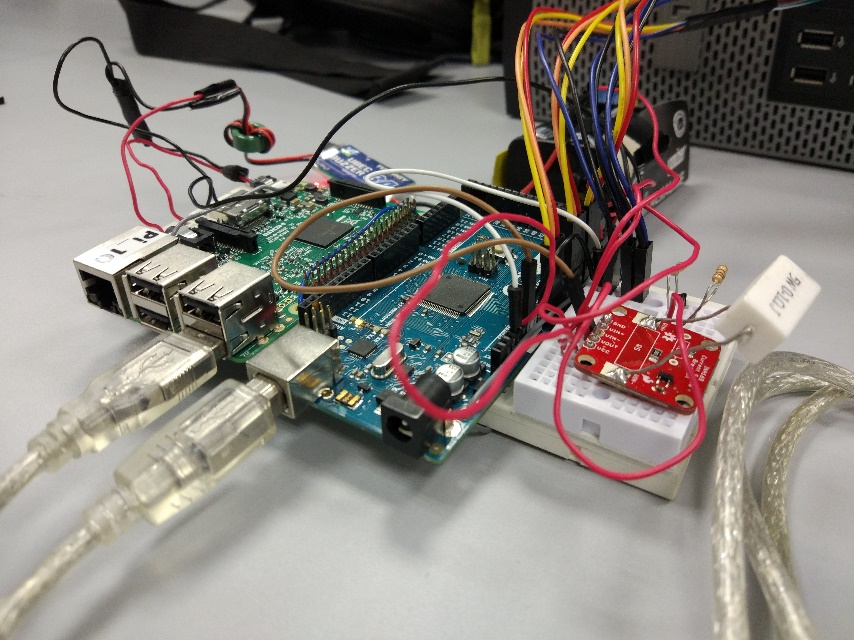


**CG3002 Embedded System Design Project**

Semester 1 2017/2018

**“Dance Dance”**

**Design Report**



Github link: github.com/evilmtv/CG3002-Dance-Dance-01

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group **01** | Name | Student # | Sub-Team | Role |
| Member #1 | Lim Jun Hao | A0147944U | Software/  Hardware | Subteam Lead |
| Member #2 | Raphael Quek Hao  Chong | A0139716X | Firmware/  Hardware | Member |
| Member #3 | How Wei Wen,  Nicolas | A0139523E | Firmware/  Hardware | Subteam Lead |

Contents

[Section 1: Project Management Plan 4](#_Toc498898368)

[1.1 Timeline 4](#_Toc498898369)

[Section 2: System Functionalities 5](#_Toc498898370)

[2.1 Feature Lists 5](#_Toc498898371)

[2.2 User Stories 5](#_Toc498898372)

[2.3 Use Cases 6](#_Toc498898373)

[Use Case Diagram 6](#_Toc498898374)

[Use Case: Dancing 6](#_Toc498898375)

[Use Case: Idle (standing) 7](#_Toc498898376)

[Section 3: System Architecture 8](#_Toc498898377)

[3.1 Overview 8](#_Toc498898378)

[3.2 Communication Protocols 9](#_Toc498898379)

[3.3 Hardware layout 9](#_Toc498898380)

[Section 4: Component Interactions and Design 10](#_Toc498898381)

[4.1 Main algorithm for activity detection 10](#_Toc498898382)

4.2 FreeRTOS in Arduino Mega………………………………………………………… 10

[4.3 Communication between Raspberry Pi and Arduino Mega 10](#_Toc498898383)

[4.3.1 Communication Verification 10](#_Toc498898384)

[4.3.2 Data Verification 11](#_Toc498898385)

[4.3.3 Baudrate 11](#_Toc498898386)

[4.4 Sample size and interval 11](#_Toc498898387)

[4.5 Coordination between Raspberry Pi and Arduino Mega 11](#_Toc498898388)

[4.6 Data processing 11](#_Toc498898389)

[4.7 Classifier 11](#_Toc498898390)

[4.8 ‘Best’ answer 12](#_Toc498898391)

