcc, 4.88V + 1.4V = 6.18V. However, we used a voltage regulator to provide 6V and the output of the op-amp was around 4.88V which could represent a slight discrepancy in our usage of the op-amp.



Input and Digital Logic

The initial section of the circuit is designed to interpret and process the input voltage, which dictates the motor speed. This section incorporates three comparators, a custom-built AND gate with diodes, and an inverter.

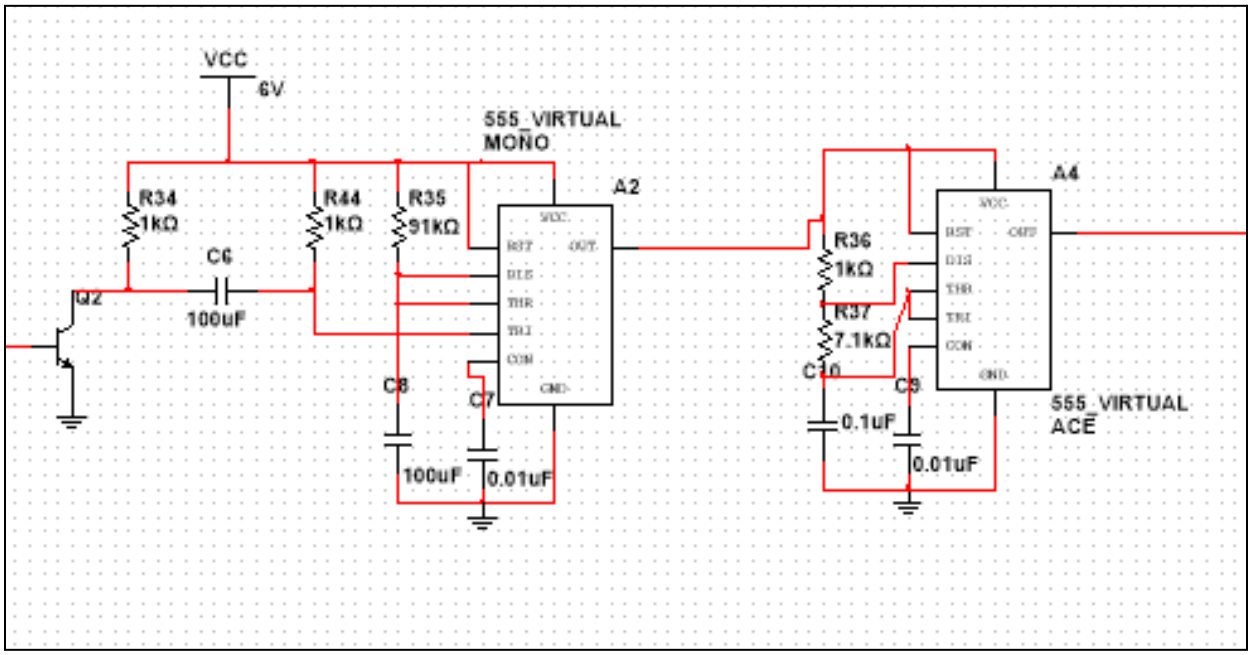
The input voltage is fed into the negative terminal of the three comparators. For the top comparator (O1), the positive terminal is connected to a voltage divider that sets a reference voltage of 3.27V, the middle (O2) is connected to a reference of 4.28V and the bottom (O3) is connected to 6.95V.

These comparators function to compare the input voltage with the set voltages associated with 45, 35, and 10 degrees (These voltages were determined experimentally by measuring the output of the Wheatstone/amplifier at those temperatures.). The power supply for these comparators is ±6V, ensuring an adequate range for input voltage levels, while still reducing power consumption from the initial design using 9V.

The output of O1 splits into two branches; one leads to an OR gate (discussed later), and the other is fed to an inverter circuit. This inverter consists of an NPN, a 1kΩ resistor, and a 6V source. The 1kΩ resistance value is sufficient so the NPN saturates. The inverter is used to reverse the logic level of the signal which is then directed into one of the inputs of an AND gate (made up of two diodes). The second input to the AND gate comes directly from one of the outputs of O2 with the other O2 output being fed to an AND gate (discussed later).

The output of O3 feeds to an NPN connected to a 6V source that acts as a switch for an LED. When the input is larger than 6.95V, the comparator turns the NPN on and the LED turns on.

