ECE 445 Final Report - Pet Health Monitor

By

Jeffery Haag
Rushill
Shah
Tanmay Thakur

May 4th, 2022

TA: Hojoon Ryu

Team Number : 59

Abstract

The primary purpose of this project is to provide cat owners an easy and effective way to monitor their pet's health and lifestyle while the pets go on with their daily activities. Body temperature, heart rate are quantitative indicators and daily activity is a qualitative indicator of a pet's health and lifestyle and with these metrics, a pet owner can make a much more informed decision.

The project was partially successful as the final prototype failed to record heart rate but was successful in recording the temperature and activity level data from sensors embedded in a portable wearable for a cat which then was transmitted to the user's phone via bluetooth. The user then can use the designed phone application to view the transmitted data in real time.

Contents

1. Introduction

A pet owner's prime concern is the well being of their pets and a trip to a veterinary doctor is usually very expensive. According to various pet care blogs the average cost of a routine check up can end up anywhere between $50 - $250 depending on the pet, tests required and the doctor's medical opinion. A short term hospitalization can cost anywhere between $600 - $1700. In order to prevent a serious complication, it is very important to monitor the overall health of one's pets. Usually owners monitor the health of their pets by behavioral changes such as eating habits, decrease in physical activity levels and sleep schedules. Often these behavioral changes are difficult to point out and thus an early onset of a health complication can go unnoticed for a long time which can lead to delayed medical attention. This delay can lead to further complications which not only increases the cost of treatment but sometimes can cause severe impacts on the pet's health and make the treatment procedures extremely difficult. Relevant health metrics and real-time updates to the owners can enable them to make an informed decision about their pet's current health and lifestyle and get them proper medical assistance accordingly. Current products in the market which record real-time vitals of pets are expensive and have a starting range of around $120, thus can be unaffordable to a lot of pet owners.

In order to help alleviate a part of this cost and ensure that an owner has a holistic view of their pet's health at a low cost, we designed a wearable sleeve to let pet owners keep track of important data during play sessions or as they go about their daily routine. As mentioned above, changes in metrics like body temperature and heart rate are good indicators of a pet's health and we aim to combine these metrics along with qualitative analysis of the pet's activity levels by measuring movement, and update the user in case of a significant change. To implement this, we decided to use specific cost effective sensors available to get the heart rate and the temperature data of the pet. For the movement data and the activity level, we used an accelerometer to measure the instantaneous acceleration of the pet which we used later to calculate the distance traveled and a quantitative measure of the daily activity level of the pet. The metric we used for this is named to be activity ratio and as the name suggests, it is a ratio of the total active time to total time. For the user end we designed an application for android phones which uses the inbuilt bluetooth interface of the phone to communicate with our device and receive the data collected from the sensors. The app then parses the data and provides the above mentioned metrics of the cat's health to the user. With a log of data over an extended amount of time can help the user spot the behavioral changes in a more statistical and metricized form.
Our design initially had only three high level requirements.

- The device should collect data from all three sensors and stream it to the control unit at least every 10 seconds.
- The device should maintain wireless connection to the user's smartphone and transfer the sensor data from the control unit to the user's phone.
- The android application should apply computations on incoming data to present accurate readings to the user in a human interpretable format.

Our final product fulfilled the second and third requirements but could only partially fulfill the first one. We were not able to integrate the heart rate sensor because of functional and ethical considerations which are discussed later in the report.

2. Outline of Subject Matter

2.1 Introduction

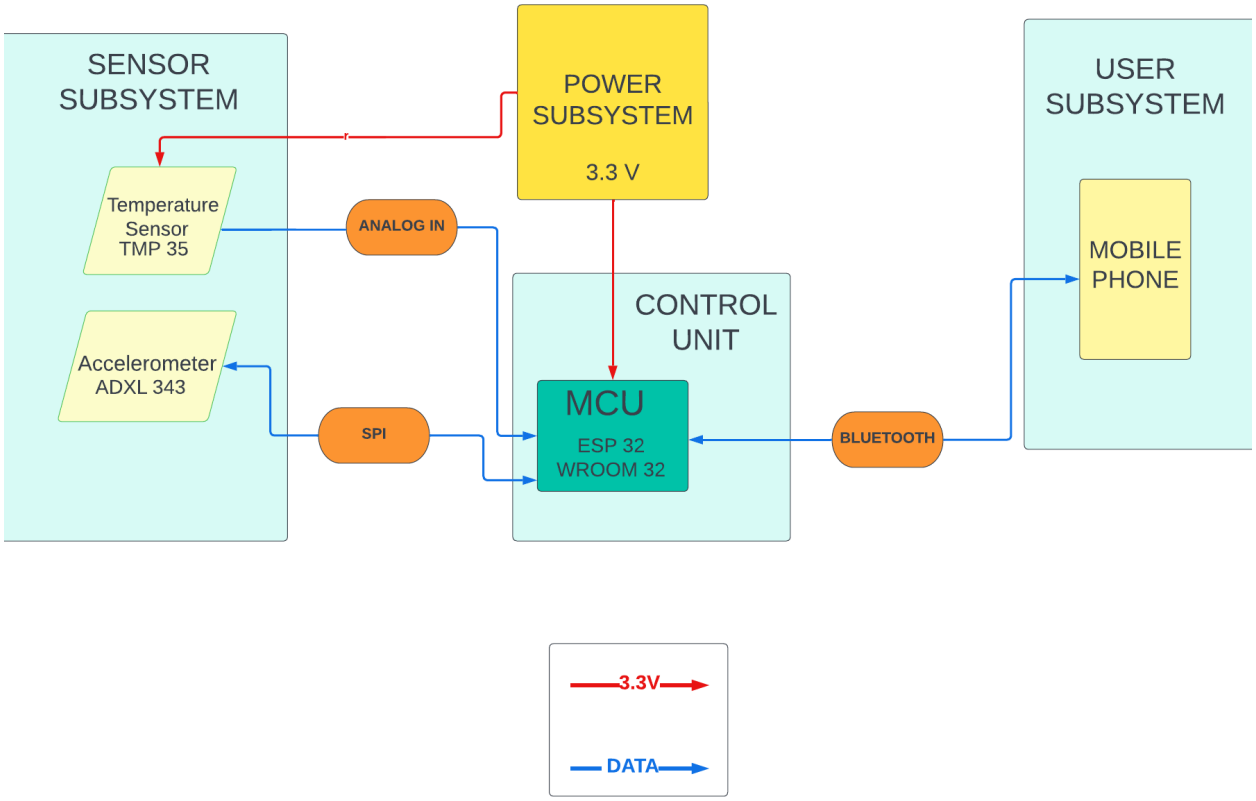
As mentioned above in the introduction (Section 1), our design's high level requirements aimed at integrating, storing and transmitting the collected data to the user end through the phone application. To achieve that we divided our design into four different subsystems which could be tested individually. Since our device is supposed to collect data from a pet, testing and calibrating the sensor data for accuracy was a very integral part and one of our main objectives of our design. Since these subsystems run independently, integrating them was a fairly simple process and hence helped us focus on the accuracy of data before moving on to integrate different components of our design. Another very important design consideration was the physical form of the device as it had to be portable and comfortable enough for the pet to be wearing it most of the time. We decided to create a cloth sleeve, resembling a sweater for pets with a pouch to hold the device. The temperature sensor needed to be in contact with the pet's body so we decided to use wires attached to the sleeve to connect the temperature sensor to our device. This design choice gave us the leeway to test for best placement of the sensor to get accurate readings.