[4.9 Communication between Raspberry Pi and server 12](#_Toc498898392)

[Section 5: Hardware Details 13](#_Toc498898393)

[5.1 Overview 13](#_Toc498898394)

[5.2 Hardware Parts 13](#_Toc498898395)

[5.3 Reasons for selected hardware parts 14](#_Toc498898396)

[5.3.1 ADXL335 vs MPU6050 14](#_Toc498898397)

[5.3.3 UBEC 14](#_Toc498898398)

[5.4 Circuit Diagram 15](#_Toc498898399)

[5.4.1 Reading of Voltage 15](#_Toc498898400)

[5.4.2 Reading of Current 15](#_Toc498898401)

[5.4.3 Calculation of Power and Cumulative Power 16](#_Toc498898402)

[5.4.4 Pin Table 16](#_Toc498898403)

[Section 6: Reflection 17](#_Toc498898404)

[6.1 Improvements 17](#_Toc498898405)

[6.1.1 Calibration 17](#_Toc498898406)

[6.1.2 Comfort & Aesthetics 17](#_Toc498898407)

[6.1.3 Overheating 17](#_Toc498898408)

[6.2 Issues faced 17](#_Toc498898409)

[6.2.1 What could have been done instead 17](#_Toc498898410)

[Section 7: Societal Impact 18](#_Toc498898411)

[7.1 Possible Future Benefits and Uses 18](#_Toc498898412)

[7.2 Concerns 18](#_Toc498898413)

[Section 8: References 19](#_Toc498898414)

# Section 1: Project Management Plan

## 1.1 Timeline

Week 4:

Plan whole project

**Design Report: (10%)**

Week 5:

Each team starts their own part

Work on the feedback given

Buy all required parts

Week 6:

Complete individual parts of 1st prototype

**Progress checkpoint: (5%)**

Week 7:

Polish individual parts of 1st prototype

**1st Prototype Evaluation: (15%)**

Week 8:

Start integrating the individual parts

Week 9:

Ensure team is on the path for 2nd prototype

Week 10:

Fulfill the Baseline requirements

Complete 2nd prototype

Collect raw data for first five moves

Week 11:

Polish 2nd prototype

Collect more raw data for first five moves

**2nd Prototype Evaluation: (30%)**

Week 12:

Further polish 2nd prototype

Implement additional improvements

Collect raw data for last 5 moves

Week 13:

Polish additional improvements

Collect more raw data for last 5 moves

**Final Demo: (40%)**

# Section 2: System Functionalities

## 2.1 Feature Lists

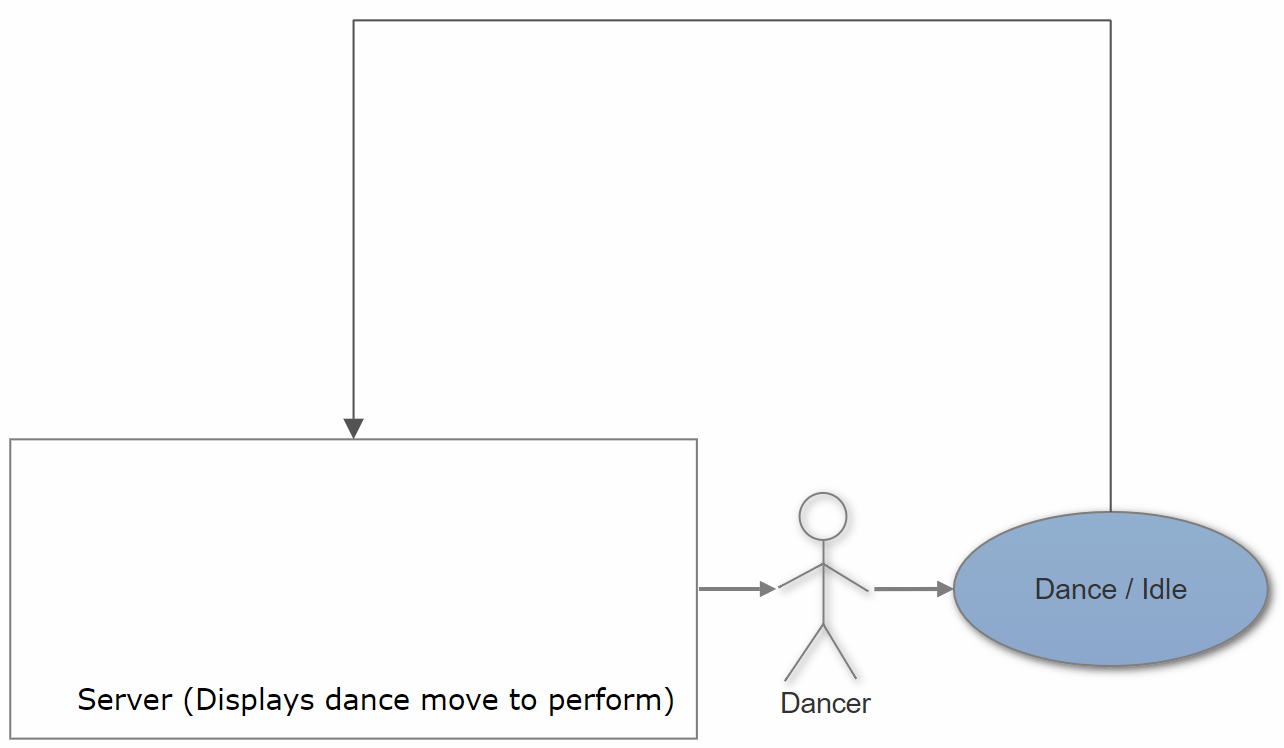
* Very low latency system (Quick and accurate results)
* Simple set-up to easily understand how to put on system and sensors
* Modular, simple and efficient raw data processing
* Simple to expand/remove set of moves as a developer/power user
* Calibration to allow for different users of varying body size/shape

## 2.2 User Stories

* As a user, I want the wearable to be comfortable.
* As a user, I want the wearable to have minimal wires so that they will not get in the way of my dancing.
* As a user, I want the wearable to be lightweight so that it will not be tiring to dance with it.
* As a user, I want to be able to do a finishing dance move to end my dance routine.
* As a user, I want the system to detect my dance moves quickly as my dance routine is usually fast paced.
* As a user, I want the system to accurately detect my dance moves.
* As a different sized user, I want the wearable to be free-size so that it can be worn by me comfortably.
* As a user, I want the wearable to last a long time without changing the batteries so that I do not have the inconvenience of frequently changing the batteries.
* As a power user, I want to know the power used by the wearable
* As a power user I want to be able to add and remove dance moves

## 2.3 Use Cases

### Use Case Diagram



*Figure 1: Use Case Diagram*

### Use Case: Dancing

**Main Success Scenario**

1. Server displays dance move to perform
2. User performs displayed dance move
3. System reads and evaluates movements to match pre-determined dance moves
4. System determines confidence level of data
5. Confidence level of data is high enough and system sends move to server
6. Server acknowledges message and generates next dance move to perform
7. Use case ends

**Alternate Scenario**

1. Confidence level of data is not high enough and nothing is sent to server
2. System continues reading and evaluating data (return to step 2)
3. Use case ends

### Use Case: Idle (standing)

**Main Success Scenario**

1. User stays idle
2. System detects idle state
3. System does not send message to server (waits for dance move to be detected)
4. Use case ends

**Alternate Scenario**

1. User idles for fixed amount of time
2. System detects and evaluates it as logout command
3. System sends logout message to server
4. Server displays “Good bye” and shuts down program
5. Use case ends

