

**CG3002 Embedded System Design Project**

Semester 1 2017/2018

**“Dance Dance”**

**Design Report**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group **01** | Name | Student # | Sub-Team | Role |
| Member #1 | Raphael Quek Hao Chong | A0139716X | Firmware | Member |
| Member #2 | How Wei Wen, Nicolas | A0139523E | Firmware | Subteam lead |
| Member #3 | Manish Dahiya | A0101045A | Hardware | Member |
| Member #4 | Lim Jun Hao | A0147944U | Software | Subteam lead |
| Member #5 | Han Xue | A0133369B | Software | Member |
| Member #6 | Narinderpal Singh Dhillon | A0097416X | Hardware | Member |

**Section 1 System Functionalities**

**1.1 Use Cases**

Use Case: Correct dance move

1. PC displays requested dance move
2. User dances requested move
3. System detects and evaluates dance move
4. System determines that move matches requested dance move
5. System sends results to server
6. PC displays the move done and that it matches

Use case ends

Use Case: Incorrect known dance move

1. PC displays requested dance move
2. User dances a different known move
3. System detects and evaluates dance move
4. System determines that move does not match requested dance move
5. System sends results to server
6. PC displays the move done and that it does not match

Use case ends

Use Case: Unknown dance move

1. PC displays requested dance move
2. User dances an unknown move
3. System detects and evaluates dance move
4. System determines the move is invalid
5. System sends results to server
6. PC displays “invalid dance move” and that it does not match

Use case ends

Use Case: Idle

1. PC displays requested dance move
2. User stays idle
3. System detects idle state
4. System waits for dance move to be detected