The output of the diode AND gate then feeds into the next portion of the circuit.



RC Pulse Generator, Monostable, and Astable timers

The RC pulse generator is responsible for creating a short 'high-low-high' pulse to trigger the upcoming monostable 555 timer. This circuit keeps the voltage across the capacitor at a steady high voltage (6V), and when needed, a switch connects the capacitor to ground allowing it to rapidly discharge and dropping the voltage across the capacitor to less than 2V. When the switch is closed, the capacitor charges back up to 6V.

The NPN acts as a switch, when the output of the comparator is high (The AND gate output is high), the base current flows through a 1kΩ base resistor, turning the transistor on (saturated) and allowing a connection between the collector and ground. When the NPN is off, the collector is disconnected from ground and the capacitor is able to charge. The output of the capacitor is connected to the trigger pin of the 555 timer; the pin itself is sensitive to a falling edge (high-low), and thus when the output of the capacitor goes from 6V to <⅓ the supply voltage, the pin is triggered.

We can use the formula, $\tau = R \cdot C$, to determine the pulse. Our circuit worked with a pulse of $\tau = 1000 \cdot 100 \cdot 10^{-6} = 0.1s = 100ms$.

The first timer (on the left) is configured in monostable mode. This chip, once it receives the trigger pulse, is responsible for producing a high fixed output for the project requirement of 30 seconds. The time duration is determined via the equation (ReviseOmatic, 2023):

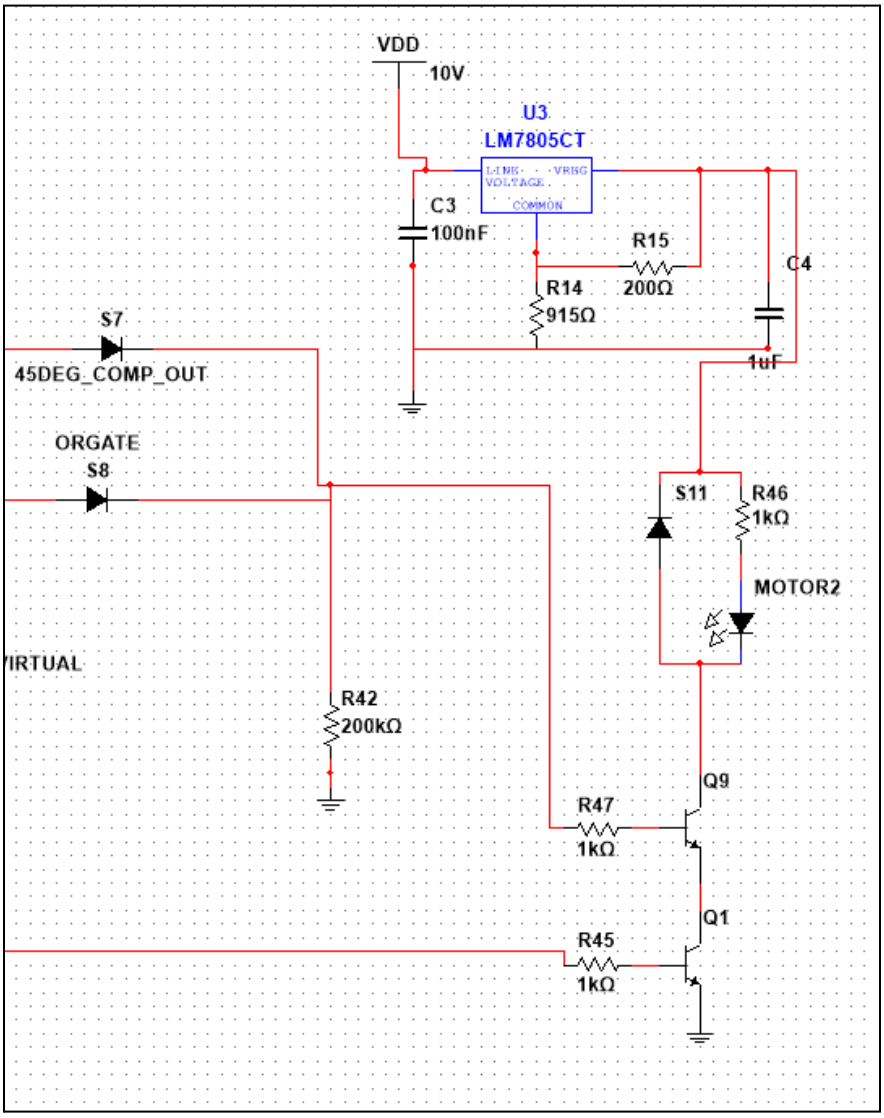
$$\begin{aligned} t &= 1.1 \cdot R \cdot C \\ &= 1.1 \cdot 91k\Omega \cdot 100\mu F \\ &= 10.01\ s \end{aligned}$$