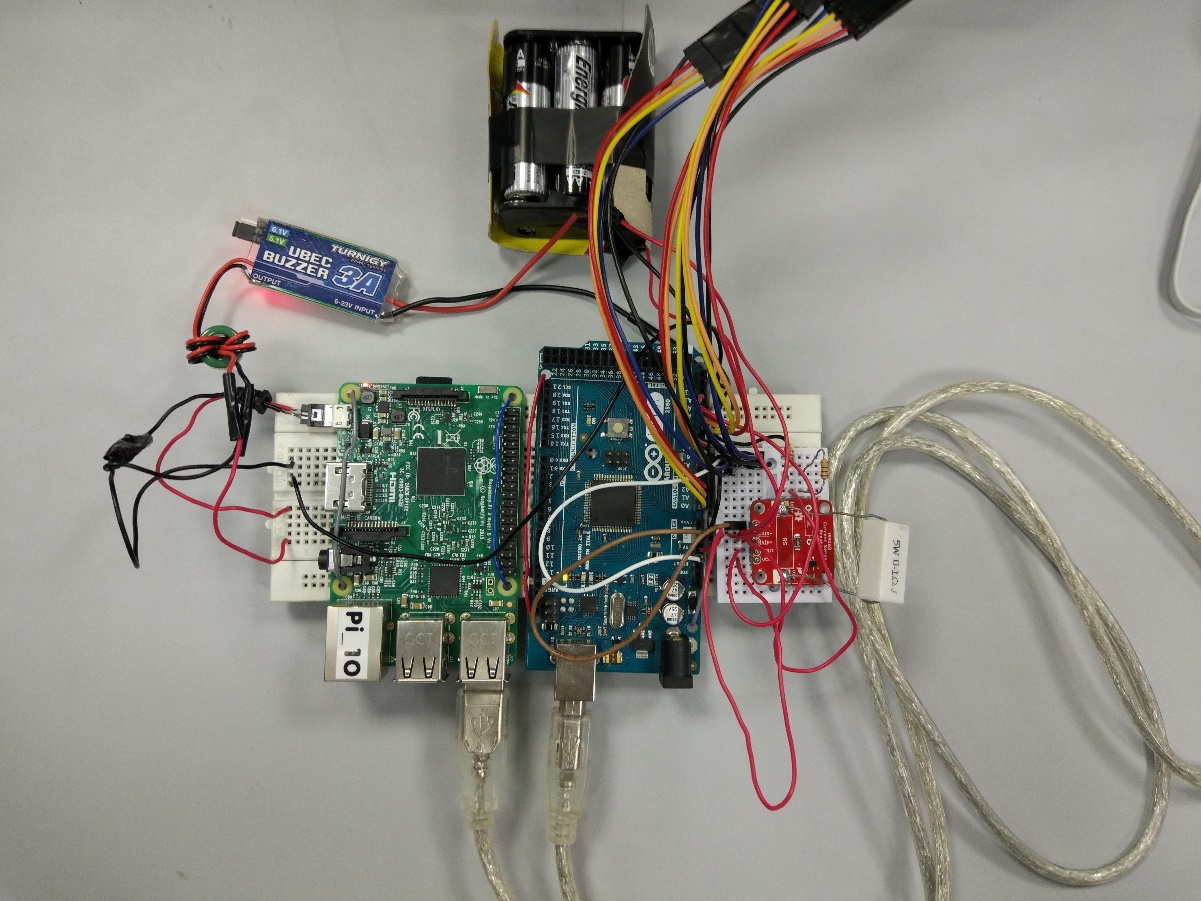
# Section 3: System Architecture

### 3.1 Overview

1. Three accelerometers strategically placed along the limbs measures the users movement
2. Raw data from the sensors will be read by the Arduino Mega and sent to Raspberry Pi
3. Raspberry Pi processes the data and a classifier is employed to determine the move
4. Move is sent wirelessly to the server



*Figure 2: Accelerometers placement on user*



*Figure 3: System*

## 3.2 Communication Protocols

**Sensors ⇄ Arduino Mega** Individual wires for each port

**Arduino Mega ⇄ Raspberry Pi**  Serial over USB

**Raspberry Pi ⇄ Server** TCP

**Server ⇄ Display** HDMI

## 3.3 Hardware layout

* Wearable pouch across the waist where the Raspberry Pi, Arduino Mega and power supply are housed in.
* Accelerometers are attached to both wrists and to the right knee via a wired connection and secured using velcro straps to the limbs.
* Additional velcro straps are used to secure the dangling wires to prevent them from interfering with the user’s movement.