2.2 Device Design

Since our project was heavily reliant on data collection and transmission, the sensors and the control unit were a big part of our device. Another independent yet very important aspect of our end product was the user interface on the phone as without presenting the collected data in a meaningful way, our product would have not served a practical purpose. Since all the

components in our design didn't need too much electrical power to operate, the power subsystem was not as significant of a design consideration as compared to the sensors.

Figure 1: Overall Block Diagram



As you can see in the figure above, the independent systems interact with each other using different protocols. The temperature sensor which we found accurate has an analog output whereas the accelerometer has a built in ADC converter and uses SPI protocol to communicate with the microcontroller. The microcontroller which was the control unit of our device had an inbuilt bluetooth module which then transmitted the data stored in its memory to the phone via bluetooth which the phone application then received by establishing an active connection with the device. Our initial design had a separate storage system as an SD card to store data for an extended duration before transmitting the data, but later we realized that since all the data to be sent is a string of characters, our device didn't need the extra SD card and the inbuilt memory of the microcontroller was enough to store a day's worth of data as a buffer.

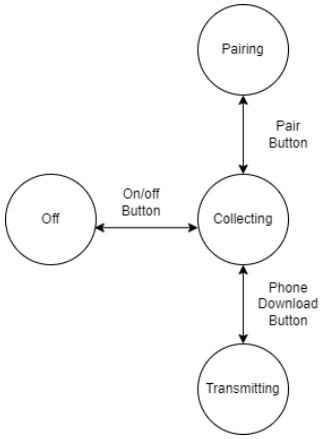
2.2.1 Power Subsystem

The power requirements of our design was limited to a 3.3 V battery as all our sensors and the microcontroller operate optimally in that range. We designed our power subsystem to regulate a 5 V battery to produce a 3.3 V output using a voltage regulator. The relevant schematic can be found in Appendix D. Our designed PCB routed the output voltage of the regulator to the microcontroller chip (Refer to Appendix D3 for the design). Even though the regulator was working perfectly, after an incident of burning the diode due to a short circuit caused by bridging two terminals of the resistor, we had to shift from our PCB to a development board. Since we could not use the power supply circuit on our development board, we ended up using a power bank outputting 5 V as there was already a regulator on the ESP 32 Wrover development board we used in our design.

2.2.2 Control Subsystem

The control system' main purpose is to interface the sensor subsystem and the app system using bluetooth and local memory as well as transferring states using button presses. Initially we were planning on using a local SD card to save information, in order to achieve the high level requirement of being able to store 24 hours of data. We discovered that the onboard memory of the microchip is sufficient for this purpose, and also doesn't require the additional complexity and weight of the micro SD card system, so we opted to just have the onboard memory.

Figure 2: Control Unit State Machine



Physically the control system includes the microcontroller, 4 buttons, and various resistors,wires, etc. to make it function properly. Since we used the esp32 Wroom, an additional bluetooth unit wasn't necessary because it comes onboard with the esp32. For buttons we have a physical power button and reset button. On the app we have a pair button,

and a transmit button, which will change the state of the control system. The default state is off, and when powered on goes into data collection mode. From data collection mode it can either go into pair mode or into transmit mode. When in collection mode, this is when sensor data is sent to memory after being modified by code to
clean it. In pair mode it pauses the transfer and establishes a bluetooth connection. While in transmit mode it reads the data from memory and transmits to the phone using the bluetooth functionality. This entire system is very simple but also extremely important, in order to deliver cleaned data to the app subsystem. To view the code that does this, consult appendix A

2.2.3 Sensors Subsystem

We originally planned on having 3 sensors for our device. Accelerometer, pulse, and temperature. Since pulse sensors are designed for human use, and don't work great under the best of conditions, getting it to work on a cat proved impossible. So on the final device we had an accelerometer sensor as well as a temperature sensor. The accelerometer takes the x,y, and z acceleration from the sensors' orientation. Because of this mounting the acceleration sensor was very important so that the z acceleration would measure -9.8m/s/s when at rest so as not to add unwanted acceleration values to other dimensions. We ended up mounting it inside the enclosure that housed the pcb, which in turn is held relatively stable on the cat's back.

For temperature, this is a relatively simple thermal coupling that we would want to be in as close contact to the cat's body as possible. It is an analog sensor that has a voltage on one of the wires that is easily converted to a value in Celsius in the control unit. In order to get as close to the cat's body as possible, the sensor runs through the material of the harness and is pressed into the cat's armpit. This value has a very good correlation with the cat's true internal temperature, but needs to be adjusted by a linear adjustment to match the core body temperature(Nutt).

2.2.4 User Subsystem

The purpose of the User-End application subsystem is to take as input the feed of live data from our control unit, process it and allow the user to view the data on an android application. The real time bluetooth data comes in and is registered by a Bluetooth Adapter, that connects only to the MAC address of our device. The data is parsed for the 3-axis accelerometer and temperature and then processed individually. This takes around 5 seconds to compute. The functions for bluetooth connection can be found in Appendix E.3.

We find the Activity Ratio by counting the number of readings where the change in axis acceleration for each was over a certain bound value (through trial and error, determined to be a 0.25) implemented as a low pass filter. We then find the ratio of the active events as defined above to the total events. We then use a double integral on the formula below for all axes in order to calculate the estimated distance covered.

Equation 1: Equation for distance used for integral

$$s = (\frac{1}{2}) * a * t^2$$

We conducted multiple trials, and found that the uncertainty was approximately 30%. We expected this rate due to the fact that our equation does not consider the initial velocity, or the '*ut*' term since we have no indication of that using the accelerometer itself (as seen in equation 2). In further work, we elaborate on GPS data that could improve this. The accelerometer parsing functions can be found in Appendix E.2.

Equation 2: Actual equation for distance

$$s = u * t + (\frac{1}{2}) * a * t^2$$

For temperature, we determine the maximum and average value and display them to the user. Our temperature sensor displayed more accurate cat temperature readings, within ±2°C of realistic values. The temperature parsing functions can be found in Appendix E.1.

3. Verification

3.1 Power Subsystem

Since the power requirements of the sensors were minimal, the only requirement we had from the power subsystem was an output of regulated 3.3 V and the battery to last for at least 6 hours before needing to be charged. To verify if our power supply was supplying a constant voltage output of 3.3 V, we used the power supply in the Senior Design lab to supply a constant DC voltage of 5 V and then used an oscilloscope to measure the output reading to be 3.3 V (Refer to appendix C.3 for the RV table).

To verify the battery life, we first computed the battery life by calculating power consumed by each device of our circuit to sum them up and find the total power consumed by the device. We looked at the datasheets to find the average consumption of each device. Refer

to appendix for the exact consumption by each device and the calculation of total power. We then looked up at the total energy of our battery to calculate the number of hours it would last. To verify we measured the current across the power supply at a 5 V output to calculate the power consumed by our device. We then used the measured current to measure the power consumed by multiplying it by the voltage across the device.

3.2 Control System

As you can view in Appendix C.2, a working control system would do the following; sensor data is accurately written to local memory, data from local memory is read and correctly transmitted using the bluetooth features, the device can be turned off and on and has an LED representing the change, bluetooth pairing is initiated by a button, with an LED representing this state. With the exception of the bluetooth pairing being represented by an LED, we were able to verify all the above requirements. To be clear, the device does pair, it just doesn't have an LED to tell the user it is pairing. All these are verified not only by the listed verification found in the table but also by the high level requirements that require the control system, also being verified

3.3 Sensor Subsystem

As you can see in Appendix C.4, a working sensor subsystem would contain the following: sensors send data at least once per second, sensors are active and transmit data to the control unit at least 95% of the time the unit is on. The pulse sensor data and the accelerometer is accurate to ±10% of actual values and the temperature sensor is accurate to ±5% of actual values. All sensors are verified to send data once per second and also transmit 100% of the time when in the recording state from our testing. As previously mentioned, unfortunately our pulse sensor is not operational so that aspect of the requirement is failed. For the temperature sensor though we achieved 5% accuracy quite easily in our tests. AS for the accelerometer, this is extremely accurate in and of itself, so it passes the requirement, but when we tried to use this data to calculate distance or activity time, a higher degree of error is introduced.

3.4 User Subsystem

As seen in Appendix C.1, a fully functional User-End subsystem would fulfill the following requirements: smartphone is able to connect to the device via bluetooth, phone can receive data from the last 24 hours, and app parses data to reveal distance and activity with temperature metrics.

We fulfilled all these requirements on the User-End subsystem. Since we moved to real-time bluetooth data streams, we were able to collect data indefinitely until a button on the device

would be pressed, sending a signal to stop collection. We were able to connect to bluetooth and enforced a connection on hitting the pair button by pairing to the MAC address of the ESP-32.

Finally, we are left with the metrics requirement. While we were unable to depict our data with graphs on the app itself due to library issues, we managed to calculate an Activity Ratio as an indication of time the cat was active during the collection session. Additionally, we use the distance formula with integrals as discussed in Section 2.2.4 to estimate the distance covered. This was within ±30% of actual distance values, but our tests were conducted on smaller distances. We find the maximum and average temperature from a collection session and display those metrics too.

Images from the android application can be found in Appendix D displaying the data pages and home screen

4. Costs

Table 1: Parts Cost Summary

Manufacturer	Model	Qty	Cost($)	Description
Adafruit	165	1	2.75	Body temperature sensor 1
Texas Instruments	LM35DZ PN JUNCTION SENSOR	1	1.66	Body temperature sensor 2
Adafruit	1093	1	25.00	Pulse sensor
Maxim Integrated	MAX30102EFD +T	1	5.06	Pulse sensor
Adafruit	1231	1	17.50	Accelerometer
Adafruit	4097	1	6.00	Accelerometer
Adafruit	387	1	0.75	LED's
Adafruit	1119	10	2.50	Buttons
Adafruit	254	1	7.50	SD Card To Spi board

Adafruit	1294	1	9.95	SD card 4 gb w/ adapter
Adafruit	2011	1	12.50	Battery 3.7V
Adafruit	1959	1	14.95	Battery 5v
Adafruit	4077	1	9.95	Bluetooth Module
Adafruit	2746	1	19.95	Bluetooth Module

Table 2: Labor Cost Estimate

Labor Type	Cost/hou	Total Hou	Cost($)
Soldering	40	12	480
Programming	60	30	1800
PCB Design	50	10	500
Testing	50	10	500
Techinical Writing	40	20	800
Parts Ordering	30	15	450
Parts			
Total	-	97	4530

As you can see, the total cost of labor is around $4,500 and the total cost of parts was $136. Of course you may notice that there are many sensors that were ordered that were not included on the final product. This is because we did testing on which sensors would work best and only the best functioning ones made it into the final product. The costs of this specific model thus has a lot of R&D costs baked into it, in order to make the most accurate tool.

5. Conclusion

5.1 Notable Features

With this project, we have managed to successfully detect cat temperature and quantify the intensity and frequency of their movements with an activity ratio. We were able to successfully test these activity ratio values for consistency with higher ranges of activity as compared to lower. While we were unable to deliver on our initial proposal of pulse sensing, we believe that even if we had an IR pulse sensor, we would be unable to measure the pulse due to damping of vibrations through the cat's fur.

5.2 Testing Uncertainty

Our device was developed for a cat of a certain size, and there is always some subjective uncertainty and we would need to adjust the temperature sensor position for each cat. As a result, our design wouldn't work for all without adjustment. The accelerometer was placed on the back in order to minimize errant movements being tracked, but a cat walking in itself makes the accelerometer move. As a result we had to measure the change in axis acceleration in order to get our activity ratio. The temperature sensor had a constant uncertainty, and with being run for longer it got more accurate.

5.3 Ethical Considerations

Since our project requires us to directly work with and test our device on a cat, we strictly abided by the regulations set out by PETA and the Human Care for Animals Act[1]. We found that 2-5 kilograms is a safe weight to carry, while our device (along with the harness) was well under that weight limit. We ensured that all electrical components were insulated and there was no direct exposure to the cat's skin (other than sensor surfaces). Additionally, due to our backup pulse sensor requiring EKG adhesive fluid, we did not go through with it to prevent any discomfort or harm to the cat. We ensured that all tests were done in the presence of the cat owners too. While PETA emphasizes animal freedom and no experimentation, we made sure to test only when the cat was deemed to be 'comfortable' by the owner.

Additionally, we followed the ACM Code of Ethics[2], with the sole purpose of our project being to contribute to the scientific field and computing, while ensuring that there is no negative externality or harm caused by our actions.

5.4 Future Work

There are many possible improvements to the project that we could make if we continued working on it beyond this course. Improvements could be made in the technical design and physical aspects of the project. In order to give users a better estimate of the distance covered and actual position of the cat, we believe that we could use the WiFi module of the ESP-32, which has specific plugins and tools for positional data. This would augment our accelerometer and give us a better metric for activity.

---

[1] HAARC, Retrieved May 4, 2022, from https://ilga.gov/legislation/ilcs/ilcs3.asp?ActID=1717

[2] *The code affirms an obligation of computing professionals to use their skills for the benefit of society.* Code of Ethics. (n.d.). Retrieved May 4, 2022, from https://www.acm.org/code-of-ethics

In order to make our actual physical design more compact and easier for cats to handle, we propose to locate the temperature sensor and the accelerometer on a collar-like attachment, which would be powered by a battery located on the harness. This improvement would greatly streamline the physical design and make better use of the sensing and real-time data transfer implementations.

6. References

1. *The code affirms an obligation of computing professionals to use their skills for the benefit of society.* Code of Ethics. (n.d.). Retrieved May 4, 2022, from https://www.acm.org/code-of-ethics
2. Costa, D. dos S., Turco, S. H. N., Ramos, R. P., Silva, F. M. F. M., & Freire, M. S. (n.d.). *Electronic Monitoring System for measuring heart rate and skin temperature in small ruminants*. Engenharia Agrícola. Retrieved May 4, 2022, from https://www.scielo.br/j/eagri/a/VSYJ448gk9DqpcxyntD6MfK/?lang=en.
3. *Maddie's Shelter Medicine Program, College of Veterinary ... - HSVMA*. (n.d.). Retrieved May 5, 2022, from https://www.hsvma.org/assets/pdfs/kelly_nutt_poster.pdf
4. Medicine, C. for V. (n.d.). *Animal & Veterinary*. U.S. Food and Drug Administration. Retrieved May 4, 2022, from https://www.fda.gov/animal-veterinary
5. Plotts, E. (2020, June 8). *How much does a vet visit cost? here's everything you need to know*. Pawlicy Advisor. Retrieved May 4, 2022, from https://www.pawlicy.com/blog/vet-visit-cost/.
6. PostedinHealth, Megan, P. by, Cats, P. S., PostedinBehaviour, & PostedinNutrition. (2021, November 25). *How strong are cats?* Tuxedo Cat. Retrieved May 4, 2022, from https://www.tuxedo-cat.co.uk/how-strong-are-cats/.
7. . (n.d.). Retrieved May 4, 2022, from https://ilga.gov/legislation/ilcs/ilcs3.asp?ActID=1717

Appendix

Appendix A- Arduino code

Appendix A.1 - Libraries & Pins Used

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL343.h>
#include "BluetoothSerial.h"
#include "esp_adc_cal.h"

#define TEMP_PIN 35 //temp pin for sensor
#define MEASURE_SWITCH_PIN 11 //GIOP21 pin connected to  upload
switch ;HI=mesure LO=idle
Adafruit_ADXL343 accel = Adafruit_ADXL343(12345);
//35 34 reserved
BluetoothSerial SerialBT;
```

Appendix A.2 - Code for Temperature Sensor

```
int setUpThermo()
{
  return 0;
}

uint32_t readADC_Cal(int ADC_Raw)
{
  esp_adc_cal_characteristics_t adc_chars;

        esp_adc_cal_characterize(ADC_UNIT_1,    ADC_ATTEN_DB_11,
ADC_WIDTH_BIT_12, 1100, &adc_chars);
  return(esp_adc_cal_raw_to_voltage(ADC_Raw, &adc_chars));
}
```

```
String getThermodata()
{
  float reading = analogRead(TEMP_PIN);
  // Calibrate ADC & Get Voltage (in mV)

  float Voltage = readADC_Cal(reading);

  float LM35_TempC_Sensor1 = (Voltage - 500) / 10;
  float temperatureF = (LM35_TempC_Sensor1 * 1.8) + 32;


  return (String)(temperatureF) ;
}
```

Appendix A.3 - Code for Temperature Sensor

```
int setUpAccel()
{
  #ifndef ESP8266
  while (!Serial); // for Leonardo/Micro/Zero
#endif
  /* Initialize the sensor */
  if(!accel.begin())
  {
     /* There was a problem detecting the ADXL343 ... check your
connections */
      Serial.println("Ooops, no ADXL343 detected ... Check your
wiring!");
    while(1);
  }
   accel.setRange(ADXL343_RANGE_4_G);
}


String getAcceldata()
```

```
{
  sensors_event_t event;
  accel.getEvent(&event);

   String returnString = (String) event.acceleration.x + "," +
(String)     event.acceleration.y     +     ","     +     (String)
event.acceleration.z;
  return returnString;
}
```

Appendix A.4 - Code for Pulse Sensor

```
int setUpPulse()
{
  //TODO:
  pinMode(35, INPUT); // Setup for leads off detection LO +
  pinMode(34, INPUT); // Setup for leads off detection LO -
  return 0;
}


String getPulsedata()
{
  return (String)analogRead(A0);
}
```

Appendix A.5 - Code for ESP32's Bluetooth Module

```
int setUpBluetooth()
{
  SerialBT.begin("Cat Thermometer");
  Serial.println("Connected Succesfully!");

   SerialBT.connect();
   return 0;
}
```

```
int sendBluetoothData(String data)
{
   SerialBT.print(data);
   Serial.print(data);
   return 0;
}
```

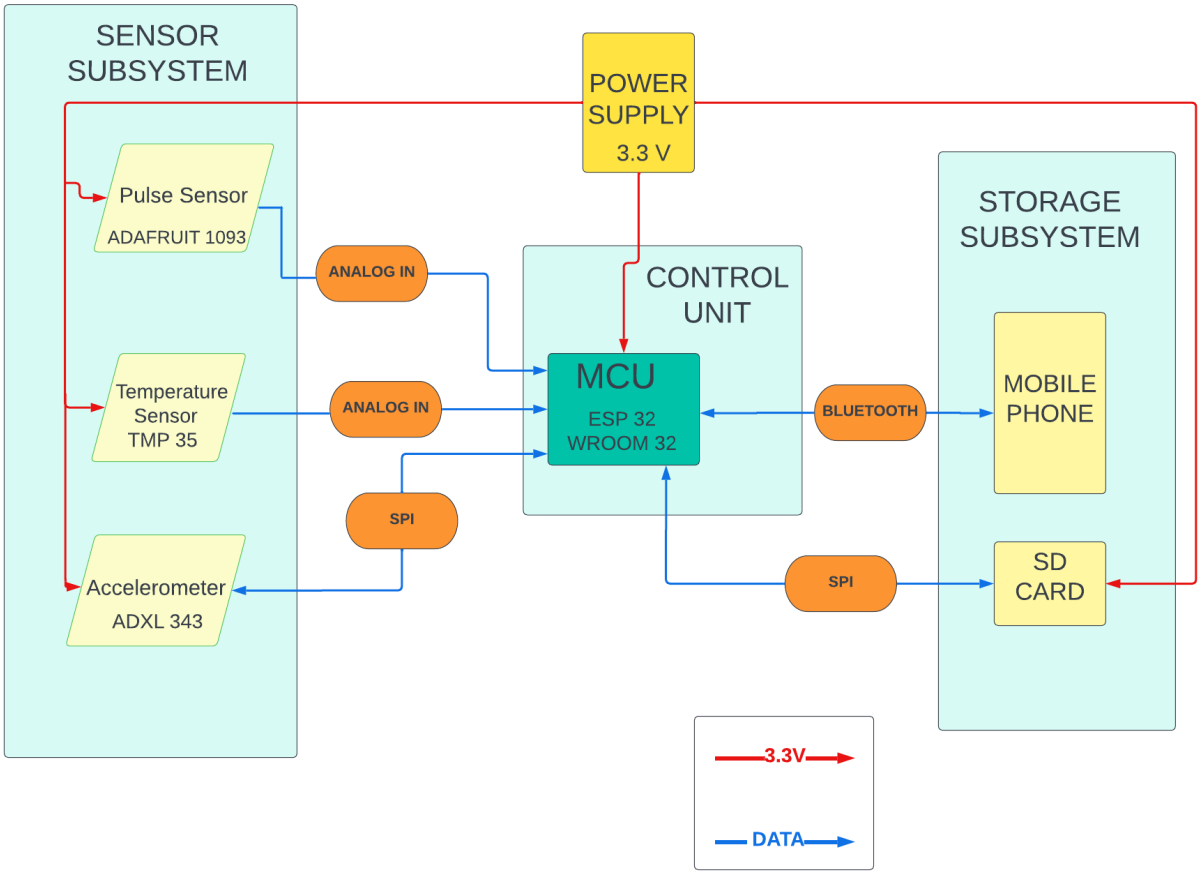
Appendix A.6 - Code for the Control Unit

```
void setup() {

  Serial.begin(115200);
  Serial.print("hi");
  // put your setup code here, to run once:
  setUpThermo();

  setUpAccel();

  setUpBluetooth();

  setUpPulse();

}

void loop() {

  int measureState = digitalRead(MEASURE_SWITCH_PIN);

  //Upload Button

  delay(1000);

    String accel_data = getAcceldata();
    String temp_data = getThermodata();
    String pulse_data = getPulsedata();
```

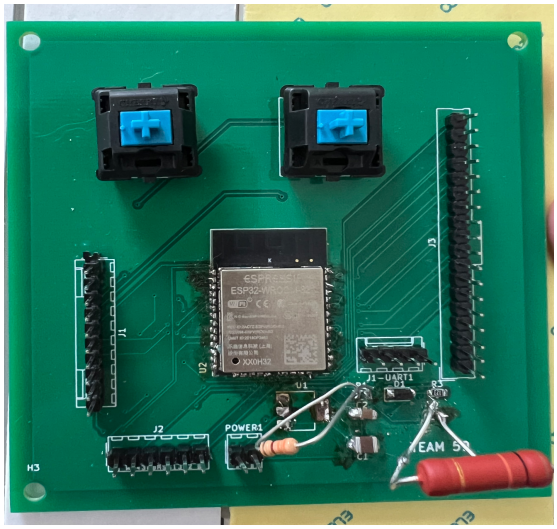
```
    String trans_string = accel_data + "\t" + temp_data + "\t" +
pulse_data + "\n";
    sendBluetoothData(trans_string);

  }
```

Appendix B - Block Diagrams and Other Images

Appendix B.1

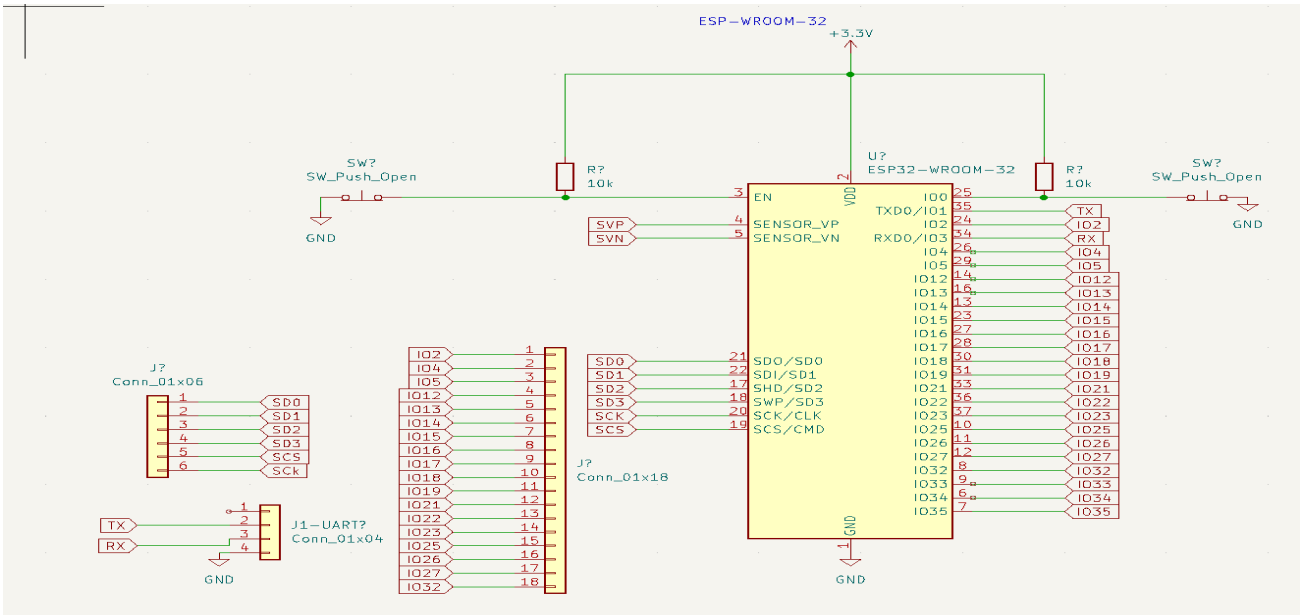


Appendix B.2



Appendix B.3

Appendix B.4



Appendix C - RV Tables

Appendix C.1

R&V Table 1: User-End Subsystem

Requirements	Verification
Smartphone is able to connect to bluetooth unit	1.User will navigate to their smartphone's bluetooth settings and search for available devices while standing within 10 feet of the device. 2.The user will attempt to pair with the device on their phone. 3. When the smartphone returns a pair complete, or similar message it is a success.
Phone can receive data from the last 24 hours	1. After successful pairing, navigating to the smartphone app and updating the app via an on screen prompt should result in the presence of data from the devices sensors

	dating back 24 hours. The presence of data visible in the app dating back 24 hours is a success.
App parses data to reveal metrics such as distance covered, sedentary time, and check for healthy vital signs.	1.Once in the app, the data should be visible in the form of distance traveled, sedentary time, frequency and total time with vitals spent outside healthy range. 2. 2. Success is, The presence of this data in readable form Frequency and time outside healthy vital range accurate to ±5% real value Distance traveled and Sedentary time accurate to ±20% real value.

Appendix C.2

R&V Table 2: Control Unit Table

Requirements	Verification
Sensor data is received and accurately written to the SD-Card.	1. We will feed custom test data such that test data is held at a certain value to the microcontroller using a lab signal generator for digital data and a power supply for analog data. 2. After a set period we will plug the sd into a computer so we can view the raw data in .txt format. 3.We will compare the data in the .txt to our test sensor data to ensure it is being recorded correctly
SD-Card data is read and fed to the bluetooth module	1. We will save custom data to the SD card using a computer, again such that the data is held constant. 2. We will then have the control unit output the data to the bluetooth subsystem by putting the control unit in that state. 3. We will check what data is being sent on the output line by using an oscilloscope, and verify it is the test data we saved in.
The unit is turned off and on by the button	1. We will observe the unit with a voltmeter

and it's state is represented by an LED light	and ensure that no wire is hot while the LED is off. 2. We will press the on button and ensure that the power LED turns on and then check with the voltmeter that the wires are now hot. 3. We will press the off button and ensure the LED turns off and the wires are no longer hot.
Bluetooth pairing state is initiated by the pairing button and is represented by an LED light.	1. The pairing LED should be off before any action is taken 2. After pressing the pairing button, the device should show as visible\pairable to any bluetooth compatible device in the area and the pair LED should turn on.

Appendix C.3

R&V Table 3: Power Unit Subsystem

Requirements	Verification
1. Battery lasts for 6 hours	1. We will measure the Potential applied and the supplied current using a multimeter and an oscilloscope while verifying individual subsystems. 2. With the recorded current and voltage values, we can calculate the power consumed by each of the subsystems. 3. Since we know the energy stored in the battery, we make sure that the battery can provide the required power for at least 6 hours.

Appendix C.4

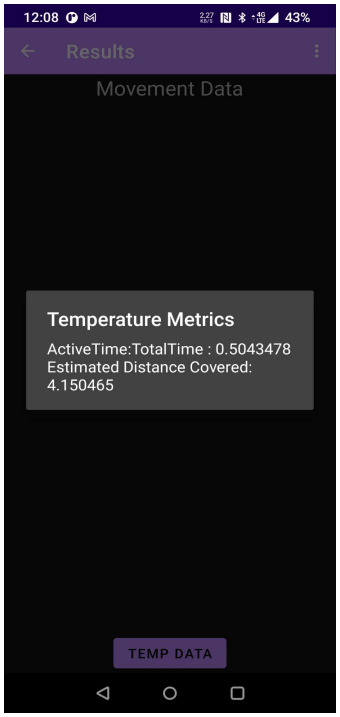
R&V Table 4: Sensor Subsystem

Requirements	Verification
Sensors send data at at least once per second	1. We will check each sensor output with an oscilloscope and ensure it is sending information at once per second.

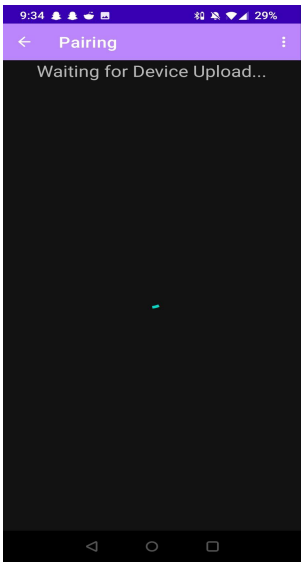
Sensors are active and transmitting data to the control unit, at least 95% of the time the unit is on.	1. We will allow sensors to collect data for 1 hour. Assuming the control unit has passed its earlier tests, we will then check the SD on a computer. 2. By seeing how many data values there are on the SD, and knowing the clock speed and thus, how many values there should be, we can calculate what percent of the time the sensors were sending data. If it is greater than 95%, it passes.
The pulse sensor data and the accelerometer is accurate to ±10% of actual values and the temperature sensor is accurate to ±5% of actual values.	1. To test temperature, we will use an outside infrared thermometer and compare it to the value of the sensor recorded on the sd card at temperatures 36,36.5,37,37.5,38,38.5, and 39 celsius using the infrared as a baseline. If the average difference at each temperature is less than ±5%, it passes. 2. To test the accelerometer, we will attach a phone with an accelerometer to the cat via the harness. We will allow it to move for 10 minutes and then get the graph for x,y,z acceleration from the sensor as well as the phone. For each x,y,z we will find $x = (Y_{phone} - Y_{Sensor)}$. We will then do error = x / Y_{phone} . If $x <= 0.05$ it passes. 3. To test the pulse sensor, after consulting a vet, we will use a person's hand to measure the cat's heart rate over a 5 minutes while recording pulse every 30 seconds. At the same time the sensor will be attached. We will then compare the manually gotten data to the sensor data. We will average the pulse sensor data to its value every 30 seconds. We will then calculate the differences and if it is under ±5% from the manual data it passes.

Appendix D - App Images

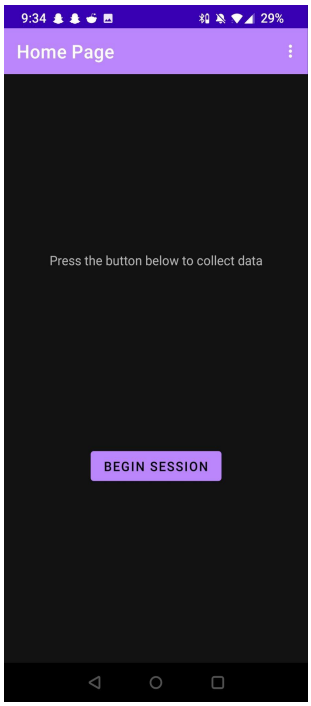
Appendix D.1 - Data Page



Appendix D.2 - Parsing Page



Appendix D.3 - Home Page



Appendix E - App Code

Appendix E.1 - Temperature Processing

```
public float[] tempHandler(SparkView sparkView, MainActivity m){
    float[] usearr = m.gettempvals();
    float sum = 0;
    float max_ = 0;
    float min_ = 1000;
    float[] actualarr = new float[usearr.length];
    int i = 0;
    int zerocnt = 0;
    for (float f:usearr){
        if(f!=0){
            actualarr[i] = f;
            i += 1; }
        else{
            zerocnt += 1; }
        sum += f;}
    for (float f:usearr){
        if(f>max_){
            max_ = f;}
        if(min_>f){
            min_ = f;}}
    float avg = 0;
    if(usearr.length!=0){
        avg = sum/(usearr.length-zerocnt);
    }
    if(sparkView!=null){
        sparkView.setAdapter(new MyAdapter(usearr));
    }
    float[] ret = new float[2];
    ret = new float[]{avg, max_};
    return ret;
}
```

Appendix E.2 - Acceleration Processing

```
public float[] accHandler(MainActivity m){
    float[] tempx = m.getXvals();
    float[] tempy = m.getYvals();
    float[] tempz = m.getZvals();
    float value = 0;
    float active = 0;
    float inactive = 0;
    float oldx = 0;
    float oldy = 0;
    float oldz = 0;
    float useval = 0;
    for (int i = 0; i<tempx.length; i++){
        if(Math.abs(tempx[i]-oldx)+ Math.abs(tempy[i]-oldy)+Math.abs(tempz[i]-oldz)>2){
            active += 1;
        }
        else{
            inactive += 1;
        }
        float ret = (float) Math.sqrt(Math.pow(Math.abs(tempx[i]-oldx), 2) + Math.pow(Math.abs(tempy[i]-oldy), 2) + Math.pow(Math.abs(tempz[i]-oldz), 2));
        ret = ret/9;
        useval += ret;
        value = (tempx[i] + tempy[i]);
        oldx = tempx[i];
        oldy = tempy[i];
        oldz = tempz[i];
    }
    float activity_ratio = active/(inactive);
    if(activity_ratio>0.25) activity_ratio += 0.2;
    if(activity_ratio<0.1) useval = 0;
    float[] sus = new float[]{activity_ratio, useval};
    return sus;
}
```

Appendix E.3 - Bluetooth Connection

```
public boolean onBluetoothPage(BluetoothManager BM, MainActivity m) throws IOException {
    BluetoothAdapter Mbt = BM.getAdapter();
    String deviceName = "Cat Thermometer";
    BluetoothDevice result = null;
    Set<BluetoothDevice> devices = Mbt.getBondedDevices();
    if (devices != null) {
        for (BluetoothDevice device : devices) {
            if (deviceName.equals(device.getName())) {
                result = device;
                Log.i( tag: "deviceName", result.getName());
                break;
            }
        }
    }
    if (result != null){
        Log.d( tag: "BT",  msg: "Trying Socket...");
        UUID MY_UUID = result.getUuids()[0].getUuid();
        socket=result.createRfcommSocketToServiceRecord(MY_UUID);
        this.serverSocket = Mbt.listenUsingRfcommWithServiceRecord( name: "test", result.getUuids()[0].getUuid()); // 1
        Log.d( tag: "BT",  msg: "Got Socket!");
    } else{
        this.socket = null;
        this.cancelled = true;
        Log.d( tag: "bluetooth",  msg: "failed bt socket");
    }

    this.cancelled = false;
    if (Mbt.isEnabled()){
    } else{
        Mbt.enable();
    }
```

```
    byte[] buf = new byte[2048];
    int bytes;
    OutputStream tmpOut = null;
    InputStream tmpIn = null;
    if(socket!=null){
        Log.d( tag: "BT",  msg: "Connecting...");
        socket.connect();
        Log.d( tag: "BT",  msg: "Connected!!"); }
    int i = 0;
    int inval = 0;
    while(true){
        boolean flag = false;
        String message = null;
            try{
                if(i==10){
                    break;
                }
                tmpIn = this.socket.getInputStream();
                inval = tmpIn.read(buf);
                message = new String(buf,  offset: 0, inval);
            } catch (IOException e) {
                e.printStackTrace();
            }
            if(message.contains("ece445") | inval == 4){
                return true;
            }
            else{
                proc(message, inval, m, i);
            }
            i += 1;
    }
    return true;
}
```

Appendix F - Battery life calculation

Energy stored in the Battery = 5 V * 2600 mAh
= 13 Watt-h

$$P_{ESP32} = 3.3V \cdot 0.5A = 1.6W$$
$$P_{Heart-rate} = 1.8 \cdot 0.6A = 1.08W$$
$$P_{Bluetooth} = 3.3V \cdot 0.0026A = 0.00858W$$
$$P_{accelerometer} = 2.5V \cdot 0.002A3 = 0.00858W$$
$$P_{temperature} = 3.3V \cdot 0.025A = 0.0826W$$
$$P_{SdCard} = 3.3V \cdot 0.1A = 0.33W$$
$$P_{total} = 3.109W$$

Device operation time = Energy Stored in Battery / Total Power Consumed
Device operation time = $\frac{13W}{1.9812W}$
Device operation time = 6.56 hours