Via experimentation, we concluded on the values of 273kOhm for the resistors and 100microFarads for the capacitor, which as seen in the equation above, yields us an output of 30 seconds.

This output feeds directly into the VCC of the second 555 timer, this time configured in astable mode. This chip is responsible for outputting a PWM signal with a 50% duty cycle, and as it is powered by the 30 second output of the monostable, it will only be able to do so for 30 seconds.
The calculations for the astable timer are as follows:

$$duty\ cycle\ = \frac{R_1+R_2}{R_1+2*R_2} = \frac{100+7.1k}{100+2*7.1k} = 50.35\%$$

$$frequency\ = \frac{1}{ln(2)*2*R_2*C_1} = \frac{1}{ln(2)*2*7k\Omega*0.1uF} = 1030.5\ Hz$$



Voltage Regulator to NPN BJT AND Gate-controlled DC Motor

We used a voltage regulator to limit our full speed motor at 7V. The voltage regulator also ensured that our motor can draw the power it needs while offering circuit isolation, since our motor ran inconsistently prior to adding the voltage regulator; this inconsistency can be attributed to a noisy VCC supplying power to our entire circuit. We increased the voltage into the voltage regulator to 10.5V to account for the drop in output voltage coming out of the voltage regulator. The resistor value between the output pin and adjustment pin is 2000Ω while the resistance between the adjustment pin and ground is 20400Ω (4 x 5100Ω), yielding the following voltage division:

$$9.1V = 10V * (\frac{20400\Omega}{20400\Omega + 2000\Omega})$$

Although 9.1V is the expected output of the voltage regulator, the actual voltage across the motor running at full speed was around 6.9V.

The NPN AND gate at the negative terminal of the motor is to guarantee that it would turn off between 10°C and 35°C. The top input of the AND gate is from the output of the OR gate between our top comparator (O1; comparing with a reference voltage of 3.27V corresponding to 45°C) and the output of the 555 timer-pair module; the bottom input of the AND gate is from the output of our middle comparator (O2; comparing with a reference voltage of 4.28V corresponding to 35°C). The logic is such that when the temperature is above 35°C, both BJTs would be on with the top BJT either always be on for full speed or on half the time for half speed; when the temperature is below 35°C, the bottom BJT would be off and the motor would be disconnected. The base resistors to our BJT AND gate are both at 1000Ω, and this is sufficient for BJT saturation.

Carbon Footprint

The electricity consumption of the smart thermostat must be calculated to determine the carbon footprint. Data from appendix B was referenced for the following calculations.

The electricity consumption without the motor running.

$$\Sigma P = \Sigma(I \cdot V) = 10.6 \cdot 0.00047 + 8.5 \cdot 0.0373 + 9.1 \cdot 0.0136 + 5.9 \cdot 0.00792 = 0.49252\,W$$

$$0.49252\,W \cdot \frac{21\,hours \cdot 365\,days}{1000\,kW} = 3.7751658\,kWh$$

The electricity consumption with the motor running at half speed.

$$\Sigma P = \Sigma(I \cdot V) = 10.6 \cdot 0.0616 + 8.5 \cdot 0.0408 + 9.1 \cdot 0.0092 + 5.9 \cdot 0.00451 = 1.110089\,W$$

$$1.110089\,W \cdot \frac{2\,hours \cdot 365\,days}{1000\,kW} = 0.81036497\,kWh$$

The electricity consumption with the motor running at full speed.

$$\Sigma P = \Sigma(I \cdot V) = 10.6 \cdot 0.1331 + 8.5 \cdot 0.03943 + 9.1 \cdot 0.010 + 5.9 \cdot 0.00593 = 1.872002\,W$$

$$1.872002\,W \cdot \frac{1\,hour \cdot 365\,days}{1000\,kW} = 0.68328073\,kWh$$

$$\Sigma Electricity\ Consumption = 0.68328073kWh + 0.81036497kWh + 3.7751658\,kWh = 5.2688115\,kWh$$

