Prototype System Synopsis

The Helios Warning System is an early heat stroke detection device that is composed of a pulse sensor, temperature sensor, humidity/temperature sensor, and liquid crystal display (LCD), all integrated into a glove. The pulse and temperature sensors monitor the user's internal conditions, their heart rate and their skin temperature, and the humidity sensor monitors the user's ambient conditions, the humidity and temperature of the environment. The LCD was used to display the collected data as well as heath reminders and warning alerts to the user. An Arduino Nano Every microcontroller was used to intake and process the collected data, and, using logic, determine if the heat stroke warning should be displayed on the LCD.

Prototype System Schematic Diagram

Figure 1:Prototype sensor location diagram of the Helios Warning System

Table 1: Component information

Component	Part Number	Manufacturer	Cost (USD)
Pulse Sensor	SEN-11574	World Famous Electronics	$24.95
Thermistor (10kΩ)	MF52A2103J3470	Cantherm	$0.51
10kΩ resistor	MRS25000C1002FCT00	Vishay Beyschlag	$0.22
Humidity Sensor	DHT22	Adafruit Industries	$9.95
LCD Screen with I2C	8541582407	Amazon	$11.43
Arduino Nano Every	ABX00033	Arduino	$13.39
Solderable Breadboard	7100-45	Twin Industries	$4.50
Battery Pack	BH48AASF	Memory Protection Device	$4.35
Suede Palm Gloves	N/A	Kodiak	$4.99

Figure 1 above displays the functional prototype integrated into a glove. The components, part numbers, manufactures and associated costs are displayed in Table 1 above. The detailed prototype system schematic can be found in Appendix A.

Circuit Diagram

The Circuit Diagram for the Helios Warning System in shown in Figure 2.

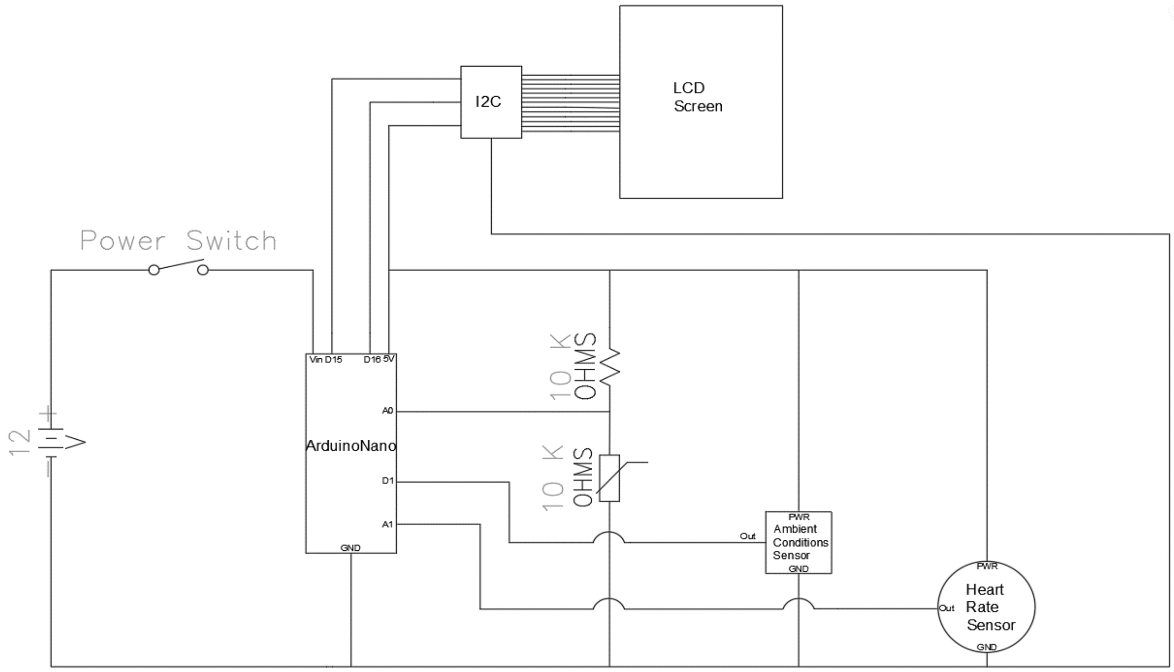

Figure 2: Circuit diagram of the Helios Warning System

Output Realization Diagram

The Helios Warning System integrates both physical measurement and digital calculation to function. The heart rate sensor and the ambient temperature and humidity sensor are integrated circuit components whose output requires custom software libraries to interpret their output signal. To interpret the thermistor's output signal, a simple voltage division calculation followed by the application of the Steinhart-Hart equation allows for skin temperature calculation. The heart rate, skin temperature, ambient temperature, and ambient humidity values are compared in order to determine which output the device should produce. The Output Realization Diagram, Steinhart-Hart equation, and Arduino code can be found in Appendices A, B, and C, respectively.