# Section 4: Component Interactions and Design

## 

## 4.1 Main algorithm for activity detection

1. Raspberry Pi requests a new set of data from Arduino Mega
2. Accelerometers are polled and read at a constant interval of 10ms
3. 50 sets of data are accumulated, forming one sample
   1. 3 accelerometers \* 3 axis = 9 values per set
   2. 10ms \* 50 = 0.5s to read
4. The sample is sent over to the Raspberry Pi
5. Sample is compared using the Random Forests classifier to determine the current move in that 0.5s window
6. Determines which move over the past 5 samples are considered to produce a ‘best’ result
7. ‘Best’ result is sent to the server only if deemed confident

## 4.2 FreeRTOS in Arduino Mega

FreeRTOS can be imported as a library into our Arduino IDE which allows us to use tasks. How we leveraged tasks was that it allowed us to read the accelerometers consistently over a constant interval of 10ms.

## 4.3 Communication between Raspberry Pi and Arduino Mega

### 4.3.1 Communication Verification

When the Raspberry Pi first starts up, it will attempt a handshake with the Arduino Mega and will only proceed when successful. Raspberry Pi will constantly send 'H' and will only proceed when it receives an 'A' back which implies that Arduino Mega has received the 'H' and is ready to begin communication with Raspberry Pi. This is to ensure the systems are properly communicating and that they are both ready to receive and send data to one another. This same handshake is also used to reset the Arduino Mega variables to a fresh state i.e. (message IDs) if the Raspberry Pi is restarted.

### 4.3.2 Data Verification

In order to ensure that the data sent by the Arduino Mega and the data received by the Raspberry Pi is not corrupted, we XOR every data together to produce a checksum. This checksum value will be appended to the back of the data string to be sent. Upon receiving the data string. the checksum value is then recalculated on the Raspberry Pi and verifies that the data has not been corrupted. Furthermore, an ID is appended at the beginning of every data strings. This ID will be checked if its incremented to ensure that duplicate data is not being received. If the data has been corrupted, the Raspberry Pi will request for a resend of data. (Implemented but not used in final version)

### 4.3.3 Baudrate

We originally used a baud rate of 9600 which felt sluggish in one of our tests and attempted to increase it to improve performance. We experimented with a baud rate of 115200 and it had much faster performance but at the same time did not compromise our accuracy of data, therefore we decided to change to 115200.

## 4.4 Sample size and interval

Upon experimenting with different intervals and sample sizes, we concluded that a sample covering a duration of half a second is sufficient to capture a motion and simultaneously differentiate between different movements. The interval to be chosen had to have a balance between increased accuracy and longer processing time. We noted that the Arduino Mega was able to read a set of data as fast as 4ms (could probably go lower but didn’t experiment further) but would result in excessive data. Furthermore, reducing the interval for accuracy comes with diminishing returns and we have settled for 10ms which is suitable for our system.

## 4.5 Coordination between Raspberry Pi and Arduino Mega

When the Raspberry Pi requests for a new sample, the Arduino Mega reads the sensors for a period of time. As it takes 0.5s for the Arduino Mega to read (explained above), we have decided to request for a new sample immediately once the Raspberry Pi has read the previous sample. This eliminates the added latency of processing and classifying the data on the Raspberry Pi as it is running concurrently with the reading of a new sample on the Arduino Mega and that the time needed to process and classify the data is shorter than the time spent accumulating a new sample.

## 4.6 Data processing

For our project, only normalization of each raw data is done before being classified. Minimal data processing is done to the data to reduce latency and simply normalization is enough for the classifier to produce accurate results.

