CG3002 Embedded System Design Project
Semester 1 2018/2019

"Dance Dance"
Design Report

Group **05**	Name	Student #	Sub-Team	Specific Contribution
Member #1	AZIZI AZFAR B ZAHARI	A0155506H	Comms	Section 4, 1.3
Member #2	CALVIN TANTIO	A0160601X	Software	Section 5
Member #3	GUOK ENJIE STANLEY	A0155128H	Software	Section 1.2, 2.2, 5
Member #4	SARAH GOH SHI NING	A0158195U	Hardware	Section 2.1, 2.2, 3.2, 3.3, 3.4
Member #5	TANG WENXUAN GARY	A0139621H	Hardware	Section 1.1, 3.1, 3.5
Member #6	YEO KAI YAN	A0143641M	Comms	Section 2, 4

Section 1: System Functionalities

1.1. Use Cases

For all use cases below, the Raspberry Pi and Arduino setup, along with connected sensors, is known as *System*, the Dancer is the *User* and the Dance Server/PC is known as the *Server*, unless specified otherwise.

- Case 1: User dances requested known move correctly
  - MSS:
    1. Server requests dance move which is displayed on monitor
    2. User dances the requested move
    3. System detects dance move
    4. System sends results to server
    5. Server requests for a new random dance move

  Use case ends

  - Extensions:

    3.a. System does not detect dance move

    3.a.1 Use case resumes at step 2

- Case 2: User dances incorrect (different) requested known dance move
  - MSS:
    1. Server requests dance move which is displayed on monitor
    2. User dances a different known move
    3. System detects dance move
    4. System sends results to server
    5. Server requests for a new random dance move

  Use case ends

  - Extensions:

    3.a. System does not detect dance move

    3.a.1 Use case resumes at step 2

- Case 3: Dance unknown move
  - MSS:
    1. Server requests dance move which is displayed on monitor
    2. User dances an unknown move
    3. System does not send results to server
    4. Server continues to request for this dance move
    5. Server will indicate that a timeout has occurred after a 60 seconds wait from the time it requested the dance move
    6. Server requests for a new random move

  Use case ends

- Case 4: Idle
  - MSS:
    1. Server displays requested dance move
    2. User stays idle
    3. Server will indicate that a timeout has occurred after a 60 seconds from the time it requested the dance move
    4. Server requests for a new random move

  Use case ends

- Case 5: Final dance move
  - MSS
    1. User dances final move at any point in time
    2. System detects and determines the move is the final move
    3. System sends results to server
    4. Server indicates that user has danced the final move and terminates program.

  Use case ends

  - Extensions:

    2.a. System does not detect dance move

    2.a.1 Use case resumes at step 1

1.2. User Stories

- As a user, I want the device to detect if I am dancing the correct dance move so that I know I am correct.
- As a bad (not so good) dancer, I want the device to be able to detect my moves even with slight inaccuracies so that I won't fail all the moves.
- As a user, I want the device to detect if I am idle.
- As a user, I want the device to detect my final dance move so that I can stop dancing.
- As a user, I want the device to be comfortable so that I can wear it for long hours while dancing with it.
- As a user, I want the device as light as possible so that I can dance easily.
- As a user, I want the device to run as long as possible so that I do not need to charge it often.

1.3. Features

- System tracks hand and leg movements for game simulator
- System records hand and leg movements to detect dance moves
- System sends dance moves detected to server
- System detects its own power usage
- System can be used for more than an hour

Section 2: Overall System Architecture

2.1. High Level System Architecture

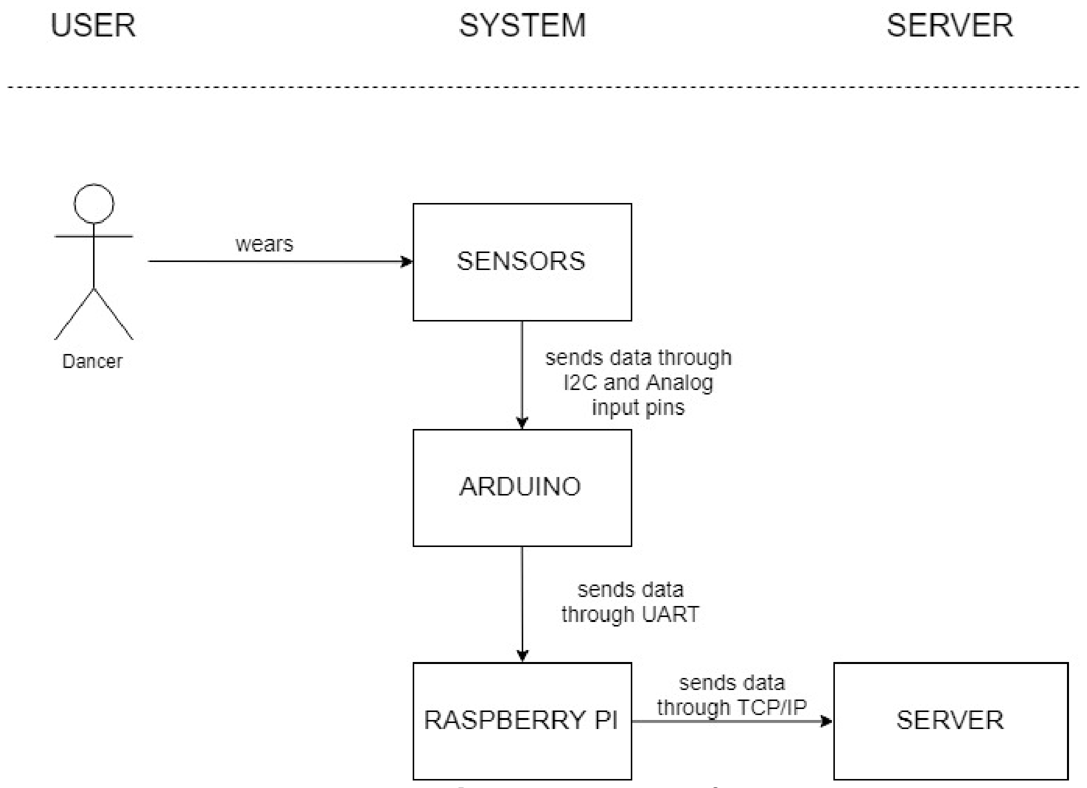


Figure 2.1. UML Deployment Diagram of System

a. Intended processes on Arduino Mega and RPi
   i. Arduino Mega
      1. Reads raw data from the sensor.
      2. Handle data according to send in the right format to Raspberry Pi.
      3. Sends data to Raspberry Pi.