Test Results and Analysis

Testing the device was of utmost importance to validate its functionality. Multiple tests were conducted to ensure that the different sensors worked in different situations, as well as the signal processing and device logic. For reference, the average resting heart rate for adults is between 60-100 beats per minute (BPM) [1], and the average skin temperature is 34^{o}C [2]. The risk of heat stroke increases when a person's heart rate is elevated above 125 BPM, and/or when their core body temperature is elevated above 40^{o}C.

The test scenarios, target sensors, and results for each test are summarized below in Table 2.

Table 2: Test results

Test Scenario	Features Tested	Results
User climbed 6 flights of stairs to induce elevated heart rate and skin temperature	• Heart Rate Sensor • Skin Temperature Sensor	• Users heart rate rose from 71 BPM to 140 BPM. • Users skin temperatures rose from 34℃ to 38℃. • Warning was successfully issued.
User entered the gym sauna	• Humidity Sensor • Environmental Temperature Sensor	• Humidity sensor was initially read 43.8% and rose to 95.8% after entering the sauna. • Environmental temperature was initially 25.3℃ and reached 72.6℃. • Warning was successfully issued.
Heated space to assess the ability to detect environmental temperature change	• Environmental Temperature Sensor	• Environmental temperature changed from 25.3℃ to 28.9℃ • Correctly, no warnings issued
System was dropped from a height of one meter onto a desk	• Durability and Strength	• Soldered circuit component broke, had to be re-soldered. • Batteries popped out of the holder. • Undesired results, improvements in device durability need to be made.

Contributions of Each Group Member

The contributions of each group member for each assignment are summarized in Table 3 below.

Table 3: Summary of each member's contributions

Group Member	**Assignment**			
	Component 1	**Component 2**	**Presentation**	**Component 3**
Matthew Bunce	Expected outcome	Design alternatives	Marketing Objective	Test results and analysis, prototype system schematic
Audra Nicholson	Hypothesis, objective and specific aims	Primary design problem analysis, design alternatives	Detailed description of prototype, sensors, test video	Prototype system schematic diagram, test results and analysis

Lienne Poon	Expected outcome	Primary design summary	Proposed testing, expected outcomes, potential improvement	Prototype system synopsis, test results and analysis
Brendan Wood	Background	Primary design problem analysis	Technical functionality	Circuit diagram, output realization diagram

M Bunce

Matthew Bunce — Dec 1, 2021 (Date)

A. Nicholson

Audra Nicholson — Dec 1, 2021 (Date)

Lienne Poon — Dec 1, 2021 (Date)

Brendan Wood — Dec 1, 2021 (Date)

References

[1] E. R. Laskowski, "What's a normal resting heart rate?," Mayo Clinic, 2 October 2020. [Online]. Available: https://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979. [Accessed 22 November 2021].

[2] H.-Y. Chen, A. Chen and C. Chen, "Investigation of the Impact of Infrared Sensors on Core Body Temperature Monitoring by Comparing Measeurment Sites," *Sensors*, vol. 20, no. 10, p. 2885, 2020.

[3] Cantherm, "rt=characteristics," Cantherm, 2014. [Online]. Available: https://www.cantherm.com/rt-characteristics/. [Accessed 29 November 2021].