The total yearly electricity consumption of the smart thermostat is 5.2688115 kWh yearly. To determine the carbon footprint the device's electricity consumption is multiplied by the carbon intensity in Vancouver, 11.5 tCO2e/GWh (Motavas, 2023).

$$5.2688115\,kWh \cdot 11.5\,tCO2e/GWh = 5.2688115\,kWh \cdot 11.5\,(10^6\,gCO2e)/(10^6\,kWh)$$
$$5.2688115\,kWh * 11.5\,gCO2e/kWh = 60.59133225\,gCO2e$$

The total carbon footprint of the smart thermostat is 60.59133225 gCO2e/year.

Cost

The total manufacturing cost will be $93.65, as shown in appendix A. The cost of operation will be the electricity consumption multiplied by the rate of electricity in Vancouver.

$$5.2688115\,kWh \cdot 0.0959\,CAD/kWh = \$0.50527902285$$

In total the cost of manufacturing and operation will be $94.1552790228 CAD.

Maximum Range

The maximum range of temperature that could be measured in our final design with 0.1°C precision is from -2°C to 66°C. The following formulas were used to achieve a range of 68°C.

$$R = 7000\Omega$$

This value was experimentally determined as the geometric mean did not prove to be accurate. Moreover, the gain was determined experimentally.

$$V_{out} = Gain \cdot 5 \cdot \left(\frac{R_s}{R_s+R} - \frac{R_{offset}}{R_{offset}+R}\right) = 1.59 \cdot 5 \cdot \left(\frac{R_s}{R_s+7000} - \frac{2011}{2011+7000}\right)$$

This range can be confirmed by calculating the voltages at the extrema.

$$\Delta V = 1.59 \cdot 5 \cdot \left\{\left(\frac{2082}{2082+7000} - \frac{2011}{2011+7000}\right) - \left(\frac{2011}{2011+7000} - \frac{2011}{2011+7000}\right)\right\} = 0.049V = 49mV$$

$$\Delta V = 1.59 \cdot 5 \cdot \left\{\left(\frac{36184}{36184+7000} - \frac{2011}{2011+7000}\right) - \left(\frac{34366}{34366+7000} - \frac{2011}{2011+7000}\right)\right\} = 0.057V = 57mV$$

Between 66°C to 65°C there is a difference of 49mV and between -2°C to -1°C there is a difference of 57mV. These values are sufficient to achieve an accuracy of 0.1°C.

Challenges and Mitigations

Challenges	Mitigations
Circuit protection	Initially, our Arduino would be exposed to voltages higher than 5V, which could lead to damage and unreliability. To combat this, we incorporated a voltage regulator that output a maximum of 6V into an op-amp that would then feed into the arduino. This meant that the voltage into the Arduino would never be negative (the -VCC of the op-amp was grounded) and would never be over ~4.6V.
Connection between Monostable and Astable Timers	The motor was drawing a significant amount of voltage from the rail that powered the two timers. To address this issue, we employed a separate power source and a voltage regulator to independently power the motor, separate from the control circuit voltages. The control functions as a switch to turn the motor on or off.
Breadboard Power Rails	Using the breadboard power rails as Vcc for the motor and circuit protection op-amp proved difficult as the voltage of the rails would fluctuate with changes to the circuit. To counteract this behavior two voltage regulators were added to the motor and circuit protection op-amp to ensure they operate as intended. The motor's voltage regulator received 10V and was regulated to exactly 7V and the saturation op-amp was given a 9V input to be regulated to 6V to ensure the output of the op-amp does not exceed 4.89V.

Theory vs. Practice

Current Going into the Motor

It was difficult to accurately simulate motors in simulations because the current often did not match what was shown in the simulation. The motor requires a significant amount of current, and the circuit must be designed to accommodate this. Frequently, the transistors were unable to meet the motor's current requirement. As a result, we designed our circuit in such a way that the NPN transistors ground the motor to make it run while connected to Vcc. The motor will be disconnected from ground when it needs to turn off.