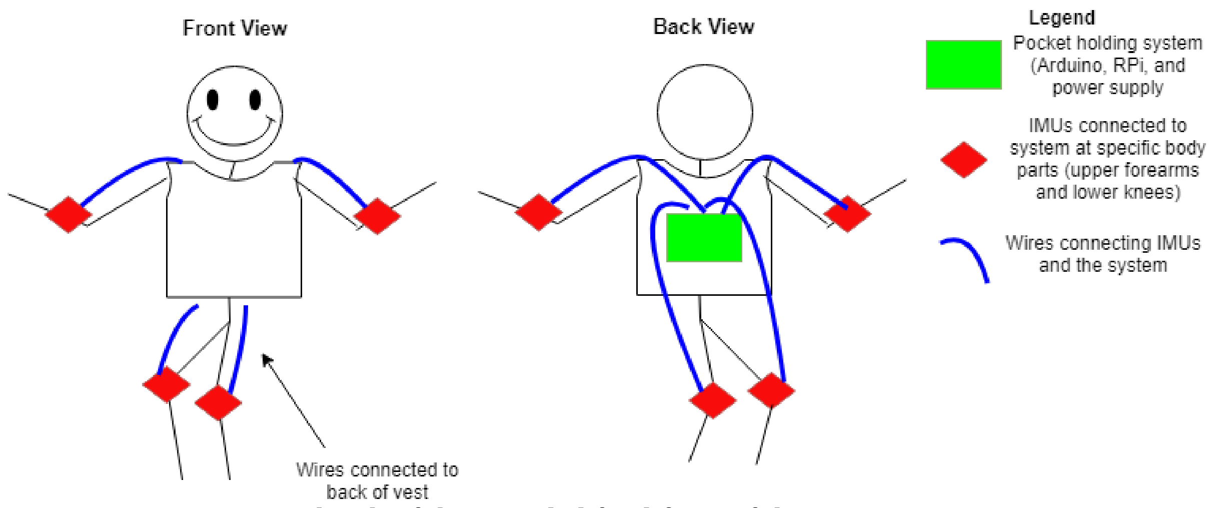
   ii. Raspberry Pi
      1. Receives data from Arduino Mega.
      2. Processes data through machine learning.
      3. Sends identified dance move to final server.

b. Communication between Arduino and RPi
   i. Arduino Mega to RPi
      1. We intend to use serial communications using UART with the help of the level shifter.

   ii. RPi to server
      1. Initially we will implement TCP/IP for socket communication and AES for secure communications between the RPi and the server.
      2. In the final system, we also intend to use wireless communication for communications between RPi and the final server using TCP/IP.

c. Interfacing hardware components with Arduino Mega and RPi
   i. 4 inertial measurement units to pass readings to Arduino Mega through I2C

ii. 1 current sensor connected between power supply and circuit
iii. 1 logic level shifter between Raspberry Pi and Arduino Mega for transferring data through UART


Front View
Back View
Legend
Pocket holding system (Arduino, RPi, and power supply
IMUs connected to system at specific body parts (upper forearms and lower knees)
Wires connecting IMUs and the system
Wires connected to back of vest
Figure 2.2. A sketch of the intended final form of the system

2.2. Main Algorithm of System

1. Sensors send readings to Arduino.
    a. Four GY-521 sensors will gather and send data through I^2C protocol to the Arduino. The INA169 sensor will send current sensor data through the Arduino’s analog input pins.

2. Arduino saves and processes sensor data.
    a. Arduino packages data to be sent to Raspberry Pi.
    b. Data will be sent in the format similar to an array of double values in the order [accelerometer x, y, z | gyrometer x, y, z].

3. Arduino sends data to Raspberry Pi.
    a. TXB1018 logic level shifter is used to change the voltage level from 5V to 3.3V between the Arduino and the RPi. 5V is the level connected to the Arduino and 3.3V is the level connected to the Raspberry Pi. This is necessary for the difference in voltages of the pins used in the two devices.
    b. Data packets will be serialised into byte streams for serial communication to the RPi. Data received is deserialised on the RPi side to obtain the data sent in structures.
    c. The two devices will follow a boot-up 3 way handshaking to indicate to each other they are ready.
    d. The serial communication between the two devices follows a communication protocol as detailed out in Section 4.

4. Raspberry Pi determines dance moves from data received by using machine learning models such as Support Vector Machine and Random Forest.
    a. Data will be used for both training the model and testing the model to produce accurate results. Data should also be validated (cross-validation).

5. Raspberry Pi calculates current power consumption and the accumulative power consumption.

6. Raspberry Pi sends dance move determined from model and power measurements to server.
    a. Results sent to server is in the format ‘#action | voltage | current | power | cumulative power’
    b. Raspberry Pi sends dance move and power measurements to server through TCP/IP.
    c. The wireless communication between the two devices are described in detail in Section 4.

Section 3: Hardware Details

3.1. Hardware Components / Devices

1. INA169 current sensor (http://www.ti.com/lit/ds/symlink/ina169.pdf)

- shunt resistor of 0.1ohms soldered onto INA169

Component	**INA169 High-Side Measurement Current Shunt Monitor**
Operating voltage	2.7V to 60V; nominal operating voltage: 5V
Operating current	Idle: 60µA; active (per pin): 10mA
Operational current range	350mA to 3.5A
Connection	Immediately after power supply

Table 3.1.1: Technical details of INA169 High-Side Measurement Current Shunt Monitor

2. GY-521 acceleration module (http://www.haoyuelectronics.com/Attachment/GY-521/GY-521-SCH.jpg)
   - with MPU6050 microprocessor on it (http://www.haoyuelectronics.com/Attachment/GY-521/mpu6050.pdf)

Component		**GY-521 MPU6050 3-Axis Acceleration Gyroscope 6DOF Module**
Operating voltage		2.375V to 3.46V
Accelerometer	Operating current	At low power: 10µA at 1.25Hz, 20µA at 5Hz, 60µA at 20Hz, 110µA at 40Hz; normal: 500µA
	Scale ranges (g)	±2, ±4, ±8 and ±16 (measured in m/sec^2)
Gyroscope range	Operating current	Standby: 5µA; active: 3.6mA
	Scale ranges (º/sec)	±250, ±500, ±1000, and ±2000 (angular velocity)
Operating current when both components are in use		Without Digital Motion Processing (DSP): 3.8mA; with DSP: 3.9mA
Connection		On Arduino Mega

Table 3.1.2: Technical details of GY-521 MPU6050 3-Axis Acceleration Gyroscope 6DOF Module

3. Arduino Mega2560 (https://store.arduino.cc/usa/arduino-mega-2560-rev3)

Component	**Arduino Mega 2560**

Operating voltage	5V
Operating current	Maximum: 800mA
Connection	Parallel to Raspberry Pi 3 and in series with Logic Level Shifter and 4x GY-521 modules

Table 3.1.3: Technical details of Arduino Mega 2560

4. Raspberry Pi 3 Model B (https://docs-emea.rs-online.com/webdocs/165e/0900766b8165e389.pdf)

Component	**Raspberry Pi 3 Model B**
Operating voltage	5V
Operating current	Recommended: 2.5A, measured under full CPU load: 0.5A
Connection	Parallel to Arduino Mega and in series with Logic Level Shifter

Table 3.1.4: Technical details of Raspberry Pi 3 Model B

5. TXB0108 Logic Level Shifter (http://www.ti.com/lit/ds/symlink/txb0108.pdf)

Component		**TXB0108 8-Bit Bidirectional Voltage-Level Translator with Auto-Direction Sensing and ±15-kV ESD Protection**
Operating voltage	Absolute maximum	V_{CCA}: -0.5V to 4.6V; V_{CCB}: -0.5V to 6.5V
	Typical	V_{CCA}: 1.2V to 3.6V; V_{CCB}: 1.65V to 5.5V
Operating current		±50 mA
Connection		Parallel to Arduino Mega and in series with Logic Level Shifter

Table 3.1.5: Technical details of TXB0108 8-Bit Bidirectional Voltage-Level Translator with Auto-Direction Sensing and ±15-kV ESD Protection

3.2. Pin Connections

1. Raspberry Pi to Arduino connected with a **logic level shifter** in the middle for transferring data through UART protocol

a. Raspberry Pi to logic level shifter

Raspberry Pi	**TXB0108 Logic level shifter**
3V3	LV
GPIO15 UART0_RXD	LV3
GPIO14 UART0_TXD	LV4

Table 3.2.1.1: Pins used to connect the Raspberry Pi and the TXB0108 Logic level shifter

b. Arduino to logic level shifter

Arduino	**TXB0108 Logic level shifter**
5V	HV
D14/TX3	HV3
D15/RX3	HV4

Table 3.2.1.2: Pins used to connect the Arduino Mega and the TXB0108 Logic level shifter

2. Arduino and sensors
    a. Arduino with current sensor (INA169)

Arduino	**INA169 Current Sensor**
V_{IN}	V_{IN-}
A_0	V_{OUT}

Table 3.2.2.1: Pins used to connect the Arduino Mega and the INA169 Current Sensor

b. Arduino with IMU (GY-521) via I^{2}C protocol

Arduino	**GY-521 IMU**
5V	V_{CC}
SCL	SCL
SDA	SDA

Table 3.2.2.2: Pins used to connect the Arduino Mega and the GY-521 IMU

c. Raspberry Pi with current sensor (INA169)

Raspberry Pi	**INA169 Current Sensor**
5V	V_{IN-}

Table 3.2.2.3: Pins used to connect the Raspberry Pi and the INA169 Current Sensor

4. Raspberry Pi and Arduino will be connected to the same point as ground for all these components.
5. Power supply will be connected to the current sensor, which will then be connected to Arduino and Raspberry Pi which are in parallel.