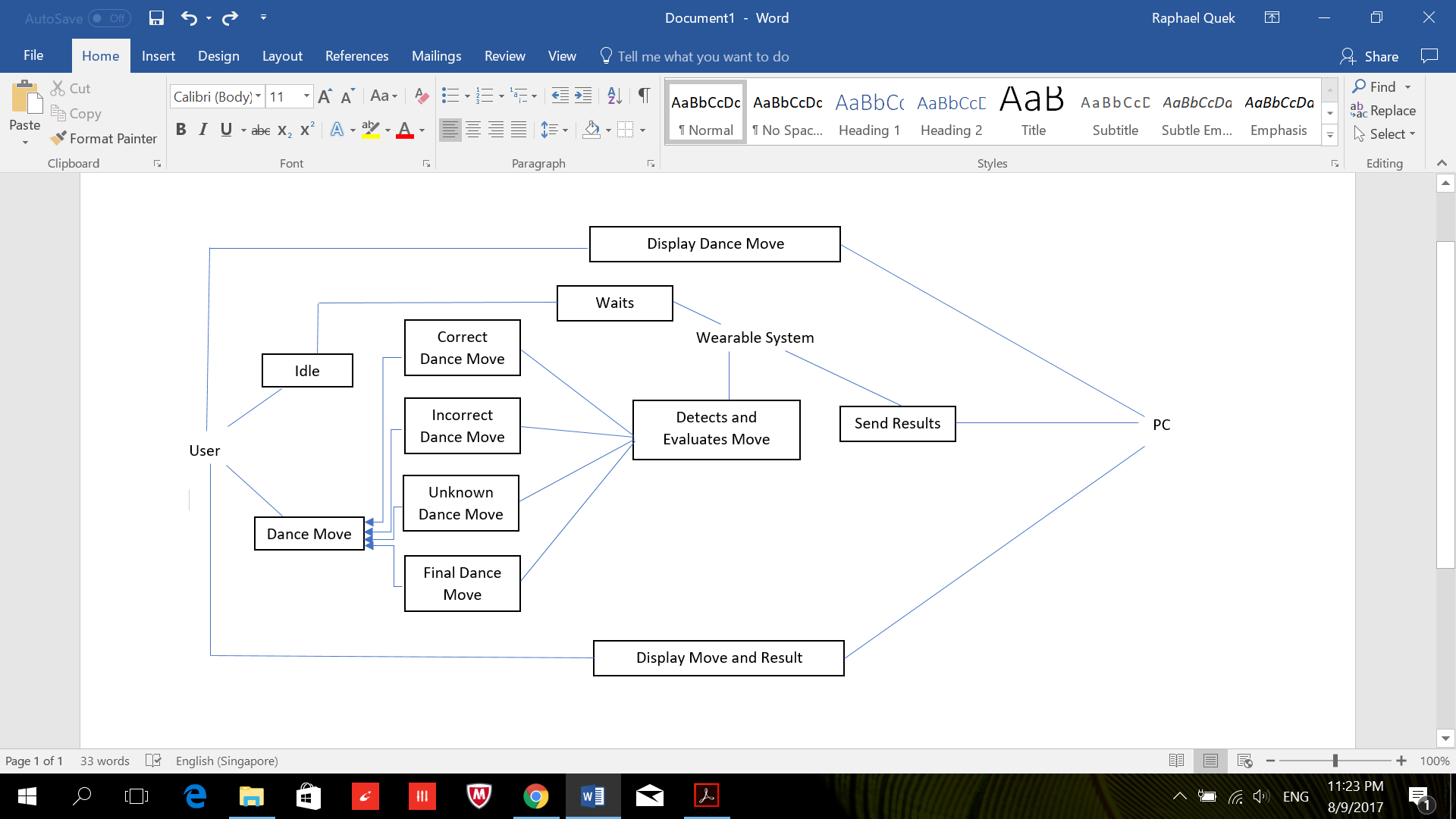
Use case ends

Use Case: Final dance move

1. User dances final move
2. System detects and evaluates dance move
3. System determines the move is the final move
4. System sends results to server
5. PC displays “The End”

Use case ends

**Use Case Diagram**



**1.2 Feature Lists**

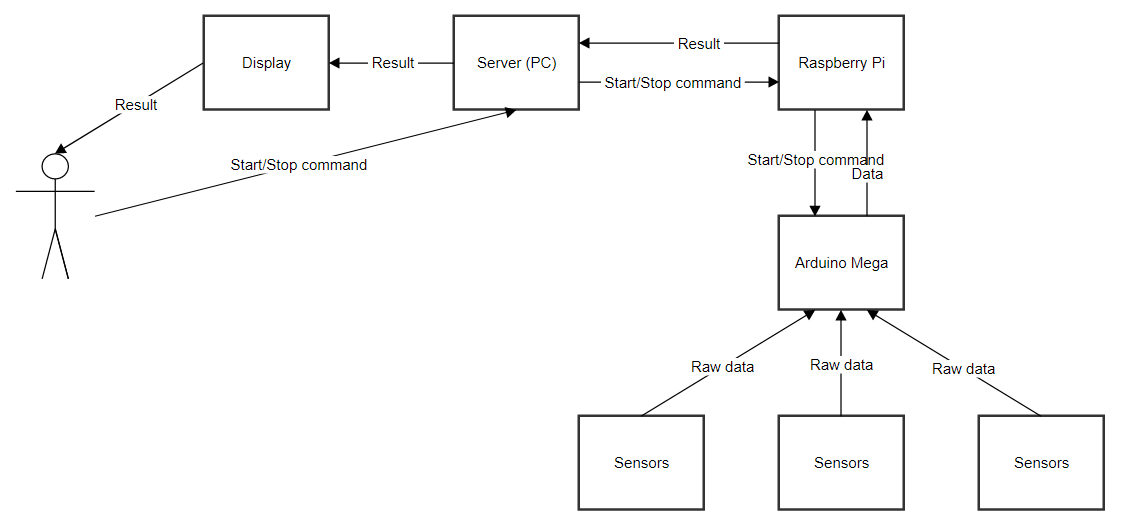
1. Detect if user is performing a known move
2. Detect if user is performing an unknown move
3. Detect if user is stationary/at rest
4. Compare detected dance move with given dance move and indicate if they match
5. Adjustable size

**1.3 User Stories**

1. As a user, I want the wearable to be comfortable.
2. As a user, I want the wearable to have minimal wires so that they will not get in the way of my dancing.
3. As a user, I want the wearable to be lightweight so that it will not be tiring to dance with it.
4. As a user, I want to be able to do a finishing dance move to end my dance routine.
5. As a user, I want the system to detect my dance moves quickly as my dance routine is usually fast paced.
6. As a user, I want the system to accurately detect my dance moves.
7. As a different sized user, I want the wearable to be free-size so that it can be worn by me comfortably.
8. 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.
9. As a user, I want to know the power used by the wearable so that power consumption can be communicated to the server.

## Section 2 System Architecture

**2.1 Overview**



1. Arduino Mega
   1. Take in raw data from the sensors
   2. Minimal/no processing of data
   3. Send data to Raspberry Pi
2. Raspberry Pi
   1. Process data for patterns and determine if it matches known dance move
   2. Send result to server
3. Server
   1. Provide a user readable format of the result on the display
   2. Allows user to stop the systems to conserve battery

**2.2 Connection**

1. Arduino Mega - Raspberry Pi: GPIO pins connected to level shifters
2. Raspberry Pi - Server: SSH over WiFi (Problems: Require own router)
3. Server - Display: HDMI

**2.3 Hardware layout**

* Wearable belt where the Raspberry Pi, Arduino Mega and their power supply will be attached
* Sensors at wrists, ankles and feet with a wired connection and secured using velcro straps to the limbs
* Additional velcro straps will be placed along the wires along the limbs to secure the wire to prevent dangling wires from interfering with the user’s movement.

**Section 3 Component Interactions and Design**

**3.1 Design**

**3.1.1 Main algorithm for activity detection**

1. User requests to begin detection -> User given time to get into rest position -> Sensors are calibrated and are now actively being read
2. Data for the past 5 seconds are being stored in the Raspberry Pi
3. It is then parsed into predetermined flags  
   i.e. isJumping(), isArmBent(), isWholeBodyMoving(), etc
4. Parsed data is now checked against patterns for the known dance moves  
   i.e. if isArmBent() & areHandsMoving() are true and the others false, then the system will regard the user as doing the ‘waving hands’ move

**3.1.2 Main algorithm for machine learning to improve detection of known moves**

1. User requests to begin detection -> User given time to get into rest position -> Sensors are calibrated and are now actively being read
2. Data for the past 5 seconds are being stored in the Raspberry Pi
3. It is then parsed into predetermined flags
4. Parsed data is now checked against patterns for the known dance moves but fails or gives incorrect move
5. User acknowledges this on the display and requests for a correction for that move
6. The flags parameters are then corrected (i.e. increased or reduced range) and updated for future use with the help of basic user inputs (i.e. user tells system that his knees are currently in a bent position when its not registering) to teach the system

**3.1.3 Main algorithm for machine learning of new moves**

1. User requests to begin learning of new move -> User given time to get into rest position -> Sensors are calibrated and are now actively being read
2. Data for the past 5 seconds are being stored in the Raspberry Pi
3. It is then parsed into predetermined flags  
   i.e. isJumping(), isArmBent(), isWholeBodyMoving(), etc
4. User is prompted to redo the move to ensure accuracy
5. Parsed data is registered as a new move
   1. If the new move matches the same flags as an known/registered move, user is requested to perform the two moves separately
   2. The flags are for the two moves are compared to look for differences between the raw data for related sensors
   3. Using K-Nearest Neighbor Classifier with a human in the loop, for the flag(s) that the system picks up a difference, they will be broken into two flags to further differentiate the move  
      i.e. isArmBent() -> isArmConstantlyBent() & isArmOpeningAndClosing()  
      isLegBent() -> isLegBentFully() & isLegBentALittle()

**3.2 Component Interactions**

**3.2.1 FreeRTOS**

In order to use FreeRTOS on the Arduino IDE we must first download its files and have it available as a library. This enables us to make use of its scheduler as well as other functions included in the library. FreeRTOS will schedule the following processes of different priorities.

**3.2.2 List and algorithm of processes**

Arduino Processes:

1. Arduino reads data from sensors and send to Queue1 [Highest priority]

2. Arduino reads from Queue1 and organises data into packets (attaching sensor ID headers, computing checksum, etc.) before sending them to RPi [Lowest priority]

* Stores a copy of the packet and waits for ACK from RPi before overwriting the data stored
* If a NAK is received, it will resend the stored data

RPi Processes:

1. RPi receives packets from Arduino and computes checksums and sends an ACK to Arduino if checksum is equal. If not it will send a NAK [Highest priority]

* If it is correct, it will store data packet in a buffer
* Else, it will discard data and wait for the resend from Arduino

2. RPi reads and processes data from buffers to determine dance move (once there are 20 items in the buffer) before sending the results to the server [Lowest priority]

We decided to make the process of reading data from sensors the highest priority because we wanted to get data as quickly as possible in fixed intervals so that they can be sent to the RPi for processing. At the same time we also want data to be received and stored in the RPi as quickly as possible because in order to determine the dance move we require around 20 sets in the buffer before we can accurately do so.

**3.2.3 Synchronization and Communication between Arduino and RPi**

In order to help synchronize and coordinate the processes in Arduino as well as in RPi we make use of queues. Semaphores were another option, but in our case since there isn’t a shared critical region that different processes are accessing at the same time, we are not going to utilize them.

Secondly, We will also use a bootup 3-way handshake once RPi has been powered on. RPi will send a message to Arduino Mega who will in return send an ACK back. RPi will once again return with an ACK so that they both know that each of them are ready to communicate with one another. This once again helps make sure that both of them are on the same page.

The method of communication that we will be using is also known as a periodic push by the Arduino Mega. Once the Mega has data available, it will send it directly to the RPi. We preferred this method to another known method, periodic poll by RPi, because in periodic poll, if RPi does not poll frequent enough, we may lose data on the Mega since Mega has a small memory and not all data can be stored. This would cause inaccuracies in determining what dance move we were doing since every data is crucial.

We decided to make the hardware connection between the Mega and the RPI via USART GPIO pins, connected with a level shifter. We had another choice of using a USB connection, but we decided that there is USB latency, it uses up a USB port which could have other purposes and finally the Mega will draw power through the RPi and lead to power instability.

We chose to to use USART1 (Pins 18 and 19) at a baudrate of 9600 because, the higher a baud rate goes, the faster data is sent and received but there are limits to how fast data can be transferred. If it is too high, it will result in seeing more errors on the receiving end. Therefore we decided to choose a very common rate of 9600 bps.

**3.2.4 Communication between RPi and Evaluation 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.

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 4 Hardware Details**

## 4.1 Hardware Components

**Ultrasonic Sensor(I2C)**

|  |  |
| --- | --- |
| Component | HC-SR04 |
| Power Supply | +5V DC |
| Working current | 15mA |
| Effectual Angle | <15° |
| Measuring Angle | 30° |
| Ranging Distance | 2 - 400cm |
| Connection | Atmega |

Data taken from: <http://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/>

### Level Shifter

|  |  |
| --- | --- |
| Component | 74LVC245 Level Shifter |
| Power Supply | 3.6V |
| Max Current Consumption | 24mA |
| Connection | Arduino & Atmega |

Data taken from:<https://cdn-shop.adafruit.com/datasheets/sn74lvc245a.pdf>

### Accelerometer/ Gyroscope (I2C)

|  |  |
| --- | --- |
| Component | GY-87 10 DoF (Gyroscope, Accelerometer, Compass, Barometer) |
| Operating Voltage | 3~5 |
| Gyroscope scale range (⁰/sec) | ±250, ±500, ±1000, and ±2000 |
| Gyroscope Average Current (mA) | 6.5 |
| Accelerometer scale range (g) | ±2, ±4, ±8 and ±16 |
| Accelerometer Average current (micro A) | 140 |
| Connection | Onboard Atmega |

Datasheet:

Gyroscope: <http://www.kamami.pl/dl/itg3205.pdf>

Accelerometer: <http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf>

### Keypad

|  |  |
| --- | --- |
| Component | Keypad |
| Operating Voltage | ~5V |
| Max Current (mA) | 500 |
| Connection | RasPi USB |

### Vibration motor(PWM)

|  |  |
| --- | --- |
| Component | Vibrating Motor mini Disc |
| Operating Voltage | 2-5V |
| Max Current (mA) | 100 |
| Connection | Digital Output |

Data: <https://www.adafruit.com/product/1201>

### Headphones

|  |  |
| --- | --- |
| Component | headphones |
| Max Operating Voltage | 2.8 V |
| Typical Max Biased Current (mA) | 1.2 |
| Connection | 3.5mm jack on raspberry pi |

Data: <http://www.ti.com/lit/ds/symlink/tpa6166a2.pdf>

## 4.2 Pin Table

|  |  |
| --- | --- |
| Component | Pin Number |
| Ultrasonic Sensor | 82 - 85 |
| Distance Sensor | 86 - 87 |
| Gyroscope | 88 |
| Vibration Motor | 1 |
| Level Shifter | 5 |
| Raspberry Pi | 12 - 13 |

Data taken from:<http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf>

<https://www.arduino.cc/en/Hacking/PinMapping2560>

## 4.3 Power Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Voltage (V) | Current (mA) | Max Power (mW) |
| Ultrasonic Sensor | 5 | 15 | 75 |
| Gyroscope | 5 | 6.5 | 32.5 |
| Vibration Motor | 5 | 100 | 500 |
| Level Shifter | 3.6 | 24 | 86.4 |
| Raspberry Pi | 5.1 | 1000 | 5100 |
| Arduino Mega | 5 | 500 | 2500 |
| Total | - | - | 8193.9 |

# 

Raspberry Pi: 2.1A power supply at 5.1V

Atmega: 5V usb power supply

**Section 5 Project Management Plan**

**5.1 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

Week 11:

Polish 2nd prototype

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

Week 12:

Further polish 2nd prototype

Implement additional improvements

Week 13:

Polish additional improvements

**Final Demo: (40%)**

**Section 6 Societal Impact <Final Report>**

**6.1 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. It would be possible to use the detector for other purposes, such as exercise aid, following these enhancements. The dance detector may 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. Further improvements on the detector can also be of the form where the server sends signal to the RPi in order to teach dancing to a bad dancer. In this case, the server will instruct the user to dance a specific move then the move will be detected and analyzed and engaging feedback will be provided to the user. However, the speed of the instructions and processing must be very fast in such case so that the system doesn’t seem to be lagging. 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.

**6.2 Issues**

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.

**References**