Implementation of Difference Amplifier

Using a difference amplifier to ensure circuit protection was thoroughly developed as it, in theory, should not ever allow voltages above the Arduino's threshold (5V) to be fed into its GPIO pins. However, due to the nature of the initial conditions of operation amplifiers varying with model, replicating a working model of a difference amplifier in practice tended to lead to undesirable outputs. These outputs would be undesirable due to the.

This is most likely due to the fact that difference amplifiers are built for the sole purpose of amplifying a voltage difference. With this in mind overvoltage conditions are inherently the main function of this amplifier and would not protect noisy spikes in voltage or current. This was prevalent in experimenting with the output of our difference amplifier as rapid spikes in voltage were displayed on our multimeter and would still reach the Arduino if connected.

Wheatstone Bridge Ratios

The geometric mean of the maximum and minimum resistance values is not an ideal way to rectify the output of the wheatstone bridge to fit in a range greater than 50°C. With the current range of 68°C, a resistance value of 7000Ω was used, while the geometric mean would be 8530Ω ($\sqrt{2011 \cdot 36184}$). It has been experimentally determined that lower values of R, will bias low resistance values, whereas higher values of R will bias high resistance values. This nature of the value of R makes it difficult to balance a large range of temperatures within 5V. Thus, to encapsulate a range larger than 50°C within 4.89V the value of R must be experimentally determined as mentioned earlier in the report.

Reflection

The final demo circuit was unable to match the accuracy of a household thermometer to the tenth of a degree, which is a critical design requirement. Focussing on fixing this facet of our design would substantially improve other aspects of our design as well. For example, ensuring we have an accurate temperature reading in the range of 0-50°C would allow for more stability in our control logic circuit with respect to precise temperature cutoffs. Accurately reading this temperature range would allow the behavior of our circuit to correlate more with the resistance vs. temperature relation of our thermistor. This will inherently allow for an extended temperature range as the behavior of our thermistor should be better mapped to temperature extrema.

Accurate temperature values would ensure that control logic signals would turn the LED or motor on at precise temperatures. An improvement that would ensure a better degree of accuracy for the thermometer circuit would begin with using a higher quality thermistor. With a thermistor that has resistance values rated for measuring a tenth of a degree centigrade or has a more linear relationship with temperature. On the topic of using higher-quality components, the power sources could be better regulated through possibly using one uniform power source. As our current design utilizes several double-A batteries and 9V batteries, the possibility for cells being disbalanced in respect to individual power usage is high. Moreover, a uniform power source would limit noisy signals in control logic.

Applications

A temperature sensing circuit working in tandem with a control logic circuit has several applications other than a household thermostat. An application of this device could be used in automating greenhouse operations. If this circuit could utilize other environmental variables such as moisture, light, or humidity, it would be incredibly useful to automate correcting actions for suboptimal conditions. Modifications for this circuit include adding sensors that can detect additional environmental variables (i.e., photoresistor, hygrometers, moisture sensor), add control logic that detects

undesirable states for the environment, and finally use appropriate countermeasures such as triggering artificial lighting, utilizing humidifiers, watering plants autonomously, and controlling the temperature.

Citations

S. Motavas, “Sustainability Discussion - 2,” in ELEC 291/ELEC 292/CPEN 291 941 2023S1, Jun. 25, 2023.
(Motavas, 2023)

"Capacitors - RC Timing," ReviseOmatic, Available:
https://reviseomatic.org/help/s-capacitors/Capacitors%20-%20RC%20Timing.php. Accessed: June 25, 2023.
(ReviseOmatic, 2023)

B. Trump, “Op Amp Voltage Ranges - input and output, clearing some confusion,” The Signal - Archives - TI E2E support forums,
https://e2e.ti.com/blogs_/archives/b/thesignal/posts/op-amp-voltage-ranges-input-and-output-clearing-some-confusion (accessed Jun. 25, 2023).
(Trump, 2012)

Microchip Technology, Inc., “Output Voltage Swing,” Output Voltage Swing - Developer Help,
https://microchipdeveloper.com/asp0107:output-voltage-swing (accessed Jun. 25, 2023).
(Microchip Technology, Inc., 2021)

Texas Instruments, “LM317 3-Terminal Adjustable Regulator, PDF.” Texas Instruments Incorporated, Dallas, Texas, Apr. 2020.
(Texas Instruments, 2020)

Appendix A: List of Components

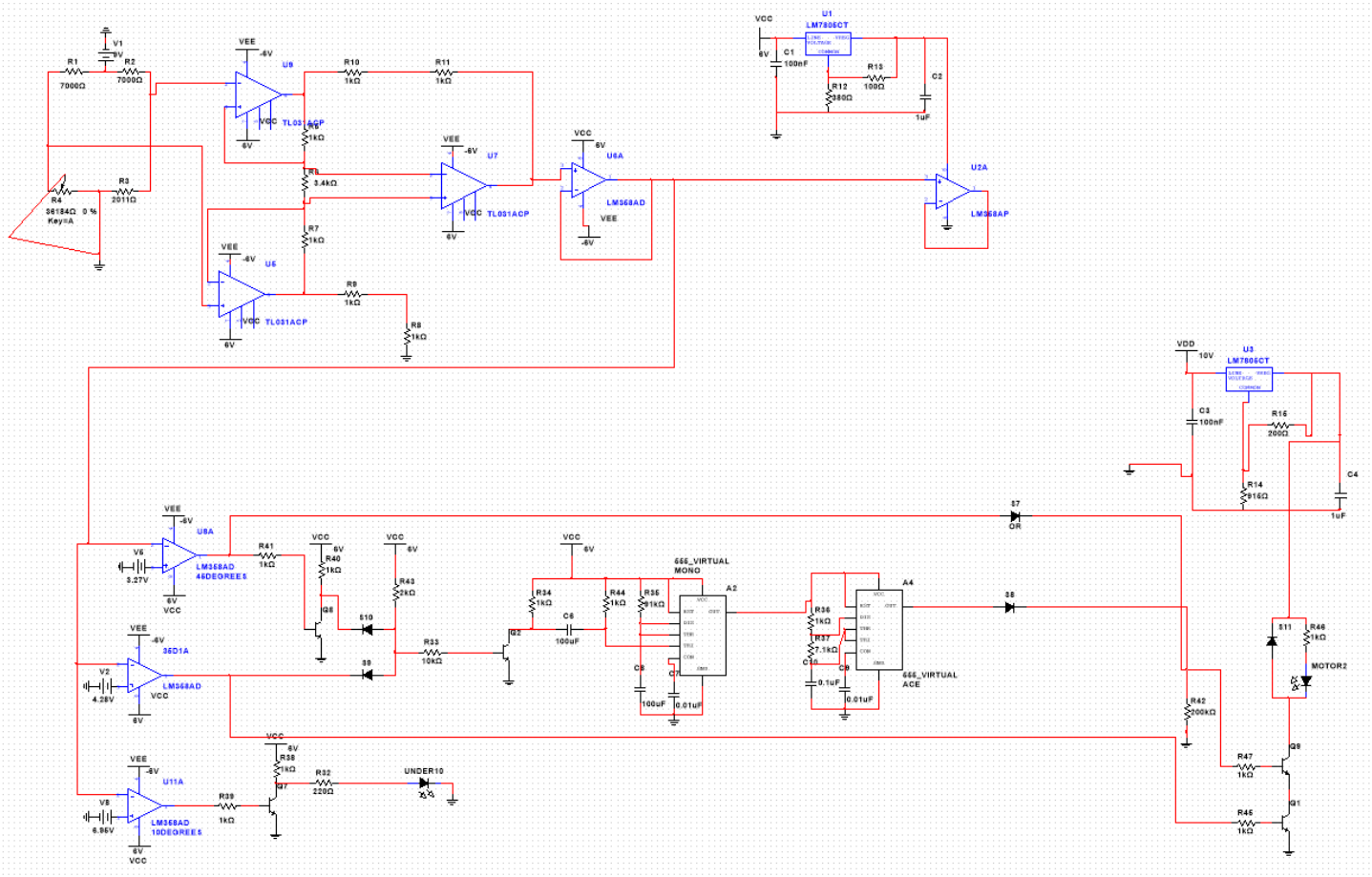
Item	Quantity	Cost
Resistors	77	0.00729
Large Breadboards	5	5.95
Mini Breadboard	1	3.68
NTC Thermistor	1	0.02885
LM317T Voltage Regulator	2	1.9
DC toy motor	1	2.80000
Ceramic Capacitors	5	0.00191
Electrolytic Capacitors	2	0.09781
Diodes	5	0.01039
NPN BJTs	6	0.66018
LED	1	0.13401
LM358P	1	0.17228
TL081ACP	9	1.18886
555 Timers	2	0.47632
Arduino UNO	1	37.83000
LCD	1	7.61886
Potentiometer	1	0.98040
1.5V AA Batteries	9	3.48714
9V Batteries	2	3.8485
Total in USD (per unit)	—	**70.8728**
Total in CAD (per unit)	—	**93.65**

***All prices were sourced from Digikey**

Appendix B: Power Data

	Source 1	Source 2	Source 3	Source 4
Voltage (V)	10.6	8.5	9.1	5.9
Current with no motor (mA)	0.47	37.3	13.6	7.92
Current at half speed (mA)	61.6	40.8	9.20	4.51
Current at full speed (mA)	133.1	39.43	10	5.93

Appendix C: Complete Circuit Diagram



Appendix D: Arduino Code

```
#include <LiquidCrystal.h>
#include <math.h>
#include <SPI.h>
#include <SD.h>

const int A = 3;

const int chipSelect = 10; // SD card chip select pin
volatile bool loggingEnabled = false;
volatile bool loggingStateChanged = false;

File beegData;


//LCD Setup
const int RS = 8, EN = 9, D4 = 4, D5 = 5, D6 = 6, D7 = 7;
LiquidCrystal lcd (RS, EN, D4, D5, D6, D7);



const int numReadings = 10;

int readings[numReadings];  // the readings from the analog input
int readIndex = 0;          // the index of the current reading
int total = 0;              // the running total
int average = 0;            // the average

int inputPin = A0;

//Struct to store data associated a temperature range
struct TempRange {
  int avgTemp;
  int lowTempResist;
  int highTempResist;
  float beta;
};


//Array containing multiple TempRange's from 0-50 degrees Celcius (in 2 degree increments)
const TempRange TempArray[] = {
  {271.65, 36184, 34366, 3804.003966},
  {273.65,32651, 31031, 3810.758964},
  {275.65,29500, 28054, 3818.816104},
  {277.65,26687, 25395, 3825.497423},
  {279.65,24172, 23016, 3832.405191},
  {281.65,21921, 20885, 3840.5026},
  {283.65,19903, 18973, 3850.157206},
  {285.65,18092, 17257, 3855.558174},
  {287.65,16465, 15714, 3862.813152},
  {289.65,15001, 14324, 3874.392256},
  {291.65,13682, 13073, 3872.933495},
  {293.65,12493, 11943, 3882.349899},
  {295.65,11420, 10923, 3889.29664},
  {297.65,10450, 10000, 3899.687958},
  {299.65,9572, 9165, 3901.395887},
  {301.65,8777, 8408, 3908.220419},
  {303.65,8056, 7721, 3916.165633},
```

```
  {305.65,7402, 7097, 3931.004088},
  {307.65,6807, 6530, 3932.120001},
  {309.65,6266, 6014, 3935.810502},
  {311.65,5774, 5544, 3948.035786},
  {313.65,5325, 5116, 3938.96101},
  {315.65,4916, 4724, 3969.374947},
  {317.65,4542, 4367, 3964.528968},
  {319.65,4200, 4040, 3968.493857},
  {321.65,3887, 3741, 3960.87622},
  {323.65,3601, 3467, 3972.285734},
  {325.65,3339, 3216, 3980.288315},
  {327.65,3098, 2985, 3988.962183},
  {329.65,2877, 2773, 4001.005097},
  {331.65,2674, 2579, 3978.804224},
  {333.65,2487, 2399, 4010.397807},
  {335.65,2315, 2234, 4012.521035},
  {337.65,2157, 2082, 4034.648652},
  {339.65,2011, 1942, 4027.714529},
  {341.65,1876, 1813, 3987.181779},
  {343.65,1752, 1693, 4045.451738},
  {345.65,1637, 1582, 4083.064017},
  {347.65,1530, 1480, 4015.660826},
  {349.65,1432, 1385, 4079.880913},
  {351.65,1341, 1298, 4030.118957},
  {353.65,1256, 1216, 4047.863749},
  {355.65,1178, 1141, 4036.570995},
  {357.65,1105, 1070, 4117.104401},
  {359.65,1037, 1005, 4054.329869},

};


//Calculating size of array
const int arraySize = sizeof(TempArray) / sizeof(TempArray[0]);

//Binary search algorithm to determine the correct temp range from a calculated resistance
int findRangeIndex (int resistance) {
  int low = 0;
  int high = arraySize - 1;
  int mid;

  if (resistance > 31031) {
     return 0;
  } else if (resistance < 3601) {
     return 9;
  }

  while (low <= high) {
    mid = (low + high) / 2;

    if (resistance <= TempArray[mid].lowTempResist && resistance >= TempArray[mid].highTempResist) {
      return mid;
    }

    else if (resistance < TempArray[mid].highTempResist) {
      low = mid + 1;
    }

    else {
      high = mid - 1;
    }
  }
```

```
  //Fallback linear search in edge case where binary search does not return a proper index
  for (int i = 1; i < arraySize; i++) {
    if (resistance >= TempArray[i].lowTempResist && resistance < TempArray[i - 1].highTempResist) {
      return i;
    }
  }

  return -1;
}


//Function that calculates the thermistor resistance from the analog input
float getResistance(float thermValue) {
  float voltage = thermValue * (4.8/1023.0) + 0.06;

  float R1 = (-7000*voltage-6188.91361913)/(voltage-6.15812300411);
  return R1;
}

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  Serial.begin(9600);
  pinMode(A0, INPUT);
  pinMode(A, INPUT);

  attachInterrupt(digitalPinToInterrupt(A), toggle, CHANGE); //interrupt for enable
  SD.begin(chipSelect);

  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }
}

void loop() {

  total = total - readings[readIndex];
  // read from the sensor:
  readings[readIndex] = analogRead(inputPin);
  // add the reading to the total:
  total = total + readings[readIndex];
  // advance to the next position in the array:
  readIndex = readIndex + 1;

  // if we're at the end of the array...
  if (readIndex >= numReadings) {

    // ...wrap around to the beginning:
    readIndex = 0;
  }

  // calculate the average:
  average = total / numReadings;
  // send it to the computer as ASCII digits
  // Serial.println(average);
  delay(1);  // delay in between reads for stability

  float resistance = getResistance(analogRead(inputPin));
  int tempIndex = findRangeIndex(resistance);
```

```
  //Error handler
  if (tempIndex == -1) {
    Serial.println("ERROR: NO VIABLE RANGE FOUND");
    return;
  }

  float foundBeta = TempArray[tempIndex].beta;
  float refResist = (TempArray[tempIndex].lowTempResist + TempArray[tempIndex].highTempResist) / 2;
  float refTempK = TempArray[tempIndex].avgTemp;

  //Temperature calculation with found beta/temp/and resistance
  float temp2;
  if (resistance > 0 && (foundBeta - (refTempK * log(abs(refResist/resistance)))) != 0) {
    temp2 = (foundBeta * refTempK)/(foundBeta - (refTempK * log(abs(refResist/resistance))));
    temp2 -= 273.15;

   } else {
    temp2 = -273.15; // Default error value
  }

  if (loggingStateChanged) { //pretty much edge detection
    loggingStateChanged = false; //reset state so we dont infinetly go thru this statement

    if (loggingEnabled) { //just print this once please
      Serial.println("Logging enabled");
      beegData.close();
      beegData = SD.open("beegdata.txt", FILE_WRITE);
    } else if(!loggingEnabled){
      Serial.println("Logging stopped");
      beegData.close();
      beegData = SD.open("beegdata.txt", FILE_READ);
    }
  }

  if (loggingEnabled) { //print this as long as logging is enabled
    beegData.println(temp2);
    Serial.println("ENABLED");
    delay(100); // Delay for 100 ms between data points (10 data points per second),
  }

  //Update the LCD
  lcd.print(temp2, 1);
  lcd.print(" C");
  lcd.setCursor(0, 1);
  lcd.print(analogRead(inputPin));

  delay(10);
  lcd.clear();
}

void toggle() {
  static int lastDebounceTime = 0;
  int debounceDelay = 1000; // 1 sec debounce delay, this is the safest amt before the response time goes bonkers

  if ((millis() - lastDebounceTime) > debounceDelay) {// we ensure that logging state is stablized by checking current time
    loggingEnabled = !loggingEnabled;
    loggingStateChanged = true;
  }

  lastDebounceTime = millis(); //this keeps track of the last time the interrupt was called between function calls
}
```