## 4.7 Classifier

In the early stages of the project, we’ve tested several classifiers, namely Support Vector Machines, Neural Network Model, K-Nearest Neighbors, AdaBoost and Random Forests against a similar format of data (multiple accelerometer readings per sample) found here:

<https://archive.ics.uci.edu/ml/datasets/Wearable+Computing%3A+Classification+of+Body+Postures+and+Movements+%28PUC-Rio%29>

Results from our benchmarks narrowed it down to K-Nearest Neighbors and Random Forests classifiers owing to both their speed and accuracy.

|  |
| --- |
| Results:  Support Vector Machines took 1950.988924 seconds |
| with an average accuracy of 82.140639% |
|  |
| Neural network model took 175.978997 seconds |
| with an average accuracy of 78.152904% |
|  |
| Nearest Neighbors took 34.353451 seconds |
| with an average accuracy of 99.276714% |
|  |
| AdaBoost took 163.459177 seconds |
| with an average accuracy of 73.236652% |
|  |
| Random Forest took 64.162376 seconds |
| with an average accuracy of 99.079289% |

Upon testing with our own raw data, both classifiers performed similarly well but Random Forests took the lead during live testing thus became our system’s classifier.

## 4.8 ‘Best’ answer

With a window of only 0.5s, the classifier may not be able to accurately classify every dance move and may occasionally give incorrect results. Hence, our system employs a confidence system to only submit the detected move to the server when the confidence level passes a set threshold. Tinkering with the values, we find that having a buffer of 5 most recent results from the classifier and a threshold of 3/5 (60%) of the same result is accurate enough with very minimal latency.

## 4.9 Communication between Raspberry Pi and server

To communicate between the RPi and the evaluation server, we would set up a data link via TCP/IP. The server program should be started and a socket is created specifying a particular port. It would wait for an incoming client. On the RPi, a client-side local TCP socket would be created, specifying the IP address and port number of the server process to bind to the server. Once created, the client TCP would establish a connection to the server TCP.

To ensure secure communication between RPi and the server, we would be using the Advanced Encryption Standard (AES) which uses a symmetric key algorithm meaning that the same secret key is used to both encrypt and decrypt the data. This standard is very secure as it is computationally infeasible to break the encryption using a brute force attack. We used AES 128 which requires a secret key that is 16 bytes long. The key we used was ‘3002300230023002’.

Before the results of the processed data, the detected dance move, is sent to the server by the RPi, the information would be encrypted using AES. It is then sent to the server where it can be decrypted using the same key used to encrypt it and the original information is then obtained and displayed.

# Section 5: Hardware Details

## 

## 5.1 Overview

Each accelerometer is connected to the Arduino to provide sensor readings. The Arduino is connected directly via USB to the Raspberry Pi to serially send the data from the sensors and to receive acknowledgement signals from the Pi. This also provides power to the Arduino. The Pi is powered by a pack of 6 1.5V batteries. The batteries are first connected to the INA169 Current Sense Breakout board which allows for current to be measured. Two resistors were used as a voltage divider to measure the voltage. A UBEC is then connected from the breakout board and resistors to provide a steady and safe power supply. The micro USB connector is connected to the outgoing wires of the UBEC. This is then connected to the Raspberry Pi to provide it with power.

## 5.2 Hardware Parts

1. Triple Axis Accelerometer Breakout - ADXL335
2. HR0245Wht 170-Point Breadboard – White
3. POL1904 2.54mm Crimp Connector Housing: 1x5-Pin
4. POL1900 2.54mm Crimp Connector Housing: 1x1-Pin
5. POL1901 2.54mm Crimp Connector Housing: 1x2-Pin
6. POL1902 2.54mm Crimp Connector Housing: 1x3-Pin
7. POL1801 Wires with Pre-crimped Terminals M-F 6-inches
8. POL1804 Wires with Pre-crimped Terminals M-F 12-inches
9. E CCH8803MUB Micro USB Type B Connector
10. ASK 6xAA Battery Holder(double layer)
11. 6x 1.5V AA Battery
12. HK Turnigy 3A UBEC with Low Voltage Buzzer
13. Xiaomi Mi Power Bank Gen 2 20000mAh
14. INA169 Current Sense Breakout Board
15. Waist Pouch
16. Velcro Straps

## 5.3 Reasons for selected hardware parts

### 5.3.1 ADXL335 vs MPU6050

While the MPU6050 includes a gyroscope, which may provide useful data in determining the user’s moves, we have decided to choose the barebones ADXL335 over it. This section will discuss the reasons for opting out of the provided sensors.