Appendix A

Prototype System Schematic

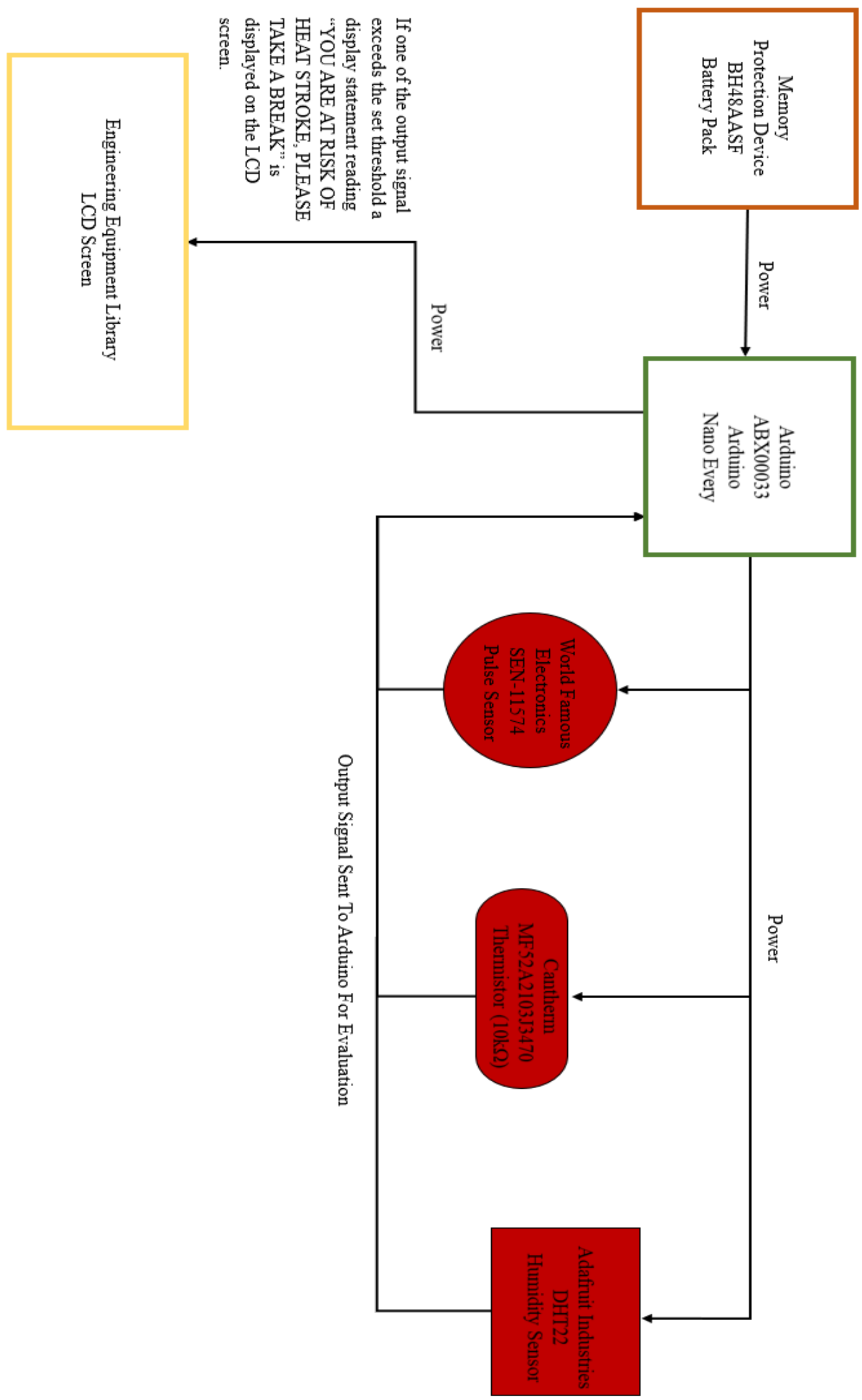

Figure 3: Prototype systemic schematic

Output Realization Diagram

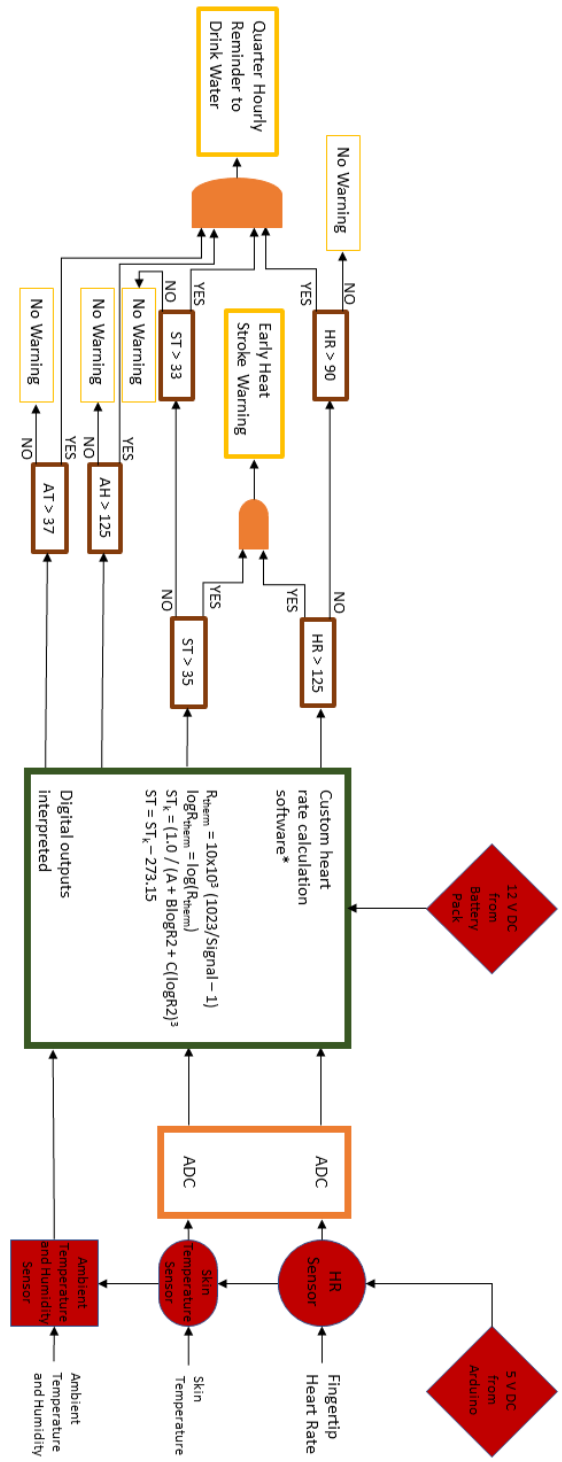

*Figure 4: Output realization diagram. Items outlined in yellow illustrate outputs to the LCD screen, items outlined in brown represent logic used for output determination, items outlined in dark green represent calculations made on the Arduino Nano Microprocessor, items outlined in orange represent physical processes carried out on the Arduino Nano, and items filled in red represent physical sensors and power supplies. *See the below paragraph for heart rate calculation description.*

The ambient temperature and humidity sensor outputs a digital signal which requires a custom library to interpret the signal, which means there is no math or calculation required to determine the signal value. The heart rate sensor uses complex math and logic that is too in depth to include in the Output Realization Diagram. To calculate heart rate, the analog signal is first read from the output lead for the heart rate device. This signal is first compared to a threshold value which varies depending on sensor's implementation, for the Helios Warning System, this value was determined to be 650 bits. If the current signal value, when converted to a 10-bit integer, is greater than the threshold, the code continues. If the time between signal readings is greater than 200 ms, the code continues. Next, the time between the previous reading and current reading is calculated, this is known as the Inter-Beat Interval (IBI). The IBI is then saved to a global array variable, removing the oldest IBI value and saving the current IBI to the most recent value. Finally, 60,000 ms is divided by the 10 most recent IBI values to calculate the number of beats per minute.

Appendix B

Steinhart-Hart Equation

$$\frac{1}{T} = A + Bln(R) + C(\ln(R))^3 \tag{1.0}$$

Equation 1.0 is the Steinhart-Hart equation. T represents the temperature the thermistor is sensing, R is the current resistance of the thermistor, and A, B, and C are constants for a given model of thermistor. Constants A, B, and C were found by solving a linear system of equations using different temperatures and the corresponding recorded resistances provided by the manufacturer [3].

Appendix C

Arduino Nano Code

```
//Libraries
#include <DHT.h>
#include<LiquidCrystal_I2C.h>

//Constants
#define DHTPIN 7     // what pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor for normal 16mhz Arduino



//Variables for humidity sensor
int chk;
float hum;  //Stores humidity value
float temp; //Stores temperature value
int ThermistorPin = 0;

//Variables for thermistor
int Vo;
float R1 = 10000;
float logR2, R2, Te;
float c1 = 0.000385937250220100, c2 = 0.000330024264854518, c3 = -9.26857851202681e-08;
double temperature;

//variables for BPM
#define pulsePin A2
int rate[10];
unsigned long sampleCounter = 0;
unsigned long lastBeatTime = 0;
unsigned long lastTime = 0, N;
int BPM = 0;
int IBI = 0;
int P = 512;
```

```
int T = 512;
int thresh = 650;
int amp = 100;
int Signal;
boolean Pulse = false;
boolean firstBeat = true;
boolean secondBeat = true;
boolean QS = false;

//detection variables
int result;

//define LCD
LiquidCrystal_I2C lcd(0x27,20,4);

void setup()
{
  dht.begin();  //initalize the humidity sensor
  lcd.init();     // initialize the lcd
  delay(20); //apparanetly this is best practice

}

void loop()
{
   //humidity sensor
   //Read data and store it to variables hum and temp
   hum = dht.readHumidity();
   temp = dht.readTemperature();
   //Print temp and humidity values to lcd screen

   //thermistor
   Vo = analogRead(ThermistorPin);  //read voltage
   temperature = calculateTemp(Vo); //calculate actual temperature

   //heart rate
```

```
  if (QS == true)
  {
    QS = false;
    }
    else if (millis() >= (lastTime + 20)) {
    readPulse();
    lastTime = millis();
  }

  //detectHS
  result = detectHS(temperature, BPM, temp, hum);
  if(result == 0)
  {
   displayWarning();
  }
  else if(result == 1)
  {
   displayReminder();
  }
  else
  {
   lcd.clear();
   lcd.noBacklight();
  }

  delay(100); //Delay 100ms

}

double calculateTemp(int Vo)
{

 R2 = R1 * ((1023.0 /(float) Vo) - 1.0);
 logR2 = log(R2);
 Te = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2));
 Te = Te - 273.15;
```

```
  return Te;

}

int detectHS(double temperature, int BPM, float temp, float hum)
{
  if((BPM >= 125) && (temperature > 35))
  {
    return 0;
  }
  else if ((BPM >= 90) && (temperature <= 33) && (temp >= 37) && (hum >= 70)
  {
    return 1;
  }
  else
  {
    return 2;
  }

}
void displayWarning()
{
  lcd.backlight();
  lcd.setCursor(0,0);
  for(int i = 0; i <= 10; i++)
  {
    lcd.print("WARNING: AT RISK FOR HEAT STROKE");
    delay(500);
  }
}

void displayReminder()
{
  lcd.backlight();
  lcd.setCursor(0,0);
  if (millis() % 900000 == 0)
```

```
 {
   lcd.print("DRINK WATER");
   delay(10000);
 }
 lcd.noBacklight();
 lcd.clear();

}

void readPulse() {

 Signal = analogRead(pulsePin);
 sampleCounter += 20;                    // keep track of the time in mS
 int N = sampleCounter - lastBeatTime;          // monitor the time since the last beat to avoid noise

 detectSetHighLow();

 if (N > 200) {                          // avoid high frequency noise
   if ( (Signal > thresh) && (Pulse == false) && (N > (IBI / 5) * 3) )
     pulseDetected();
 }

 if (Signal < thresh && Pulse == true) {  // when the values are going down, the beat is over
   Pulse = false;              // reset the Pulse flag so we can do it again
   amp = P - T;               // get amplitude of the pulse wave
   thresh = amp / 2 + T;          // set thresh at 50% of the amplitude
   P = thresh;                // reset these for next time
   T = thresh;
 }

 if (N > 2500) {              // if 2.5 seconds go by without a beat
   thresh = 512;              // set thresh default
   P = 512;                  // set P default
   T = 512;                  // set T default
   lastBeatTime = sampleCounter;        // bring the lastBeatTime up to date
   firstBeat = true;
```

```
    secondBeat = true;
  }

}

void detectSetHighLow() {

  if (Signal < thresh && N > (IBI / 5) * 3) {
    if (Signal < T) {
      T = Signal;                // keep track of lowest point in pulse wave
    }
  }

  if (Signal > thresh && Signal > P) {
    P = Signal;                  // keep track of highest point in pulse wave
  }

}

void pulseDetected() {
  Pulse = true;                               // set the Pulse flag when we think there is a pulse
  IBI = sampleCounter - lastBeatTime;          // measure time between beats in mS
  Serial.print("IBI = ");
  Serial.println(IBI);
  lastBeatTime = sampleCounter;                // keep track of time for next pulse

  if (firstBeat) {                            // if it's the first time we found a beat, if firstBeat == TRUE
    firstBeat = false;                        // clear firstBeat flag
    secondBeat = true;
    return;
  }
  if (secondBeat) {                           // if it's the second time we found a beat, if secondBeat == TRUE
    secondBeat = false;                       // clear SecondBeat flag
    for (int i = 0; i <= 9; i++) {
      rate[i] = IBI;
    }
```

```
  }

  word runningTotal = 0;                    // clear the runningTotal variable

  for (int i = 0; i <= 8; i++) {            // shift data in the rate array
    rate[i] = rate[i + 1];                  // and drop the oldest IBI value
    runningTotal += rate[i];                // add up the 9 oldest IBI values
  }

  rate[9] = IBI;                            // add the latest IBI to the rate array
  runningTotal += rate[9];                  // add the latest IBI to runningTotal
  runningTotal /= 10;                       // average the last 10 IBI values
  BPM = 60000 / runningTotal;               // how many beats can fit into a minute? that's BPM!
  QS = true;                                // set Quantified Self flag (we detected a beat)
}
```