3.3. Schematics

The power supply we use will first be connected directly to the INA169 current sensor, which will send the data to Arduino Mega through an Analog input pin, connected by V_{OUT}.

1. The current will continue flowing out to supply power to the Arduino Mega and Raspberry Pi in parallel from the V_{IN} pin.
2. The Arduino Mega2560 then powers the 4 GY-521 sensors arranged in parallel through its 5V output. These sensors will send data to the Arduino Mega through the I^2C protocol where data is sent through the SDA and SCL buses.
3. The Arduino Mega2560 sends data to Raspberry Pi 3 through the logic level shifter using UART protocol.
4. All components are also connected to the same point considered ground.

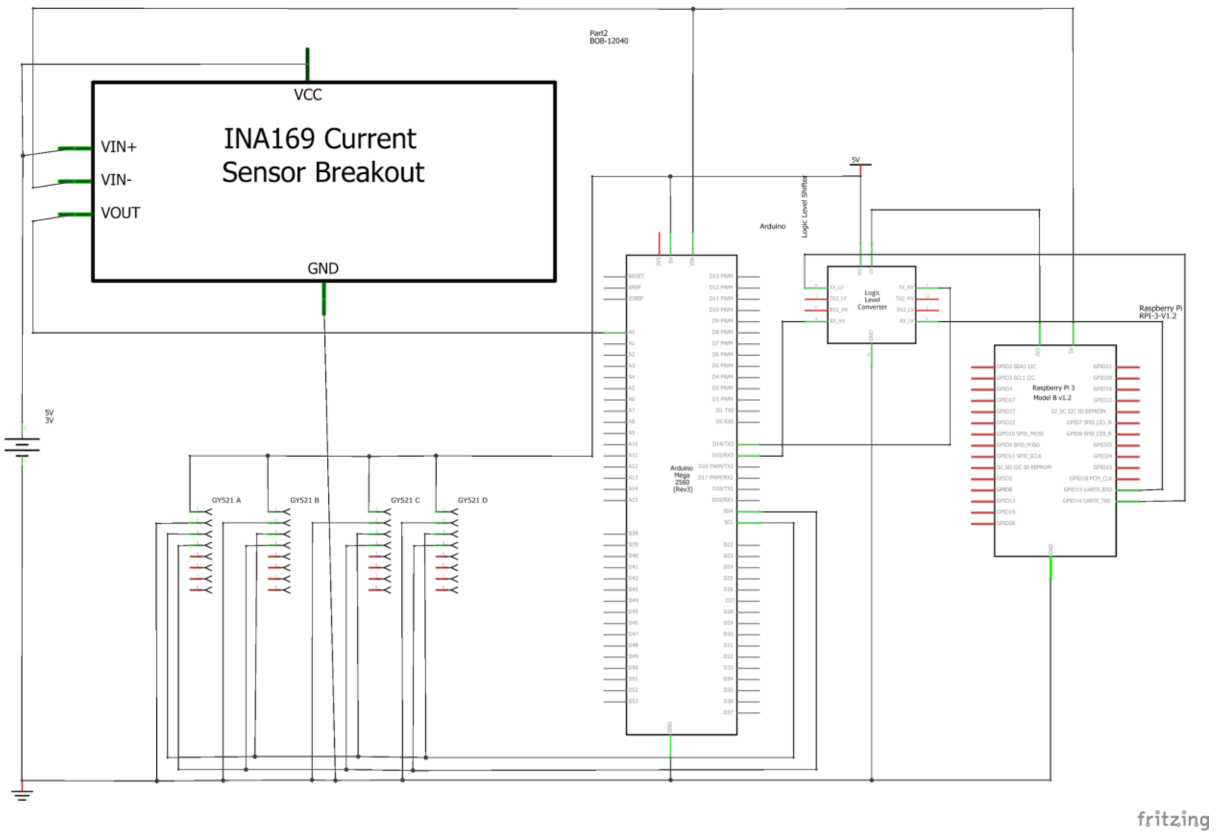


Figure 3.3.3.1: Schematic diagram of system and all components used in it

3.4. Interfacing Hardware

Some idea of the algorithms / libraries we would be using to get these hardware to work.

1. For GY-521 with MPU6050 microprocessor
   - On Arduino: use I2Cdev.h and MPU6050.h (https://github.com/jrowberg/i2cdevlib) (More information available at

https://www.i2cdevlib.com/devices/mpu6050#source and https://playground.arduino.cc/Main/MPU-6050)

3.5. Power Consumption

1. Subsystems and their power usage
   a. Current sensor (INA169)
      - R_S = 0.1 ohm
        Since current has a maximum value of 3.5A when R_S = 0.1ohm,
        P_{OUT} = I^2 R = 3.5 x 3.5 x 0.1 = 1.225W
   b. Accelerometers (GY-521) in parallel arrangement
      - Operating current provided by datasheet = 3.8mA per sensor
        Voltage provided by Arduino is 5V
        Total current through 4 sensors = 3.8 x 4 = 15.2mA
        ∴ P_{IN} = 15.2 x 5 = 50.16mW = 76mW = 0.076W
   c. Arduino Mega 2560
      - Maximum current = 800mA
        Operating Voltage = 5V
        ∴ P_{IN} = 800 x 5 = 4000mW = 4W
   d. Raspberry Pi 3 B
      - Maximum current draw when Raspberry Pi at 100% CPU usage = 0.5A
        Voltage across Raspberry Pi = 5V
        ∴ P_{IN} = 0.5 x 5 = 2.5W
   e. Assuming 100% power efficiency of components,
      - Arduino and GY-521 are arranged in series
        → P_{IN} = P_{OUT} = 0.076 + 4 = 4.076W
      - Arduino with GY-521 and Raspberry Pi are arranged in parallel
        → P_{IN} = 4.076 + 4 = 8.076W
      - INA169 current sensor is arranged in series with the rest of the circuit
        → P_{IN} = P_{OUT} = 1.225 + 8.076 = 9.301W
   f. The system's power consumption is calculated to be **9.301W**.
   g. Assuming our system runs for 2 hours (more than 1 hour),
      Energy required = 9.301W x 2h = 18602mAh
   h. Therefore, our total energy required over a two-hour period is **19000mAh** after rounding up.

2. Battery Design
   - Based on initial projected power consumption values, we will be using a standard 5V 1A power bank with a custom circuit (most probably settled via a DIY breadboard) and cables to connect the components together.

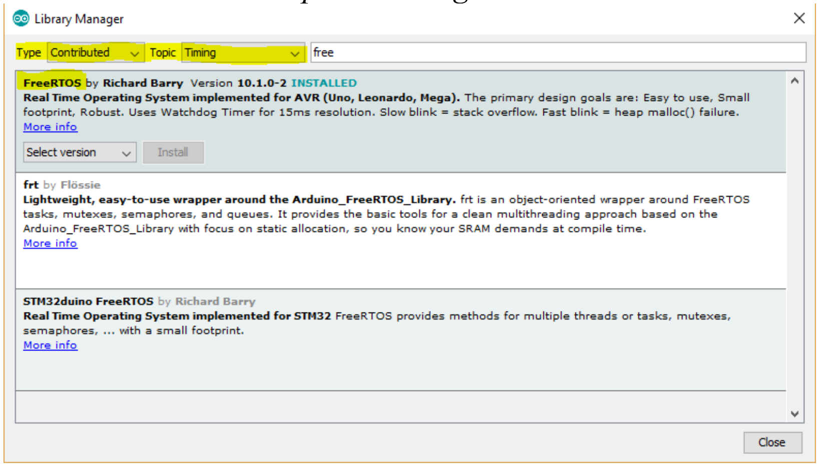
3. Optimising power consumption at runtime
   - Lowering the baud rate will allow the Arduino and Raspberry Pi to communicate less and hence require less power during communication.
   - Minimising the CPU usage on the Raspberry Pi with less intensive algorithms will also help lower the power consumption by it.

Section 4: Firmware & Communications Details

4.1. Using FreeRTOS

4.1.1. Porting FreeRTOS for Arduino Mega

1. Referencing online tutorial on Arduino website, look in the Arduino Libraries marketplace (*Sketch > Include Library > Manage Libraries...*) for the FreeRTOS library under *Type: "Contributed"* and *Topic: "Timing"*.



FreeRTOS by **Richard Barry** Version **10.1.0-2** INSTALLED
Real Time Operating System implemented for AVR (Uno, Leonardo, Mega). The primary design goals are: Easy to use, Small footprint, Robust. Uses Watchdog Timer for 15ms resolution. Slow blink = stack overflow. Fast blink = heap malloc() failure.
More info

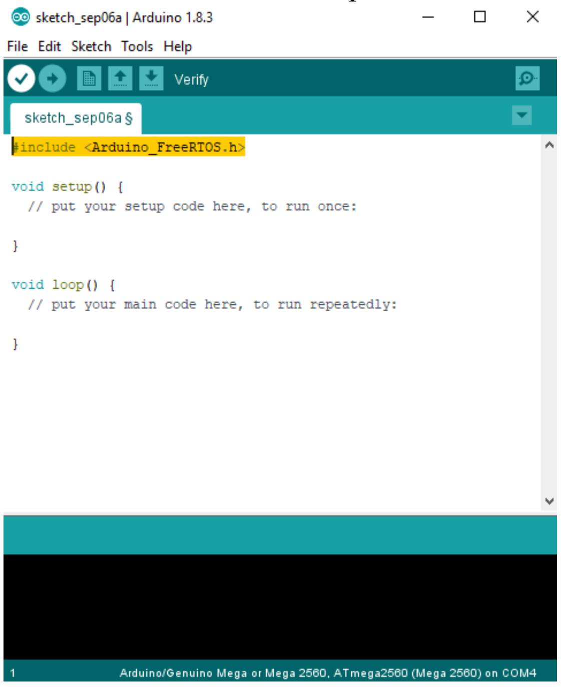
frt by Flössie
Lightweight, easy-to-use wrapper around the Arduino_FreeRTOS_Library. frt is an object-oriented wrapper around FreeRTOS tasks, mutexes, semaphores, and queues. It provides the basic tools for a clean multithreading approach based on the Arduino_FreeRTOS_Library with focus on static allocation, so you know your SRAM demands at compile time.
More info

STM32duino FreeRTOS by Richard Barry
Real Time Operating System implemented for STM32 FreeRTOS provides methods for multiple threads or tasks, mutexes, semaphores, ... with a small footprint.
More info

Figure 4.1.1.1.: Installing FreeRTOS from Arduino Libraries Marketplace

2. Click install to install the library for use in Arduino IDE.

3. After installation, to use the FreeRTOS library, go to Sketch > Include Library and select FreeRTOS to include it for our current sketch. If the library is included, the sketch should have *#include <Arduino_FreeRTOS.h>* at the top of the sketch, such as below:

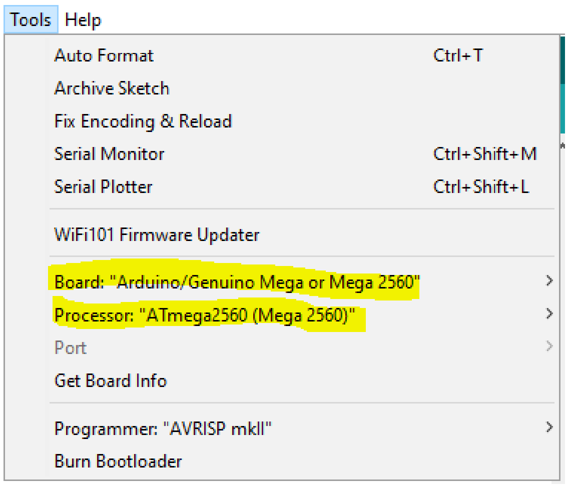


```
#include <Arduino_FreeRTOS.h>

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

1 Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM4

Figure 4.1.1.2.: Arduino IDE with FreeRTOS Library

4. Ensure that our Arduino Mega is connected correctly by checking that *Arduino/Genuino Mega or Mega2560* is selected under *Tools > Board* and that *ATmega2560 (Mega 2560)* is selected under *Tools Processor*, as below:



Board: "Arduino/Genuino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Programmer: "AVRISP mkII"

Figure 4.1.1.3.: Checking if the Arduino Mega is connected correctly to the PC

5. FreeRTOS has been installed and is ready for use on our Arduino Mega.

4.1.2. Intended Usage

Using the FreeRTOS with the Arduino Mega, we intend to leverage on the real time scheduling abilities of FreeRTOS, priority assignments, and synchronisation mechanisms available with the library

- Real time scheduling to achieve a predictable (known as deterministic) execution pattern. Because we are detecting dance moves with different actions happening at the same time, we need to know when dance actions are executed one frame after the other in order to determine the right move executed by the dancer. Without this, we will not be able to differentiate well the movements executed.
- Priority assignments to help processor determine what threads to run based on priority. This helps us collect the right data in the correct sequences by running the threads in order of priority.
- Synchronisation mechanisms to prevent race conditions by allowing data access one thread at a time. This ensures reliable and accurate data when collected.

4.2. Processes on Arduino and RPi

The following are the processes that we intend to create, with a discussion of the priority of each process at the end.

The **Arduino** will do the following:

- Read sensor data from the 4 IMUs.
- Read current and voltage for power measurement.
- Transmit through UART these data after they have been organised into data packets and ready for sending.

The **RPi** will do the following:

- Receive data packets from Arduino through UART. Request for retransmission if necessary.
- Use the data received for training of the algorithm and for detection of dance move.
- Calculations for power measurement. Store the result.

- Send results to server through secure communications

We decided on a **periodic push** by the Arduino Mega to the RPi, with the following reasons:

- Time is required for data to be collected. The Arduino Mega will need some memory space for data to be stored while the full packet is being collected and getting ready to be sent.
- Since all data is essential to ensuring the right training by the machine learning algorithm, there may be inaccuracies in determining dance moves. The Arduino Mega has a much smaller memory compared to the RPi, making it difficult to store too much data. With a larger memory on the RPi, storing data, even in buffers (before the data is used), will then be easier. Less data is likely to be lost due to lack of memory space for storage.
- Because the system wants to detect dance moves at the current time and to determine dance moves quickly, the oldest data packets received will be overwritten by the new data should the RPi memory run out of space in order to ensure the most updated data is available for a quick detection of dance moves.
    - This is based on the assumption that the oldest data may be less relevant by the time there is a buffer overflow.

4.3. Arduino Tasks

4.3.1. Obtaining Sensor Data

- 4 IMUs to detect the movement at each arm and leg
- General steps:
    - Read values from each sensor through I^2C (based on hardware devices communication protocol)
    - Values read are organised into data packets to be sent to the Raspberry Pi
    - Each data packet contains the data values from all 4 sensors (tentatively like in the format of an array of double values in the order of [accelerometer x, y, z | gyrometer x, y, z]), size and the checksum that is to be received by the Raspberry Pi.

The following are some cases whereby the system detects an error:

- <u>Case 1</u>: one of the sensors are not providing data (due to malfunction or corrupted readings)
- <u>Case 2</u>: no full data packet is available after more than 30 seconds (all sensors are providing data at some point)
    - Error detected by the system and flagged. Actions must be taken by us to reduce such errors, ideally to zero.

4.3.2. UART Sending Data

- Interrupts will be used whenever a full data packet is ready to be sent to the RPi.
- Sensors are read periodically (< 1 second) in order to find a balance between detecting dance movements (data collection), ensuring enough space for data collection and reducing power consumption. Hence, while not obtaining data, operations carried out on the Arduino are kept to a minimum.
- The general steps of sending data from the Arduino to the RPi:
    - The sensors are read periodically and data is first saved in data structures on the Arduino. When a full structure is obtained, the data will be prepared for sending by framing the data with heading, putting the data in appropriate formats, and adding checksum.
    - Once the data packet is ready for sending, the data structure is serialised before the data packet is pushed to the RPi. This step is necessary in order to send the data structure as a stream of bytes through UART. This is to provide extra reliability on the relatively huge amount of data collected from the 4 IMUs, as communication errors may result in data corruption or lost data.
    - In order to ensure that complete data is collected for use, once a complete data packet is ready for use, it will be serialized to be sent to the RPi through UART.
- Data continues to be read periodically while Arduino waits for a response from the RPi. If the RPi sends an ACK, the data packet can be discarded as the RPi has received the correct data. If the RPi sends a NAK, the data packet has to be sent again.
- With all the above considerations, the Arduino will use a small part of its memory for saving data (data structures) while building the data packets to be sent. The Arduino memory should be enough for this purpose, so that data will not be lost.

4.3.3. UART Receive

- Mainly for communication with RPi. To know whether to retransmit data because of a NAK bit, or for an indication to discard latest sent message because of an ACK bit (RPi received correct data).

* Message queues (FIFO data structure) to be used for UART communication tasks in order to preserve the order of which data is collected to ensure a more reliable sequence of dance move detected.

4.3.4. Power data collection

- Data collection from current sensor and voltage readings in order to calculate power consumption. Least crucial of all tasks, considering the objective of the system is to detect dance moves and send results to server.

4.3.5. Priorities

Task	Priority (lower number = higher priority)	Reasoning
Obtaining Sensor Data	2	Data collection is necessary for the system to work. However, because data is needed by the RPi for dance moves to be determined, this task is second in priority as compared to UART communication.We need to ensure that the data is collected regularly such that there is no loss of information.
UART Send (Organising Data into Packets)	1	Collected data is crucial to the functioning of the entire system as the machine learning algorithm needs these data to work on the RPi and determine dance moves. Hence communication with the RPi takes highest priority.The data has to be organised such that we are able to send it properly to the RPi through UART.
UART Receive (Sending Packets to RPi)	1	
Power Data Collection	4	Main objective of system is to detect dance moves and send results to server. Power data collection is secondary.

Table 4.3.5: Arduino Mega's priority ranking table

4.4. RPi Tasks

* Buffers to be maintained on the RPi of the data received from the Arduino. Data will be retrieved from the buffers when the machine learning algorithm requires them.

4.4.1. Running of Algorithm

- Machine learning model obtains the received data packets from the RPi buffers whenever ready.
- The data set is used for determining a dance move.
- Determined dance move sent to the evaluation server.

4.4.2. UART Receiving Data

The general steps to receiving data from Arduino through UART:

- Receives data packets from Arduino by reading the UART ports. Deserialize the data packets received to convert the data byte stream back to structures.
- Check whether data has any errors (checksum matches). If there are errors, send back NAK to request for the data again. If not, send an ACK to let the arduino the data packet has been successfully received.
- Stores the deserialized data packet received in the buffer to be used later.

The machine learning algorithm needs time to run before obtaining the next set of data for its dance detection. Hence, although a periodic push of data is done, the machine learning algorithm will only obtain the latest amount of data when it needs to.

4.4.3. UART Sending Data

- Mainly for communication with Arduino, according to communication protocol.

4.4.4. Server Communication

- After dance move is determined and power calculated, an interrupt is generated to trigger this task.
- The results are put together in the format '#action | voltage | current | power | cumulative power' and then sent to the evaluation server through secure communications. More details on server communications below.
- Since the server is only involved in a one way communication from the RPi to the server, this task should only run when necessary, hence implemented as an interrupt.

4.4.5. Power Calculations

- Calculate power consumption with the current and voltage data: P = I x V
- Power consumption to be tracked for ensuring a stable and efficient system. Accumulative power is also tracked and sent as part of the result string to the evaluation server.

4.4.6. Priorities

Task	**Priority (lower number = higher priority)**	**Reasoning**
Server Communication	1	It is critical for the system to be able to send its detection result to the evaluation server. Without any result sent to the server, the system is deemed to have failed. Hence, when the result string is ready, this task must be triggered with the highest priority.
Running of Algorithm	2	The main objective of the system is to detect the dance moves using the machine learning algorithm. Hence, with enough data, the algorithm must have the greatest priority to run, as long as there is no server communication..
UART Receiving Data	3	Data is crucial for the algorithm to determine the dance moves. Hence, data transfer has be of greater priority than power calculation but lower priority than the algorithm itself.
UART Sending Data	3	
Power Calculation	4	Main objective is to detect dance moves and send results to server. Power calculation is secondary.

Table 4.4.6: Raspberry Pi's priority ranking table

4.5. Synchronisation and Communication between RPi and Arduino

We intend to use mutexes to synchronize the processes both on the Arduino and on the RPi. To prevent race conditions on the message queue on the Arduino and the buffers on the RPi, mutexes will protect these shared resources on each device.

4.5.1. Adhering to Voltage Difference

The RPi works with 3.3V pins, while the Arduino Mega typically works with 5V pins. In order to interface the two devices, we use the level shifter [TXB0108] to shift from 5V to 3.3V. This will allow the Arduino Mega to communicate with the RPi using the UART ports, in the right voltage ranges. The connection of the two boards are as below:

5V pin from Mega to **HV**.
3.3V reference voltage (from Mega or Rpi) to **LV**.
HV3,HV4 to Mega (Connected to digital **pins 19**, Rx and **18**, Tx)
LV3, LV4 to RPi (Connected to **pins 8**, Tx, and **10**, Rx)

We had the option to use a USB connection instead, but we decided against it as it may lead to power management issues since by using a USB connection, the Arduino will be drawing its power directly from the RPi instead.

4.5.2. Communication Protocol

The following settings will be used:

USART Port used on Arduino	USART3 will be used. USART0 is avoided to prevent data corruption when sending data through USB from our devices (laptop), especially during testing and initial development.
UART Port used on RPi	RPi only has one UART port. Will be used for serial communication with Arduino.
Bit Level Protocol Settings	Baud Rate: 9600 Data Length: 8 (bits) # of Parity Bits: None # of Stop Bits: 1 More details below, along with reasoning.

Table 4.5.2: Settings used for communication between components

4.5.3. Handshaking

Referencing Prof Peh's Comms lecture, a bootup 3-way handshake will be implemented for both the Arduino and RPi to know they are both ready for sending and receiving data.

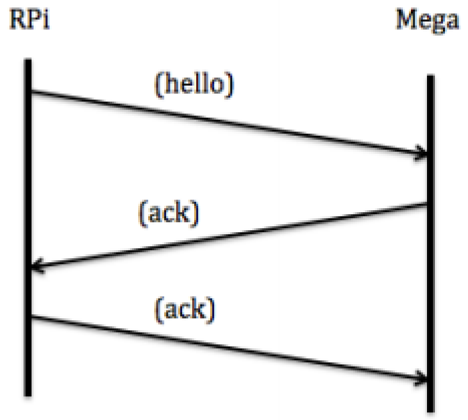


Figure 4.5.3.1: Bootup 3-Way Handshake between Arduino and RPi

4.5.4. Packet Types

These are some of the packet types that will be used. As we go into greater detail when implementing our communication protocol, these will be modified.

Packet Type	**Purpose**
ACK	● Acknowledge correct data sent. ● Signals device is ready in bootup handshake sequence.
NAK	● Incorrect/problematic data received. ● Signals device is not ready at bootup handshake sequence.
Bootup Hello	On each device, to signal bootup ready and requests if the other device is ready.
Data Packet	To signal that the incoming packet is a data packet.

Table 4.5.4: Packet Types Variants and their respective purposes

4.5.5. Packet Format

UART communication serves as the first layer of communication between the Arduino and RPi. The two devices will have the above bit protocol settings for the following reasons.

<u>Baud Rate</u>
We agreed on using a baud rate of 9600. This is in consideration of the possible differences between data sending/receiving speeds on the Arduino and the RPi. If the baud rate is too high, data sending may not be as reliable. If the baud rate is too low, we may end up having too much data waiting on the Arduino to be sent, possibly causing memory problems. Hence, we will use the standard baud rate of 9600. We may change it to something faster in the future if the need arises.

<u>Data Length</u>

We will use an 8 bit data length, as this is the standard and should be enough for us to send each bit we are using. As data we are collecting and sending comes in the form of digits and possibly some characters (eg. labelling data as x, y, z to indicate the accelerometer axes), 8 bits should be sufficient.

Parity Bits
Although parity bits are used as a simple form of low level error checking, it can slow down data transfer and extra work has to be done for error handling on both sender and receiver side. Hence we are opting for a checksum at the end of the data packet instead.

Stop Bits
1 stop bit should be sufficient for our use to indicate the end of data sending. Using 1 stop bit will allow our data to be passed more efficiently as well, since there will be less bits sending non-data information.

4.5.6. Communication between RPi and Server

To communicate between our RPi and the evaluation server, we intend to use a data link via TCP/IP. TCP is reliable and byte-stream oriented, suitable for our use. When the server program runs, a socket will be created. It will wait for an incoming client (our RPi) to communicate with through the port we assigned it to (on RPi side).

On the RPi, a client-side local TCP socket would be created. Here, we will specify the IP address and port number of the server process to bind to the server. Once created, the client TCP will establish a connection to the server TCP.

Message to be sent to the server will be of this format: '#action | voltage | current | power | cumulative power'. This is also the format expected on the server side.

4.5.7. Secure Communication

To ensure secure communication between RPi and the server, we would be using the Advanced Encryption Standard (AES) which uses an identical secret key (provided by us) for both encryption (on client side) and decryption (on server side). Authentication is achieved by the checking of matching keys on the client and server side. All data will be encrypted before being sent to the server and decrypted when received on the server. This ensures that our communications are secure and third parties won't be able to interfere with our communications with the evaluation server.

We will use the Cryptodome Cipher library available in python to implement the AES scheme and to achieve the above security in communications with the server. This library provides packages to enable us to verify authenticity (Crypto.Signature), encrypt and decrypt data (Crypto.Cipher), as well as facilitate secure communications between the RPi and server (Crypto.Protocol).

Section 5: Software Details

5.1. Preprocessing

5.1.1. Segmenting the Sensed Data

During the real-time detection of the dance moves, we will require 100 sensors readings per second based on guesstimation. This, we estimate, will be able to capture around 2° change per second. We try to avoid too many data inputs per second as it can drain power faster[1] and increase signal noise.

Our guesstimation is justified by the fact that Fitbit fitness watches' accelerometers also have a default sampling rate of 100 Hz (100 samples per second).[2]

The data will be segmented into frames of 300 to 500 sensor readings (3 to 5 seconds for each frame). This is, again, a guesstimation on the average duration for all the dance moves.

We will have a 50% overlapping frames, which means that 50% the $(k + 1)^{th}$ frame data will be taken from the last 50% of the data from the k^{th} frame; the other 50% of the $(k + 1)^{th}$ frame data will be new data from the sensors. We choose to use overlapping frames instead of the non-overlapping ones as it is difficult to determine the transition between one dance move to another. We believe that 50% overlap (1.5 seconds to 2.5 seconds) is the time required for the user to transition between dance moves.

There will not be any detection of the start of a movement (e.g. when there is a sudden spike in the sensor readings). The overlapping nature of the frames will be able to take into account the unpredictable starting time of a dance move. The system will continue to sample data frame by frame once it is started until it is eventually stopped.

5.1.2. Features

For the time being, we will take into account every values that are sent by each sensor, which are the three axes of rotation (gyroscope) and the three axes of acceleration (accelerometer). In total we will be considering 24 features (4 sensors (2 arms, 2 legs) with 6 values each). As the project progresses, we may be discarding some of the features deemed not important in classifying dance moves.

As each sensor data frame (refer to section 5.1.1.) corresponds to 1 data point in the 24 dimensional space, we need a way to flatten the 300 to 500 sensor readings to one 24 dimensional vector. Note that each sensor reading is a 24 dimensional vector.

We will be considering 3 different methods to flatten the data within each frame:

1. Average value analysis
   The idea is to take the average values of each of the 24 features. We will most probably try this method first as this is considered the simplest way of flattening the data. We would like to see whether the average feature values of each dance move are good enough to help us classify the dance move itself.

[1] https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/23353/Jiang.pdf?sequence=1
[2] https://dev.fitbit.com/build/guides/sensors/accelerometer/

A possible reason why this method might not work is that this method may fail to capture the unique characteristics of each dance move, causing the feature values from various dance moves to be similar to one another. For instance, a back and forth movement characteristic may not be able to be captured by the average of their accelerations, which may result to be around 0 (positive acceleration for forward movement is cancelled out by the negative acceleration for the backward movement).

2. Peak value analysis
   The idea is to detect the change in movement in all directions (through accelerations) and rotations. A peak will indicate a change in either direction or rotation.

   In this case, the detection of highest and lowest values of each feature may not be very crucial as they are merely an indication of how fast / slow a change is happening. We believe that it will be more fruitful to count the number of peaks of each features within a given frame. While we do not discriminate between positive and negative peaks, it is intuitive that each peak indicates a change of movement in the opposite direction.

   For this analysis to work properly, we need to filter out the noise. In this way, we will only consider peaks caused by an actual change in directions.

3. Energy frequency band
   Since the data recorded is in time domain, we need to transform the data to frequency domain using fast fourier transform. Since there is movement when dancing, we can track the increase or decrease in energy level for a particular frequency band.

5.1.3. Normalisation of Data

There may not be a need to normalise the sensor data as both the gyroscope and accelerometer data is converted to a 16-bit digital signal by the sensor (lowest value of -32,768 and highest value of 32,767). This in itself can be considered normalisation.

In the case where normalisation is necessary, it can be done by dividing all the values of each feature by the largest value of that feature. The largest value will not be based on the data collected but rather by the largest values stated in the sensor datasheet.

The values used for normalisation will be used to normalise the testing dataset.

5.1.4. Separation of Data into Training, Validation and Testing Sets

We will be partitioning the data as follows:

- 90% training data
- 10% test data

As we will be using 10-Fold Cross-Validation to check the validity of our data, 10% of the training data will be used as the validation data. The validation process will be repeated 10 times. Each time with different validation data sets from different folds of the training data. The validation data in each fold will also be stratified, meaning each fold will have the same proportion of observations for a given dance move.

We aim to have a Mean Squared Error (MSE) value of less than 0.1 before proceeding to generate the model using 100% of the training data. In case of a large MSE value, we can do one of the following:

- Generate more data and redo the partitioning, training and validation process
- Plot the data points and remove the outliers / remove outliers through table observation (a little bit hard to do for a 24 dimensional space and large dataset)
- Remove one / some feature(s) and redo the training and validation process

We can also remove outliers before data partitioning is done.

5.2. Machine Learning Models

5.2.1. Random Forest

A random forest is a collection of decision trees. As its name suggest, a decision tree is a tree data structure consisting of nodes and edges. It has a root node, which is the starting point of a decision tree. All the nodes in the tree, except the leaf nodes, represent questions / feature value checks. The edges, on the other hand, represent the answers to those questions. The decision tree will be traversed all the way from the root to one of the leaf nodes based on the answers given to the nodes. This means that all parent node must have at least 2 child nodes. In the case of multiclass classification problem, like the one we are dealing with in this project, each leaf node of the decision tree represents a unique class or label. This label is the prediction given by the decision tree. An example of a decision tree can be seen below.

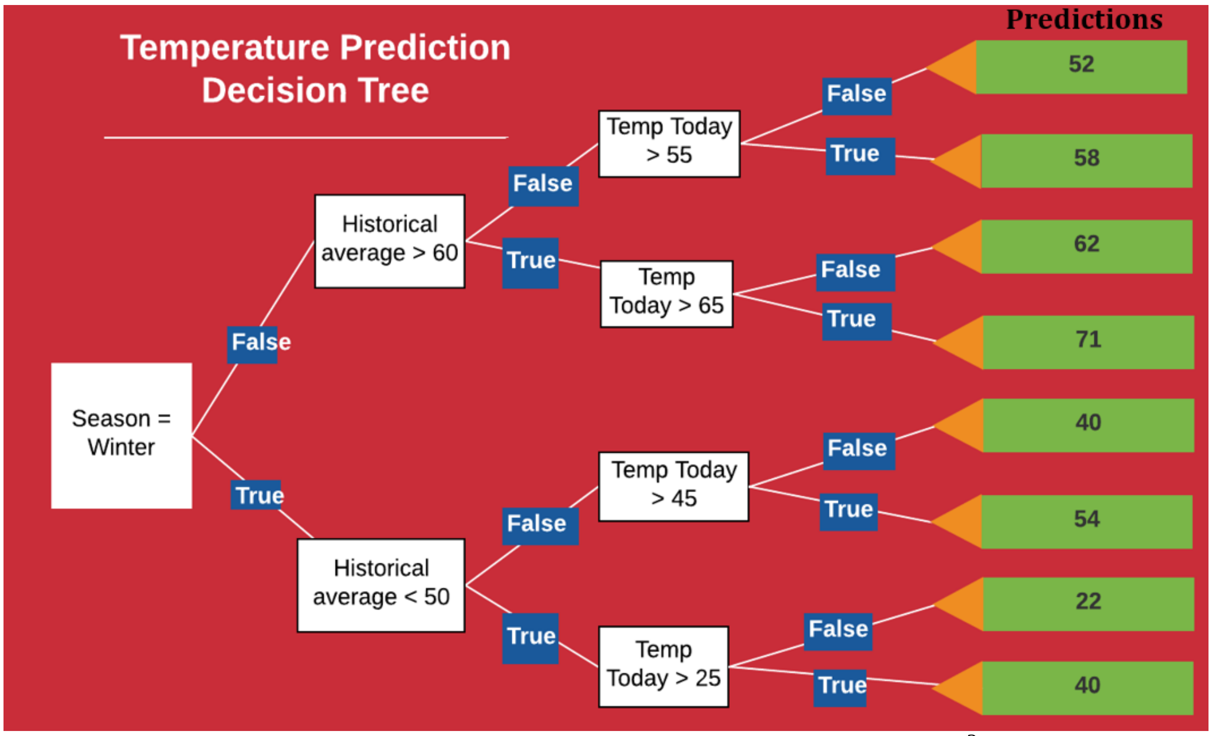


Figure 5.2.1.1. Decision Tree for Temperature Prediction[3]

However, decision tree in itself is not good enough. This is because the predictions will be widely spread around the right answers (the predictions have variance). Hence, instead of having one decision tree to decide on the prediction, we can have multiple trees, each with different structure (nodes and edges values) to come up with one final consensus. The

[3] https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

consensus is decided based on the outputs of all the individual decision trees. The final output of a random forest should be the label that has the highest occurrence frequency.

5.2.2 Support Vector Machine

Support Vector Machine is a supervised machine learning model that categorize classes by hyperplane. Data will be plotted as a point in n-dimensional space where n is the number of features there are and each feature have a value corresponding to the value of a particular coordinate.

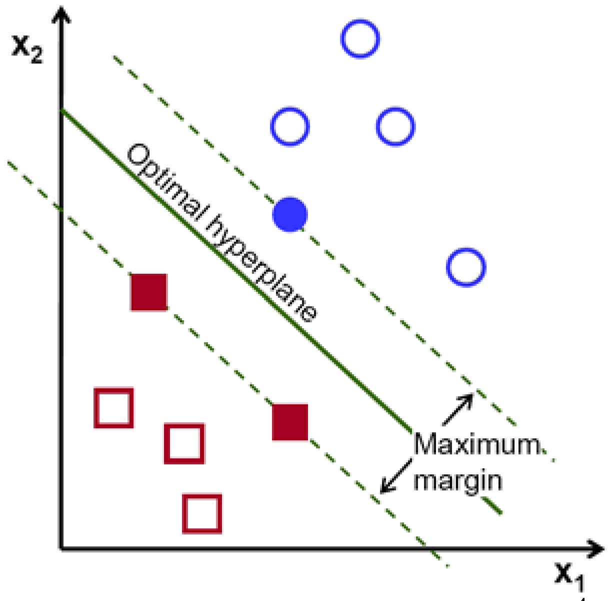


Figure 5.2.2.1. Optimal hyperplane[4]

If the line is too close to the points, it will not be generalize correctly and hence, it is important to find the line that pass as far as possible from all the possible points. The above diagram shows the optimal hyperplane from all the points. For sklearn, SVC and NuSVC implement "one-against-one" approach while linearSVC uses "one-vs-rest" approach[5].

However, SVM does not estimate the probability.

5.3. Training

5.3.1. Random Forest

Training a random forest requires the training of each decision tree. This means that we need to segment our training data. The number of segments depends on the number of decision trees we want our random forest to contain. Ideally, we should pick a number that can minimise the chance of having two or more labels having equal number of occurrence, for example 7.

There are two ways in which we can segment our data to train each of the decision tree.

Method 1: Segment the data randomly while maintaining equal number of data of each of the 10 dance moves in each segment.

[4] https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
[5] http://scikit-learn.org/stable/modules/svm.html

Method 2: Segment the data based on the dancer (each member in our team). This will result in 6 segments of data (6 decision trees). Each segment consists of data for 10 dance moves done by one member.

Specific tuning of the hyperparameters can be done after first iteration of training. Some of the hyperparameters that we can explore includes the number of decision trees in the random forest, maximum tree depth, etc.

5.3.2. Support Vector Machine

Training a support vector machine requires finding the optimal hyperplane/s. We would need first divide the data into attributes and labels from the csv file. For example, if we have 5 attributes, we would need to give it a label for each attribute and label the class as well. Then we would use scikit-learn to train the model with either polynomial, Gaussian, and sigmoid kernels.

We need to tune the parameters of the support vector machine which are kernel, regularization, gamma and margin to ensure higher accuracy of prediction for our model.

5.4. Testing

There are two different testing processes throughout the project:

1. Using the partitioned data

   This training process uses the testing data partitioned immediately after data collection (10% of all of the data collected) stored in a CSV file.

2. Real-time testing

   Data is collected from the sensors and fetched to the machine learning model while the user is dancing.

For each type of testing, the accuracy and the confusion matrix will be used to evaluate the model. Confusion matrix will be especially useful to potentially find out which dance moves are similar (and hence often misclassified).

5.5. Week 6 Testing Data

We will be using Weight Lifting Exercises monitored with Inertial Measurement Units[6] as our testing data to train our two selected machine learning models which are Random Forest and Support Vector machine. We are planning to use scikit-learn libraries for our machine learning. After testing the data, we would be evaluating both models and decide which model performs better for this particular dataset. We hope that our analysis may give us an insight to come up with a hypothesis on which Machine Learning model may perform better on our actual project.

[6]http://archive.ics.uci.edu/ml/datasets/Weight+Lifting+Exercises+monitored+with+Inertial+Measurement+Units

Section 6: Project Management Plan

	Hardware	Firmware & Comms	Software
Week 4	Submit design report		
Week 5	- Purchase sensors and start making wearable system - Use Arduino to record readings from sensors	**(Focus on FreeRTOS implementation + basic serial communication)** - FreeRTOS set up (done) and implementation of tasks - Handshaking between Arduino and Raspberry Pi - Protocol implementation of serial communication between Arduino and Raspberry Pi	- Research more on the hyperparameters of the two ML models - Learn **pandas** and two ML models libraries (**sklearn**) - Start implementing ML models on sample dataset
Week 6	- Complete circuit with all sensors on it and get sensor readings for 2 dance moves	**(Focus on finalising serial communications + server communication; preparation for subsystem evaluations)** Ensure that Arduino is able to send dummy data to Raspberry Pi Implementation of socket communication for Raspberry Pi to connect to server Implementation of secure communications Testing of all communications with dummy data	- Complete the subsystem project - Do final evaluation on the two ML models
Recess	Subcomponents integration and data collection		
Week 7	- Ensure all hardware components are working correctly - Fix anything if necessary - Come up with ideas to help improve power efficiency or lower power consumption	**(Focus on testing and ensuring comms work with data obtained from hardware)** - Ensure that readings from the sensors can be obtained, organised and sent accurately. (Arduino; hardware	- Train the 2 ML models using the data collected - Data validation (To experiment and come up with the most suitable data flattening method)

		should be set up already) - Ensure that data is collected and sent correctly to the RPi. - Data can be passed correctly to machine learning algorithm	
Week 8	- Improve power consumption - Provide test data for ML	**(Focus on ensuring data is sent securely to server)** - Test server communication using actual data obtained from algorithm; make necessary modifications	Test and compare the performance of the two ML models on the test data Decide on the final model to be used for real time classification (e.g. combination of both, use one of the models, etc.)
Week 9	- Help other subteams - Provide test data for ML	- Implement changes that can make the sending/receiving more efficient/faster (after testing for significant improvements) - Fix bugs arising from integration	- Real time classification testing and hyperparameters tuning - Compare the performance between ML models
Week 10	Baseline 5 moves evaluation		
Week 11	- Help other subteams - Provide test data for ML	- Integration checks and modifications	- Integrating the remaining 5 moves if previously unable to come up with models that are able to classify all 10 moves - Train and compare performance among models
Week 12	- Final testing and integration checks	- Final testing and integration checks	- Final testing and hyperparameters tuning
Week 13	Final demonstration (10 moves)		

References

Hardware:
Bi-Directional Logic Level Converter Hookup Guide. (n.d.). Retrieved September 9, 2018, from https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

GY-521-SCH.jpg. (n.d.). Retrieved August 30, 2018, from http://www.haoyuelectronics.com/Attachment/GY-521/GY-521-SCH.jpg

How to use the accelerometer- gyroscope GY-521. (n.d.). Retrieved September 9, 2018, from https://create.arduino.cc/projecthub/Nicholas_N/how-to-use-the-accelerometer-gyroscope-gy-521-6dfc19

Hymel, S. (n.d.). INA169 Breakout Board Hookup Guide. Retrieved August 30, 2018, from https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide

Hymel, S. (2013, November 15). Sparkfun/INA169_Breakout. Retrieved September 6, 2018, from https://github.com/sparkfun/INA169_Breakout/tree/master/Fritzing/source_files

InvenSense (2012, May 16). MPU-6000 and MPU-6050 Product Specification Revision 3.3. Retrieved September 4, 2018, from http://www.haoyuelectronics.com/Attachment/GY-521/mpu6050.pdf

MPU-6050 Accelerometer Gyro. (n.d.). Retrieved August 30, 2018, from https://playground.arduino.cc/Main/MPU-6050

Rowberg, J. (2018, April 25). I2C Device Library. Retrieved September 4, 2018, from https://github.com/jrowberg/i2cdevlib

Rowberg, J. (n.d.). MPU-6050 6-axis accelerometer/gyroscope. Retrieved September 4, 2018, from https://www.i2cdevlib.com/devices/mpu6050#source

Texas Instruments. (2014, November). TXB0108 8-Bit Bidirectional Voltage-Level Translator with Auto-Direction Sensing and ±15-kV ESD Protection. Retrieved September 6, 2018, from http://www.ti.com/lit/ds/symlink/txb0108.pdf

Texas Instruments. (2017, February). INA1x9 High-Side Measurement Current Shunt Monitor. Retrieved September 4, 2018, from http://www.ti.com/lit/ds/symlink/ina169.pdf

Comms:
J. (n.d.). Serial Communication. Retrieved from https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial

One or two UART stop bits? (n.d.). Retrieved from https://electronics.stackexchange.com/questions/29945/one-or-two-uart-stop-bits

Savage, R. (n.d.). The Pi4J Project – Pin Numbering - Raspberry Pi 3 Model B. Retrieved from http://pi4j.com/pins/model-3b-rev1.html

Using FreeRTOS multi-tasking in Arduino. (n.d.). Retrieved from https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc

Additional references from :
EE2024 and EE4204 lecture slides
Prof Peh's lecture slides (CG3002)

Software:
Fitbit Development: Accelerometer Sensor Guide. (n.d.). Retrieved September 9, 2018, from https://dev.fitbit.com/build/guides/sensors/accelerometer/

Introduction to Support Vector Machines¶. (n.d.). Retrieved September 9, 2018, from https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

Jiang, C (2015, June 3).Sampling Frequency Optimization And Training Model Selection For Physical Activity Classification With Single Triaxial Accelerometer. Retrieved September 9, 2018, from
https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/23353/Jiang.pdf?sequence=1

Koehrsen, W. (2017, December 27). Random Forest Simple Explanation – William Koehrsen – Medium. Retrieved September 9, 2018, from
https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

UCI Machine Learning Repository: Weight Lifting Exercises monitored with Inertial Measurement Units Data Set. (n.d.). Retrieved September 9, 2018, from
http://archive.ics.uci.edu/ml/datasets/Weight%20Lifting%20Exercises%20monitored%20with%20Inertial%20Measurement%20Units

1.4. Support Vector Machines¶. (n.d.). Retrieved September 9, 2018, from http://scikit-learn.org/stable/modules/svm.html