#### 5.3.1.1 Power usage

The ADXL335 uses 350 μA under typical conditions whereas the MPU6050 uses a whopping 3.6mA for the gyroscope and 500µA for the accelerometers under typical conditions. Voltages are similar for both sensors. While relative to the Raspberry Pi and the Arduino Mega, power usage of these sensors are somewhat negligible but is still a consideration.

#### 5.3.1.2 Sufficiency of Accelerometers

The accelerometer readings were good enough to provide the classifier sufficient data for an accurate result. Therefore, even without the use of gyroscope, we are able to come to an accurate detection of the dance moves.

#### 5.3.1.3 Interface

The MPU6050 runs on I2C serial interface which requires additional programming and hardware tweaks to allow for more than two sensors owing to the limitation of the Arduino Mega I/O whereas the ADXL335 provides analog readings, allowing for up to 5 sensors.  
As we intend for the system to use at least 3 sensors, this is a major factor.

As such, we have decided against the MPU6050 since the system can detect well enough without gyroscope readings and the amount of power the MPU6050 uses does not provide much incentive nor significantly boost our performance. Furthermore, the ease of being able to set up more than two sensors without going through additional implementations further reinforced our decision.

### 5.3.3 UBEC

A simple UBEC is required to ensure a steady and safe power supply to the systems. The additional low voltage buzzer is a useful add-on as it gives us and the user information that the battery is running low.

## 5.4 Circuit Diagram

*Figure 3: Circuit Diagram of Power Management System*

### 5.4.1 Reading of Voltage

Firstly, the voltage from the battery was split via voltage divider rule using two 110Ω resistors and passed into pin A15. The voltage was measured using the analogRead() function in the Arduino. Since the analogRead() function returns a value from 0 to 1023, where 1023 corresponds to 5V, we have to convert the received value into it's corresponding voltage reading. Furthermore, because of the voltage divider rule, the voltage being measured is only half of the actual voltage of the battery. Therefore, we will have to multiply it by 2 in order to attain the actual voltage value.

### 5.4.2 Reading of Current

The breakout board used was the INA169 Current Sense Breakout board. It consists of a shunt resistor, the INA169 chip, and an output resistor (RL). By default, the shunt resistor has a resistance of 10Ω and RL has a resistance of 10kΩ. An additional resistor of 0.1Ω was added in parallel to the shunt resistor. The resultant resistance (Rs) is calculated as (1/((1/0.1)+(1/10))) = 0.0990Ω. To calculate current (I), the following formula is used:

I = (Vout \* 1kΩ) / (Rs \* RL)

Vout is measured at the output of INA169 at pin A13 of the Arduino.

### 5.4.3 Calculation of Power and Cumulative Power

Power (P) can be calculated by using the formula: P = V \* I where V is the voltage measured at pin A15 and I is the current calculated as specified in Section 5.4.2. Cumulative power was calculated by multiplying the current power value by the time elapsed and adding it to the previous value of cumulative power.

In the final demonstration, we used the power bank to power the Raspberry Pi directly.

### 5.4.4 Pin Table



# Section 6: Reflection

## 6.1 Improvements

### 6.1.1 Calibration

Currently the system classifies the moves based on the raw uncalibrated data from the sensors. As each user has different body shapes and even on the same user, the user will wear the sensors at a slightly different angle each time. This results in inconsistent readings across each run. Implementing an initial calibration which will be run before each run can mitigate some of the issues by resetting all sensors to a new baseline when asking the user to stay still in a standing(resting) position. Furthermore, being reset to a new baseline allows for more distinct values between different moves as they currently range around 200~400 without resetting to a baseline. With a baseline of 0 set, the values can go negative/positive from 0 and have a clearer distinction for the classifier to work with.

### 6.1.2 Comfort & Aesthetics

While the current prototype is lightweight and acceptably comfortable, we could possibly obtain more comfortable and easier-to wear straps to replace the currently used velcro straps.

Wire management is also a major concern from both an aesthetics standpoint as well as sturdiness of the system. A proper mount and harder casing for the system (as opposed to a soft bag) would allow for better security of the devices inside the pouch. The system could also employ a clear case rather than an opaque one so that we will be able to observe if any problems should occur. Following which further adjustments of the length of the wires could be done in order to reduce the amount of excess wires.

### 6.1.3 Overheating

Being in an enclosed case, the batteries as well as the systems overheat over long periods of time. While it does not overheat to the point of failure or instability, comfort may be an issue. Basic ventilation is required to address this issue. Opening vents on the case and adding heatsinks to the systems should suffice to maintain a passive cooling solution.

## 6.2 Issues faced

An issue we faced were the lack of communication, since the progress of each sub-group were dependent on each other. There were times when progress became stagnant because time had to be spent debugging. Furthermore, we did not communicate properly what each sub-group needed from the other, for example, the rate at which data should be sent.

### 6.2.1 What could have been done instead

We could have communicated more and not work on our own sub-groups independently, since knowing what each other clearly wanted would have minimized the amount of changes we had to do in the future.

# Section 7: Societal Impact

## 7.1 Possible Future Benefits and Uses

The dance detector can be improved further by considering more types of movements and increasing the range of current detectable movements. Following these enhancements, .it would be possible to use the detector for other purposes, such as an exercise aid. The dance detector could also be integrated with a server which records the dance moves and, after processing the data, can update the user with useful information such as calories burnt. It could also be used as an easy way for experienced dancers to record down their choreography into teachable dance steps. Their exact moves and positions would be detected and used as a standard to follow when other dancers attempt to do the same dance. The system can be also extended for other uses such as sports where form and movement can be monitored and analysed to provide feedback to the user learning the sport.

## 7.2 Concerns

As is the case with any technology, there would be some issues regarding the detector as well. For instance, people may be concerned with the privacy of their personal movement data. However, we can assure the user with the use of advanced technology to improve privacy that could be implemented into the detector. One such technology is Network Access Control (NAC) which is an approach to computer security that attempts to unify endpoint security technology (such as antivirus, host intrusion prevention, and vulnerability assessment), user or system authentication and network security enforcement. Network Access Control is essentially a mechanism that allows access to network resources only to devices that are compliant with a specific security policy. Another issue that might arise may be the over-reliance on digital technology. There may be misinformation provided by the dance detector such as the wrong amount of calories burnt and it may affect the user's lifestyle. The detector isn’t meant to control their lives but rather serve just as a guideline and to supplement the user’s life. Therefore, there should be a disclaimer along with the sale of the dance detector.

# Section 8: References

https://exploreembedded.com/wiki/Setting\_Up\_FreeRTOS\_on\_Arduino

https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial

http://www.python-exemplary.com/index\_en.php?inhalt\_links=navigation\_en.inc.php&inhalt\_mitte=raspi/en/communication.inc.php

http://etutorials.org/Networking/802.11+security.+wi-fi+protected+access+and+802.11i/Appendixes/Appendix+A.+Overview+of+the+AES+Block+Cipher/Steps+in+the+AES+Encryption+Process/

http://www.inetdaemon.com/tutorials/internet/tcp/index.shtml

https://oscarliang.com/raspberry-pi-and-arduino-connected-serial-gpio/

https://stackoverflow.com/questions/22737838/raspberry-pi-server-client-socket-in-python

https://docs.python.org/2/howto/sockets.html

https://raspberrypi.stackexchange.com/questions/13425/server-and-client-between-pc-and-raspberry-pi

https://blog.algorithmia.com/machine-learning-with-human-in-the-loop/

<https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/>

<https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf>

<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>

https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide