Fall 2020

Smart Wearable Device for Reduction of Parkinson's Disease Hand-Tremor

Bruno Ribeiro
CCT College Dublin

Leopoldo Medeiros
CCT College Dublin

Rodrigo Aguiar
CCT College Dublin

Juan Carlos
CCT College Dublin

CCT College Dublin

Bio Protech Academic Research

Wearables & Software Engineering

Smart Wearable Device for Reduction of Parkinson's Disease Hand-Tremor

Bio Protech Systems

CCT College Dublin

Bruno Ribeiro - 2017138

Leopoldo Medeiros - 2017288

Rodrigo Aguiar - 2017302

Juan Carlos - 2016181

ABSTRACT OF THE RESEARCH

Smart Wearable Device for Reduction of Parkinson's Disease Hand-Tremor

by Bio Protech Project, Bachelor of Science in Information Technology
CCT College in Dublin, Ireland 2020

Project Supervisor: Professor Greg South
Project Manager: Bruno Ribeiro
Bio Protech Team: Juan Carlos, Leopoldo Medeiros & Rodrigo Aguiar

Parkinson's disease is a neurodegenerative disorder that affects over 10 million people worldwide (Health Unlocked, 2017). People diagnosed with Parkinson's Disease can experience tremors, muscular rigidity and slowness of movement. Tremor is the most common symptom and external agents like stress and anxiety can make it worse, which may cause complications to complete simple day-to-day tasks.

Therefore Bio Protech proposes the development of a smart wearable device for reduction of the hand-tremors based on medical evidence that by applying vibration to the wrist may result in a reduction of the involuntary tremor. The device imitates the shape of a wristwatch and the vibration is supplied by motors placed around the wrist. The users will be given the possibility to regulate the frequency according to their needs using a mobile application connected via Bluetooth.

CCT COLLEGE IN DUBLIN, IRELAND
College of Computing Technology

Academic Research Examination Committee:
Ken Healy
Greg South
Graham Glanville

A research on the
wearable solutions for Parkinsonians tremors reduction
by
Bio Protech Systems.

A research presented to the College of Computing Technology Dublin, Ireland
in partial fulfilment of the
requirements for the degree of
Bachelor of Science in Information Technology

May 2020
Dublin, Ireland

1. Introduction

I. Purpose

Parkinson's disease (PD) is the second most common age-related neurodegenerative disorder after Alzheimer's disease. There are more than 10 million people in the world who suffers from Parkinson's Disease (Health Unlocked, 2017).

The group has been inspired by a project called *"Emma Watch"*, which consists of a smart band device developed by Haiyan Zhang and her team (*Microsoft Research Department*). Emma Watch was developed to help Emma Lawton, a 33-years-old-Parkinsonian patient, to perform her job as a creative director. In a short documentary called "This invention helped me write again" produced by BBC (2016), Emma Lawton shows a comparison of her drawing a straight line with and without the watch, the difference is easily noticeable. In the same documentary, it is said that Emma Watch was specifically built for Mrs Lawton's tremor patterns. Therefore, it would not have the same results in other patients.

II. Understanding Parkinson's Disease

Parkinson's disease (PD) is the second most common neurodegenerative (Entering, 2018) disorder and also the most common movement disorder. Characteristics of Parkinson's disease are progressive loss of muscle control, which causes tremor of the limbs and head while at rest, stiffness, slowness, and impaired balance. This project intends to reduce the most common symptom of the disease, which is the hand-tremor

A. Feedback Loop

To a better understanding of the relation between our brain and tremors caused by PD, it is important to mention the feedback loop that happens within the human body. Feedback loops can be found in any system that requires equilibrium or controlled disturbance, the self-adjustment of a system acquires an input, internal or external, which will trigger a particular outcome. This outcome can be either a positive or a negative feedback loop, it will escalate the change to the system (positive) or decrease it to bring the system back to its constant state (negative).

A significant example of a positive feedback loop can be observed in the relation of water vapour and global warming. This is a positive feedback loop because water vapour is a greenhouse gas thus the rise in water vapour in the air causes further warming, which results in more water vapour (Zuckerman, 2016). Regardless of the importance of positive feedback loops in a determined system, for this project, it will be required to keep a constant state, without hand-tremor. Therefore, the focus of this report will be on the negative loop or homeostasis.

Negative feedback loops are important because they allow living organisms to maintain constant homeostasis (Fatsy, 2016). In living system thermoregulation is the most common negative loop, the system tries to keep a steady and constant temperature by receiving inputs from thermoreceptors and using its information to self-adapt to the desired temperature.

It has been suggested that Parkinsonian tremors are due to oscillations within an unstable internal feedback loop involving ascending (Evarts, et al., 1979) impulses from the spinal cord to motor cortex, and back to the spinal cord . The Parkinsonian brain senses a false movement of the arm and tries to prevent it. Therefore the involuntary tremor is triggered by a false input. In an experiment executed by Teräväinen and collaborators (1980), it was possible to break the feedback loop between Parkinsonians patient's brain and arm by providing vibrations that would send an impulse opposite to the deceptive one, resulting in the uncontrolled tremors to reduce or completely stop.

B. Homeostasis

Homeostasis means maintaining a continuing internal environment seen in any self-adjusting process by which an organism tends to ensure that the stability is established.
If homeostasis is successful, life continues; if it is unsuccessful, it leads to a disaster or death of the organism. The "stability" that the organism reaches is not around an actual point (such because of the idealised flesh temperature of 37ºC [98.6ºF]). Stability takes place as a part of a dynamic equilibrium, which may be thought of as a cloud of values within a decent point which continuous change occurs. The result's that relatively uniform conditions prevail. Equilibrium, in physics, the condition of a system when neither its state of motion nor its internal energy tends to alter with time (P. Guillan, 2015). A simple mechanical body is claimed to be in equilibrium if it experiences neither linear acceleration nor angular acceleration; unless it's disturbed by an out of doors force, it'll continue therein condition indefinitely. For one particle, equilibrium arises if the resultant of all forces acting upon the particle is zero. For instance, imagine the cockpit of a plane is tactful of dials. It is the pilot's job to make sure all of them stay in the right place.

Whether any of them change, he will correct the planes keep them within the right place and also the plane within the air. This can be a small amount, like how a brain works. There are sensors all around the body for measuring various things and sending the impulses back to the brain. The brain does its best to stay stable and constant to keep the body working properly. This is homeostasis, maintaining a continuing internal environment.

C. Negative Feedback

Negative feedback is the process where the brain uses either hormone or nervous system to send a signal to the part of the body, that can rectify the problem. For instance, to eat a meal your blood sugar level increases, this is detected and the pancreas will release a hormone called insulin that causes the sugar to be stored in the liver. Therefore, the blood sugar level returns to normal.

Negative feedback is to live from which the body detects of stimulus reacts by responding and brings the body back to the normal levels. Organisms also have feedback mechanisms that maintain dynamic homeostasis billowing them to respond to changes in their internal and external environments. Negative feedback loops maintain optimal internal environments.
Homeostasis and Parkinson's Diseases

Neurons are metabolically active cells with high energy demands. Thus, neurons are particularly reliant on mitochondria function, especially on the homeostasis properties of mitochondria.
This is reflected by the observation that mitochondrial abnormalities diseases, like Parkinson's disease (PD). Mitochondria are highly complex and dynamic organelles continuously undergoing different alterations. The dynamic property of mitochondria is termed as mitochondria homeostasis. Imbalance of mitochondrial homeostasis is a neurodegenerative disease, such as Parkinson's diseases.

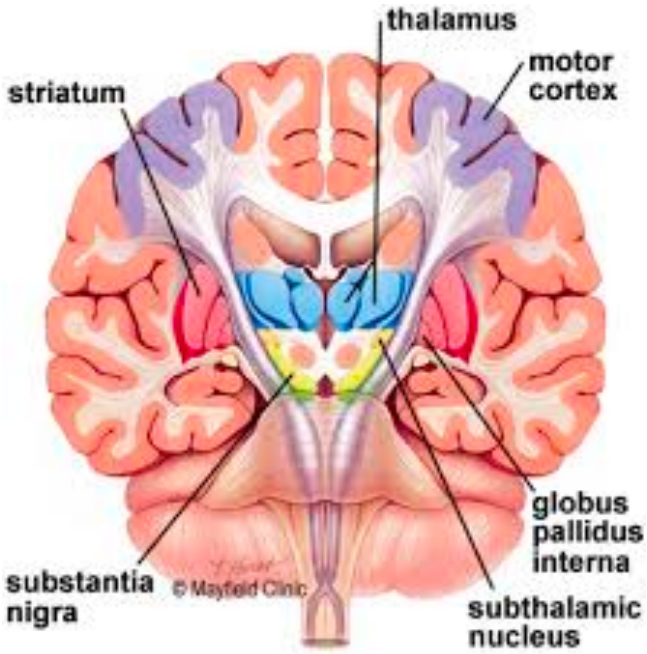


Fig 2. A cross section of the brain. The impulse for body movement begins in the motor cortex of the brain. The basal ganglia are responsible for activating and inhibiting specific circuits or feedback loops. Picture collected from: https://d2uko0fg0ksrq5.cloudfront.net/pe-parkinsons-snsi.pdf

III. Goals and Objectives

Bio Protech motivation reflects our desire to help Parkinson's disease patients to regain their ability to execute simple daily tasks such as writing and eating by offering a technological solution. The main goal of the project is to develop the prototype of a medical device that is easy to use, comfortable to wear, affordable and controlled by a friendly user-interface to provide a better quality of life for those who suffer from Parkinson's Disease. The objectives of the project include:

- The collection of data through medical experiments to support our goal and to apply the same principals with technological knowledge and skills;
- Researching about types of equipment and hardware to make possible the creation of the device;
- Improving and learning new skills about planning, analyse and development of mobile applications;
- To acquire knowledge on topics never encountered throughout the course such as Python programming language, Raspberry Pi, Biology, Electronics and Physics.

IV. Solution Proposal

Inspired by Project Emma and based on Teräväinen scientific experiment, we are developing a wearable medical device, like a wristband, that will have five vibrating motors discs around the wrist. What it does is that it is short-circuiting whatever feedback loop there is between the brain and the hand that is causing the tremor. Bio Protech's wearable device will be called **Park Med**.

Park Med needs to supply different levels of vibrations to counteract the tremors seen on Parkinsonian patients. At this point, it is believed that the frequency of the vibration is a fundamental matter in this project because it is the key to the tremors' control. This approach is based on the fact that each patient has variations of hand-tremors. Through a mobile application, the users will be able to control the intensity of Park Med's invoked vibrations according to their needs.

Park Med will be powered up using a Raspberry Pi, which will be connected to a smart device such as smartphones or tablets. The device will receive instructions from the user sent through a mobile application installed on the smartphone or tablet. As this solution is aimed at people with a disease, we believe that Bluetooth connection is the best way to connect the app to the Raspberry Pi, so the patient does not have to depend on a WIFI signal, and most smartphones have Bluetooth.

It might be an option if the time is enough that we could potentially eliminate the wires from the wrist band and the Raspberry Pi, and use Bluetooth between them. The mobile will be as friendly and straightforward as possible. It will consist of a layout with a stop button and three or four buttons to start the vibrations according to the intensity needed for the user.

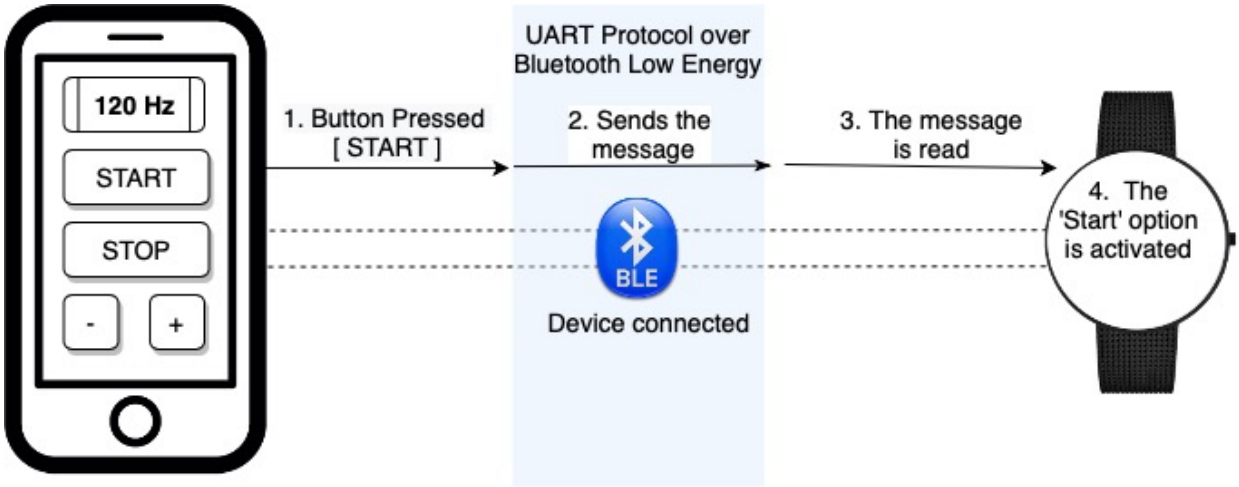


Fig. 3. Illustrative representation of connectivity and communication between a smartphone and Park Med. A text message is transmitted over Bluetooth, so that its corresponding command is activated.

V. Available Solutions

There are some medical devices which can be related to Park Med and to our goal of improving patient's quality of life. Those devices are in the development stage or already available to a small number of people.

A. Gyrogear

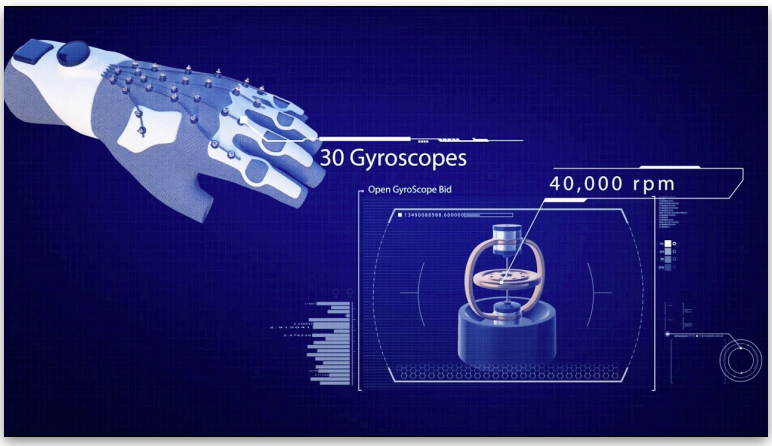
"The GyroGlove is a wearable technology currently in development to provide a sort of hand stability. Gyroscopes inside the glove are used to reduce hand tremors, to relive boldness and easy daily tasks". Available at: http://gyrogear.co

B. Parkinson Smartwatch

"With a Parkinson Smartwatch you can easily keep track of the severity of your Parkinson's, a really simple way to keep a diary. Whenever you notice a change in your condition you can record this using the "Status Rainbow". Simply touch the colour on the Rainbow that matches how you are feeling: yellow means 'off' or 'bad' (slow, stiff, tremor etc), green is 'good', and red is for involuntary movements". Available at: https://parkinsonsmartwatch.com

C. iStopshaking

"iStopshaking device is powered by 30 gyroscopes spinning at 40,000 rpm per gyroscope and are controlled by an intensive switch and driven by a battery. iStopshaking gyroscopes provide more precise control over the tremors on their hand. The device has a wireless connection to iPhone devices to send data regarding time-frequency of tremor degree of trembling and plot a chart on smartphones. Available at: https://www.youtube.com/watch?v=kgRgcmiS-vI



VI. Technologies

In this section will be listed the technologies and resources necessary for the performance of the project, the same will be detailed in further stages of the project. The group has selected the Raspberry Pi for collecting and processing data due to its computing performance, size, cost and also availability.

Park Med prototype is aiming to deliver a vibration system functionality to the user. However, developing the device will demand knowledge about some subjects such as programming, electronics and biology.

Based on that prototype, the main focus during all the process is trying to understand and make some tests regarding the language we will be using for it, which are Java and Python applied on Raspberry Pi.

Resources		
Programming Language & Application	**Hardware**	**Accessories**
Python	Raspberry Pi	Vibrating motor discs
Java	Monitor	HDMI cable
GUI (*Graphical User Interface*)	Mice	Raspberry Pi's board
-	Keyboard	Cables to connect the board into the Raspberry Pi

A. Why is Java efficient for this project?

When we talk about Programming Language, more specific Java, it could run in any IDE software. Fundamentally, the approach is remote development using NetBeans. NetBeans runs on a workstation (a desktop or laptop computer) connected via Wifi to the Raspberry Pi.
Once everything is set up correctly, Java can be written and compiled in a single click which builds the program on the workstation, the file then must be transferred to the Pi, which will run the program and deliver the output to the NetBeans console.

B. Why is Python efficient for this project?

Raspberry Pi is designed for educational purposes. There is no surprise that Python is one of the most popular choices as a programming language for the Raspberry Pi Foundation promotes Python, which is the primary language for the Raspberry Pi.

The most advantages of Python is that it emphasises indentation, which gets students writing clear code. For programming the Raspberry Pi, Python is considered one the most choices for building educational applications that control some pins, retrieve data from an internet service.

For pin control, we are getting to include the wiring library. This wiring library is called WiringPi, which is specially designed to run on the Raspberry Pi, and it allows tp control LEDs, motors and buttons through the GPIOs.

In resume, by using Java to develop all the Graphical Interfaces, designing the visual composition and temporal behaviour of a GUI is a very important a part of software application programming within the area of human-computer interaction. In Python, all the software are going to be geared toward the prototype functionality within the background.

Example of how a graphical user-interface works:

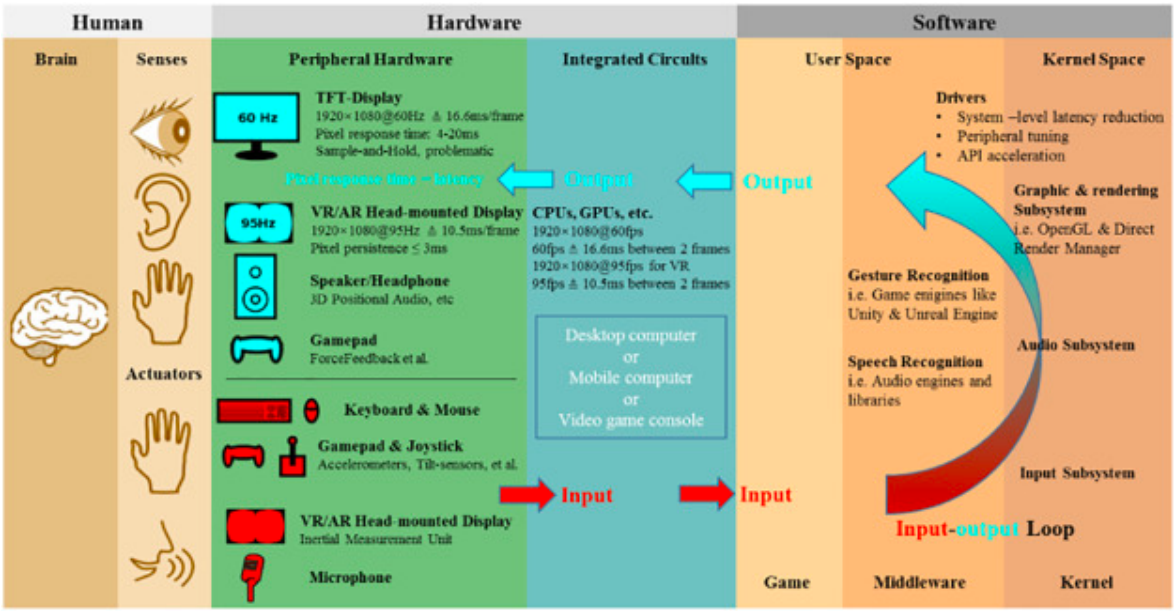


Fig.4 The graphical user interface is presented (displayed) on the screen. It is the result of processed user input and usually the main interface for human-machine interaction.

2. Literature Review

I. Medical Knowledge

A. Hand-tremor reduction

In this chapter, we will describe a number of scientific experiments that will be used as the medical knowledge necessary to build Park Med. The methodology applied in those experiments is our direction and the results obtained is our proof that by inducing vibrations Park Med is capable to achieve the desired results.

The nervous system disorder that affects the movements of a person who is suffering from Parkinson's disease provokes severe tremors that compromise the daily activities. Nowadays, many medical treatments have been developed to decrease this involuntary movement, and induced vibrations are getting, even more, acknowledge by organizations that can develop devices to minimize the amplitude of the tremors. Project Emma by Microsoft Research Team is one of them.

The previous researches have shown the relation between induction vibration and the tremors of Parkinson's and its effectiveness. In this part of our research, we will see how this vibration was applied, in which part of the body specifically and which frequency it was used to get the best results. The tremors are not the same in every person. Consequently, the results are not the same for everyone.

This external vibration modifies the tremor of Parkinson's person by stimulating the tendon, improving the impaired motor function. Proprioceptive stimulation is caused by muscle tendon vibration. The sensory system which is found in our muscles and joints is responsible for giving us a way of body awareness. According to the Elsevier Journal specialist in science and health, "Proprioceptive stimulation by muscle tendon vibration (MV) is also another means to influence tremor, as MV even up to vibratory frequencies of about 120 Hz ends up in muscle spindle activity".

B. First Experiment

Twenty-seven patients were investigated with moderate Parkinson's Disease resting tremor. The patients had the right forearm fixed just to allow the movement in the flexor-extensor direction of the wrist-joint.

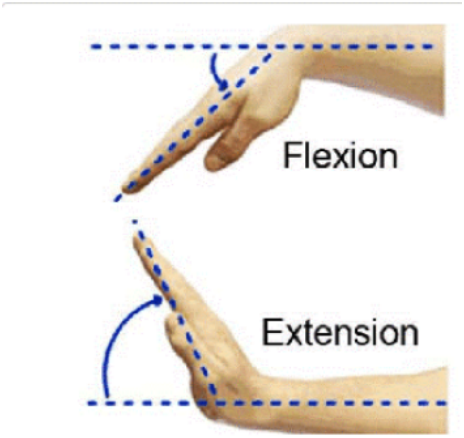


Fig-1. Flexor and extension wrist movement

Muscle vibration was applied between belly and tendon of M extensor carpi radialis longus and M. flexor carpi ulnaris by two vibrations. The frequency of 80 Hz and a peak pressure up to 2kg/m^2 was applied and was with the most effective result.

According to the medical article "Vibratory proprioceptive stimulation affects Parkinsonian tremor" from Elsevier Journal specialist in science and health, were performed five consecutive trials without interruption. Six consecutive sections in which trial followed this sequence: (1) 30 s without vibration, (2) 30s vibration of M.extensor carpi radialis longus, (3) 30 s without vibration and (6) 30 s vibration of both muscles. In the last five trials, the type (1) 'one' section was appended. In total the sequence of was section was:

(1),(2),(3),(4),(5),(1),(2),(3),…(5),(6),(1),(2),(3),(4),(5),(6),(1).

These pictures below show the muscles in these experimental sections for a better understanding:

Fig – 2. Etensor Carpi Radialis Longus

Fig – 3. Flexor Carpi Ulnaris

Electromyography results.

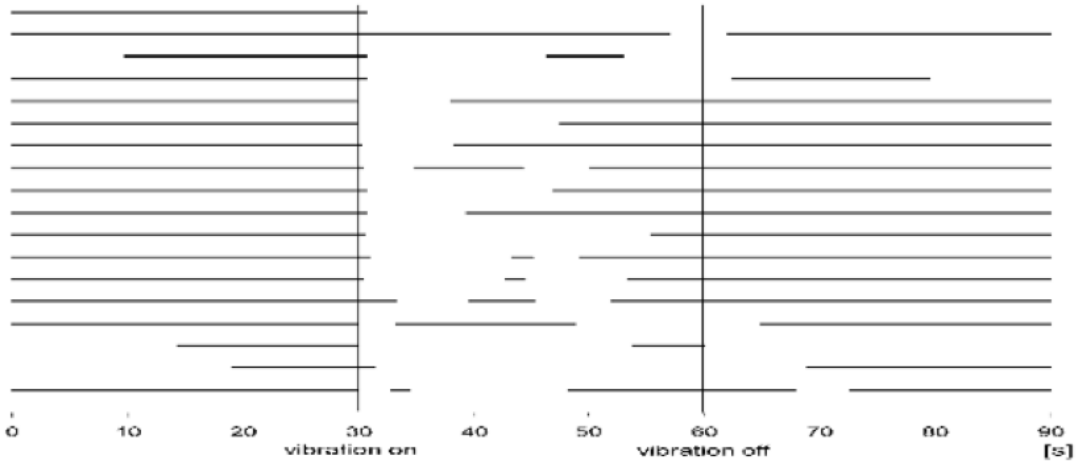


Fig - 4. Traces show vibration-induced tremor abolition. Horizontal lines indicate the tremor. The sequence was: 0-30s no MV (muscle vibration), 30 – 60s MV on, and 60-90s no MV.

In the first experiment, we could see some vibration frequencies that are being used to decrease the involuntary movement from the person who is suffering from PD. A second test was executed, keeping the same line related to induced vibration. This test shows vibrations that were applied to the patient's wrists. In the article called Proprioceptive Movement Illusions Due to Prolonged Stimulation: Reversals and Aftereffects published in 2008 and available on journals.plos.org says: "A widely-used tool for investigation of movement perception is tendon vibration; it activates muscle spindle endings and induces an illusory sensation of movement". Creating this illusion in the brain these vibrations can help the patients to hold objects while it decreases the tremors.

The neuro system department of the European Journal of Neuroscience published an article called "Temporal features of human tendon vibration illusions" in 2012. In this article, they mention "A recognized method for studying muscle spindle signalling is tendon vibration. Superficially vibrating muscle tendons at approximately 80 Hz stimulates muscle spindles, and induces illusions of altered joint position and movement (Goodwin et al., 1972; Burke et al., 1976; Roll & Vedel, 1982; Lackner, 1988)". They also say "Moreover, some individuals do not experience tendon vibration illusions, despite (apparently) having normal muscle spindle signals."

C. Second Experiment

In 1993 the Scandinavian Journal of Work, Environment & Health published an article called Contribution of the tonic vibration reflex to muscle stress and muscle fatigue.

The experiment was made to check the vibration displacement amplitude. Different vibrations (40,80,100,120,150,200 Hz) was analysed. Ten students were submitted in this vibration-induced muscle activity. The electromagnetic vibrator (Ling Dynamic System, type 203) while the person should hold the hand dynamometer for measuring the strength of the muscle contraction.

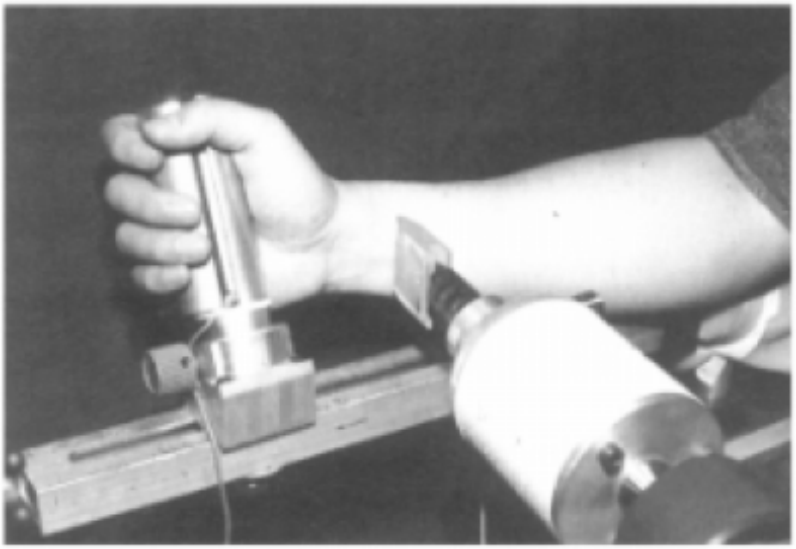


Fig 5- Vibration being applied to the muscle-tendon

In this particular experiment, during vibrations, the students reported a sensation of an increase of the grip force, but vibrations beyond 100 Hz did not have effective results.

D. Third Experiment

In the scientific article called “High-frequency peripheral vibration decreases completion time on a number of motor tasks” published in 2018 by European Journal Neuroscience where the vibration was applied to the wrist of the right arm while Parkinsonian patients had to deal with some simple activities.

The experiment consisted of 30 s of 80 Hz vibration or no vibration or a control condition (20hz) on the right wrist. The vibratory stimuli were delivered via an electromagnetic mechanical stimulator with a 3-cm-diameter circular probe under the palm wrist of the right hand. The vibration of 80Hz was applied with slight pressure on the wrist of the right hand.

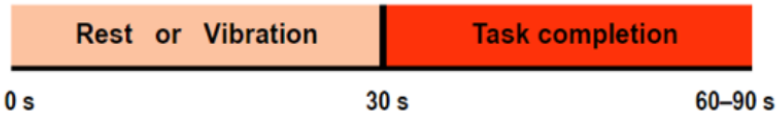


Fig – 6 Task Condition

The patients were submitted to do the following tasks:

(b) the box and blocks test;

(c) the nine-hole peg test;

(d) a reaction time task;

(e) 1 min right hand tapping test with the cybernetic glove;

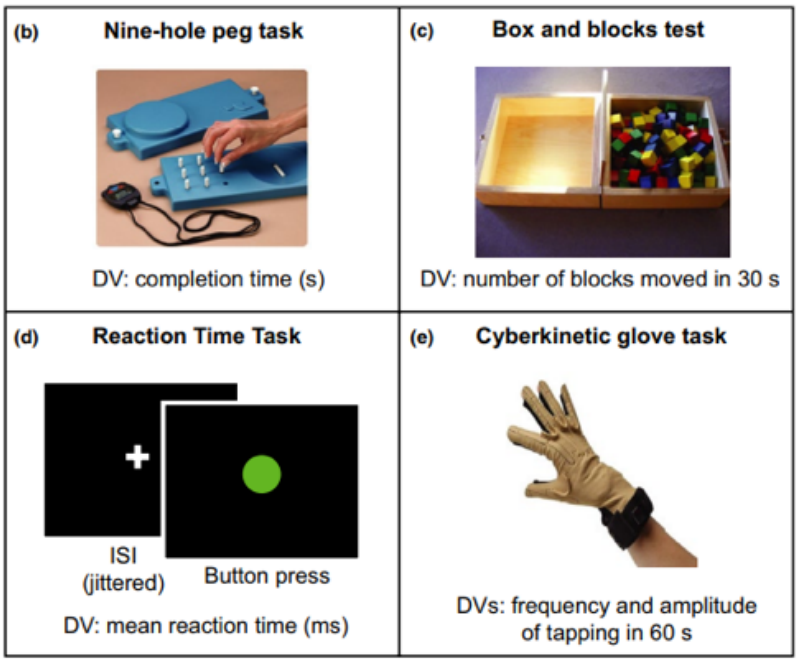


Fig-13 Tasks

Tasks descriptions

B - The Nine-hole peg task the patient needs to take the pegs from a container, one by one, and place them into the holes on the board, as quickly as possible.

C - In the Box and blocks test, the patient should grasp one block at a time with the dominant hand, transport the block over the partition, and release it into the opposite compartment.

D - Reaction time test is to check the time reaction of the patient. In this case, it will be measure in milliseconds how long it will take to the patient to press a button when it gets green on the screen. The patient should stay looking to the cross sign before hitting the bar button.

E - The cyber glove was used to measure the frequency and amplitude of tapping over 1 min of a time window. The tapping was between the first two fingers of the right hand.

The tasks were repeated three times and three different conditions:

(a) absence of vibratory stimuli;

(b) 30 s of 80 Hz vibration on the right wrist;

(c) 30 s of 20 Hz vibration on the right wrist.

Results of this test

With the results of this experiment, the article says: “Improved performance on all motor tasks (except the amplitude of finger tapping) was also seen for a sample of 18 PD patients ON medication ” and “ Overall, it's clear that peripheral vibration at 80 Hz improved motor performance on a spread of control tasks…” and below it says “In the present study, there was only a major effect of vibration on behavioural performance at 80 Hz.

Conclusion

Induction vibration has influenced the amplitude of the person who suffers from Parkinson's disease. However, in advanced cases, MV (muscle vibration) could not be properly applied. We could realise some frequencies of vibrations that are being used in those particular scientific experiments and understand in which part of the forearm it was applied. In the medical article “Vibratory proprioceptive stimulation affects Parkinsonian tremor” from Elsevier Journal specialist in science and health, some patients with moderate Parkinson's disease had their tremors abolished.

This research was built aiming for a better understanding of the action of the vibration on the limb and to have a close range of vibration to be applied. The group saw that is needed different frequencies for each patient to create tendon vibrations illusion and in some cases it may not be achieved. Although some of these experiments have used the same frequency of 80 Hz that does not prove that this is the pattern for all patients but we could see that with this frequency is possible to achieve great results. Therefore, we have concluded that allowing the user to control Park Med’s vibration is vital to the success of this project.

II. Raspberry Pi

The Raspberry Pi is a well known single-board computer developed by the Raspberry Pi Foundation in the UK. The main reason that they created was to promote basic computer science in schools. Nowadays, it has been used widely for research projects regarding many different subjects, such as robotics, monitoring weather, automation and so on.

As it has a low cost in the market, it has been chosen by the group to have this device as the "heart" of the project implementation by developing many experiments. The Raspberry Pi was used in the early stage of the development, which allowed the group to get familiar with programming, hardware and electronics before more sophisticated implementations were executed.

Raspberry Pi specifications

Chipset: Broadcom BCM2837
CPU: 1.2GHz quad-core 64-bit ARM cortex A53
Ethernet : 10/100 (Max throughput 100Mbps)
USB: Four USB 2.0 with 480Mbps data transfer
Storage: MicroSD card or via USB-attached storage
Wireless: 802.11n Wireless LAN (Peak transmit/receive throughput of 150Mbps), Bluetooth 4.1
Graphics: 400MHz VideoCore IV multimedia
Memory: 1GB LPDDR2-900 SDRAM
Expandability: 40 general purpose input-output pins
Video: Full HDMI port
Audio: Combined 3.5mm audio out jack and composite video
Camera interface (CSI)
Display interface (DSI)

For further information regarding specs of different models of Raspberry Pi: https://www.raspberrypi.org/products/

III. Circuit Playground Bluefruit

Circuit Playground Bluefruit (CPB) is a microcontroller board developed by Adafruit Industries, this board was found in the middle of the development phase with Raspberry Pi (Pi). Due to its size and shape, the group decided to buy one to replace the Pi. However, before the final decision was made, a member of the group carried out the same experiments previously done with the Pi to get familiar with the board. The decision came after it was possible to complete all the experiments in a much shorter time, which demonstrated the easy development with the newest board. The group concluded that the replacement would be beneficial to the project, and it was decided to continue the project implementation with the CPB instead of the Pi. In a called "Phase 2".

The specifications seen on CPB is nothing close to the Pi, but it has the necessary processing power required by Park Med. According to Adafruit, the newest board is the third generation of a microcontroller series intended to an educational and learning environment. Therefore it was intentionally developed to be simple and straightforward even for non-experienced programmers.

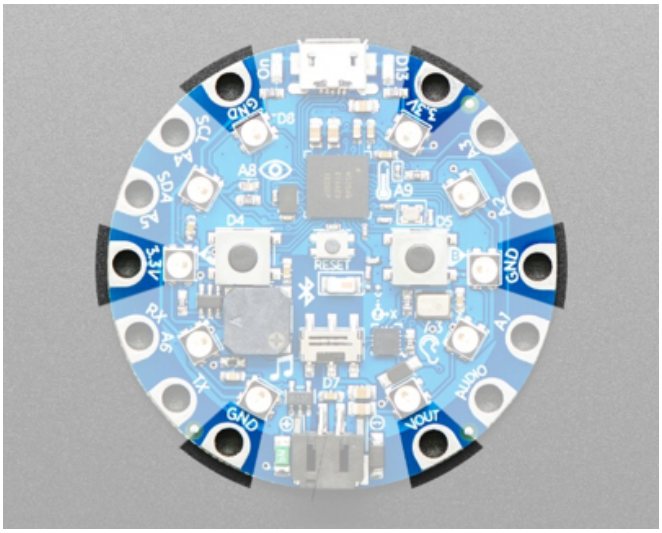
CPB comes with an nRF52840 chip which is not only powerful but also offers Bluetooth Low Energy support for wireless connectivity. It also has different built-in sensors: Motion, light, temperature and sound along with 2 built-in buttons and LEDs.

The board can be programmed using CircuitPython or Arduino. It is highly suggested to use CircuitPython for this project since it is a programming language designed to run on boards and has strong hardware support. As the name may suggest, CircuitPython is based on Python and has a similar syntax which is being used to develop on the Pi.

CPB Pinouts

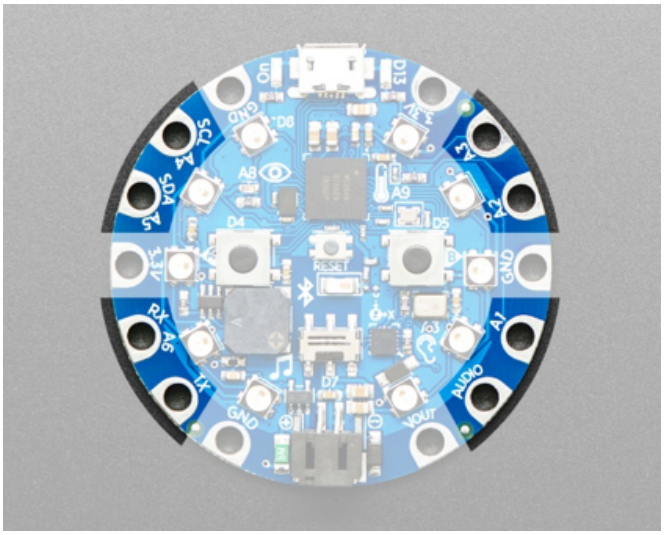
According to Adafruit's website, CPB comes with 14 pads, 8 GPIO and 6 power pads available.
Power pads:

- GND pads. There are three Ground pads all connected together and are all signal/power ground connections.
- 3.3v pads. There are two 3.3-volt output pads, and they are connected to the output of the board regulator. The regulator can provide about 500mA max, but that includes all the built-in parts so the energy budget should be no more than 300mA for functionality purposes.
- Vout pad. There is one Volt output pad. This is a special pad that will be connected either the USB power or battery input, it will decide for whatever has the higher voltage.



Input/Output Pads

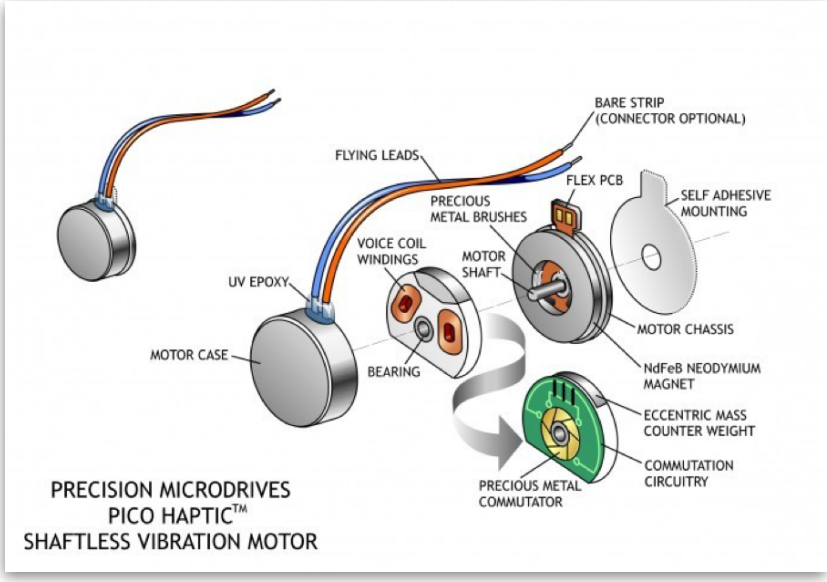
For purposes of this projects the GPIO pads will be used just for transmit the energy needed to the coin motors (coin motors will be explained further) Each pad can provide up to ~20mA of current. Do not connect a motor or other high-power component directly to the pins! All the GPIO pads are 3.3V output level and should not be used with 5V inputs. In general, most 5V devices are OK with 3.3V output though.



IV. Electrical Components

A. Vibrating Motor Discs

Vibrating motor discs is a form of ERM (eccentric rotation mass) motor. Different from Cylinder Coreless Motors, the thickness of Coin Vibration Motor is smaller, which makes it is such a coin. Moreover, that s why people call it a coin vibration motor generally. They are widely used in many designs because they have no external moving parts and should be fixed in place with a robust permanent self-adhesive mounting system which makes them perfect for purposes of this project. The motors will be attached to a wristband following a particular positioning so that its vibration will be enough to counter-react the hand-tremor.



Picture collected from https://www.precisionmicrodrives.com/vibration-motors/coin-vibration-motors/

Applications

A collaborator from Precision Microdrives(2019) has said that motor discs are great for haptics, particularly in handheld instruments where space is crucial such as:

- Mobile phones
- RFID scanners
- Industrial tools or equipment user interfaces
- Portable instruments
- Medical applications **(Useful for Bio Protech project)**

The full term 'Maximum Start Voltage' is that the lowest voltage that only can apply to the motor and still ensure that will start.

Coin vibration motors have high start voltage (compared to cylinder pager vibration motors) which must be considered in designs. Typically, it can be around 2.3v, and failure to respect this might cause motors not starting when the applying is lying in specific orientations.
This problem arises because, within the vertical orientation, the coin vibrating motor must force the eccentric mass over the right of the shaft on the initial cycle.

B. Transistors

According to a description provide on the portal The Engineer Project (TEP) the PN2222 transistor is "A conjoint NPN bipolar junction transistor which is used for common persistence less power intensifying or swapping circuits applications" (2017).

It means that through the PN2222 transistor, it is possible to control the power intensity outcome. For Bio Protech's project this will allow controlling the frequency that the motors vibrate, a mediator, as higher the power intensity, more intense the vibrations and vice versa.

Connection Circuit Playground – Transistor – Motor.

The connection is quite simple as soon as we have identified the right way to do it. The first thing would be to connect the negative end of the motor with the transistor's collector because that is the one that receives the current, the positive end of the motor is connected directly to the Vout pad on the CPB. After doing that, connect the base of the transistor to one of the GPIO pads. Lastly, the transistor's Emitter will be connected to the GND pad.

V. Wireless Connectivity

BLE (Bluetooth Low-Energy)

Bluetooth Low Energy (BLE) is a type of protocol regarding Bluetooth connectivity. Let us start with the fact that Bluetooth connexions offer a range of distance between 1 and 30 meters depending on the power of the transmission, as longer the distance, more power required, therefore, more energy consumption.

Nowadays most of the times it is not needed such a long distance, assume. Nokia developed the first version known as Bluetooth Low-End Extension, which then became WiBee.

Due there was already a Bluetooth developer standard know as Bluetooth Special Interest Group (BSIG) they take in the WiBee and BLE born in 2010 to be integrated inside to the protocols of the Bluetooth 4.0.

BLE fundamentals are the reduction on the distances range and power consumption, therefore minimise the transmission power and energy intake maximising battery's life, which makes it perfectly suitable for wearable devices, smartphones and short distances Bluetooth connections. With the advances on this technology, it is possible to have a distance range up to 100mts under the BLE consumption.

Inside of the BLE technology

- Physical Layer. This has all the communication circuits able to modulate the analogic signals and subsequently transform them into digital symbols. The BLE tech is able to use up to 40 channels of 2 MHz through the 2.4GHz band. This standard applies the Frequency Hopping technique, which consists of a sequence of random hoppings among the channels providing robustness against interference.

- Link Layer. It is in charge of managing characteristics such as the standard's temporary requirements, message checking and forwarding of received erroneous messages, management and address filtering. It also defines roles (Advertiser, Scanner, Master and Slave) that allow to logically identify the role of each device in the communication process. The LL level is similarly responsible for control processes such as changing connection parameters or encryption.

- Logic Link Control and Adaptation Protocol (L2CAP Layer). It is responsible for two fundamental tasks in the communication process. First, the multiplexing process, that is, the ability to format messages from the upper OSI layers and encapsulate in standard BLE packets, as well as the reverse process. On the other hand, fragmentation and recombination. Packets that at the application level suppose large byte datagrams are fragmented correctly, conforming to the BLE MTU (27 bytes of maximum payload).

- The L2CAP layer is also in charge of giving access and support to the two fundamental protocols. On the one hand, ATT (Attribute Protocol), a protocol based on attributes presented by device, with client-server architecture, which allows the exchange of information. On the

other hand, SMP (Security Manager Protocol), a protocol that provides a framework to generate and distribute security keys between two devices. At the highest level of the protocol layer, we will find the GAP and GATT layers in parallel. This first, GAP (Generic Access Profile), allows a device to be visible to other devices and also determines how one device can interact with another. Sets different rules and concepts to standardise lower-level operations. On the other side, GATT (Generic Attribute Profile), which defines how two BLE devices transfer information. This process takes place when two devices have passed the phase of establishing communication or the famous Pairing (controlled by GAP), and the transfer of information begins, being able to be bidirectional.

Universal Asynchronous Receiver/Transmitter (UART)

It is a critical device of a serial communication system, the most popular protocol used for talking to another device over a serial port. The primary function is to convert serial to parallel when it comes to receiving data and the opposite for transmitting. It also provides other functions such as the handshake signals necessary for interfaces, as well as audio, video and game transmissions by Bluetooth technology.

The parameters used to define the correct design of a UART communication interface are the following:

- Synchronisation in data transmission is carried out by first placing a start bit, then data is sent starting with the least significant bit and lastly, a stop bit is sent. An error detection or starting bit is also added, and this is done sequentially until the transmission is complete.
- Data encoding. The encoding can be any binary code. The most used is the ASCII code that uses 7 bits to encode 96 printable characters and 32 control characters.
- Transmission rate. It is measured in the number of bits that are transmitted per second (bps), and in the UART component, it could be from 4807 to 31250 depending on watch frequency.

One difference from the standard Bluetooth that provides the Serial Port Profile (SPP), Bluetooth Low Energy offers more profiles, but the most regularly used is GATT. GATT exposes characteristics and attributes which are a little like variables that can be read from and written to an external device. Writing and reading data is usually limited to 20 bytes.

VI. CircuityPython and Mu Editor

Adafruit Industries was founded to offer the best place online for learning electronics and making the best-designed products for makers of all ages and skill levels (L. Freid, 2005). Circuit Playground Bluefruit (CPB) is not an exception, due to its educational purpose, CPB is a versatile piece of hardware. It can be programmed in C for Arduino or CircuitPython. Because of the familiarity with Python, which is the language we have been using on Raspberry Pi, we will be programming CPB with CircuitPython.

CircuitPython is an open-source programming language designed to simplify experimenting and learning to code on low-cost microcontroller boards (K. Rembor, 2017). The most significant advantages of choosing CircuitPython over other languages were explained by Kattni Rembor and Limor Freid(2017), they mentioned the strong hardware support, the educational purpose, the possibility of data-logging and that the code does not need to be compiled by an IDE before being uploaded to a microcontroller. It is crucial to mention that for each microcontroller, there is a different version of the language package. In January of 2020, the latest one for CPB is the CircuitPython 5.1, the best approach for the project is to be up to date regarding the language, we are aware that we may need to download and update to a newer version halfway through the implementation.

Although it is not required to compile the code before uploading it into a microcontroller, doing so can help to debug and troubleshoot any error encountered while testing the code. Any code editor that supports Python is also capable of running CircuitPython. As we will be following and making use of the excellent support provided by Adafruit Industries, we have decided to go along with the suggestions given on numerous tutorials found on their learning website. The python code editor recommend called Mu-editor.

Mu-editor is an open-source code editor created and mainly written by Nicholas Tollervey, nowadays it is maintained by a considerable community of developers. Mu-editor was created following four principles; less is more, make it simple, walk the path of least resistance and have fun. Therefore, teachers, kids and beginner programmers can start coding on it without any obstacle. In other words, developing on Mu-editor is very straightforward and easy.

Raspberry Pi Foundation has supported the work done by Nicholas Tollervey, the result of that could be seen when we were implementing the early stage of phase 1, and we needed to install Raspbian on our Raspberry Pi, Mu-editor was one of the python editors installed by default with Raspbian. However, we will be using mostly the Mu-editor version for macOS, which should not have a huge discrepancy from other operating systems' versions.

VII. Android System

Android Operating System was created by Android Inc., a Californian company that was bought by Google in 2015. However, since 2018 Android is an open-source platform dedicated to mobile applications and belongs to a consortium consisted of leading tech companies such as Google, Motorola, Samsung, Qualcomm and many others (J, Callaham, 2019).

Android 1.0 version was released in 2008 still under Android Inc. domain and its latest version was released in 2019 called Android 10 or Android version 10.0. When a mobile application is being developed, it is necessary to choose a minimum android version target. Developing an Android 10 application offers features that can be crucial to projects. However, it will not be compatible with previous versions. Even though Android 10 is the latest and most advanced one, its presence in mobile devices is trivial compared to other versions. It is desirable to develop an application that is compatible with as many devices as possible to offer Bio Protech's device to a substation number of PD patients. According to a chart published by Statista.com, the best balance between compatibility and features can be seen when developing Android 6.0 applications, reaching over 80% of android devices and offering reasonable and updated features.

3. System Analyses and Design

I. Description of the Device and App.

The purpose of this report is to represent and analyse Bio Protech's Park Med device from a System Analysis and Design (SaaD) perspective by identifying the system requirements and how users will be able to interact with it to achieve its functionality from a high-level perspective to a lower one. **Please note that this report shall not cover medical or a precise range or intervals of frequency need it.**

The aim of the Bio-Protech working team is the development of a Park Med device to emit regulated vibrations aiming to counter the tremors caused by Parkinson's Disease. This vibration will be controlled by Bio Protech's mobile application, through which the user will be able to control the Park Med entirely.

The user will be able to start vibration by selecting the frequency desired by pressing the relevant "button" on the screen, and also it will be able to turn it off when the user wishes to.

Bio-Protech Park Med will have motors attached to it with an Adafruit Circuit Playground Bluefruit (CPB) built-in. This CBP will receive external power supply from power bank connected by USB and also is going to receive users input through the Bio-Protech App via Bluetooth Low Energy technology (BLE)

II. System Requirements

- The Park Med should turn on once it is connected to the power bank. The default behaviour should be null; this means no vibrations waiting for the input of the user.
- The Park Med should have an Adafruit CPB built-in to be able to communicate to the Bio-Protech App.
- The Bio Protech App installed on the smart device (e.g. a smartphone) should communicate to the Park Med via BLE.
- The Park Med should start a Bluetooth Advertise so it could be detected by user's smart device and therefore by the Bio Protech App.
- The Bio Protech App will display a "home Screen". The home screen will consist of:
    - A Menu Button at the top of the screen. If the user taps this button, it will display the features of the app – these features will be explained further on this section.
    - A Connectivity Indicator. This indicator will illustrate to the user whether the app has established a connection with the Park Med Band or not.
    - The following what the user will be able to see is the vibration frequency select represented by a decimal number and Hertz (Hz), e.g. 120Hz, where the number represents the intensity and hertz the unit of measurement of frequency according to the International System of Units.
    - On this screen, there will be displayed a frequency regulator from the bottom to the top. The regulator consists of oval-shapes segments, and each of them represents different levels of vibrations. It will be controlled with a swipe up and down movement.
    - Start/Stop button: This button will be displayed at the very bottom of the screen just below the, if the Park Med is in non-motion at the moment of the user press this button, the Park Med will start to vibrate in the lowest range of frequency, if the Park Med is currently vibrating at the moment the user presses this button it will stop gradually.

- Menu Displayed: If the user presses the menu button, a Menu Screen will be displayed with the following options:
    - **A Home Button**. This button will take back the user to the home screen.
    - **Connect Button**: This button will initialise the connectivity process between the Bio Protech App and the Park Med. Like any other Bluetooth device connection, whether or not the connection is successfully achieved, the app will this display a legend confirming either both cases. With the CBP integrated on the Park Med, there is no need to do the "Pairing" process with the smartphone.
    - **Graph button:** This button will display an Accelerometer Graph to show the frequency of the tremors that the Park Med will receive from the user through motion sensors the CPB has built-in (integrated into the Park Med), so the user has a visual idea or parameter of how the vibrations worked on the tremors. Also, it will be displaying an option to save the graph on its smart device as an image (e.g. smartphone' memory).
    - **Settings button**: This button will provide an option to register a new Park Med. Assuming a hypothetic situation where either, the current Park Med breaks, get lost, stop working, or any other issue that forces the user to get a new one. This is because Bluetooth connectivity works with MAC addresses.
    - **Info button**: At the moment the user taps this button, it will display on the screen a user-manual with a step-by-step tutorial of how the app and Park Med works.
    - Any of these screens and options will have the navigation menu so the user can navigate free around the app.
- The Bio Protech app should send the user's input to the Park Med.
- The Park Med should receive and read the user's input, then vibrate according to the frequency selected by the user.
- The Park Med should be able to register the tremors of the user and store them for a short period and send them to the Bio Protech app.
- The Bio Protech app should be able to receive and read the tremors data, then be able to represent them on an Accelerometer Graph on the same Bio Protech app.
- The Park Med should have the coin motors integrated around the wrist without being harmful or uncomfortable to the user.
- The Park Med should be comfortable to wear and harmless for the user.
- No validation required to be as user friendly as possible.

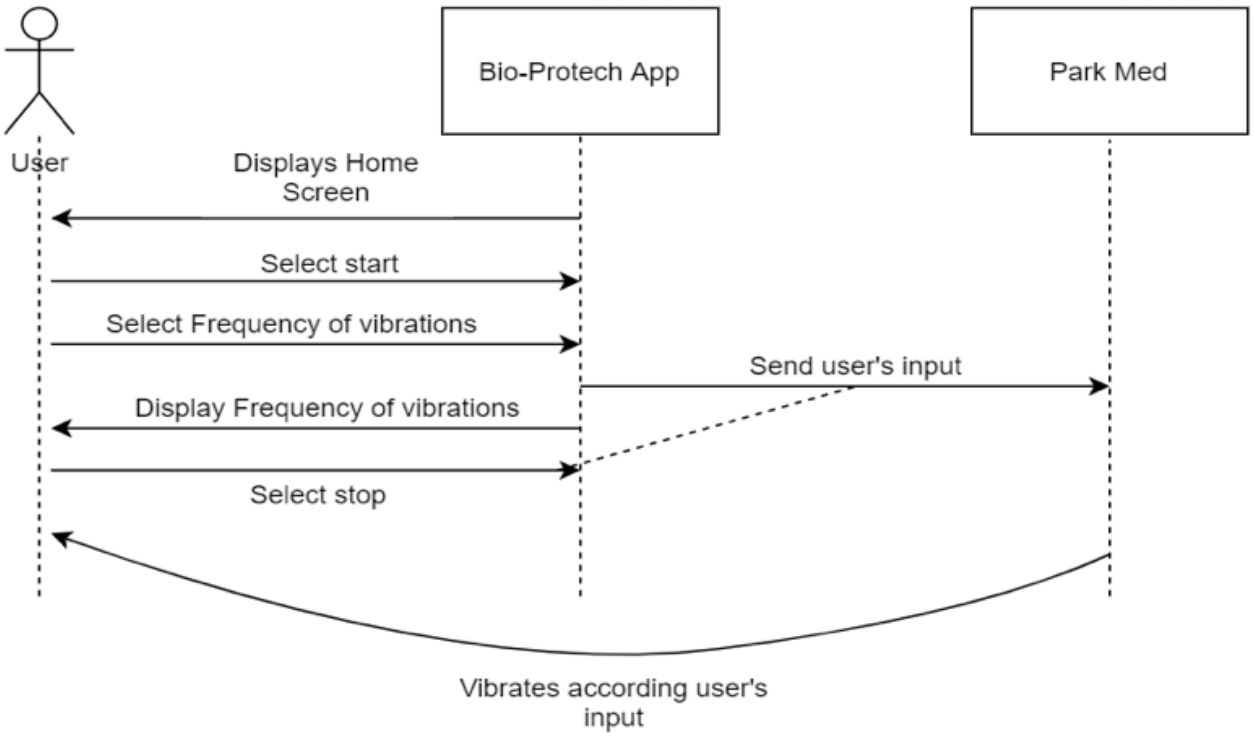
III. Use Cases

A. Selecting and stopping vibrations Use Case Scenario

In this section will be represented how the user will interact with the app to activate or stop vibrations.

- The app will display the home screen previously explained.
- User shall press start; vibrations will begin from the lowest range.
- The user will have the option to select another frequency with the help of the ovals.
- Bio-Protech app will send the user's input to the Park Med.
- Bio-Protech App will display option selected.
- User can if decide it so stop vibrations.
- Park Med shall vibrate according to the user's input if the input is Stop vibrations will be null.

The following diagram represents what has been mentioned above

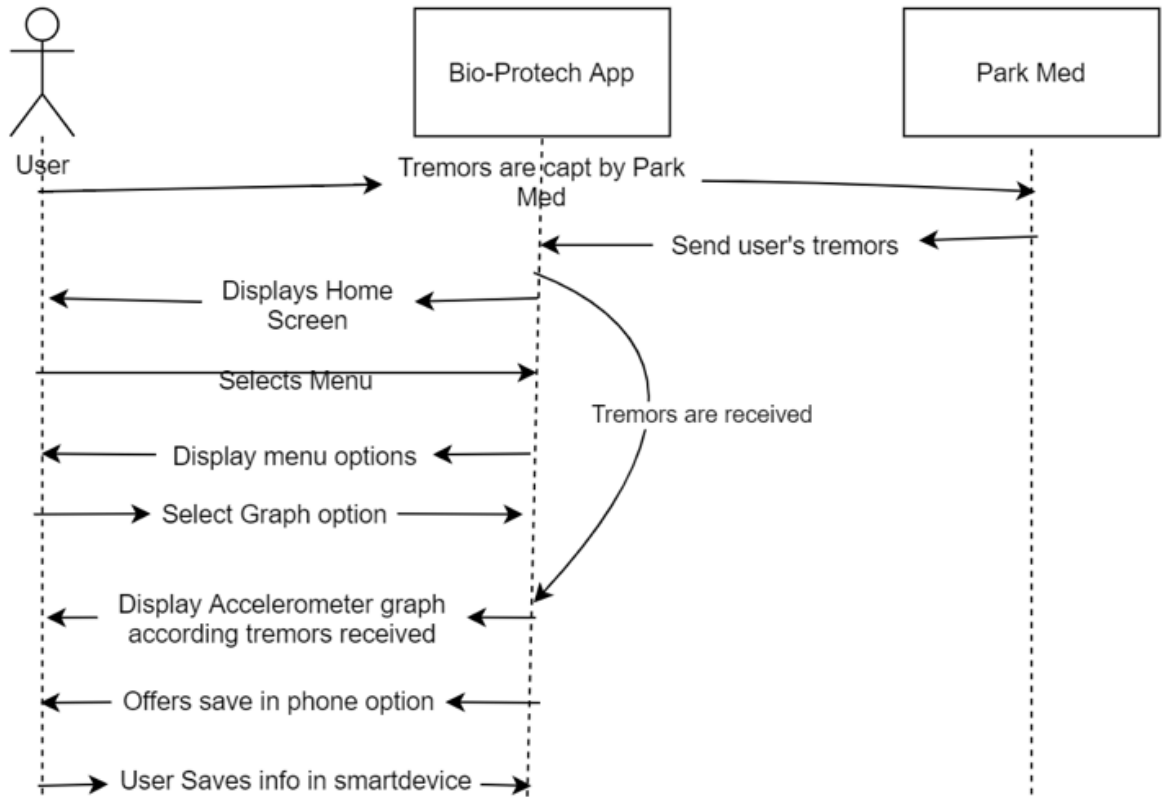


B. Receiving and saving Tremors frequency

In this section will be represented how the user will interact with the app and the Park Med to be able to see the Accelerometer graph representing how the tremors response to the vibrations after the user starts the vibrations.

- The Park Med will receive on monitoring the tremors form the user through its built-in motion sensor.
- It will send the information to the Bio Protech app, and it will hold the information.
- The user will have to press the menu button, the app will display the options available and then the user will have to select the Graph option.
- The app will represent and display the information visually received from the Park Med for the customer through an Accelerometer Graph. Also, there will be an option for the user to store the graph in its smart device.
- The graph will be stored as an image in the device.

The following diagram represents what has been mentioned in this section.



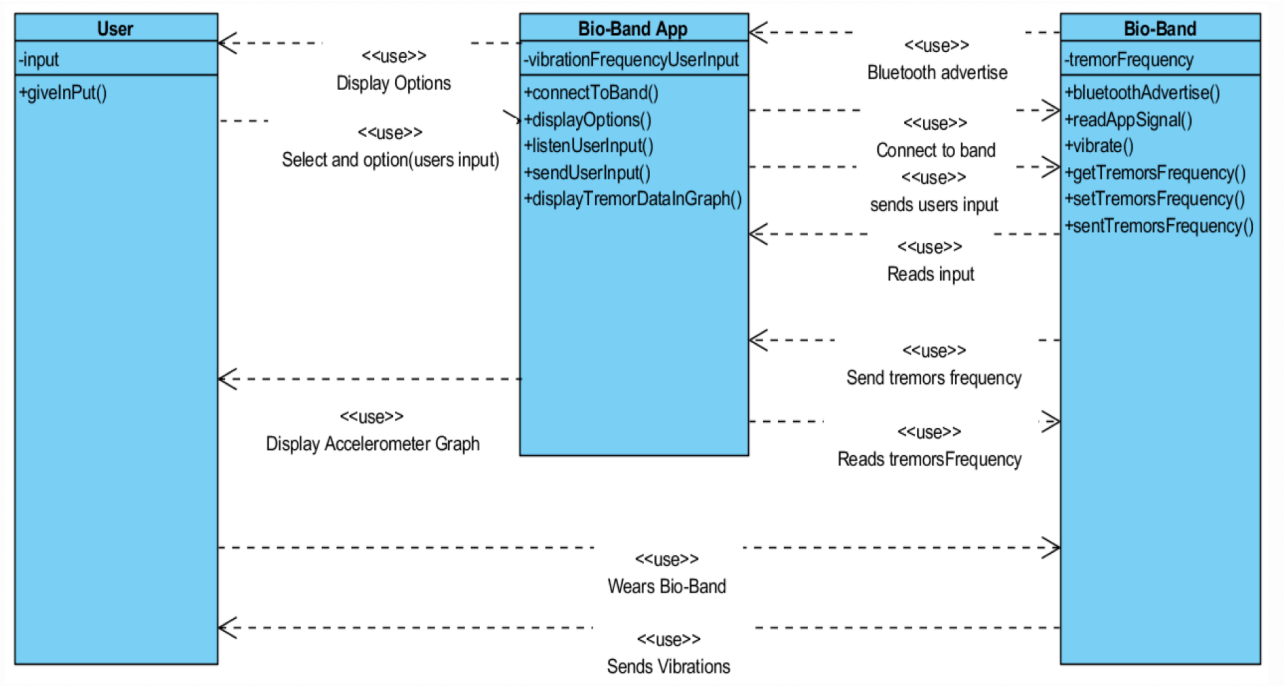
IV. Class Diagrams

Bio Protech Class Diagram User – Bio Protech's App – Park Med

The intention of the following class diagram is modelling the user's interaction in a lower level scale. The user will provide an input through the Bio Protech's app, that input will be processed by the app and send it via Bluetooth to the Park Med, the Park Med will receive and read the signal coming from the app and start the vibrations according to the user's input if nothing has been selected by the user, the default behaviour will be null.

The Park Med will also be monitoring the frequency tremors coming from the user by a motion sensor built-on the Adafruit CPB, then it will send the tremors data collected to the Bio Protech's app which should receive it, read it and present it on the accelerometer graph.

This report guides how the System is expected to work regarding how the user will be interacting with the Park Med and its App. It also offers guidance to the users through use case scenarios in the most 2 important functionalities this project is required and expected to do. As we can appreciate it, is an easy straight forward application/device due to the need to be for the users to whom this application is addressed.



V. User Interface

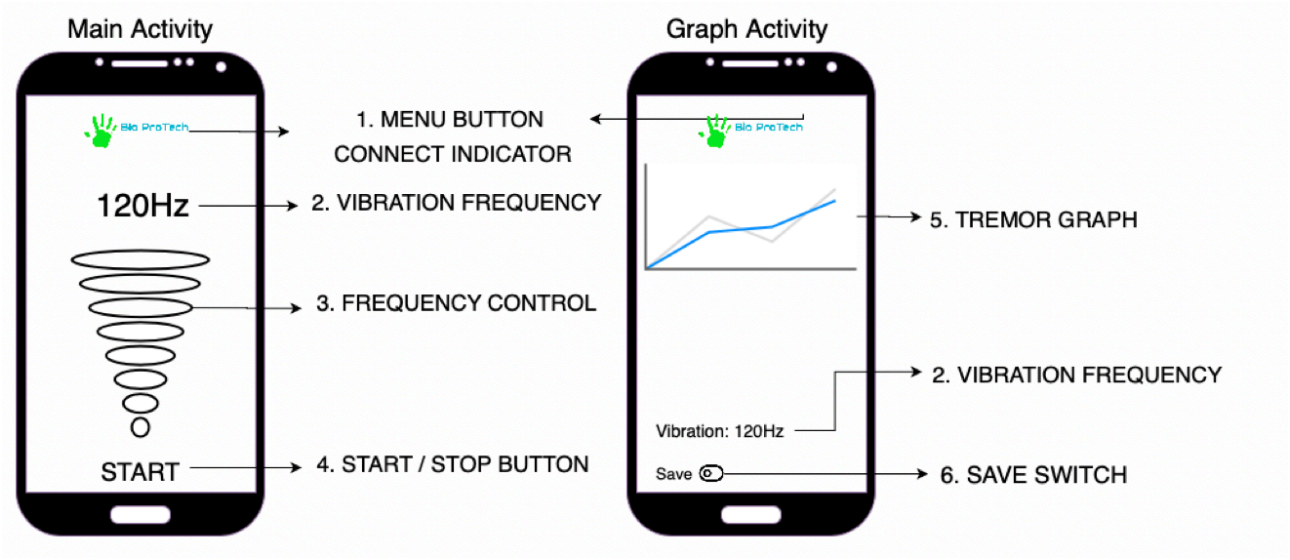
Mobile Application Interface

Bio Protech's app is developed to allow users to connect and control a Park Med device, meeting all the specified requirements. The application must be as user-friendly as possible, simply offering all functionalities.

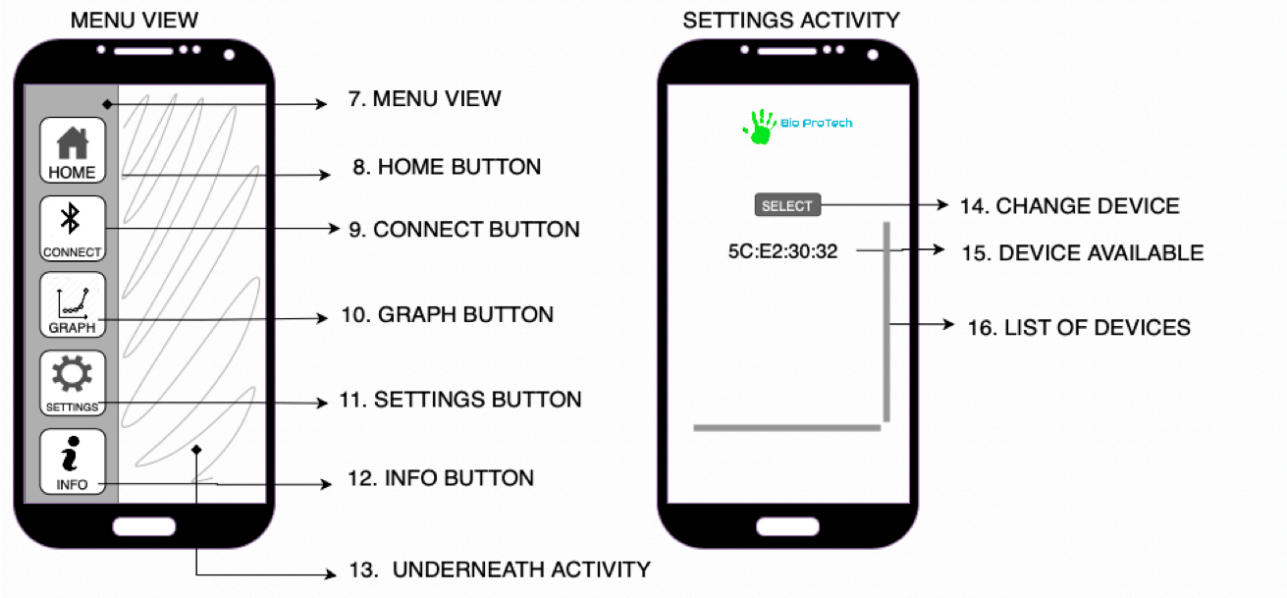
Considering that it will be mostly used by Parkinson's Disease patients. There are two main design guidelines that we must follow:

- Size: Due to their tremors, users may experience difficulties while doing simple tasks, including pressing a button on a smartphone screen. In order to avoid any barrier that will affect Parkinsonian patient to use the mobile application, we must design buttons with suitable surface size.

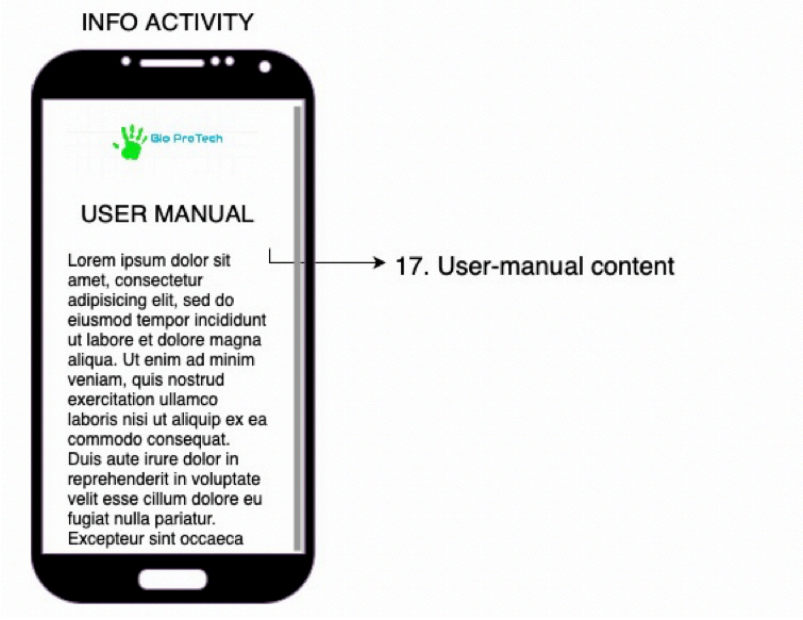
- Colours: According to ParkinsonsDisease website (2019), there are several kinds of visual disturbances that may be experienced by people with Parkinson's. Therefore, the colours on Bio Protech's app must have a big tonal contrast to facilitate the readability.



Mock-up of the Main Activity on the left and the Graph Activity on the right. It shows names of different elements within the layouts. Mock-ups created on Draw.io.



Mock-up of the Menu Demo on the left and the Settings Activity on the right. It shows the names of different elements within the layouts. Mock-ups created on Draw.io.



Mock-up of the Info Activity. It show names of different element within the layout. Mock-ups created on Draw.io.

1. Menu Button / Connect Indicator

This is Bio Protech's logo. It will work as a clickable image to open a side menu (item 8). The hand drawing will change colour to indicate whether Park Med is connected or not.

2. Vibration Frequency

It is intended to display the current frequency that motors on Park Med are vibrating. It will dynamically change conforming the Frequency Control (item 3) moves.

3. Frequency Control

It is a slider intended to meet the minimum requirements for the project. It will allow users to regulate the frequency that most suits them. If the user slides the finger upwards, the frequency will be incremented, and if it slides downwards, the frequency is decremented. When the user releases the slider, the new frequency is sent to Park Med by Bluetooth. It is set to a limit of 180Hz and a minimum of 20Hz conforming the limitations of the motor discs. The slider was designed to simulate different levels of frequency in a graph and to offer a visual representation of levels by colours.

4. START / STOP button

This button is intended to allow the user to START and STOP the vibrations. It is a button that changes the text according to the vibration status. If Park Med is vibrating, it will show STOP and if it is not, it will show START. When the user presses the button, a start/stop command and the frequency displayed (item 2.) is sent to Park Med via Bluetooth.

5. Tremor Graph

The graph will allow the user to analyse his/her tremor intensity represented in a graph. This graph is plotted in real-time with data coming from Park Med's accelerometer sensor via Bluetooth (y-axis). The time frame (x-axis) will be set for one minute. After one minute, the graph will be plotted following a FIFO order.

6. Save Switch

This switch allows the user to choose if he/she wants to save the data from the graph (item 5). The data is stored in the internal memory of the smartphone.

7. Menu View

This is a navigation view with five different buttons that will be displayed on top of the current activity. It will open when the user pressed the Menu Button (item 1) and closed when any area outside the menu view is pressed.

8. Home Button

It sends the user to the Main Activity. If the user has the Main Activity open, the button will only hide the menu.

9. Connect Button

As per the requirements, the user must have the option to connect the Bio Protech's app to a Park Med device. If the connection is already established, the connect button is changed to a disconnect button.

10. Graph Button

It sends the user to the Graph Activity. If the user has the Graph Activity open, the button will only hide the menu.

11. Settings Button

It sends the user to the Settings Activity. If the user has the Settings Activity open, the button will only hide the menu.

12. Info Button

It sends the user to the Info Activity. If the user has the Info Activity open, the button will only hide the menu.

13. Underneath Activity

The current activity that is being displayed under the Menu View (item 8). If the Menu View is open, the user will not be allowed to make any changes to the underneath activity. However, if he/she tries to make changes, the Menu View will be closed. Eg. If the user tries to change the frequency while the Menu View is open, this action will close it.

14. Change Device

This button triggers the Bluetooth scanner trying to find nearby Park Med devices.

15 - 16. List of devices available

This is a list of Park Med devices available to connect. The list is populated for each Park Med found by the Bluetooth scanner.

17. User-manual

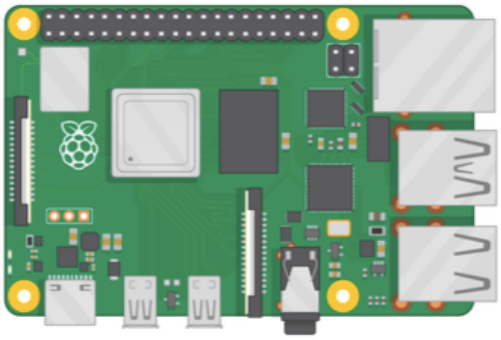
This a step-by-step guide of how to use Park Med devices and Bio Protech's app.

4. Implementation

I. Phase 1 - Raspberry Pi

A. Setting up Raspberry Pi

There are several models of Raspberry Pi, and for most people, Raspberry Pi 4 Model B is the one to choose. Raspberry Pi 4 Model B is the newest, fastest and easiest to use. Raspberry Pi 4 comes with either 1GB, 2GB, or 4GB of RAM. For most educational purposes and many hobbyist projects, 1GB is enough; for use as a desktop computer, we recommend 2GB.



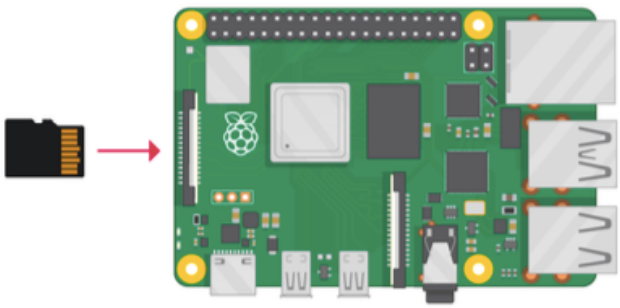
- **Power Supply**

To connect to a power socket, all Raspberry Pi models have a USB port (the same found on many mobile phones): either USB-C for Raspberry Pi 4 or micro USB for Raspberry Pi 3, 2 and 1.

- **A MicroSD**

The Raspberry Pi needs an SD card to store all its files and the Raspbian operating system.

It is required a microSD card with a capacity of at least 8 GB. Many sellers supply SD cards for Raspberry Pi that are already set up with Raspbian and ready to go.



- **A keyboard and a mouse**

To start using the Raspberry Pi, it is needed a USB keyboard and a USB mouse for the first setup.

- **A TV or computer screen**

The screen can be a TV or a computer monitor. If the screen has built-in speakers, the Pi is able to use these to play sound.

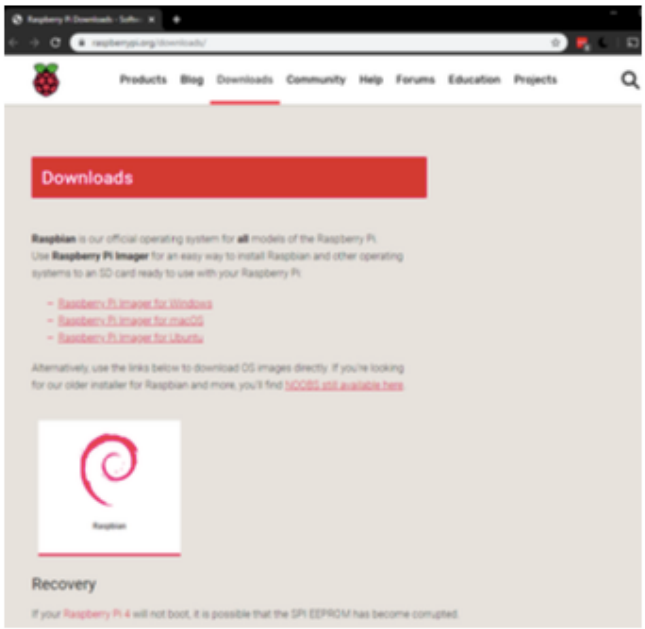
- **HDMI**

The Raspberry Pi has an HDMI output port that is compatible with the HDMI port of most modern TVs and computer monitors. Many computer monitors may also have DVI or VGA ports.
It is needed either a micro HDMI-to-HDMI cable or a standard HDMI-to-HDMI cable plus a micro HDMI-to-HDMI adapter, to connect Raspberry Pi to a screen.

- **An Ethernet cable**

The large Raspberry Pi models (but not Pi Zero/Zero W) have a standard Ethernet port to connect them to the internet; to connect Pi Zero to the internet, it is needed a USB-to-Ethernet adaptor. Raspberry Pi 4, 3, and Pi Zero W can also be wirelessly connected to the internet.

The Raspbian operating system via the Raspberry Pi Imager



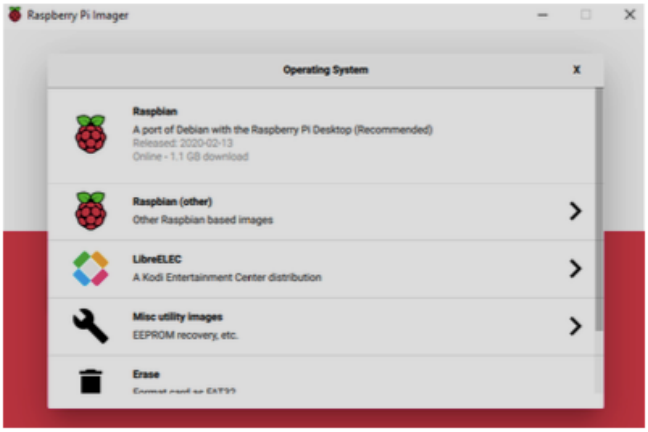
Using the Raspberry Pi Imager, which can be found on the Raspberry Org's website, is the easiest way to install Raspbian on the SD card.

- Click on the link for the Raspberry Pi Imager that matches the operating system of the desktop being used to download it.
- When the download finishes, click on it to launch the installer.
- Follow the instructions to install and run the Raspberry Pi Imager.
- Insert the SD card into the computer or laptop's SD card slot.
- In the Raspberry Pi Imager, select the OS that will be installed and the SD card to install it.

Using the Raspberry Pi Imager

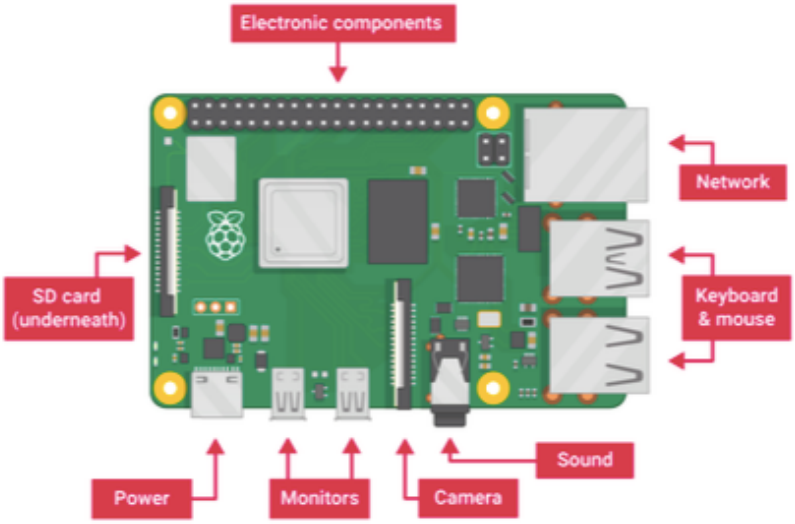
Anything that is stored on the SD card will be overwritten during formatting. So if the SD card on which the Raspbian will be installed has any files on it, e.g. from an older version of Raspbian, it is recommended to backup these files first to not lose them permanently.

- Follow the instructions to install and run the Raspberry Pi Imager.
- Insert the SD card into the computer or laptop.
- In the Raspberry Pi Imager, select the OS that will be installed and the SD card to install it.

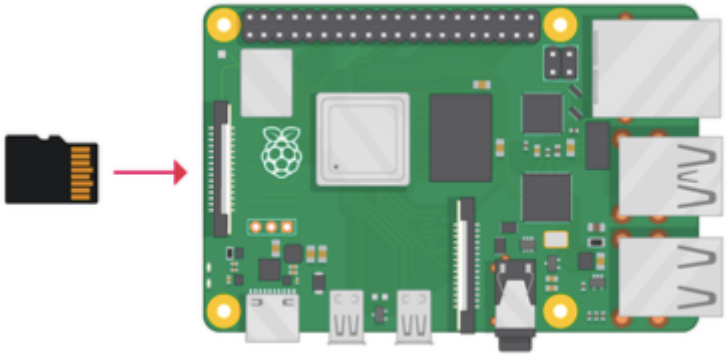


Connecting the Raspberry Pi

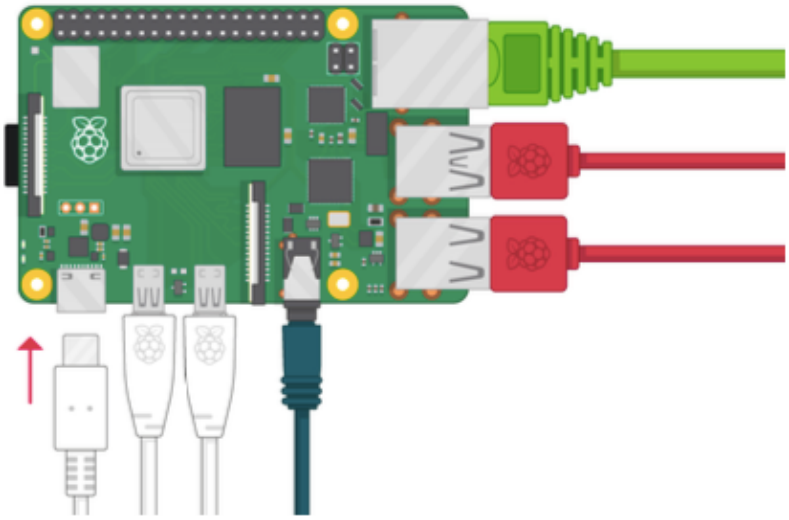
Now get everything connected to the Raspberry Pi. It is important to do this in the right order so that all the components will not suffer any damage.



- Insert the SD card with Raspbian (via NOOBS) into the microSD card slot on the underside of your Raspberry Pi.



The Raspberry Pi does not have a power switch: as soon as it is connected to a power outlet, it will turn on. Plug the USB power supply into a socket and connect it to your Raspberry Pi's power port.



After a few seconds the Raspbian desktop will appear.

B. Raspberry Pi Configuration Experiment

Required List of components:

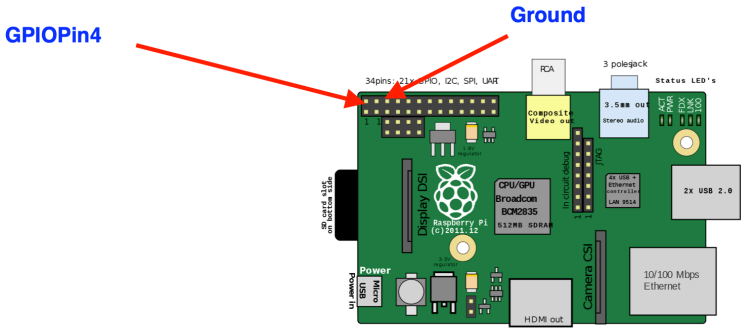
a. 1 - Raspberry Pi device with a 5 V power supply.
b. 1 - Bread Board
c. 1 - Male to Female jumper cable
d. 1 - LED

I. Firstly, connect the GPIO pin to the breadboard as shown in the figure.

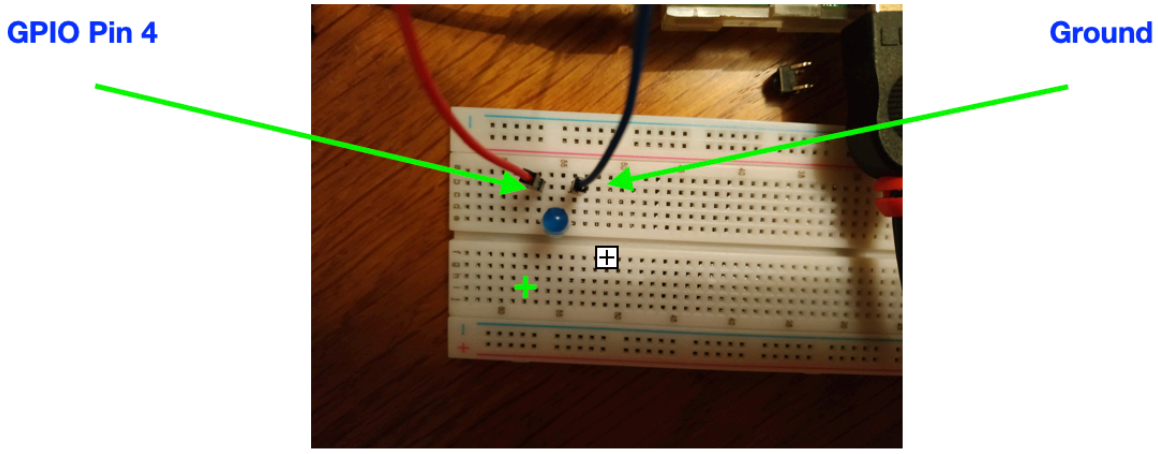
Pin#	NAME	Connection		Connection	NAME	Pin#
01	3.3V			5V (Cupcade)	5V	02
03	GPIO 2			5V (Powerboost)	5V	04
05	GPIO 3			GND (Powerboost)	Ground	06
07	GPIO 4	START			GPIO 14	08
09	Ground	GND (Cupcade)			GPIO 16	10
11	GPIO 17	UP		SELECT	GPIO 18	12
13	GPIO 27	DOWN		GND (Select/Start)	Ground	14
15	GPIO 22	LEFT		RIGHT	GPIO 23	16
17	3.3V			A	GPIO 24	18
19	GPIO 10	B		GND (ABXYR)	Ground	20
21	GPIO 09	X		Y	GPIO 25	22
23	GPIO 11	L Shoulder		R Shoulder	GPIO 08	24
25	Ground	GND (L)			GPIO 07	26
27	ID_SD				ID_SC	28
29	GPIO 05				Ground	30
31	GPIO 06				GPIO 12	32
33	GPIO 13				Ground	34
35	GPIO 19				GPIO 16	36
37	GPIO 26				GPIO 20	38
39	Ground				GPIO 21	40

Connect the GPIO pin to the breadboard

II. For this experiment, it was used the PIN Number: "4" (As per number 4(pin rank 7), third from the top – left). For getting the output from the Raspberry PI connect the jumper wire from Raspberry PI GPIO pin "4" to the breadboard as shown in the image.



III. Now take LED and put it on the bread board as shown in the image, connect the plus terminal of LED to the jumper cable (blue cable in image) coming from the GPIO Pin "**4**" as shown in the image.



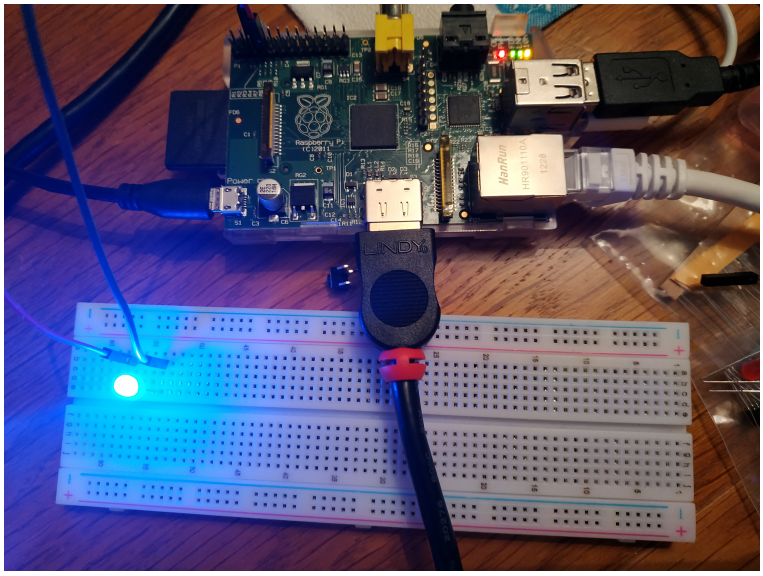
Connect the Plus terminal of LED to the jumper cable

IV. Now write the python program for blinking LED every 1 second.

```
1. import RPi.GPIO as GPIO
2. import time
3. #assign numbering for the GPIO using BCM
4. GPIO.setmode(GPIO.BCM)
5. #assingn number for the GPIO using Board
6. #GPIO.setmode(GPIO.BOARD)
7.
8.  cnt = 0
9.  MAIL_CHECK_FREQ = 1 # change LED status every 1 seconds
10. RED_LED = 4
11.
12. GPIO.setup(RED_LED, GPIO.OUT)
13.
14. while True:
15.
16.     if cnt == 0 :
17.
18.          GPIO.output(BLUE_LED, False)
19.          cnt = 1
20.     else:
21.          GPIO.output(BLUE_LED, True)
22.          cnt = 0
23.
24.     time.sleep(MAIL_CHECK_FREQ)
25.
26.          GPIO.cleanup()
```

V. Now run the program and the following output can be seen:

This simple experiment has been done in order to get familiarised with Raspberry Pi, and it was use da LED to simulate intervals time to blink the LED as we did not have vibrating motor discs on that opportunity. The blinking LED would be the same approach as the motor vibrating.



C. Simulation Motor Vibration in Python I

The code line itself has been created in order to create a background concept at how it would acts. It was offered a CLI menu to interact with the system. Through the menu, it is possible to start the functions and see the results in the CLI output.

In order to make all function effectively, we had to use concepts such as conditions such as if, elif and else statements and also While loops. So far the program only starts, access the menu and the function “vibrate” works. The next step is to find out a way to stop it manually by using any key from the keyboard. This would be possible adding a sort of external library installing pip install, and using a function called keyListener.

Program functionality

This is a basic program which provides the main menu with six options on it. The feature that the program meant to do is provided below:

```
def menu():
    print("1. Switch on")
    print("2. Vibrate")
    print("3. Stop")
    print("4. Regular")
    print("5. Random")
    print("6. Quit/Exit")


input("Press Enter for menu")
pass

command = int
started = False

print("""
      ____________________________________
      |                                  |
      | Press 1 - to switch on           |
      |__________________________________|
      |                                  |
      | Press 2 - to vibrate             |
      |__________________________________|
      |                                  |
      | Press 3 - to stop the vibration  |
      |__________________________________|
      |                                  |
      | Press 4 - to setup regular frequency |
      |__________________________________|
      |                                  |
      | Press 5 - to setup random frequency  |
      |__________________________________|
      |                                  |
      | Press 6 - to exit                |
      |__________________________________|

      """)
```

@author: Leopoldo Medeiros_2017288 - def menu()

In the next screenshot, it shows the while loop function that gives a possibility of an option chosen

```
Please, type a number for your choice:
> 1
Device turned on...

Welcome to BioProtech Device!

Please, type a number for your choice:
> 2
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
vibrating
```

@author: Leopoldo Medeiros_2017288 - CLI output performance

To be accurate, in this case, it must turn on the equipment (option 1) and vibrates the coin motor (option 2) by using a function time.sleep(1), it means the coin motor will vibrate in an interval of 1 sec. In this screenshot below, it can be seen how the program performed with the result displayed in the CLI output.

D. Simulation Motor Vibration in Python II

Python's features and Concepts used

The system uses some concepts used in program language as Boolean data type and logic gate to compare the input of the user and to take a specific action. Some features of the language as "if " conditional and "elif " and "else" commands are used to make the action occur and "while" and "for loop" are responsible for keeping the system working without any interruption until it reaches the "break" command. In order to make the system works with an interval between the vibration, the "time" library was imported to reach this goal. The "print" feature is used to show the output of our system in a "String" format.

System functionality explanation

The system provides a simple menu with three options: vibrate, strong, off and "choose the vibration" message at the bottom. Each option is enabled by typing the proper name and pressing the enter button.

By choosing the "vibrate" option, the coin motor vibrates five times with a two seconds interval by using a "for loop". The output with a "String" typed message "motor on" is printed every two seconds.

The following is the "strong" option which involves the same functionality and characteristics of the first one. However, the inactivity is decreased by one second, which makes the coin motor works with more intensity.

The last is the "off" option. Once off is typed and the user presses enter, the program will stop working immediately by reaching the "break" command at the end of the "for loop". If the user types some option that does not exist in the program, the "type again" message shows up as many time as needed until the right choice provided by the system be identically typed. All the concepts and features that were applied and all the code is provided in the image below:

```
import time

choice = ""
vibrate = False
strong = False

print("""
vibrate - to start the vibration
strong  - for strong vibration
off - to quit
        """)

while choice != "vibrate" or
strong":
    choice = input ("choose the
vibration ")

    if choice == "vibrate" :

        for x in range(5):
            vibrate = True

         # put the pin number of
the raspberry to send electronic
signal
            print("motor on")
            time.sleep(2)
            print("motor on")
            time.sleep(2)
            print("motor on")
            time.sleep(2)
            print("motor on")
            time.sleep(2)
            print("motor on")
            break
```

@author: Rodrigo Aguiar

E. Bluetooth Connectivity to a Smartphone

For the sake of this prototype, the need for a Raspberry PI has been established to make Park Med work, so an important part of this project is how the RPI will be connected to the smartphone that will be responsible for having the app (GUI) integrated into whereby the user will be able to control the vibrations of the smart bracelet. This report will guide us through the first tests in order to prove connectivity between and android App and the RPi.

First, it is needed to install PyBluez which is a module that allows the system to access Bluetooth resources, and then the Bluetooth Python library with the following commands:

sudo apt-get update
sudo apt-get install python-pip python-dev ipython
sudo apt-get install python-pip python-devipython
sudo apt-get install bluetooth libbluetooth-dev
sudo pip install pybluez

Now for test purposes is needed to install the “minicom” which is a text-based serial port communication program which will be used to transmit text to the smartphone.

Apply the following command:

sudo apt-get install minicom -y

After these two steps, we need to open a file known as Bluetooth service configuration using the following command:

sudo nano /etc/systemd/system/dbus-org.bluez.service

And set up a compatibility flag -C:

ExecStart=/usr/lib/bluetooth/bluetoothd -C

Now the PI is ready to use the Bluetooth module. We can proceed to enter to the control mode using the following command:

bluetoothctl

And then, it is needed to power up the Bluetooth module and set the discoverable mode to on.

power on
discoverable on

The terminal console is going to display the MAC address of the RPi and it will change the pair mode to “yes”

[CHG] Device XX:XX:XX:XX:XX:XX Paired: yes

After the smartphone is paired, we can exit the Bluetooth mode on the RPI and run the following command for listening incoming connections via RCOMM port.

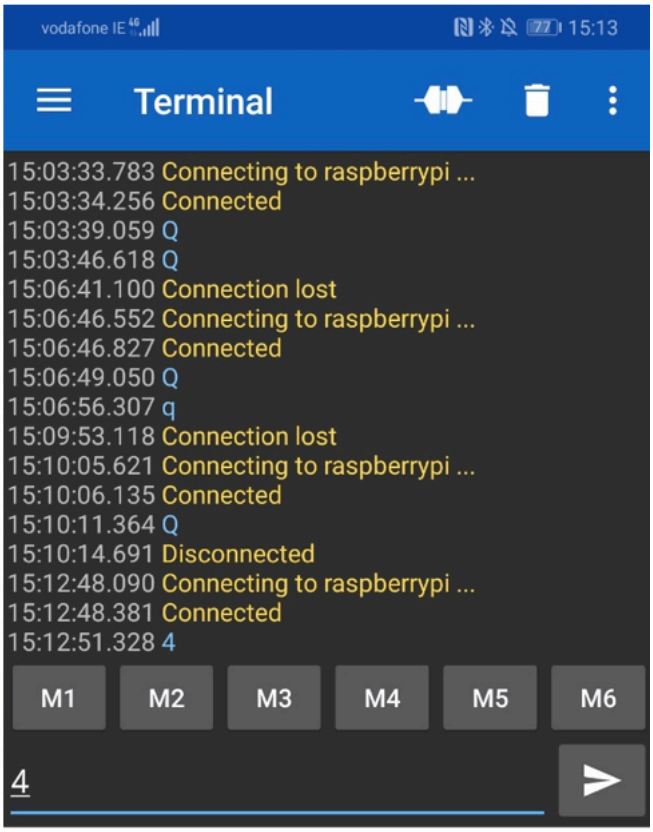
sudo rfcomm watch hci0

In our smartphone, we opened the “Bluetooth Terminal” app (or similar) to establish connection with the RPI and for last step, we run the command **minicom -b 9600 -o -D /dev/rfcomm0** to run the serial terminal. Finally, type something in the app and it will be displayed on the RPi’s console and the other way around. The following code is a very simple test with no validations.

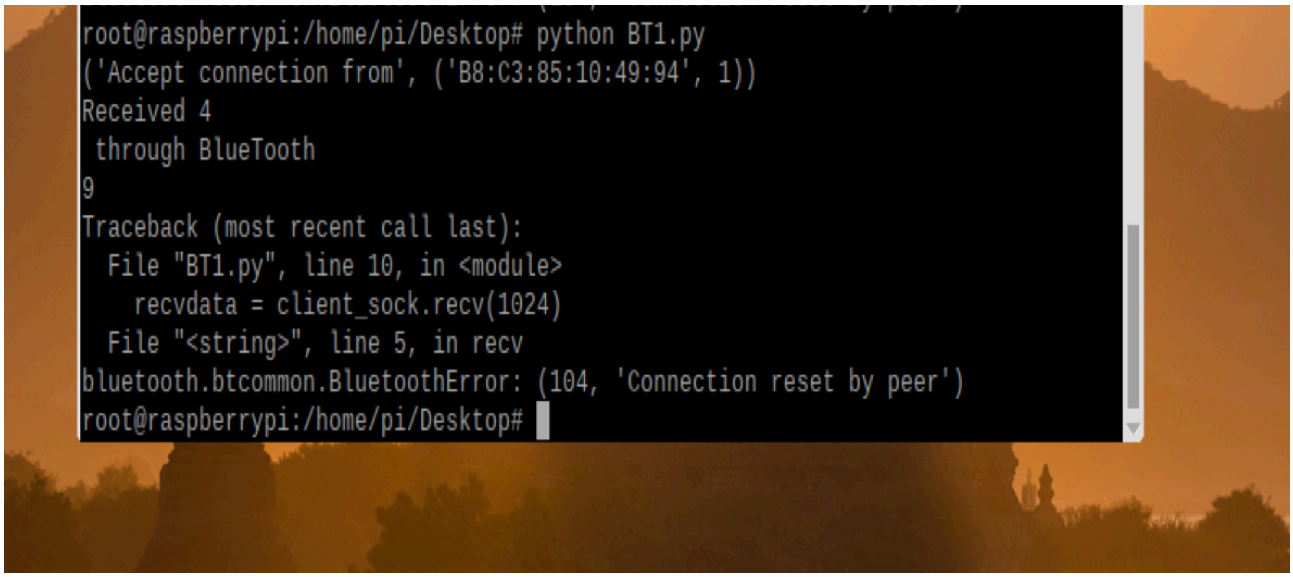
Whatever number we type in the terminal android app, it will be printed in the console of the RPI and be added to 5, the result will be also printed.

```
import bluetooth
server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
port = 1
server_sock.bind(("", port))
server_sock.listen(1)
client_sock, address = server_sock.accept()
print ("Accept connection from",address)
while True:
    recvdata = client_sock.recv(1024)
    print ("Received "+recvdata+" through BlueTooth")
    if (recvdata is "Q"):
        print("Exiting")
        break
    x = 5
    y = int(recvdata)
    print (x + y)
client_sock.close()
```

Once the app showed the RPi was connected, we typed the number 4 and pressed the send button.



On the RPi's terminal, it was printed the number 4 and also the result of the sum, in this case 9.



This experimented demonstrated that it is possible to contro a Python program with texts coming from the Bluetooth connection. This methodology can be applied to the development of Bio Protech's mobile application, but instead of sending a number to make a simple sum, we could send a text message 'start vibration' or even 'set the frequency to 80Hz'.

II. Phase 2 - Circuit Plaground Bluefruit

A. Circuit Playground Bluefruit Configuration

Due to the advantages of replacing the Raspberry with a Circuit Playground Bluefruit, which was previously described, the group decided to divide the implementation of the Park Med device in two parts, the first part consisted of the development with a Raspberry Pi. In this second phase, we will describe how the implementation of the Circuit Playground Bluefruit (CPB) occurred, starting from how the environment was set up.

Even though Circuit Playground boards are intentionally easy to set up, there is still a minor effort necessary to configure it. Some steps must be done before any code is written. The first step is to plug CPB into a computer using a USB cable, wait for a green light and press the reset button in the middle of the board as described on the full tutorial written by Kattni Rambord (2019). At this stage, a new disk drive called CPLAYBTBOOT must be seen on the computer.

The next step is to download the version of CircuitPython for the CPB board. The downloadable file can be found on CircuitPython.org website. As mentioned earlier, it is important to download the version designed only for Circuit Playground boards. The downloaded file is called adafruit-circuitpython-circuitplayground_express-en_US-5.1.0.uf2, which contains the manufacturer and board's name, the language and the version, the file format is UF2, a firmware developed by Microsoft. According to Cian B at Arduino Project Hub, the UF2 format allows for flashing microcontrollers over USB and appears as a USB drive when connected to a computer (2019). When the download is done, and the CPB is plugged in, it is just needed to drag-and-drop the file on the CPLAYBTBOOT disk driver. This procedure allows CPB to be programmed with CircuitPython code.

Based on the tutorial provided on Adafruit's website, when mu-editor is opened up, it should recognise CPB as a programmable microcontroller as long as it is plugged-in and with CircuitPython. The expected resulted was not achieved at first, but this issue was simple to solve. The solution was to reset the board and drag-and-drop the UF2 again, while the mu-editor was open. After that, it was possible to see an "Adafruit" label on the bottom-right corner of the editor indicating that the CPB was ready to be programmed,

B. Exploring Libraries

Bio Protech's intention of building Park Med with vibration motor discs around the wrist that may reduce involutory tremors on Parkinson's patients remains the same. The previous system was using Raspberry Pi and GPIO's. When the Raspberry PI system was being implemented, we were required to import some python libraries such as RPi.GPIO and time, the first one was responsible for allowing access and control GPIO's by providing a specific number, and the second was used to create sleeping times throughout the code. Likewise coding on Raspberry Pi, coding on Circuit Playground also requires additional libraries.

Adafruit provides a package of CircuitPython libraries that can be imported onto the code to facilitate the use of all features offered on CPB, including sensors, Bluetooth module, GPIOs and LEDs. This package can be download on CircuitPython official website.

The library bundle needs to be the same version as the CircuitPython being used, since we are using the version 5, we have downloaded the Bundle Version 5. The file adafruit-circuitpython-bundle-5.x-mpy-20200407 comes in a zip format with two folders, examples and lib. The lib folder is the one where we could find over 220 libraries and modules developed or supported by Adafruit Industries.

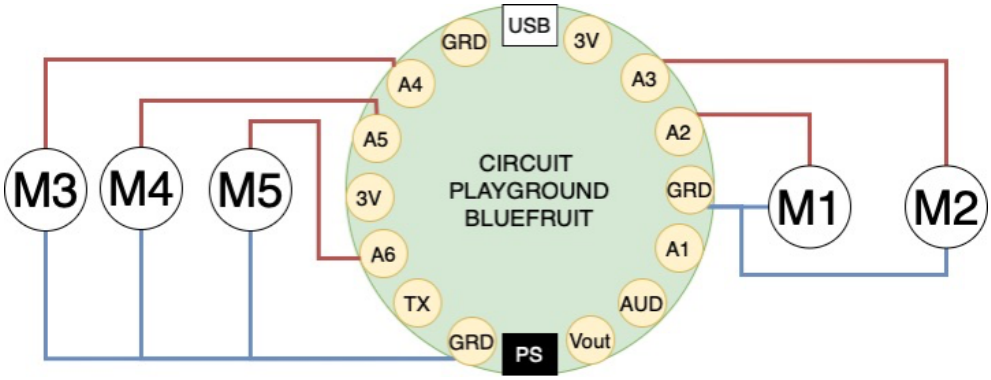
In order to reproduce the same system previously done with Raspberry PI and motor discs, we will need libraries that allows accessing and controlling GPIO pads on CPB. On the learning section of Adafruit's website, there is a tutorial of how to a Switch ON/OFF a LED, Kattni Rembor mentions that three libraries will be imported: board, digitalio, and time. In addition, she reassures that we don't need anything extra to make it work. Board gives access to the hardware on CPB by providing its unique identifier, digitalio allow us to treat the hardware as inputs or outputs ports and time is to create sleeping times throughout the code.

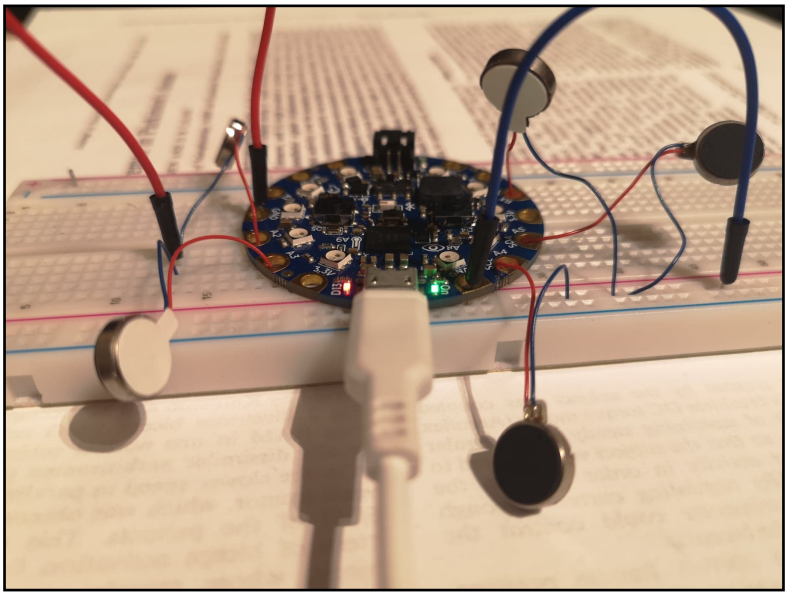
The three libraries need to be copied from the lib folder to CPB disk driver before being imported to the code. It is recommended to create an extra folder within the driver to store all the additional libraries and modules. Having done that, we can start our code by importing those libraries using the import command.

All extra libraries or modules that may be needed in the future must be imported following the same procedure, copy and pasting it into the driver and using the import command at the start of the code. The full description for each library available can be read on CircuitPython documentation website.

C. Vibrating Motor Discs System

The vibration delivered to the wrist of Parkinsonian patients has a crucial role in the project. A system that generates vibration thought motor discs has already been implemented by Bio Protech's members on Raspberry Pi. The idea of using CPB is to replace the Raspberry Pi in order to make the device more versatile and user friendly. Therefore we will be implementing the equivalent system with Circuit Playground Bluefruit. The whole system consisted of five motors disks, a circuit playground board and was done with the help of a breadboard.





Vibrating motors discs and Circuit Playground system. M represents each motor and the CPB illustration provides GPIO pads real positioning. Each motor is connected to ground port and different GPIO pads. M1 is connected to A2. M2 is connected to A3. M3 is connected to A4. M4 is connected to A5. M5 is connected to A6. Diagram created on draw.io.

Even though choosing different GPIOs pads from the diagram above does not make a difference in the final result, it necessary to keep a note of the pad's identifier being used, since it will be addressed in the code. As mentioned earlier, the Board library provides access to the GPIO pads. E.g. for using the pad A1, we would need to address it using board.A1. In addition, we need to define each pad as digital ports, which means they will be ready to receive low and high values, which in our system are translated into TURN DOWN and TURN UP the motors.

```
1  # author Bruno Ribeiro
2  # importing libraries
3  import board
4  import digitalio
5  import time
6
7  # Defining each pad being used as an digital port
8  # using their physical location and addressing it into a variable
9  m1 = digitalio.DigitalInOut(board.A2)
10 m2 = digitalio.DigitalInOut(board.A3)
11 m3 = digitalio.DigitalInOut(board.A4)
12 m4 = digitalio.DigitalInOut(board.A5)
13 m5 = digitalio.DigitalInOut(board.A6)
14
15 # Defining the direction of the ports, as we will be sending out signals
16 # We need to define them as output ports.
17 m1.direction = digitalio.Direction.OUTPUT
18 m2.direction = digitalio.Direction.OUTPUT
19 m3.direction = digitalio.Direction.OUTPUT
20 m4.direction = digitalio.Direction.OUTPUT
21 m5.direction = digitalio.Direction.OUTPUT
```

Screenshot of Mu-editor. In the code, five GPIOS pads are being defined as digital ports and being addressed using the Board library. Each GPIO needs to receive the signal been sent out.

The intention of this test is to learn how the GPIO pads work and to make all the motors vibrates simultaneously for 5 seconds and stop. Since we are dealing with time, we will use the function sleep of the well-known library called *time*.

```
16 # Turning on all motors
17 m1.value = True
18 m2.value = True
19 m3.value = True
20 m4.value = True
21 m5.value = True
22
23 # Wait 5 seconds
24 time.sleep(5)
25
26 # Turning of all motors
27 m1.value = True
28 m2.value = True
29 m3.value = True
30 m4.value = True
31 m5.value = True
```

Screenshot of Mu-editor. In the code, all motors from M1 to M5 are being turned on for 5 seconds and then turning down.

Analysing the whole code, we could see how easy and straightforward is to program a Circuit Playground. All the libraries provided by Adafruit make everything more straightforward, but efficient enough to allow Bio Protech to achieve its goal. Making sure that the CPB is currently plugged-in via USB cable, we were able to run and save the code into the board. As expected, the five motors started to vibrate immediately and went to a complete stop five seconds later.

D. Changing Vibration Frequency

In earlier researches by Bio Protech group's members, it has been found that different Parkinson's patients have different intensity of involuntary tremor. Based on that fact, the vibration provided by motor discs must counter-react the tremors with accordingly intensity. Therefore, we need to allow Parkinsonians patients to adjust the frequency range until they find the best result for their specific tremor pattern. Each vibrating motor disc used by Bio Protech supports frequencies between 20 and 183 Hertz, however stronger vibrations require higher voltages and results in more current draw. All the GPIO pads presented on the CPB supplies a constant output of 3.3 Volts.

The relationship between the voltage supported by motor discs and the voltage supplied by the board can be seen on the table below.

The voltage supplied (Volts)	Frequency of the vibration (Hertz)
2	20
3.3 (GPIO pad)	**121**
5	183

Vibration, Frequency and Voltage relation.

The GPIO can be set to only two values 0 and 1. In Python, it can be translated to LOW and HIGH. Therefore there is no way to change the voltage value other than 0V and 3.3V by only sending different output signals to the GPIOs from a python program.

Due to the common relationship between voltage, frequency and resistance, it is possible to control any of them by changing the other two. In a simple electrical circuit, the best choice to change the vibration's intensity is by controlling the voltage provided to the motors. This can be done with the assistance of resistors. Resistors are electrical components developed to maintain a constant resistance within a digital circuit. There are different types of resistors for different resistance values, which are measured in Ohms (W). A small circuit consisting of Circuit Playground, one motor disc and two resistors (500W and 250W) was built as shown below. (Figure 1)

Figure 2.

$$V_{R2} = \frac{V_{IN}\,(R_2)}{R_1 + R_2}$$

V_{R2}	Voltage needed
V_{IN}	Initial voltage (3.3V)
R_2	Second resistor (500Ω)
R_1	First resistor (250Ω)

Figure 1.

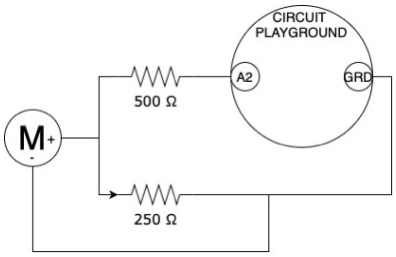


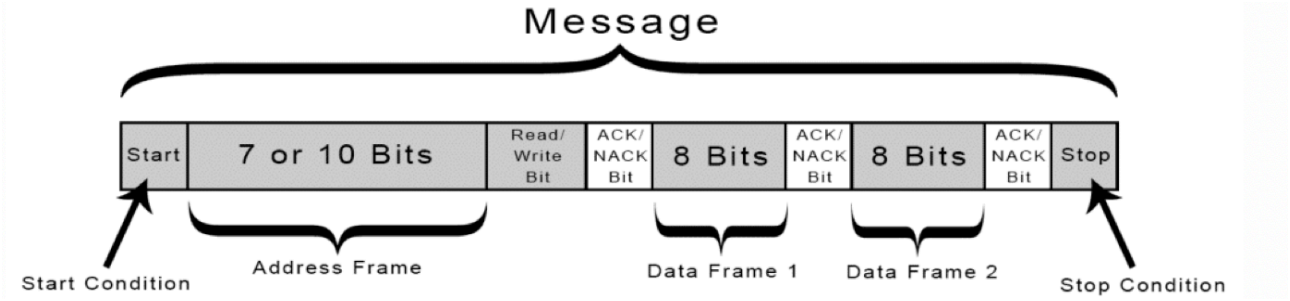
Figure 1 show the digital circuit logic to implement the test.
Figure 2 shows the formula to determine the output voltage of a digital circuit when two resistors are used.

In this experiment, the motor was able to vibrate in a frequency of approximately 80Hz (4800RPM) when 2.2V was provided. The resistance needed to achieve a certain voltage can be calculated using the formula above (Figure 2). Despite successfully changing the frequency with the assistance of resistors, this method does not allow the user to dynamically control it.

E. Controlling Levels of Frequency

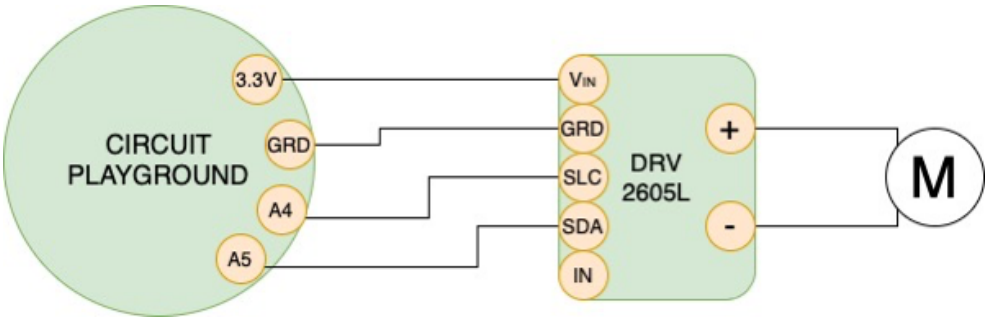
The purpose of the Bio Protech device is limited when a CircuitPython program is designed to send LOW or HIGH signals to the GPIO or when only hardware components are used. In order to allow the user to set different frequencies, it is necessary to combine both digital and electronic techniques. Texas Instruments has created a motor controller or motor driver called DRV2605L for low-voltage devices that supports I2C bus processing. I2C is a protocol built to allow easy communication between microcontrollers with GPIO and peripheral devices, it only requires SDA (serial data) and SLC(serial clock) wires, along with power source (3.3V) and ground.

In order to understand I2C adequately, it is needed to mention the analogy of master and slave, on Bio Protech system, the master is a CPB board, and the slave is a motor controller. The data transferred between master and slave is referred as I2C messages, each message consists of frames that indicate the binary address of the slave, followed by a bit known as Read/Write bit that specifies whether the master is sending or requesting data, one or more data frames that contain the content being transmitted. Between each frame, there is an acknowledgement bit that returns to the sender whether the frame was successfully received or not.



Representation of an I2C message. Picture extracted from Circuit Basics website. Accessible at: http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-I2C-Message-Frame-and-Bit-2-1024x258.png

The electric current supplied by the DRV is insufficient to power up more than one motor, so it is necessary individual motor controller for each motor used for the Park Med. The diagram below simulates the use of a DRV2605L with Circuit Playground and a single disc motor.



Digital circuit consisted of CPB, DRV 2506L and M. Illustrative image, does not provide GPIO real position.

This motor controller includes a pre-configured library containing over 120 different haptic-effects created by Texas Instruments and tested by Carter Nelson at Adafruit Industries. This library has an underlining array of effects, each of them that can be accessed by referring the index wanted. DRV2605L driver and library can be easily imported to a Circuit Playground.

EFFECT ID NO.	WAVEFORM NAME	EFFECT ID NO>	WAVEFORM NAME	EFFECT ID NO.	WAVEFORM NAME
35	Short Double Sharp Tick 2 – 80%	76	Transition Ramp Down Long Sharp 1 – 100 to 0%	117	Transition Ramp Up Short Sharp 2 – 0 to 50%
36	Short Double Sharp Tick 3 – 60%	77	Transition Ramp Down Long Sharp 2 – 100 to 0%	118	Long buzz for programmatic stopping – 100%
37	Long Double Sharp Click Strong 1 – 100%	78	Transition Ramp Down Medium Sharp 1 – 100 to 0%	119	Smooth Hum 1 (No kick or brake pulse) – 50%
38	Long Double Sharp Click Strong 2 – 80%	79	Transition Ramp Down Medium Sharp 2 – 100 to 0%	120	Smooth Hum 2 (No kick or brake pulse) – 40%
39	Long Double Sharp Click Strong 3 – 60%	80	Transition Ramp Down Short Sharp 1 – 100 to 0%	121	Smooth Hum 3 (No kick or brake pulse) – 30%
40	Long Double Sharp Click Strong 4 – 30%	81	Transition Ramp Down Short Sharp 2 – 100 to 0%	122	Smooth Hum 4 (No kick or brake pulse) – 20%
41	Long Double Sharp Click Medium 1 – 100%	82	Transition Ramp Up Long Smooth 1 – 0 to 100%	123	Smooth Hum 5 (No kick or brake pulse) – 10%

Table of content, the haptic-effects library created by Texas Instruments.

For Bio Protech device, only haptic-effects that creates a continual vibration with different levels of intensity will be used. It was found 5 different effects that suit the purpose of the experiment, each of them offers different levels of vibration in a range of 10 to 50 per cent of the maximum frequency (121Hz at 3.3V). For this experiment, the Effect ID number 119, 121 and 123 will be used and to use them it is needed to specify the ID number in the program. It is also possible to build a sequence of up to seven effects and save it to the motor controller memory, as shown below.

```
#author Bruno Ribeiro
#importing libraries
import board
import busio
import adafruit_drv2605

#Setting SDA and SCL ports to establish an I2C connection
#between circuit playground and the motor controller
#bord.A4 is an SCL port board.A5 is an SDA port
i2cPorts = busio.I2C(board.A4, board.A5)

#Setting the I2C protocol to the motor controller
motorController = adafruit_drv2605.DRV2605(i2c)

#Building a sequence of effects
motorController.sequence[0] = adafruit_drv2605.Effect(119)
motorController.sequence[1] = adafruit_drv2605.Effect(121)
motorController.sequence[2] = adafruit_drv2605.Effect(123)

#Executing the sequence
drv.play();
```

While updating the code above into CPB an error occurred “RuntimeError: SDA or SCL needs a pull up” and according to Kesharwani at KritiKal Solutions, it occurred due to the absence a constant high state on both SDA and SCL ports on CPB. It is crucial to mention the three logical states that can be found within a digital circuit, which are low, high and floating.

In this case, the floating state happened because SDA and SCL ports could not be set to a high value and at the same time be used to establish a proper I2C communication thus the motor controller could not be turned on. One way to solve this fault was provided by Resistor's Guide and collaborators, the solution given was to connect the *Vout* pad on CPB to SDA and SCL ports using a 10k resistors, using this method the motor controller could be pulled to a high state and was able to establish a suitable I2C connection.

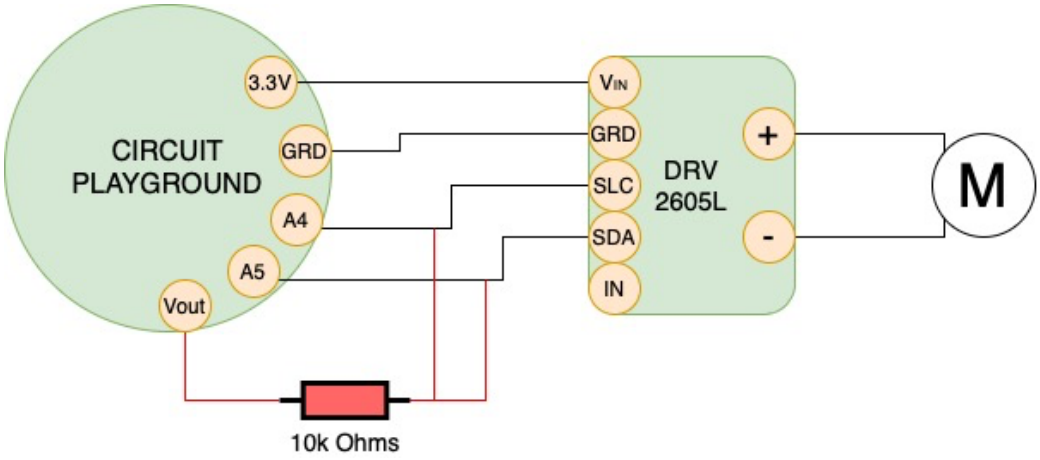
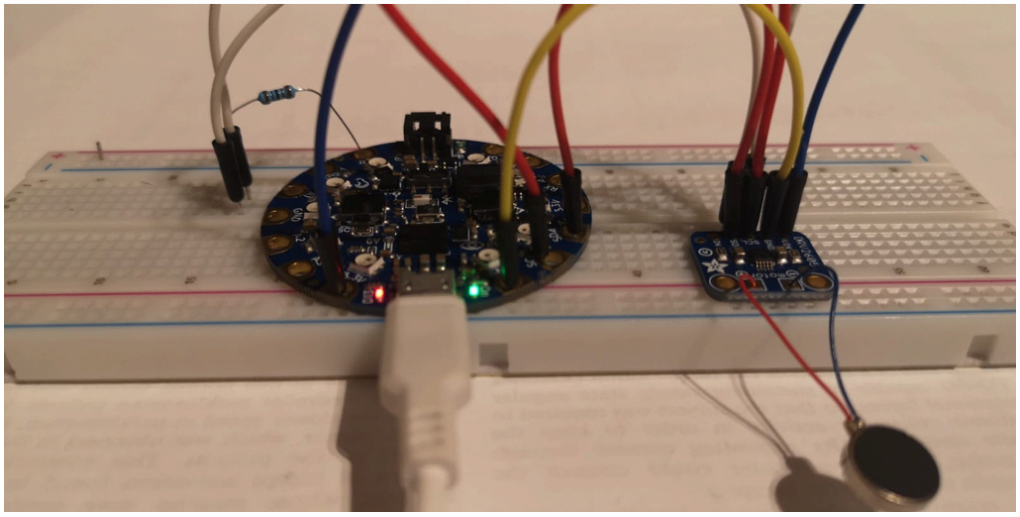


Diagram showing the digital circuit connection with CPB, a motor disc and, a DRV2605 and a 10kOhms resistor. Illustrative image, it does not provide GPIO real position.

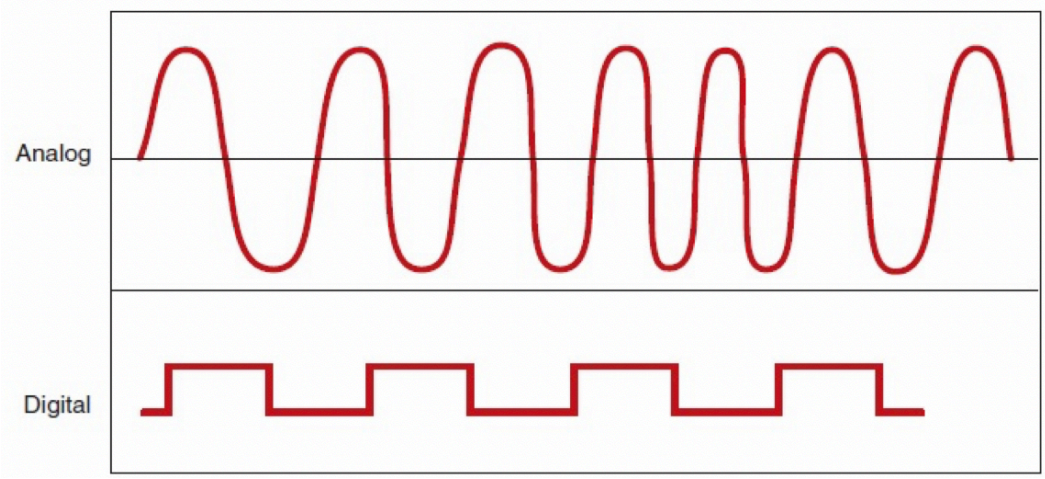


This test resulted in a motor disc vibrating with three different levels of frequency, even though the frequency can't be measure manually without the assistance of external equipment, there is another way to find an approximate value. Considering that the normal frequency a motor disc vibrates when supplied a voltage of 3.3V is approximately 121Hz and DRV2605L pre-configured effects 119, 121 and 123 provide a constant vibration with 50%, 35% and 10% of the normal frequency that the motor vibrates, it was possible to calculate them and it resulted in frequencies reaching approximately 60Hz, 36Hz and 10Hz.

Using a code written in python, a motor controller and libraries developed by Texas Instruments, Bio Protech device allowed the user to set a continual vibration in 5 different levels of intensity.

F. Full Frequency Control with PWM

In an electric circuit, there are two main types of signals that are used to carry information between components, Digital and Analogue. Digital signals are represented by square waves and are translated into binary form, usually 0's and 1's. The analogue signal is represented by a continuous wave that keeps changing over a period and is described using amplitude, frequency and phase as described in Tech Differences (2016).



Visual representation of Analogue and Digital signals. The Analog signal imitates a wavelength, meanwhile, digital signals is translated into low and high. Picture extracted from http://autosystempro.com/wp-content/uploads/2013/07/

Pulse with Modulation(PWM) is a technique widely used in a variety of electrical circuits. It is capable of generating analogue signals from digital inputs. Therefore, by using PWM pulses, it is possible to change 0's and 1's into a broader signal range. PWM consist of two parameters that allow this conversion to happen, duty cycle and frequency. Frequency controls how fast a cycle is completed and the duty cycle controls how much time a single cycle is in a high state.

If a PWM pulse is supplied by 5V from a power source, uses 50Hz(50 cycles per second) and has 25% of duty cycle, the digital input will be set in a high(5V) state during ¼ of a cycle, which occurs 50 times every second. The duration combined with the velocity results in PWM average output of 1.25V. The speed to complete a full cycle is vital to generate the desired voltage; for example, for a DC motor is recommended to use between 5kHz and 10kHz(National Instruments, 2019).

Adafruit Industries has developed a CircuitPython library called Pulseio that supports PWM signals. Pulseio (pulse-input/output) was originally created to control the brightness of LEDs and the frequency of infrared signals, but it can be implemented to Bio Protech's device using the same principles.

Circuit Playground Bluefruit also supports PWM, however a motor disc can't handle a pulse by itself, it is required extra component called transistors. For this test, a PN2222 will be used as a PWM receiver. As a safety measure, a diode 1N4001 pin will also be added to the logical circuit. Diodes are used to conduct the electric current in one direction and avoid reverse polarity. In a small voltage system such as CPB and motors discs, diodes are redundant and can be discarded.

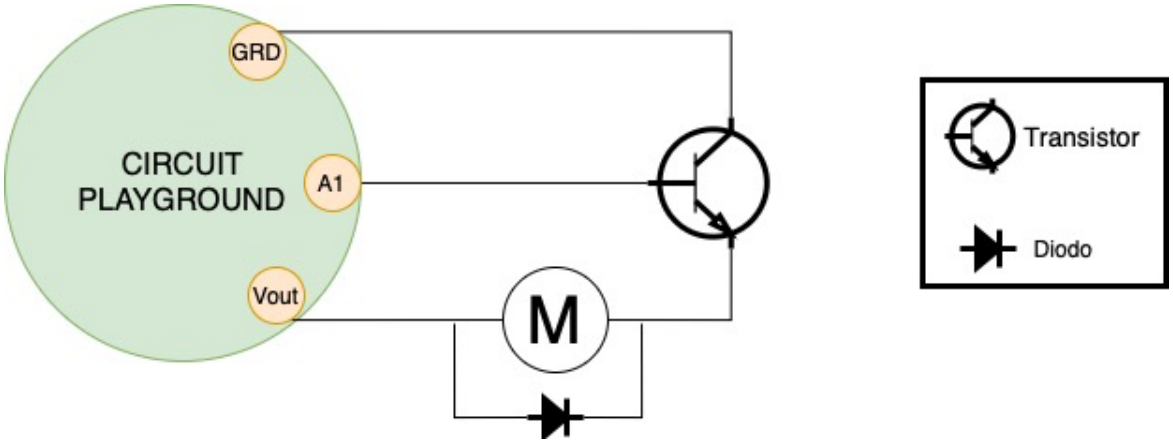


Diagram showing the digital circuit connection with CPB, a motor disc, a transistor and a diode. Illustrative image, it does not provide GPIO real position.

In the circuit above, with the help of PWM pulses, the transistor PN2222 acts as a variable frequency driver (VFD). A VFD is a type of controller that drives an electric motor by varying the frequency and voltage of its power supply. The VFD can also control ramp-up and ramp-down of the motor during the start or stop, respectively.

As a mean to send the right PWM pulse, containing the correct duty cycle and frequency, it is needed to understand that the frequency sent via a pulse is not the frequency that the motor will vibrate, as previously mentioned the frequency of the pulse determines how many times per second a cycle is restarted, however for a small motor disc it does not make an impact, anything above 50Hz will work.

For the purpose of finding the correct duty cycle, it will be considered the information that was given by the manufacturer. It says that the motor disc vibrates around 183Hz (11000RPM) at 5V. It is also known that pulseio from Adafruit interprets duty cycle as a 16-bit message, going from 0 to 65535. After gathering all the information presented and given a frequency f, the formula to estimate the suitable duty cycle regarding f is the maximum duty cycle less a constant number of 358 times the frequency f. Any decimal value is not considered given that it would not make a reasonable interference in the final result.

0 Duty Cycle = 5V results in 183Hz
65535 Duty Cycles = 0V results in 0Hz

$$dc = maxDuty - \frac{(maxDuty \quad f)}{maxFr}$$

$$dc = 65535 - \frac{65535\, f}{183}$$

dc = duty cycle
f = frequency requested

$$dc = 65535 - 358\, f$$

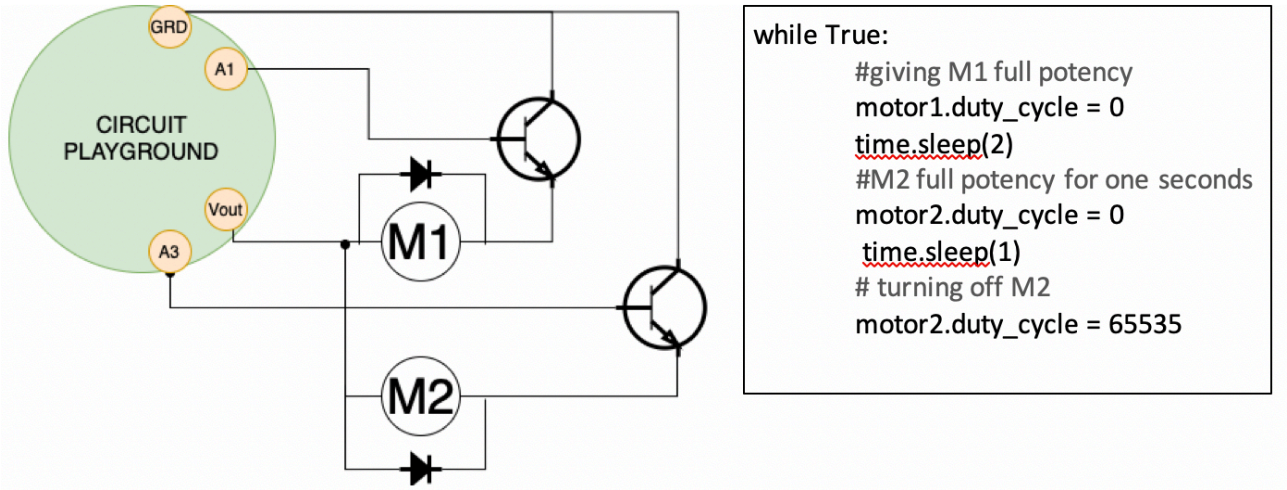
Formula to calculate the necessary duty cycle, it is originally created by Bio Protech members and uses rule of three to find the duty cycle based on the information given by Texas Instruments and Adafruit Industries.

Setting up the motor with PWM signals is easy with the assistance of the pulseio library, to initialise the motor, it will be used a PWM frequency of 5000 (5kHz) and the GPIO displayed as A1 on Circuit Playground. This test aims to generate a vibration of 120Hz for 5 seconds and then send a low value for 2 seconds.

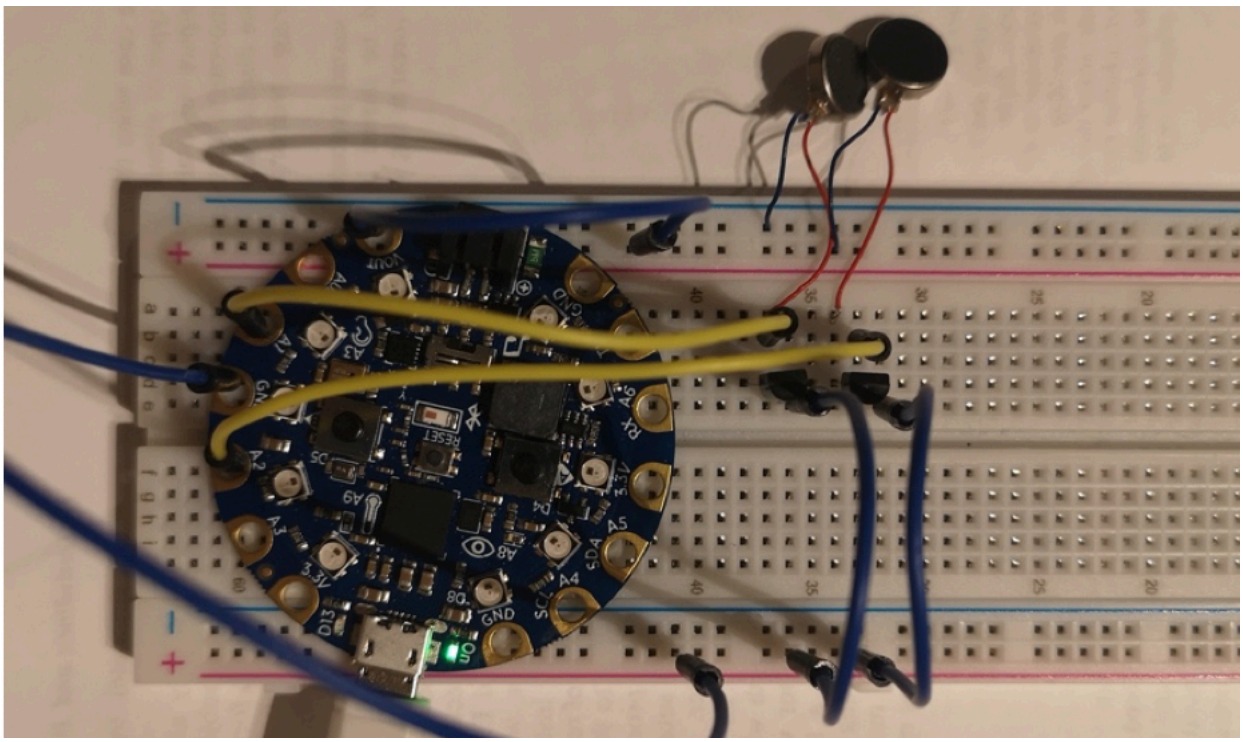
```
#author Bruno Ribeiro
#importing libraries needed
import board
import time
import pulseio
#setting the motor as a PWM outbut and a constant frequency of 5kOhms
motor = pulseio.PWMOut(board.A1, frequency=5000)
#this is the frequency we want the motor to vibrate
f = 125
#this is the duty cycle needed to make the frequency reach f
dc = 0
#formula to determine dc
dc = int(65535 - 358 * f)
#program will run while CPB is on
while True:
  #vibrating the motor at f Hz
  motor.duty_cycle = dc
  time.sleep(5)
  #shuting down the motor for 2 seconds
  motor.duty_cycle = 65535
  time.sleep(2)
```

The aim of the test was achieved with success. The change of frequency was clearly perceptible within a range of 1Hz up to 183Hz as expected. However, even when the duty cycle was set to 65535 a small vibration could be seen when the motor disc was suspended in the air (without touching any surface).

Intending to ensure that using a system with PWM and transistors is an appropriate approach to give the full control over the vibration intensity to the user, a second motor(M2) was added to the circuit so that the interference with the first motor's vibration(M1) could be tested, both motors were supplied by the same 5V port. The sequence started with the M1 vibrating at full potency without stopping and every two seconds M2 was given full potency for 1 second. It resulted in M1 vibrating continuously and M2 vibrating with peaks happening every 2 seconds.



On the left: Diagram showing the digital circuit connection with CPB, two motor discs, two transistors and two diodes. Illustrative image, it does not provide GPIO real position. On the right: A snapshot of the code: M1 vibrating continually and M2 vibrating with peaks happening every 2 seconds.



There was no noticeable interference with each other. The same experiment was done with 2 LEDs replacing the motor discs and it ended with equivalent results. Further tests must be done to get more precise results using a multimeter or VOM to measure the voltage within the circuit. However, Bio Protech did not possess such equipment at the time this experiment was carried out.

G. Bluetooth Connectivity

Being able to regulate the frequency accordingly to the patients' needs is satisfactory to Bio Protech's goals. However, we want to avoid any obstacle that restricts a beneficial user experience. With that in mind, we need to develop a user interface that any user is capable of using it. An ideal solution is to allow users to adjust the vibration intensity through a mobile application connected wireless. This is where the built-in Bluetooth module on CPB has an important role. The nRF52840 chipset with Bluetooth Low Energy presented on CPB is perfect for the project, due to its low energy consumption and simple usability.

Adafruit has its own mobile applications for Android and iOS that are capable of communicating with different microcontrollers and Bluetooth modules produced by them. Despite the fact that Adafruit's App has more features than we need, it will be perfect for testing Bluetooth connectivity on our Circuit Playground. We will be using the Android version and the app was downloaded directly from Play Store, Google's official application store. A simple experiment will be carried out to learn and understand how the BLE communication works, which libraries are needed and how to program CPB to establish Bluetooth connection.

There are several projects using BLE connection done by the community at Adafruit, but none of them is really describing how to configure CPB to able to connect to an external device. The solution to determine what must be done was to analyse all common aspects from the tutorials. In other words, we needed to gather enough information related to BLE connection and analyse the repeated CircuitPython code from different tutorials, but to make the whole process clear, we have filtered the tutorials so that only the ones using the same Bluetooth module as CPB were analysed.

The projects analysed were:

- CircuitPython BLE Controlled NeoPixel Hat by Ruiz Brothers (2019)
- BLE Light Switch with Feather nRF52840 by John Park (2019)
- Halloween Pumpkin by Lady Ada (2013)
- BLE Synth with the Feather nRF52840 by Liz Clark (2020)

After analysing the CircuitPython code from the projects above, we could see that there were some familiarities. Firstly, all of them were using the same three libraries: BLERadio, ProvideServiceAdvertisement, and UARTService, all of them were being imported from the module adafruit_ble. According to CircuitPython documentation, the adafruit_ble module provides the necessary low-level functionality for communicating using BLE. Meanwhile, BLERadio provides interfaces for BLE advertising, scanning for advertisements, and connecting to peers. ProvideServiceAdvertisement advertises what services that the device makes available upon connection. UARTService handles data transmission using the UART protocol. Since all of them use the same module, the folder adafruit_ble is the only one needed to be copied from the library bundle and pasted into CPB disk driver.

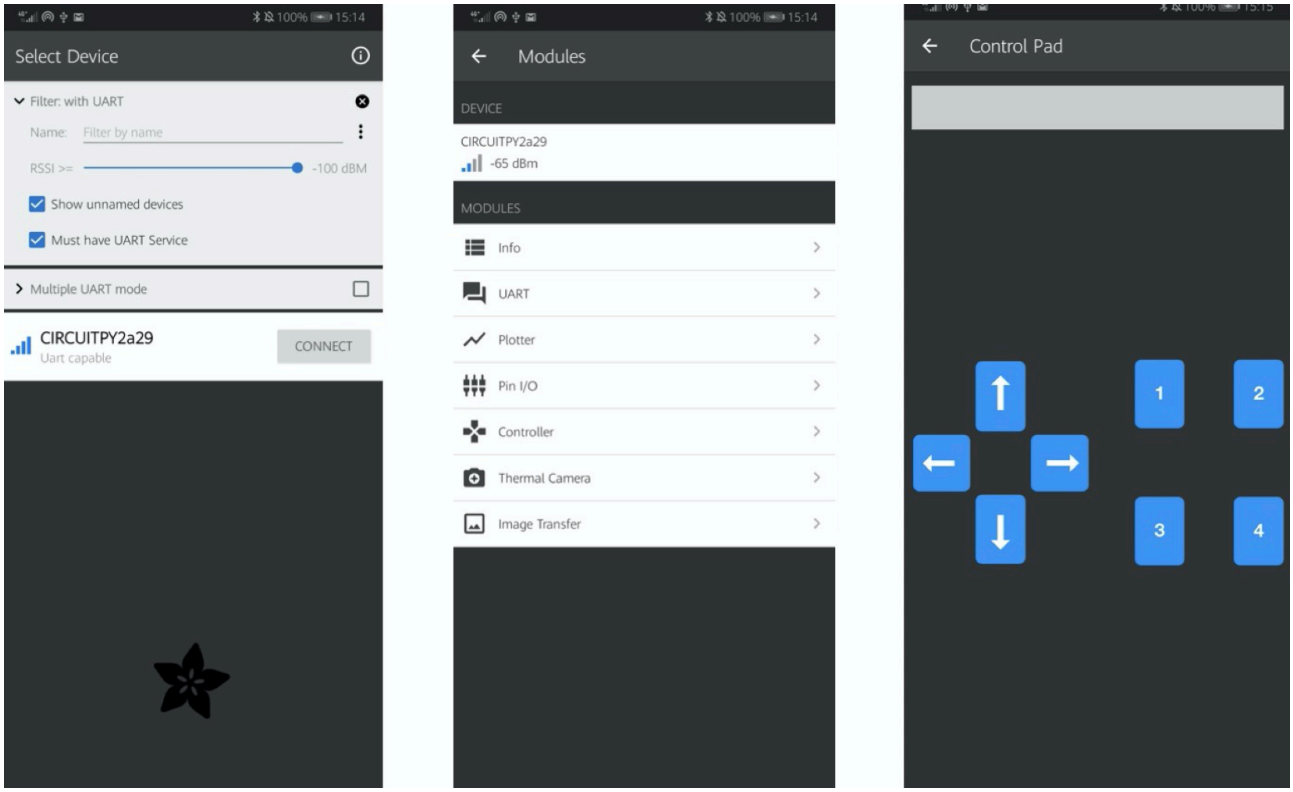
All projects mentioned were following the same structure of code and the same way to establish a Bluetooth connection. While loops were being used for advertising the Bluetooth service on CPB, within these while loops there were two more while statements, one to verify it the device was connected and another one to check if the device was not connected. We followed the same structure of code to program our CPB and to test it. We programmed the LED (D14) to be turned on when the connection is confirmed. The code was saved onto CPB.

```
1  # importing libraries for BLE connection
2  from adafruit_ble import BLERadio
3  from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
4  from adafruit_ble.services.nordic import UARTService
5
6  # initiliazing the imports
7  ble = BLERadio()
8  uart_service = UARTService()
9  advertisement = ProvideServicesAdvertisement(uart_service)
10
11 # Advertises without stopping
12 while True:
13     ble.start_advertising(advertisement)
14
15     while not ble.connected:
16         # Do while BLE is not connected
17         led.value = False
18         pass
19     while ble.connected:
20             try:
21                 # Do while BLE is connected
22                 led.value = True
23             except ValueError:
24                 continue
```

Screenshot of the Mu-editor: After importing the required libraries(line 2-4), we need to store each one of them into variables (line 7-9). Using a while loop that will run uninterrupted as soon as CPB is started up (line 12), we advertise the device as a Bluetooth service (line 13). Inside the while loop we can check whether the device is connected or not (line 19 and line 15). Code was written by Bruno Ribeiro based on 4 different projects provided by Adafruit Industries.

Bluefruit Connect, Adafruit's mobile application, was opened up and a message was prompted asking to *Allow Bluefruit to enable Bluetooth*. After accepting it, we were sent to the application's first page "Select Device". This page shows all Bluetooth enabled devices around, but at this time, only the CPB was being advertised within a reasonable distance. The application also offers the option to filter devices by name or distance. We could see important information regarding CPB, including name, address, signal strength and also a "Connect button" in front of the name. When the "Connect" button was pressed, a "Connecting…" message was shortly displayed and then it changed to "Discovering services", as soon as the second message disappeared we were redirected to the application's dashboard and the LED on CPB turned on, demonstrating a successful connection.

On the dashboard, we could see different types of features to control CPB such as ColorPicker, which allows the user to change the colour of the LedPixels on CPB, and Control Pad, which offers an interface similar to a remote with 8 buttons, but because we do not have anything programmed other than turning on the LED, we were not able to test them.

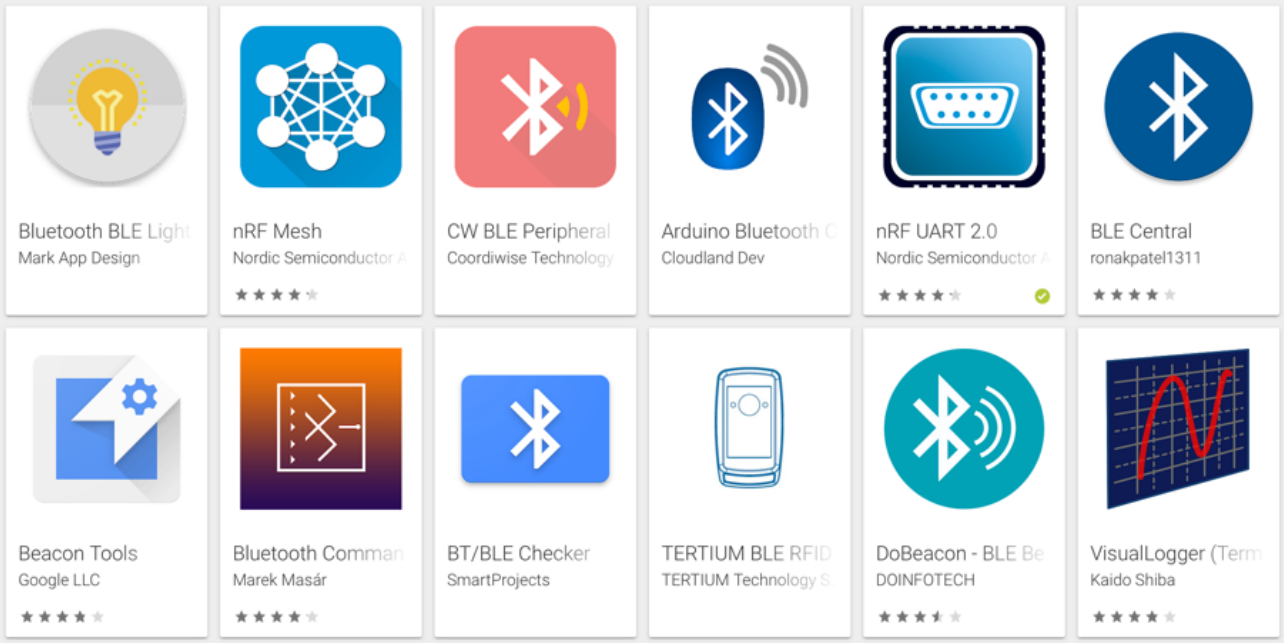


Screenshot of Bluefruit Connect app. From left to right: 1. First page of the app (Select Device). 2. The page that opens when the connection is completed (Modules available). 3. A page with a remote controller interface (Control Pad).

The aim of this test was to connect CPB and the smartphone via Bluetooth and by using Adafruit's mobile application, we accomplished the desired result. However, after concluding the procedure, we realised that we needed to test the connection using a different mobile application. In other words, we needed to test if CPB is able to connect with third-party mobile applications.

At this point, we are fully aware that CPB is only compatible with Bluetooth Low Energy technology, with that in mind we searched for Android mobile applications that allow BLE connection. The search was executed on Play Store's website and the result page had several options of applications, some of them were intended for scanning, analysing or connecting. Even though there were numerous options to choose from, one application grabbed our attention, "nRF UART 2.0" developed by *Nordic Semiconductor ASA*, the same company that manufactured the

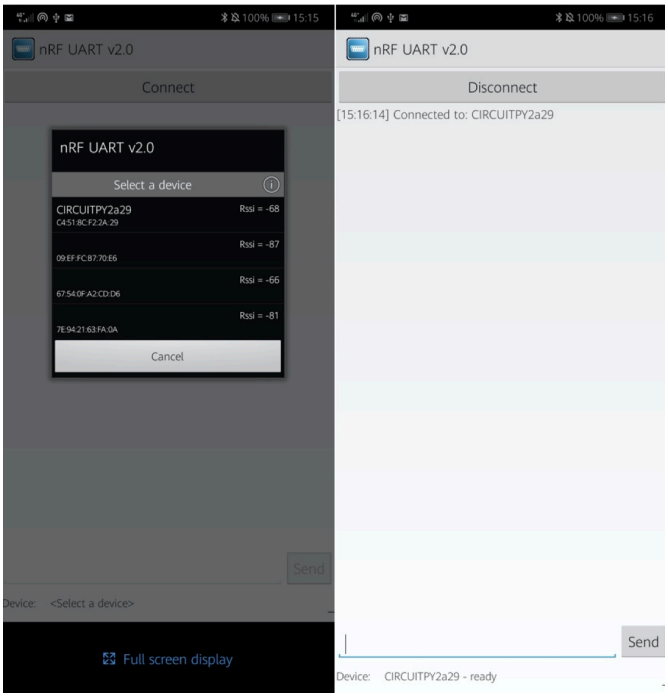
nRF52840 chipset.



Screenshot of Play Store's partial result page when the term BLE was searched. Nordic Semiconductor's mobile application, nRF UART 2.0 can be seen on the fifth column of the top row.

The interface from nRF UART 2.0 is clearly simpler than the one from Bluefruit. The first page of the application consists only of one input field and two buttons, one "Connect" and one "Send". When the button "Connect" is pressed, a list of available devices is displayed and in order to connect to any of them, it is only needed to press on top of the name.

CPB was started up running the same code used to test Adafruit's application. After a few seconds, CPB's name was amongst the list of devices. After pressing on CPB's name, the LED was turned on and a text message was printed on the screen confirming that the connection was confirmed.
The test reassured us that CPB is able to connect via Bluetooth with different mobile applications from different developers and also allowed us to find a simpler application that can be used to execute further Bluetooth-related tests.



Screenshot of nRF UART 2.0 Android application.

H. Control Over Bluetooth

Considering that now CPB is able to connect to mobile devices via Bluetooth, the next step is to rearrange the code so that CPB can accept data from the mobile device and translate the received data into commands. First, we need to define what type of data can be interpreted as commands. As per the project requirements, the user must be given the options to start, stop and also change the frequency of the vibrations and these options will be displayed in the form of buttons.

As mentioned earlier, UARTService library from the adafruit_ble module is the one responsible for handling data transmission using the UART protocol created by Nordic Semiconductors. This particular communication protocol takes bytes of data and streams bit by bit in a sequential manner. CPB receives the stream with the Read() method from the UARTService class, then the bits are put together with the help of a Buffer library. When a new stream is captured by the Buffer, the data being received is stored inside of an array of bytes, which can be later transformed into text, numbers or even objects. Using Adafruit libraries, this whole process happens underlying, but it is worth knowing how it works, further explanation is given on CircuitPython documentation website.

Now it is known that anything CPB receives from the Bluetooth stream is automatically stored into arrays of bytes, the matter to solve now is how to transform it in command. W3Resource (2020) website provides different suggestions of how to manipulate the array of byte, one of them is to transform it into a string data type, which is the equivalent of a text in the programming field. The best option to activate certain commands is to validate the streams being received and with strings, this validation can be easily done using IF/ELSE statements, so we will follow the examples given by the collaborators at W3Resources.

Code:

```
# create a string using the decode() method of bytes.
#This method takes an encoding argument, such as UTF-8, and optionall
x = b'El ni\xc3\xb1o come camar\xc3\xb3n'
s = x.decode()
print(type(s))
print(s)
```

Output:

```
<class 'str'>
El niño come camarón
```

Screenshot of the W3Reources website. Code of how to transform an array of bytes into a string. Example 2. Accessible at: https://www.w3resource.com/python/python-bytes.php

The previous code that was written to connect CPB and the mobile application was rearranged to allow CPB to get the data being transmitted, to convert it into a string using the example from W3Resource and to validate it using an IF statement. The mobile application used to test data transmission is the nRF UART 2.0, its easy usability makes it the perfect option for this test. The Bluetooth connection was established following the same procedure done before.

```
while ble.connected:
    try:
        # read the data from the UART service
        stream = uart_service.readline()
        # transform it into a string
        text = x.decode()
        # validate it to print a message
        if text is "hi":
            print("Hello Bruno")
    except ValueError:
        continue
```

Screenshot of Mu-editor: In the code, while the Bluetooth is connected, read the data from the Bluetooth UART service and store it into a ‘stream’ variable. Convert whatever is inside of the stream variable to a string and store it into a text. Validate the text to check if “hi” is stored inside. If so, print ‘hello Bruno’. If not, do nothing.

The nRF UART 2.0 has a text field where we can enter the text message that we want to send to the connected device. Because CPB was programmed to check if the messages received were equal to “hi”. We typed “hi” and pressed the “Send” button, the application confirmed that the message was sent. However, there was no visible information that the message was received on CPB and the expected “Hello Bruno” was not printed to the console. It is important to mention that the message was completely lowercase. In order to see where the problem was, we changed the code to print the stream and text variable as soon as it is received on CPB.

```
19  while ble.connected:
20      try:
21          # read the data from the UART service
22          stream = uart_service.readline()
23          print(stream)
24          # transform it into a string
25          text = x.decode()
26          print(text)
27          # validate it to print a message
28          if text is "hi":
29              print("Hello Bruno")
30      except ValueError:
31          continue
```

Screenshot of Mu-editor: In the code, two print methods were added to troubleshoot the program and see what was causing the validation to fail. Line 23 and Line 26.

We followed the same procedure to retry the test, this time we could notice that as soon as both devices were connected empty lines started being printed uninterruptedly every 1 second, this unexpected outcome revealed that the data from the Bluetooth takes some time to be read. Despite the fact that there was nothing on the stream awaiting to be read in, the code interpreted it as a null or none value but printed it anyway

When the “Send” button was pressed, two text messages were printed to the code editor’s console, again it was noticed a delay of around 1 second between the button being pressed and the text being printed, confirming that BLE has a short delay when it is transmitting data. Strangely the two messages were printed as arrays of bytes, therefore the issue was not in receiving or validating the data, but on the conversion to a string using the decode method.

Screenshot of Mu-editor console. The first message printed corresponds to the variable stream, which is clearly an array of bytes. The second message printed corresponds to the variable text, which should be a string rather than another array of bytes.

More unsuccessful tests using the Decode method were carried out including with Adafruit's mobile application and the reason for the problem could not be discovered. While searching for different approaches, we could find another solution to complete the required conversion without using the Decode method. One of the collaborators at EDUCBA(2018) described a process that has similar results, it is suggested to iterate through each byte of the array, convert it to char values and assemble it inside of a string. The process seems confusing, but it is easy to implement.

First, we need to loop through all elements of the array, it can be done by using a simple for each loop "*for b in stream*", where *b* is a byte and *stream* is the array of bytes. Second, it is needed to convert each byte into a char value, Python has a built-in function to do this, "*char(b)*" where *b* is a byte. The last part is to assemble each char value in order to make a string, Python has a function called join that puts together two values, the current one and the one being passed. This whole process can be done in one line of code.

```
19  while ble.connected:
20      try:
21          # read the data from the UART service
22          stream = uart_service.readline()
23          print(stream)
24          # transform it into a string
25          text = ''.join([chr(b) for b in stream])
26          print(text)
27          # validate it to print a message
28          if text is "hi":
29              print("Hello Bruno")
30      except ValueError:
31          continue
```

Screenshot of Mu-editor: In the code, the method Decode() was swept by another methodology with the same result.

This time when the “Send” button was pressed, three messages were printed to the console. The first one represented the array of bytes, the second one was a simple “Hi” string and the third one was another string “Hello Bruno”. It demonstrated that the conversion from an array of bytes to string and the validation using a simple if statement was successful. Having said that, using text messages sent wirelessly from the mobile application we are able to activate different commands on CPB with IF/ELSE statements. For example, we can activate the vibrations with a “start” command or “stop” to make the vibrations stop. It will allow us to proceed to the development of Bio Protech’s mobile application.

III. Android Mobile Application

Bio Protech aims to deliver a mobile application to allow Park Med users to have full control over the device. In earlier researches done by the group, we have discovered it is needed to generate different vibration intensity to reduce the hand-tremor in different Parkinson’s disease patients. The microcontroller board that is used to power Park Med was already programmed to offer such feature, our intention of this chapter is to develop an Android mobile application to connect to Park Med and be able to control it.

The requirements the application must meet includes:

- Offering the possibility to establish a Bluetooth connection to a Park Med device;
- Controlling the vibration;
- Offering an easy and simple user interface;
- Plotting a graph using data from the accelerometer sensor on the Park Med device.
- Offering an option to save the graph’s data

The main reason for choosing Android as the first platform is that according to Statcounter Globalstats, in February 2020 Android had 73.3% of presence in the mobile market worldwide meanwhile iOS was running on 25.89% of devices and other platforms such as KaiOS, Samsung and Windows Phone had less than 1% altogether.

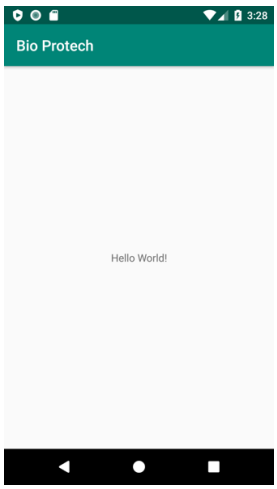
Other two good reasons are the programming languages and the integrated development environments (IDE) that can be used to develop an android application. Android SDK (System Development Kit) had Java as its first official language however nowadays it supports Kotlin, C++ and many other programming languages as described by Adam Sinicki (2020). Due to the familiarity amongst different group members and being android's primary supported language, Java was chosen for the development of Bio Protech's mobile application. As IDE we have chosen Android Studio for code editing, testing and debugging. This decision was based on an article written by a collaborator at AltexSoft (2018), this article mentions advantages such as official supported IDE by Google, templates of common app features, drag-and-drop layout builder, easy integration with other Google services and free to download.

A. New Android Project

Android Studio is compatible with Windows, Mac, Linux and Chrome OS. For this project we will be using the one for Mac version 3.5, the latest one found in January of 2020. The installation was simple and followed the recommended options, keeping in mind that any additional resource or library needed can be added later on.

We started a new android project with Empty Activity, which means that Android Studio will generate a ready to use activity class along with its layout. The application is named as Bio Protech, and it will use Java and the minimum API level of 23, for Android Marshmallow (6.0), as justified earlier. The project was created by Android Studio automatically using the Gradle guidelines.

Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software (Gradle Org, 2020) and it is the default build tool on Android Studio, but Maven is also available. After creating and building, the yet empty application was tested using a real Android smartphone. It is possible to use an emulator on Android Studio to create a virtual device, but due to its high processing consumption while running an application, we opted to use a physical smartphone with developers' options enabled connected via a USB cable. Every time an application is launched, Android Studios sends to the connected smartphone an Android Package(APK), and for each launch, the previous version is replaced by the new one.



Screenshot of Bio Protech's application running on an Android Smartphone. By default, Android Studio starts a project with automated layout generation containing a top bar with the application's name, a text field "Hello World!" at the middle of the screen and navigation bar at the bottom.

B. Importing Additional Dependencies

As previously mentioned, Circuit Playground Bluefruit (CPB) has an nRF52840 Cortex M4 processor with Bluetooth Low Energy created by Nordic Semiconductor. The same company has developed the open-source android application that was used to test Bluetooth connectivity and data transmission between a smartphone and the CPB. This application can be download at Google Play Store, and its source code can be found on Nordic Semiconductor's Github page. The source code is licensed under the Apache License, Version 2.0 that allows redistribution with or without modifications when the following conditions are met give any other recipients of the work a copy of Apache License, any modified files to carry prominent notices stating that the files were changed and retain all copyright, patent, trademark, and attribution notices from the source form(Apache License, 2004). Therefore, Bio Protech will follow all the requirements throughout the development of our app while using part of any source code from Nordic Semiconductors.

The main reason for using Nordic's mobile application as a foundation is that it comes with a java class called UartService that contains all Universally Unique Identifiers (UUIDS) and GATT events required to establish a Bluetooth connection and transmit data between CPB and an android smartphone. GATT events are related to Bluetooth Adapter status such as connected, disconnected, service discovered, available and extra data. The UUIDS shown below are the ones from the processor nRF52840 presented on CPB, and each one of them is responsible for a different connectivity task such as advertisement, read to, write from and UART availability.

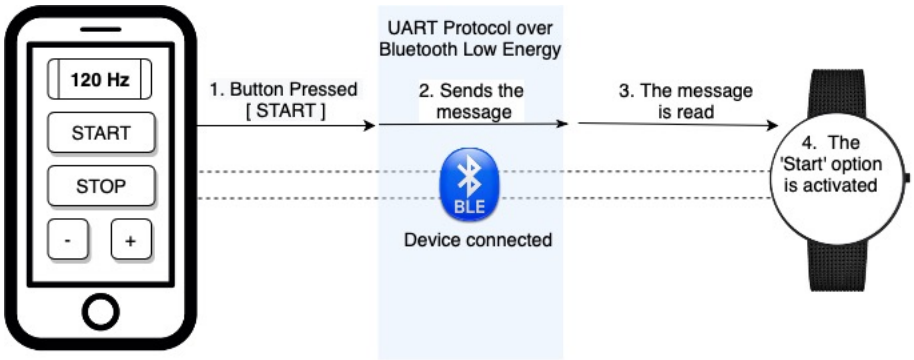
```
public final static String ACTION_GATT_CONNECTED =
        "com.nordicsemi.nrfUART.ACTION_GATT_CONNECTED";
public final static String ACTION_GATT_DISCONNECTED =
        "com.nordicsemi.nrfUART.ACTION_GATT_DISCONNECTED";
public final static String ACTION_GATT_SERVICES_DISCOVERED =
        "com.nordicsemi.nrfUART.ACTION_GATT_SERVICES_DISCOVERED";
public final static String ACTION_DATA_AVAILABLE =
        "com.nordicsemi.nrfUART.ACTION_DATA_AVAILABLE";
public final static String EXTRA_DATA =
        "com.nordicsemi.nrfUART.EXTRA_DATA";
public final static String DEVICE_DOES_NOT_SUPPORT_UART =
        "com.nordicsemi.nrfUART.DEVICE_DOES_NOT_SUPPORT_UART";

public static final UUID CCCD = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");
public static final UUID RX_SERVICE_UUID = UUID.fromString("6e400001-b5a3-f393-e0a9-e50e24dcca9e");
public static final UUID RX_CHAR_UUID = UUID.fromString("6e400002-b5a3-f393-e0a9-e50e24dcca9e");
public static final UUID TX_CHAR_UUID = UUID.fromString("6e400003-b5a3-f393-e0a9-e50e24dcca9e");
```

Screenshot from Android Studio: UartService.java class showing UUIDS and GATT events provided by Nordic Semiconductors. Licensed under Apache License, Version 2 from 2004. Copyright (c) 2015, Nordic Semiconductor All rights reserved.

C. Establishing Bluetooth Connection

We are aiming to develop an application that is capable of controlling the Park Med device via Bluetooth. The idea is that for each button pressed, a text will be sent via Bluetooth and the board will translate this text into a command. For example, to start the vibration a button “START” is pressed, a “start” message is transmitted through the Bluetooth adapter, the board receives the new message and activates the corresponding command. UART protocol has a restricted limit of characters allowed for each message. Therefore the length of the message must be considered to avoid any loss of data transmission.



Illustrative representation of connectivity and communication between a smartphone and Park Med. A text message is transmitted over Bluetooth so that its corresponding command is activated.

Most of the app functionalities will be executed within the MainActivity Java class, including wireless connection, receiving and sending data and handling button events. Therefore, any method related to these functionalities must be declared inside of the MainActivty. In order to use the Bluetooth service provided by Nordic Semiconductors, the class UartService extracted from the source code was added under the same package as the MainActivity class and all methods related to BluetoothAdapter, BluetoothService and UartService components were imported to the MainActivity. Moreover, the Bluetooth Service requires the MAC address of the target device to establish a connection. For the first stage, the current Circuit Playground MAC address will be hardcoded; thus, it is the only device that will be connected with the smartphone.

A ButtonView called “connect” was added to the MainView layout using the drag-and-drop feature presented on Android Studio. In order to handle any click event from a button, it is required to follow two steps. First, we needed to tag the ButtonView to a Button java object using the findViewById method as shown in the figure below. The second step is to add an OnClickListener for each button and by doing this, the IDE will automatically implement the onClick method, anything that must be done resulting from a button pressed must be added within the onClick method. Consequently, every time a button is pressed, the listener will be triggered and the equivalent method is executed.

The “connect” button is responsible for first creating a new BluetoothDevice object that stores information regarding the targeted device such as name, type, address and UUIDS capabilities. The next step is to use an instance of the UartService class provided by Nordic Semiconductors to connect with the device, for this project the instance will be called mService, following the naming convention from its supplier.

```
Button buttonConnect = (Button) findViewById(R.id.connect);
buttonConnect.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
            BluetoothDevice device = BluetoothAdapter.getDefaultAdapter().getRemoteDevice( address: "C4:51:8C:F2:2A:29");
            mService.connect( address: "C4:51:8C:F2:2A:29");
    }
});
```

Screenshot from Android Studio: buttonConnect hooked up with onClickListener. When the button is pressed the application will use the Bluetooth adapter on the smartphone running it to create a Bluetooth Device with MAC address of C4:51:8C:F2:2A:29, which is the one from Circuit Playground. The mService is an instance of the UartService class provided by Nordic Semiconductors.

D. Transmitting Data Over Bluetooth

A second button “Send” was added to the MainView layout. In order to make the newest button functional, we followed the same procedure as before. The purpose of the send buttons is to make use of the writerRXCharacteristic from the UartService to transfer a short text message over Bluetooth. This approach to transmitting data occurs only with bytes. Consequently, we need to convert the string message to an array of bytes before initiating the transfer.

```
Button send = ((Button) findViewById(R.id.send));
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
            String message = "Send";
            byte[] value;
            try {
                value = message.getBytes( charsetName: "UTF-8");
                mService.writeRXCharacteristic(value);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }

        }
});
```

Screenshot from Android Studio: Send button functionality. 1. Parsing a string value to an array of bites. 2. Using writeRXCharacteristic feature to transmit short text messages over Bluetooth. The mService is an instance of the UartService class provided by Nordic Semiconductors.

Circuit Playground Bluefruit (CPB) was programmed following the procedures described in the chapter Control Over Bluetooth. This test occurred without any setback, Bio Protech mobile application was able to establish a successful wireless connection with CPB when the button “Connect” was pressed. While both devices were connected and CPB was waiting for new messages, the button “Send” was pressed and the text message was printed on the CPB side, demonstrating an effective data transmission. However, it was noticed a short delay of around 1 sec, which confirms that the UART technology for Bluetooth Low Energy communication has its limitation concerning data length and transfer speed. Despite the fact of these limitations, it is believed that BLE is still effective for the scope of this project. On the other hand, it should be reconsidered if any other feature may be implemented in the future.

E. User Interface Components

Figure 3. From left to right: Design for Main Activity, Graph Activity, Menu View, Setting Activity and Info Activity. Mock-ups done on Draw.io.

Now that we know Bio Protech's app can connect and transmit data to a Circuit Playground Bluefruit via Bluetooth. It is time to start implementing the design, all the code related to Bluetooth connection will not be changed throughout the design implementation. Bio Protech's app is being developed to be used by Parkinsonians patients, we must remind that Parkinson is a neurologic disease that may produce several symptoms, including tremor, vision impairment and dizziness, which causes blurred vision (NHS UK, 2019). It is notorious that those symptoms make simple day-to-day tasks tougher, including pressing a button on a smartphone screen. Considering the difficulties that a Parkinsonian patient will possibly encounter while using our application, the design goal is to minimize those barriers as much as possible.

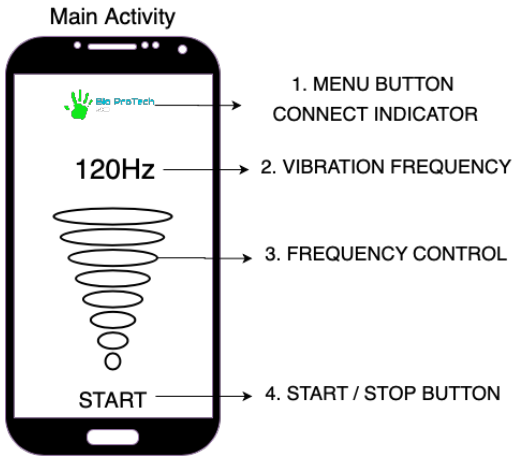
Tremor is the main symptom seen on a Parkinsonian patient and to facilitate the use of our mobile application, Bio Protech will design any clickable element with appropriate dimensions, the greater the element, the easier it will be pressed.

Vision impairment and blurred vision caused by dizziness will directly affect how the user interacts with the application, likewise clickable elements, any text will be displayed with an appropriate dimension or in the form of large icons. In addition, Nick Babich at Smashing Magazine wrote an article “The Underestimated Power Of Color In Mobile App Design” (Babich, 2019), he mentions guidelines that must be followed when developing a mobile application to allow easy usability and to express the application purpose. Considering the guidelines and the visual impairment symptom, the application’s colours will be chosen aiming to display expressive contrast between background and element.

Most of the elements we need are offered on Android Studio such as TextViews and Buttons, but for some of the elements which we will use, we need to design them using external image-editors and import them inside the drawable folder on the Bio Protech’s Android Studio project. The drawable folder is commonly accepted as the place where all the drawings and image resources are stored within an Android application following project structures convention. This folder can be found on Pro Tech/Bio-Protech/app/src/main/res/drawable/.

Main Activity

The Main Activity is the first page displayed and the most important one. In this page, the user will be offered the option to start, stop and regulate the vibration as per the minimum requirement. The user is also provided with the real-time frequency that Park Med is vibrating. On top of the page, there is the Bio Protech’s logo that works as a button and also to indicate whether Park Med connected via Bluetooth or not. The Menu Button/Logo and the Frequency Control must be designed outside Android Studio, while Vibration Frequency and Start/Stop button can be created using the tools presented on the IDE.

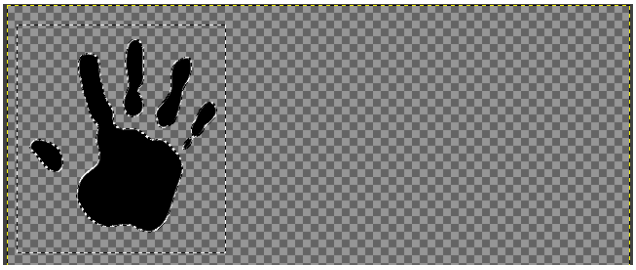


1. Menu Button / Connect Indicator & Bio Protech's logo

Bio Protech's logo consists of two elements, the drawing of a hand and the group's full name "Bio Protech Systems". The hand seen on the logo represents the part of the body that will be affected by the vibration induced by the Park Med, reducing the hand-tremor is the primary goal of the project.

The logo was created using an image editor called GIMP, which is a freely distributed and developed for tasks such as photo retouching, image composition and image authoring. After downloading and installing GIMP, the image editor was opened and a new file called 'logo.png' was created with transparent background, 1920px wide and 800px high. The dimension of components is measured by pixels (px).

The hand icon was created by Adrián Castañeda and is hosted on The Noun Project's website licensed under the Creative Commons, which requires to give the full credit for the icon's creator. To download the icon with good quality and transparent background, we needed to sign up on the website. After the registration was completed, we downloaded the icon and opened it inside the 'logo.png'.

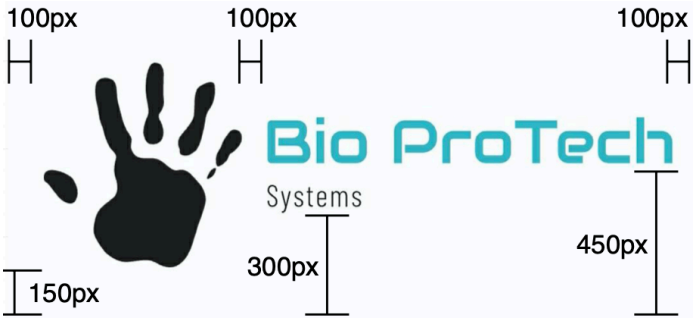


Screenshot of GIMP: The hand icon created by Adrián Casteñeda was added to the left-hand side of the logo.png. The transparent background is represented by chessboard tiling alternating between two shades of grey.

The Bio Protech's full name was written using the Text Tool feature on GIMP. This feature offers the possibility to choose from several types of fonts and sizes. It was tested with different combinations of colours, fonts and size. The final decision was the following: "Bio Protech" was written using the Fontzillion font, size of 200px and the colour turquoise, which is represented by the HEX value of #27BAD1. This colour was chosen for two reasons: It is easy to read on both white and black background and also it is associated with what the group wants to transmit to the users.

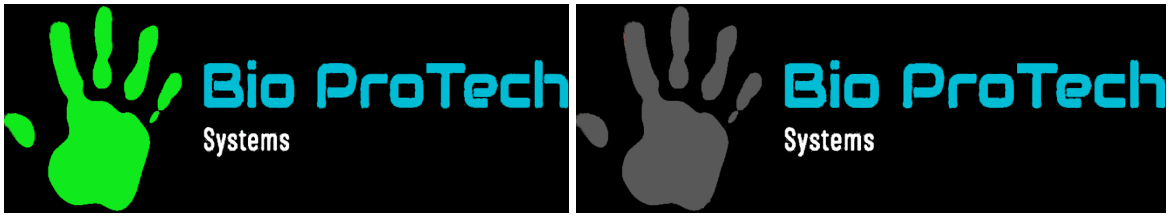
According to Dena Przybyla, a member of the Colour Psychology organization, turquoise is a shade of blue that lies on the scale between blue and green. It has characteristics associated with both of these, such as the calmness of blue and the growth that is represented in green (2020). “Systems” was written with Fontspring, size of 30px and the colour black.

Screenshot of GIMP: The name Bio Protech System was divided into two parts, Bio Protech and System. Each part was written using different types of fonts, colours and sizes.



Screenshot of GIMP: Distance between elements and border. The white background was added only for better visibility.

As mentioned earlier, the hand on the logo will change colour to indicate the Bluetooth connection status. Therefore, using the same image editor, we need to create two new variations of the logo, one to show that the mobile application is connected (green hand) and another to show that it is disconnected (grey hand). The Menu Button will be used on a black background, so we need to change the colour of the word “System” to white to make it visible on the screen. The logo with the green hand was named “logoconnected.png” and the one with the grey hand was named “logodisconnected.png”. The three variations of the logo were added to the drawable folder of the Bio Protech Android Studio’s project.

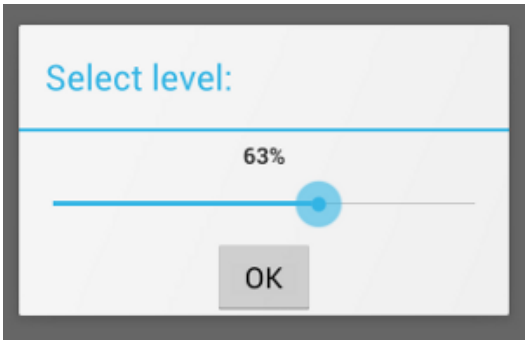


Screenshots of GIMP: On the left, the green hand demonstrates that the app is connected. On the right, the grey hand demonstrates the app is disconnected.

3. Frequency Control

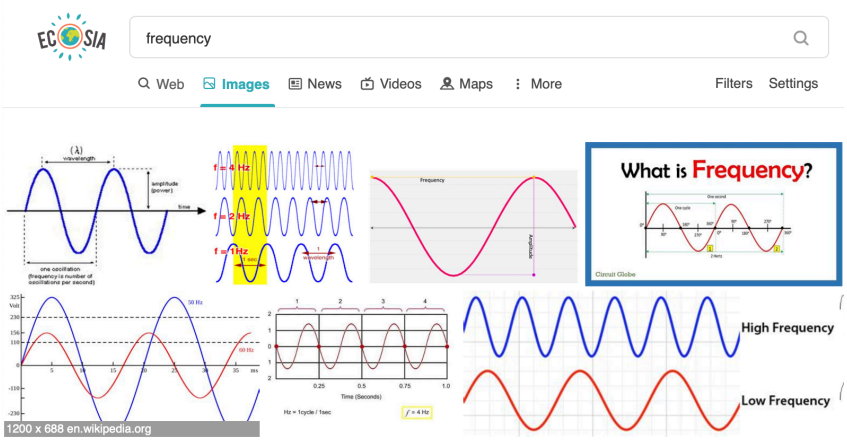
As per the minimum requirement, the users must be allowed to change the vibration frequency accordingly with their needs. Therefore, the Frequency Control element is one of the most important in the whole application and will be the most used one. The first idea was to display two buttons, one “plus” button to increase and one “minus” to decrease the frequency. However, after further considerations, Bio Protech discarded that idea because it would limit the control of the frequency. For example, the frequency range will be from 20Hz to 180Hz, if the frequency is set to 20Hz at the start, and the plus button is programmed to increase it by 10Hz, the user would be limited to choose the frequency in multiples of 10 and it would take 16 button presses to reach 180Hz. If the plus button is programmed to increment the frequency by 5Hz, it would be required to press the button 32 times to reach the maximum allowed. The solution to found was to use the SeekBar widget on Android Studio.

The SeekBar allows us to set the minimum and the maximum range and by sliding the pointer the user can pick any number in between but looking in a design perspective it does not complement any usability for the user. Therefore, we need to design our own SeekBar that will be used on top of the normal SeekBar.

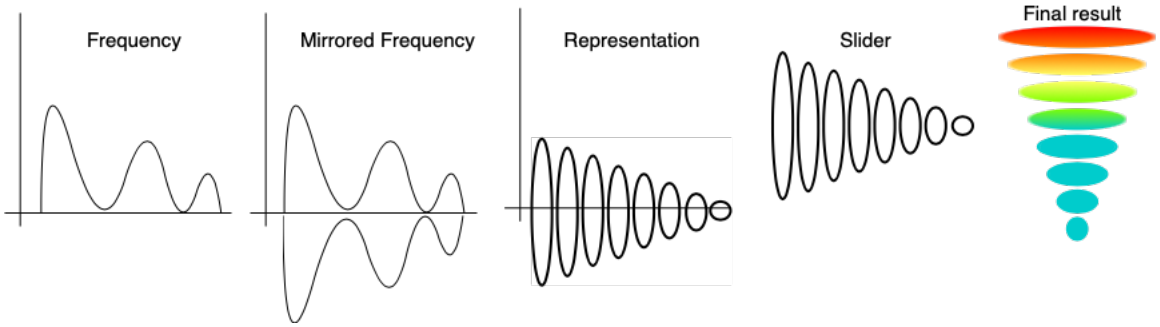


Screenshot of a SeekBar element. It allows the user to pick any number between the range defined.

For the design of our own SeekBar, we wanted something that can represents or has some familiarity to frequency levels. To get some inspiration the word “Frequency” was searched on the web, the result page shows different types of graph that represents frequency, all of them following the same wave format as seen below:



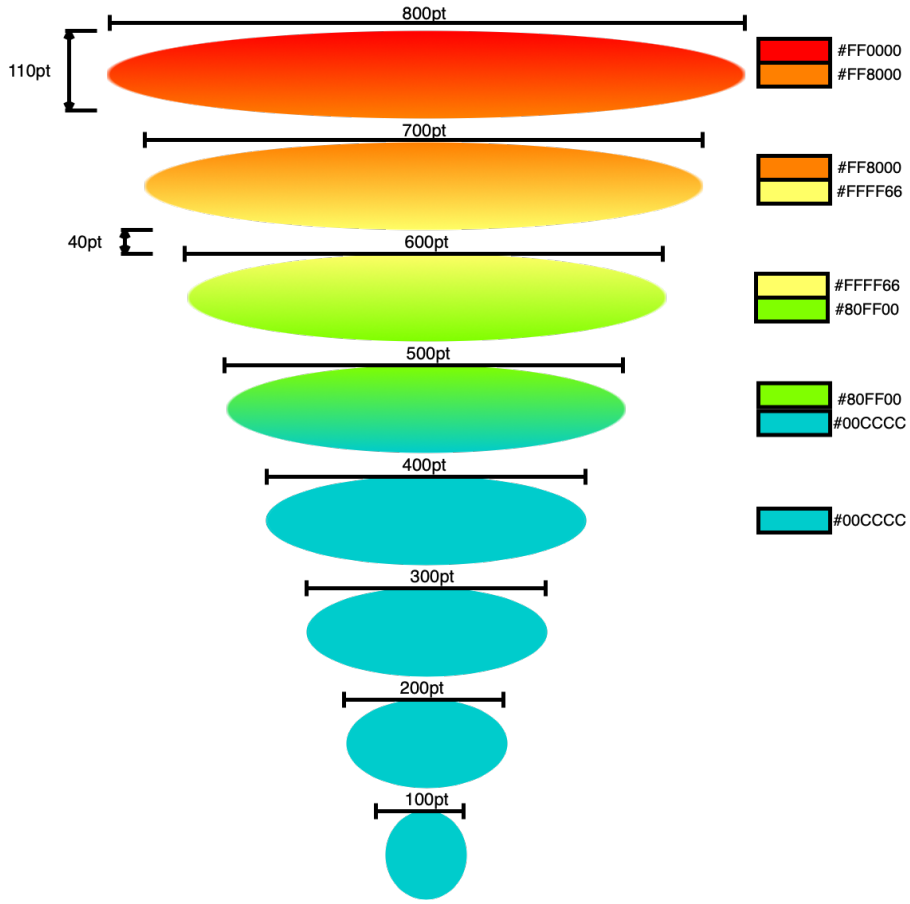
A member of Bio Protech's group came up with an idea inspired by how frequency is usually represented in a graph. The process to create the SeekBar was imagined following the steps: 1. Take any graph representing different levels of frequency as a reference. 2. Mirror the line plotted in the graph. 3. Divide the mirrored line into segments, it will result in oval-shaped segments. 4. Rotate all segments, so that the wider one is on the top, the result will be:



From left to right: 1. Frequency representation in a graph following a wave-length format. 2. The result when the wave is mirrored. 3 and 4. Design of the slider simulating a mirrored wave. 5. Displaying different levels of frequency by wave-length and colours.

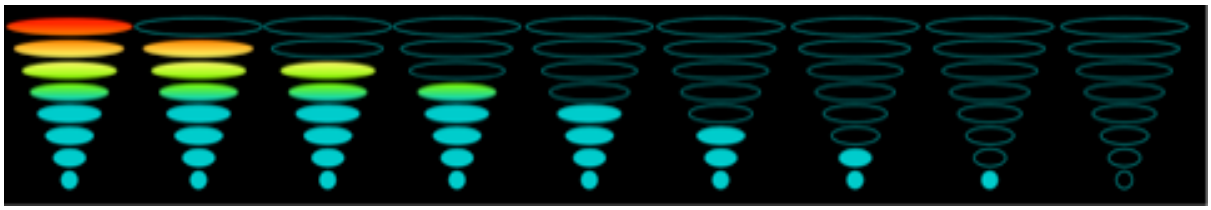
Draw.io is a flowchart maker and online diagram software, it is free and can be used inside a web browser. The slider with oval segments was created to simulate different levels of frequency. We believe that with this type of design the user can visualise the range of frequency allowed. The slider will also indicate the intensity of the vibrations by different shades of colours.

When Draw.io is open with an empty diagram, we can see a frame of 830px of width and 1170px of height, to draw the oval shape, we use the ellipse tool on the general section seen on the left-hand side of the window. Park Med will vibrate in a range from 20Hz up to 180Hz, which means 160Hz of range. If we divide 160 by 20, we will find the number 8 and for that reason, our slider will have 8 ellipses. The dimension and colours used can be seen in the diagram below.



Picture of the final result of the slider showing dimensions and colours selected.

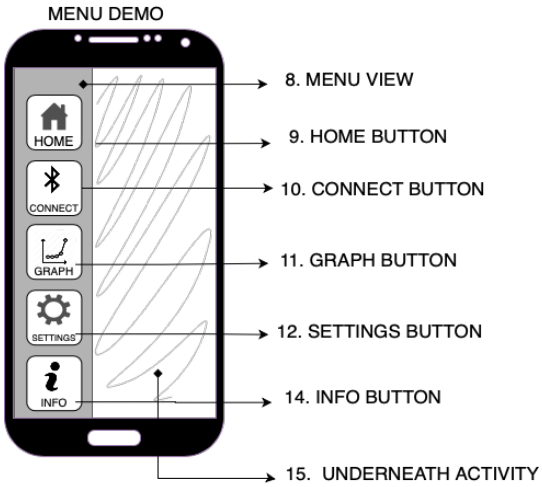
Each time the user moves the finger over the slider, the ellipse will change colours to demonstrate a type of "activated state". This will be done by creating 9 different images of the slider and for each movement over it, the image will be changed. In addition to the one with all ellipses "activated", we need to create 7 more variations. The process to do it was simple, starting from the top, going downwards and for each ellipse that has its colour removed a border of 2pt and colour turquoise (#00CCCC in HEX value) is added. Each variation of the slider was saved on the Drawable folder on Android Studio.



Eight different variations of the slider. They were named from left to right: slider8, slider7, slider6, slider5, slider,4, slider3, slider2, slider1 and slider0. All of them was saved on the drawable folder on Bio Protech's Android Studio Project.

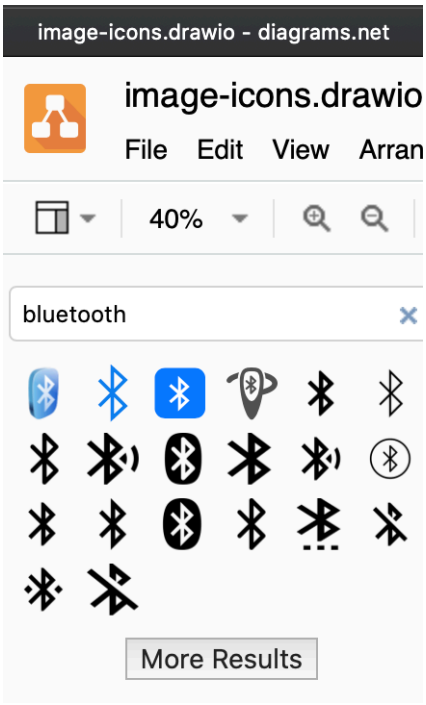
Menu View

Android Studio offers a component called Navigation Drawer, which is a sidebar with the main purpose of offering a type of menu, this bar opens on top of whatever activity the user is when the Drawer is invoked. The biggest problem with using the Navigation Drawer on an Android application is that it usually follows Google's Material Design guidelines thus it usually has the same structure and layout. It can be redesigned, but Bio Protech's purpose is to create buttons and clickable surfaces with suitable width and height to facilitate Parkinsonian patients clicking on it even if they experience severe hand-tremor. The best approach is to create our menu system that works as a Navigation Drawer.



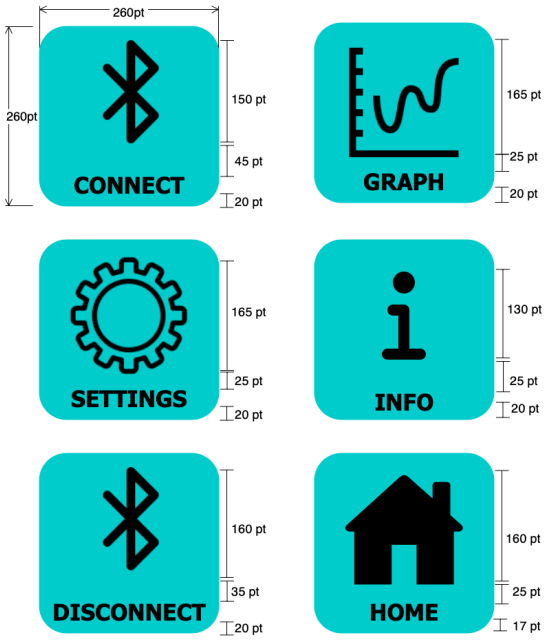
Once open, the menu will display five different buttons: Home, Connect, Graph, Settings and Info. Creating an actual button element for Android demands a lot of work, it is much easier and simpler to draw an image and use it as a clickable ImageView. The biggest advantage of using clickable-images is that we can personalise it as we wish. All the images will be designed by Bio Protech's members using the image-editor Draw.io, including its options of icons.

All the images were drawn using the Rounded Rectangle tool seen on the general section of Draw.io. All of them shared the same size and background colour, 260pt high and 260pt wide and for the colour we have chosen the same turquoise seen on Bio Protech's icon. The turquoise is represented by the HEX value of #00CCCC. Each image consists of a text and an icon related to the text and to find the icon we used the search tool from draw.io. The word searched was the same seen on each button excepting the connect one, which we used the word Bluetooth to find better representation. The Bluetooth button was also the only one to have two variations, one for "connect" and one for "disconnect".



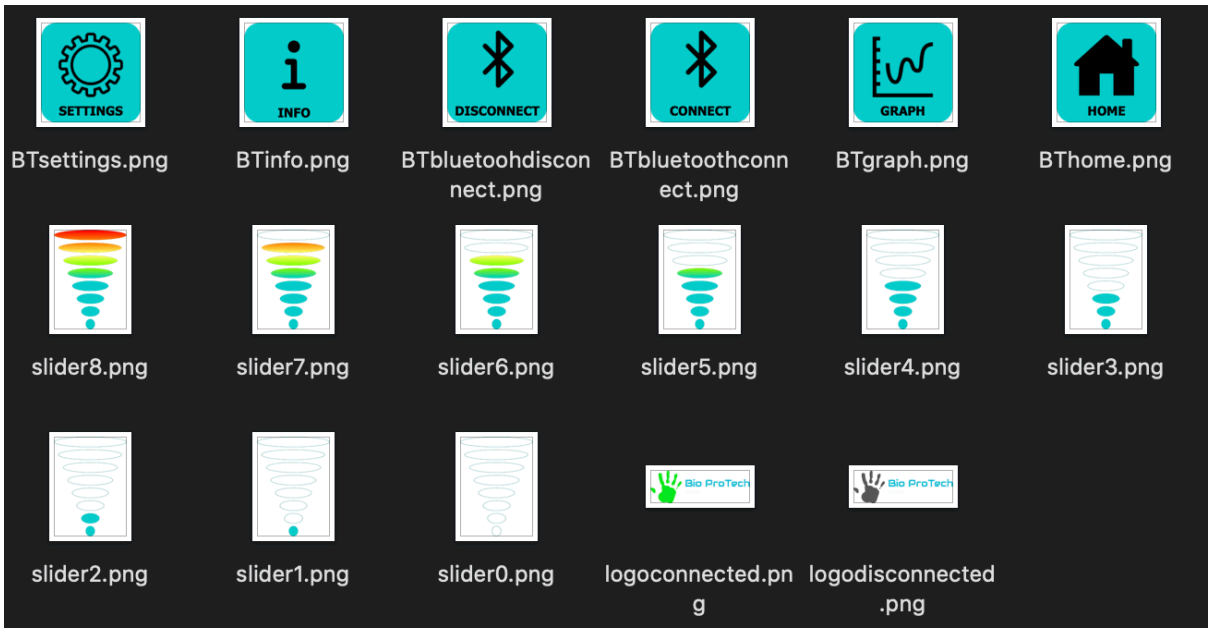
Screenshot from draw.io search tool. The search returns different types of icons.

The image below shows the sizes and spacing used for each button. It is important to mention that icons and texts are horizontally centred, and the font sizes are 32pt. Each image was saved in png format and was named as the junction of BT in upper case plus the name seen on each of them. For example BTsettings.



The design of the five buttons (and Bluetooth variation) showing the size of each icon and spacing between elements and borders.

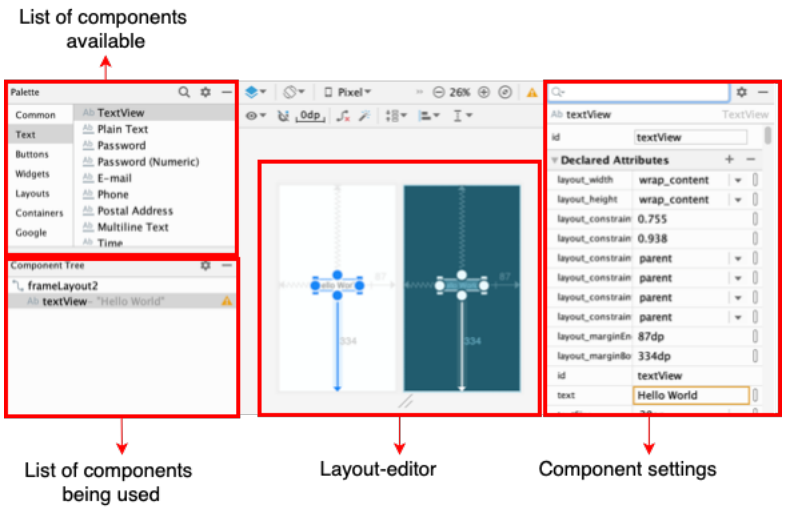
At this point, the drawable folder must contain 17 images in total, which are 2 variations of the Bio Protech's logo, 9 variations of the slider and 6 different image-buttons, including the 2 variations of the *connect* button. These are all the components that were designed outside Android Studio.



Screenshot of the Drawable folder storing the image of all components designed by Bio Protech members.

F. Design Implementation

The language used for design implementation on Android Studios is XML by default. In addition, Android Studios offers a layout-editor that allows developers to drag a component from the list and drop it where it is needed. There are several options of components to choose from such as TextView, Button, CalenderView, ProgressBar and many others. There also different types of Layout, for this project we will be using ConstraintLayout. This type of layout allows us to choose the right position of a component and restrict it there, the restriction works based on the distance to other components and the layout's border. Moreover, with that feature is easy to centre-align components.



Screenshot from Android Studio, using the layout-editor feature. It is easy and simple to add components and choose their positions.

Main Activity

From left to right: 1. The current layout of the MainActivity; 2. MainActivity mock-up. 3. The final design of the MainActivity.

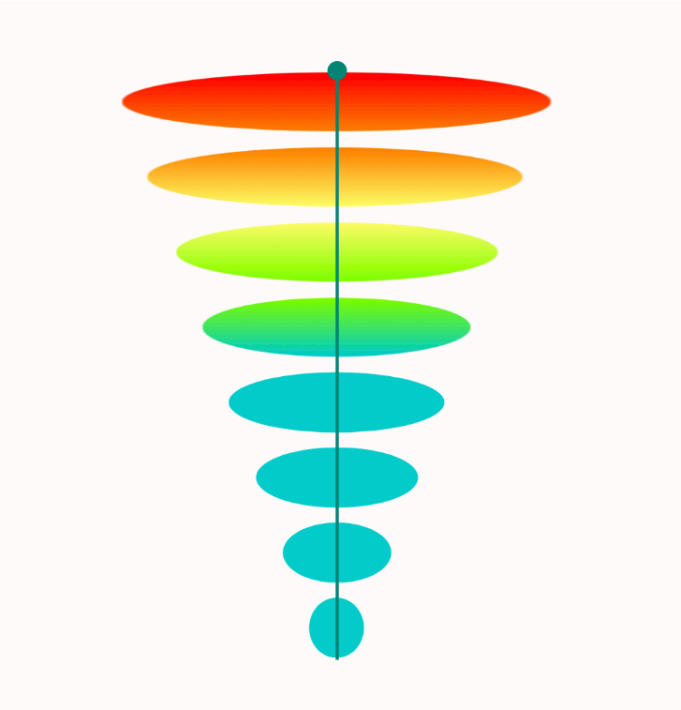
Due to the relative-positioning of ConstraintLayout is recommended to start adding components from the top. In the Main Activity, the first one is Bio Protech's logo that will work as a menu-button and indicate the connection status. For the menu-button will be using an ImageView that can be found amongst the components list on the left-hand side of the screen and to added it, it is just needed to drag and drop inside of the layout so that it will stay right on the top of the screen.

For each component added on a ConstraintLayout, it is required to restrict its position using at least two layout_constraints, the first constraint added for the menu-button was the layout_constrainedWidth, which when set to true it horizontally centre the component. The second constraint added was the distance of 16px from the top of the screen. In order to use the logo as the ImageView, we need to define the source with the logodisconnected path. It can be done through the components settings our adding the following line: android:src="@drawable/logodisconnected". In Android development when a new component is created, we need to define a unique ID so that it can be referenced in the java class. The ID chosen was menuButton.

The next component in the sequence is a TextView intended to display the frequency chosen by the user, it will change accordingly to the value selected on the slider. The new TextView was added below the menu-button following the same layout_constraint to align components to the centre and 16px of distance to the menu-button. The text size was set for 60sp and coloured with a brighter tone of the turquoise being used, which has the HEX value of #00BCD4. The ID was set as *frequencyTF*.

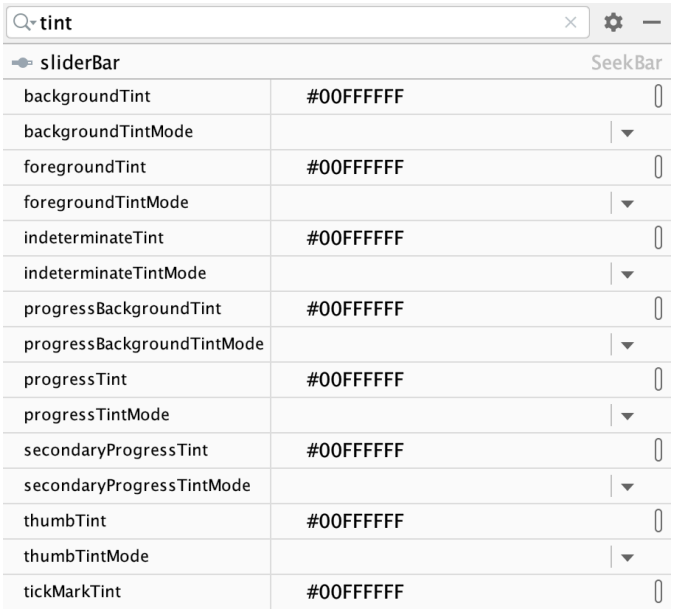
The next element in line is the slider, as mentioned earlier, the slider will enable the option to change the frequency needed by the user. The slider that has been designed by Bio Protech does not have a real programmable function, it is only a set of different images that changes according to the user's touch on the screen. A transparent SeekBar will be used under the slider to capture the input of the user so that when they swipe the slider up or down, this movement will be capture by the SeekBar. Therefore, SeekBar will be responsible for changing the slider images accordingly.

The implementation of this method requires extra caution, the position and size for both slider and SeekBar need to be carefully adjusted to make this adaptation unnoticeable by the user. We will start creating an ImageView and setting the source image to android:src="@drawable/slider8". The image of the slider is centre-aligned, 315px wide and 360px high. The ID was set as "slider". At the exact same position, we added a SeekBar and tagged it with the ID "sliderBar". Now it is needed to align the ends of both components, by default, SeekBars are displayed horizontally and the minimum value is on the left and our slider must be vertical and the minimum value must be on the bottom. The solution to align them is to rotate the SeekBar by 270 degrees using the rotation command *(android:rotation="270")*, it makes the minimum value go to the bottom. Using the resize-view tool on the layout-editor, we can resize the SeekBar so that its ends meet with the ends of the slider.



Screenshot of the Android Studio layout-editor: The ends of the slider meets with the ends of the SeekBar. The green line seen in the middle is the SeekBar.

Now that we have the position and size correct, it is needed to make the SeekBar transparent. On the component settings tool, there is a search bar that allows us to find specific settings to the component selected. If the word “tint” is searched, a list of different settings regarding the colour will be displayed and to make them invisible we need to change the colour value to transparent, which is represented by the HEX value of #00FFFFFF.



Screenshot of Android Studio component-settings: The world tint was searched and all settings regarding the colours were changed to transparent.

The next element is the Start/Stop button. It will be done using a clickable TextView. It follows the same alignment from other components and 25 pixels of distance to the slider. The default text was set to "START" with size set for 36sp and colour changed to the same turquoise tone seen on the Frequency TextView, #00BCD4. Now that all components needed were added to the MainActivity, we will define the background colour to black, HEX value of #000000. At this point, MainActivity's layout is completed and it is a good place to start working on the MenuView.

Menu View

1. Mock-up of the MenuView; 2. MenuView final result over the MainActivity

The MenuView consists of a side-panel with 5 buttons, the panel will open when the menu-button is clicked and closed when any area outside the panel is clicked. The technique used to open and close is by making the entire MenuView visible and invisible. We can start by adding a vertical LinearLayout on the left-hand side of the screen. The dimension was set to 190dp of width, the height matching with the parent view and to make it appears on top of all the other components of the screen, we needed to set the elevation to 1dp.

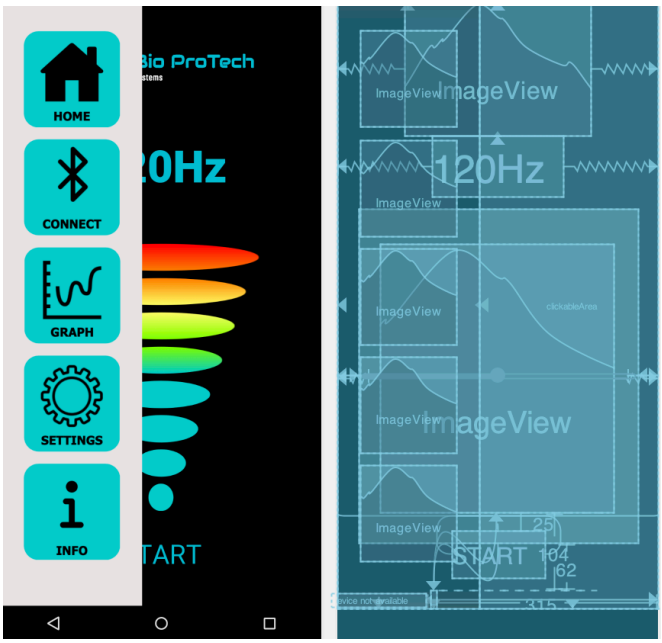
The five buttons are also going to be implemented using clickable ImageViews. We will start by adding an ImageView inside of the menu's LinearLayout. The size was set to 130dp by 130dp and to make it centre-aligned the layout-gravity was set to centre. The first button on the list is the HomeButton so we need to define the source image with the file BThome path and define the ID as homeButton. After we defined the source image, Android Studio was accusing "file not found", even though all images were in the drawable folder, they could not be found by the rest of the application.

This issue was being caused by the uppercase letters in the name of the files, the solution was changing from “BThome” to “button_home” and following the same naming convention for all images.

Having done that, we can copy and paste the button 4 times, it will create 4 buttons with the same size and alignment. After that, it is just needed to change the source image and the ID for each button. Following the order Home, Bluetooth, Graph, Settings and Info.

Button Name	Source Image	ID
HomeButton	@drawable/button_home	homeButton
BluetoothButton	@drawable/button_bluetooth	bluetoothButton
GraphButton	@drawable/button_graph	graphButton
SettingsButton	@drawable/button_settings	settingsButton
InfoButton	@drawable/button_info	infoButton

The command to close the MenuView will be executed when the user clicks anywhere outside of the panel. To capture this click, we need to create a new layout beside the MenuView covering the rest of the screen. It will not hold any component inside; the only purpose of this layout is offering a clickable surface outside the MenuView and for that reason, the ID defined was clickableArea. In this case 250dp of width was enough to cover the rest of the screen and the height was set to fill the parent view (fill_parent attribute).



On the left: The final result of the MenuView over the MainActivity, this is how the menu will be displayed once opened from the MainActivity. On the right: The blueprint of the MainActivity with the MenuView open. It is possible to see all types of components used.

The MenuView must be hidden by default so that when Bio Protech's app is open, the user won't see it unless the MenuButton is pressed. In order to hide it, we need to set the visibility of the entire LinearLayout to invisible using the command android:visibility="invisible". Going back to the MainActivity Java class, we need to create Java objects that correspond to the ones added in the MainActivity layout.

```
private TextView frequencyTF, startButton;
private ImageView menuButton, sliderView, bluetoothButton, graphButton, settingButton, homeButton, infoButton;
private FrameLayout clickableArea;
private SeekBar sliderBar;
private LinearLayout menu;
```

Screenshot from Android Studio: All the components added to the layout need to be created as Java objects inside the MainActivity java class.

Inside the onCreate method in the MainActivity java class, we need to reference each Java object to its corresponding view using the findViewByID method. Also, we need to enable the clickable function to all ImageViews and to the start/stop TextView.

```
//This is the textview intended to display the current frequency
frequencyTF = (TextView) findViewById(R.id.frequencyTF);

//This is the start/stop button, it needs to be set to clickable
startButton = (TextView) findViewById(R.id.startStop);
startButton.setClickable(true);

//Tagging each ImageView with their respective views.
homeButton = (ImageView) findViewById(R.id.homeButton);
menuButton = (ImageView) findViewById(R.id.menuButton);
bluetoothButton = (ImageView) findViewById(R.id.bluetootButton);
graphButton = (ImageView) findViewById(R.id.graphButton);
settingButton = (ImageView) findViewById(R.id.settingButton);

//Making them clickable to simulate a button
homeButton.setClickable(true);
menuButton.setClickable(true);
bluetoothButton.setClickable(true);
graphButton.setClickable(true);
settingButton.setClickable(true);

//ClickableArea is the layout added to fill the rest of the screen
//when the menu is open
clickableArea = (FrameLayout) findViewById(R.id.clickableArea);

//sliderBar is the transparent SeekBar under the slider
sliderBar = (SeekBar) findViewById(R.id.sliderBar);
//sliderView is the slider-images designed by Bio Protech
sliderView = (ImageView) findViewById(R.id.slider);

//Menu is the sidepanel with five buttons
menu = (LinearLayout)findViewById(R.id.menu);
```

Screenshot from Android Studio: All the java objects need to be referenced to their corresponding view and some of them is set to be clickable.

The MenuButton has three functions: First, it will open the MenuView by changing its visibility to visible. Second, it will enable the clickableArea by setting its isClickable attribute to true. Lastly, we need to avoid that a click event will be captured by the correct element, it can be done by disabling all the clickable elements seen in the MainActivity, which are the Slider and the StartButton. It is recommended to check if the MenuView is currently invisible. Once pressed, the clickableArea will have the opposite function from the MenuButton.

```
//The menu button is responsible for opening the MenuView
menuButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Check if the menu is invisible
        if (menu.getVisibility() == View.INVISIBLE) {
            //Make the menu visible
            menu.setVisibility(View.VISIBLE);
            //Enable the clicklableArea
            clickableArea.setVisibility(View.VISIBLE);
            clickableArea.setClickable(true);
            //Disable all the buttons of the main activity outside the menu
            startButton.setClickable(false);
            sliderBar.setVisibility(View.INVISIBLE);
        }
    }
});
```

Screenshot from Android Studio: When the menuButton is pressed, it will check if the menu is invisible, then it will make the menu visible using the setVisibility and enabling the clickableArea.

```
//The clickableArea will have the opposite function of the menu button
clickableArea.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //first check if the menu is visible
        if (menu.getVisibility() == View.VISIBLE) {
            //Then make the menu invisible
            menu.setVisibility(View.INVISIBLE);
            //Make itself non clickable and invisble
            clickableArea.setVisibility(View.INVISIBLE);
            clickableArea.setClickable(false);
            //Enable all the buttons of the main activity outside the menu
            startButton.setClickable(true);
            sliderBar.setVisibility(View.VISIBLE);
        }
    }
});
```

Screenshot from Android Studio: When the clickableArea is pressed, it will check if the menu is visible, then it will make the menu invisible and disable its own clickable function.

The implementation of the slider requires some extra steps. SeekBars do not use onClickListener to capture the click events, we need to use setOnSeekBarChangeListener instead. When this listener is used, it is required to implement three methods: onProgressChanged, onStartTrackingTouch and onStopTrackingTouch. In our project we will use the onProgressChanged and the onStopTracking.

The onProgressChanged is invoked when the user slides the finger over the SeekBar, resulting in changing the value. In our case, it will be responsible for changing the value of the frequencyTF and updateing the slider image according to the frequency range. Bio Protech's slider has 8 different levels represented by ellipse-shaped segments.

The frequency supported by the motor discs is from 20Hz to 180Hz. Therefore, 160Hz of range. If we divide the frequency range (160Hz) by the number of segments (8), we will find the number 20. We can use this number to specify the frequency covered by each segment. For example, the first segment on the top of the slider will represent the frequency between 160Hz and 180Hz, the second will represent the range of 140Hz to 159Hz.

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    //First check if the menu is not open
    if(menu.getVisibility() == View.INVISIBLE){
        //Update the progress (frequency) to the frequencyTF
        frequencyTF.setText(progress+"Hz");
        //Anything bellow 20hz will 'deactivate' all the segments
        if(progress <= 20){
            sliderView.setImageResource(R.drawable.slider0);
        }
        //This is the first segment starting from the bottom
        //Anything between 20hz and 40Hz will only activate the first segment
        else if(progress <= 40 && progress >= 20){
            sliderView.setImageResource(R.drawable.slider1);
        }
        //This is the second segment starting from the bottom
        //Anything between 40hz and 60Hz will only activate the first two segments
        else if(progress <= 60 && progress >= 40){
            sliderView.setImageResource(R.drawable.slider2);
        }
        //This is the third segment starting from the bottom
        //Anything between 60hz and 50Hz will only activate the first three segments
        else if(progress <= 80 && progress >= 60){
            sliderView.setImageResource(R.drawable.slider3);
        }
        //This is the fourth segment starting from the bottom
        //Anything between 80hz and 100Hz will only activate the first four segments
        else if(progress <= 100 && progress >= 80){
            sliderView.setImageResource(R.drawable.slider4);
        }
        //This is the fifth segment starting from the bottom
        //Anything between 100hz and 120Hz will only activate the first five segments
        else if(progress <= 120 && progress >= 100){
            sliderView.setImageResource(R.drawable.slider5);
        }
        //This is the sixth segment starting from the bottom
        //Anything between 120hz and 140Hz will only activate the first six segments
        else if(progress <= 140 && progress >= 120){
            sliderView.setImageResource(R.drawable.slider6);
        }
        //This is the seventh segment starting from the bottom
        //Anything between 80hz and 100Hz will only activate the first seven segments
        else if(progress <= 160 && progress >= 140){
            sliderView.setImageResource(R.drawable.slider7);
        }
        //This is the eighth segment starting from the bottom
        //Anything between 80hz and 100Hz will activate all eight segments
        else if(progress <= 180 && progress >= 160){
            sliderView.setImageResource(R.drawable.slider8);
        }
```

Screenshot from Android Studio: Implementation of the onProgressChanged method. This method belongs to the setOnSeekBarChangeListener.

The setOnSeekBarChangeListener is invoked when the user releases the slider. Ideally, the user will first select the frequency needed and then release the slider, with this is mind, we can use this event to send the frequency select over Bluetooth. It is important to mention that all Bluetooth connectivity and data transmission were tested in earlier stages. See chapter Transmitting Data Over Bluetooth.

```
    //When the user releases the finger from the slider this method is invoked
    public void onStopTrackingTouch(SeekBar seekBar) {
        //First check if the app is connected to a park med device
        if (mState == UART_PROFILE_CONNECTED) {
            //Converting the integer value to a String
            //The F letter will help the Park Med device to identify that the new message is a frequency
            String message = String.valueOf("F" + frequency);
            try {
                //Creating a array of bytes extracted from the message variable
                byte[] value = message.getBytes( charsetName: "UTF-8");
                //Sending the new frequency to the
                mService.writeRXCharacteristic(value);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
        } else {
            //if not connected, show a Toast message
            showMessage( msg: "Park Med is not connected!");
        }
    }
});
```

Screenshot from Android Studio: Implementation of the onStopTrackingTouch method. This method belongs to the setOnSeekBarChangeListener

The clickable TextView Start/Stop was named startButton because it will act as a button. It will be responsible for changing the text shown on the screen and sending the command to start or stop the vibrations via Bluetooth. In order to know which command to send over Bluetooth, we will use the text of the TextView as reference.

```
startButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //First check if the app is connected to a park med device
        if (mState == UART_PROFILE_CONNECTED) {
            //Create an empty string
            String message = "";
            //First check if the button is currently "START"
            if (startButton.getText().equals("START")) {
                //Set the message value to "START"
                message = "START";
                //Change the text of the startButton to STOP
                startButton.setText("STOP");
            } else {
                //Set the message value to "STOP"
                message = "STOP";
                //Change the text of the startButton to START
                startButton.setText("START");
            }
            try {
                //Creating a array of bytes extracted from the message variable
                byte[] value = message.getBytes( charsetName: "UTF-8");
                //Sending the new message over bluetooth
                mService.writeRXCharacteristic(value);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
        } else {
            //if not connected, show a Toast message
            showMessage( msg: "Park Med is not connected!");
        }
    }
});
```

Screenshot from Android Studio: The startButton, which is actually a clickable TextView, will send START or STOP commands via Bluetooth and update the text seen on the screen.

For the buttons inside of the MenuView, we can start from the HomeButton, which will redirect the user for the MainActivity. In this case, the user is already in the MainActivity, so the only thing that needs to be done is hiding the MenuView. We could copy and paste the same code from the clickableArea listener method, instead we can simulate a click event using the *performClick* method.

```
homeButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //The home button and the clickableArea have the same function
        //We can simulate a click in the clickableArea instead of copying the dode
        clickableArea.performClick();
    }
});
```

Screenshot from Android Studio: Implementation of the onClick method of the HomeButton.

The second button is the BluetoohButton, for this we will reuse the code implemented in the chapter "*Establishing a Bluetooth connection*". In addition, we need to disconnect the Bluetooth using the same button.

```
bluetoothButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Check if is disconnected
        if (mState == UART_PROFILE_DISCONNECTED) {
            //Show a message to ensure that the device is on
            showMessage( msg: "Make sure Park Med is on!");
            //Create a new blutooth device using the bluetooth adapter.
            //For now, the MAC address "C4:51:8C:F2:2A:29" will be hardcoded
            mDevice = BluetoothAdapter.getDefaultAdapter().getRemoteDevice( address: "C4:51:8C:F2:2A:29");
            //Connect to "C4:51:8C:F2:2A:29"
            mService.connect( address: "C4:51:8C:F2:2A:29");
        } else {
            //If is connected
            if (mDevice != null) {
                //Disconnect
                showMessage( msg: "Disconnecting...");
                mService.disconnect();
            }
        }
    }
});
```

Screenshot from Android Studio: Implementation of the onClick method of the BluetoothButton. It will connect or disconnect the application depending on the Bluetooth status.

The other three buttons GraphButton, SettingsButton and InfoButton will open different activities once pressed. First, we need to create three empty activity by right-clicking on top of the project folder and selecting New, Activity and then Empty Activity. This opens a window where we can enter the name we want to define and then it is just needed to click on Finish.
The name of the three new activities are GraphActivity, SettingsActivity and InfoActivity. All of them were created using the same procedure.

This is the first time that we will start a new activity once a button is pressed, we need to find a way to do such thing. On the Android Developers Docs website, there is a tutorial of how to do it. In this tutorial, it is suggested to build an Intent, which is an object that provides runtime binding between separate components, such as two activities. The Intent represents an app's intent to do something (Android Doc, 2020), in this case the app intents to start a new activity. The tutorial also shows how to pass information between activities by using the putExtra method. In our GraphActivity, we need to display the frequency that the motors are vibrating and because we already have this information on the MainActivity, we can pass it into the GraphActivity.

```
graphButton.setOnClickListener((v) → {
        //Build a new Intent with the MainActivity as context and GraphActivity as destination
        Intent intent = new Intent( packageContext: MainActivity.this, GraphActivity.class);
        //Getting the value from the frequency text field, parsing to a string
        //And put it into the intent
        intent.putExtra( name: "Frequency", frequencyTF.getText().toString());
        //Start the new Activity
        startActivity(intent);
});
```

Screenshot from Android Studio: Implementation of the onClick method of the GraphButton. It will build a new intent, put the current frequency into it and start the new activity.

The same procedure was followed to start the other two activities once their respective button is pressed. The only difference is that we don't need to pass any information between the activities.

```
settingButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Once settings button is pressed, build a new intent and start the new activity
        Intent intent = new Intent( packageContext: MainActivity.this, SettingsActivity.class);
        startActivity(intent);
    }
});

infoButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Once info button is pressed, build a new intent and start the new activity
        Intent intent = new Intent( packageContext: MainActivity.this, InfoActivity.class);
        startActivity(intent);
    }
});
```

Screenshot from Android Studio: Implementation of the onClick method of InfoButton and SettingButton, it follows the same procedure to start a new activity

G. Starting Vibration and Changing Frequency

Starting Vibration and Changing Frequency

At this point, the Bio Protech's app is configured to establish a Bluetooth connection with the Park Med device. It is also configured to start, stop and change the frequency of the vibrations. As explained in the Control Over Bluetooth chapter, Circuit Playground Bluefruit (CPB) can capture a stream of bytes coming from the Bluetooth module, convert it to a text and activate different functions using if/else statements. We need to match the commands sent from the app and received by Park Med to their respective functions. In other words, we need to match the commands with if/else conditions.

Currently, Bio Protech's app sends three types of commands to Park Med:

- START: It should start the vibrations.
- STOP: It should stop the vibrations.
- F120: It should change the frequency of the vibration. It is divided into 2 parts. First, the letter 'F', which is hardcoded to facilitate Park Med recognise it as the new frequency. A second part is an integer number taken from the user's input.

Combining this information with the configuration and tests previously done with CPB, we can create the conditions needed to activate those functions. In the image below, we can see how the variable data_string is used in the if statement to validate the stream coming from the Bluetooth. It is good practice to make the large code as much modular as possible, so instead of writing everything inside of the ble.connected loop, we will use the if statements to call different methods.

```
if data_string == "START":
    print('START')
    # Initialise all the motors with PWM
    motor1 = pulseio.PWMOut(board.A1, duty_cycle= 65534, frequency=50000)
    motor2 = pulseio.PWMOut(board.A2, duty_cycle= 65534, frequency=50000)
    motor3 = pulseio.PWMOut(board.A3, duty_cycle= 65534, frequency=50000)
    motor4 = pulseio.PWMOut(board.A4, duty_cycle= 65534, frequency=50000)
    motor5 = pulseio.PWMOut(board.A5, duty_cycle= 65534, frequency=50000)
    # Call the start method, if a new message from the bluetooth is captured
    # inside of the start method, it will be returned
    data_string = start()
if data_string == "STOP":
    print('STOP')
    stop()
```

Screenshot of Mu-editor: Circuit Playground Bluefruit validates the START and STOP command coming from the Bluetooth stream using the same methodology described in the chapter Control Over Bluetooth.

In order to validate the new frequency command, we need to check if the stream contains the letter F, then we need to extract the number attached to the stream and convert it to an integer value. The validation can be done using the "is equal" condition, but we need to specify the position of the letter F. In this case, the letter F can be found in the first position of the text. The extraction can be done using the split method, this method returns a list of strings with pieces of texts between each 'F'. For example, if the text is F100F200F300, the first index of the list would be null, because there is nothing before the first F. The second index would be 100, because this is the number before the second F and so on. Due to the format 'F100' received from the stream, the number will be stored in the second index of the list.

```
49
50                  # Check if the first character in the data_string is 'F'
51                  if data_string[0] == "F":
52                      # Extract the number
53                      f = data_string.split("F")
54                      # Convert the number to an integer value.
55                      frequency = int(f[1])
56                      print(frequency)
57              except ValueError:
58                  continue    # or pass.
```

Screenshot of Mu-editor: Circuit Playground Bluefruit checks if the first letter of data_string is an "F". If so, the number is extracted from the text and converted to an integer value.

In the chapter Full Frequency Control with PWM, Circuit Playground was configured to make motor discs vibrate in different frequencies using the pulse with modulation technique and transistors. Following the same coding procedure seen in the referred chapter can be adapted to the start() method. Also, to test the first Park Med's prototype connected to the Bio Protech's mobile application, five motors will be used instead of only two, which is the number of motors for the final product.

It is important to program Circuit Playground so that it will keep checking if there is a new stream coming from the Bluetooth, if the user changes the frequency while the motors are vibrating, Park Med should be able to detect it and update the frequency without stopping the vibrations. Changing the frequency dynamically may help the users to find the most suitable intensity to them.

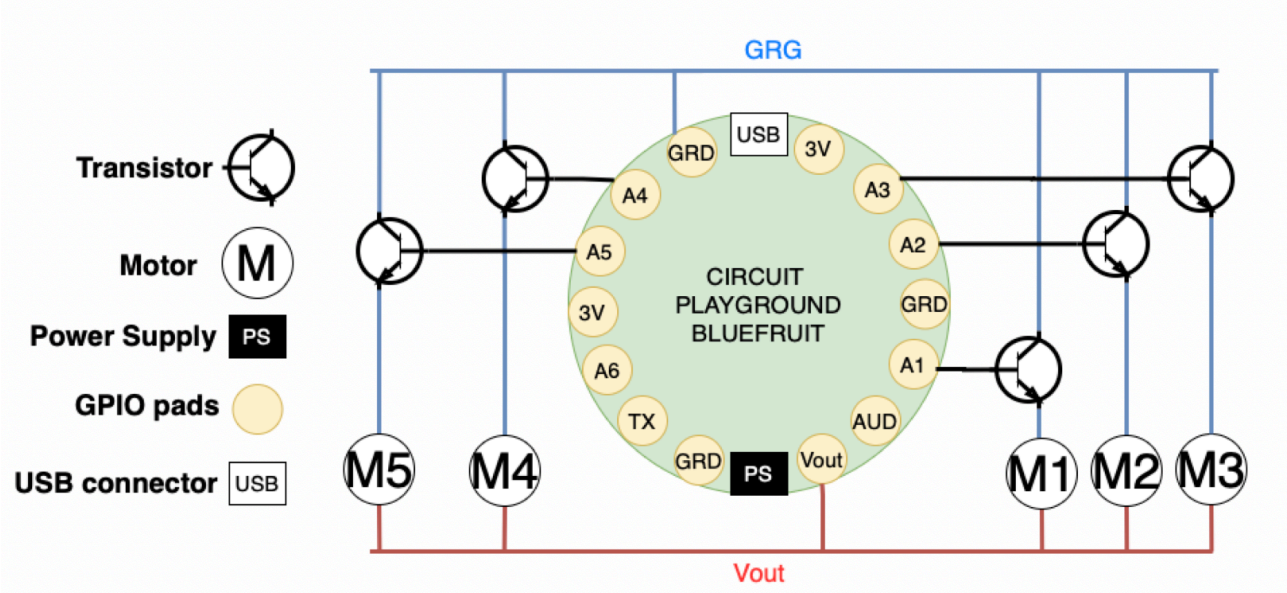


Diagram showing the digital circuit connection with Circuit Playground Bluefruit, five motor discs and five transistors. The diagram above was created based on the knowledge acquired in the chapter Full Control with PWM.

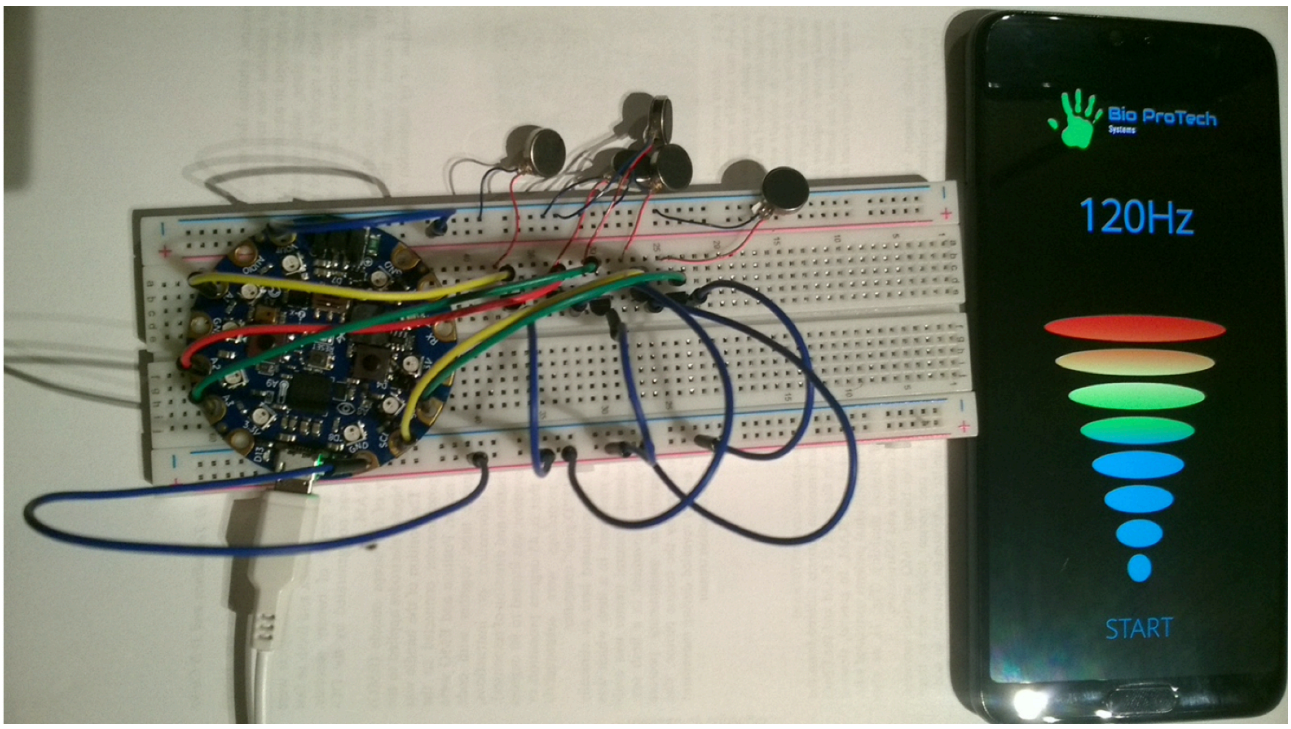
It is important to program Circuit Playground so that it will keep checking if there is a new stream coming from the Bluetooth, if the user changes the frequency while the motors are vibrating, Park Med should be able to detect it and update the frequency without stopping the vibrations. Dynamically changing the frequency may help the users to find the most suitable intensity to them.

```
42  def start():
43      # Initialise all the motors with PWM
44      # Loop until there is a new message from the Bluetooth
45      streamMessage = None
46      data_string = ""
47      while len(data_string) == 0 and ble.connected:
48          # Define the intensity of the vibration using pwm duty_cycle
49          motor1.duty_cycle = duty
50          motor2.duty_cycle = duty
51          motor3.duty_cycle = duty
52          motor4.duty_cycle = duty
53          motor5.duty_cycle = duty
54          # Check if there is new message from the Bluetooth
55          streamMessage = uart_service.readline()
56          data_string = ''.join([chr(b) for b in streamMessage])
57          if len(data_string) > 0:
58              return data_string
```

Screenshot of Mu-editor: Implementation of the start() method, using the same procedure done on the chapter Full Control with PWM. In the code, five motors are initialized using the pulseio library and receive a duty_cycle of 65535, which is stored in a global variable called duty. In addition, inside of the while loop, the program must check if there is any new message coming from the Bluetooth, if there is, it should return to the main loop, where the new message is going to be validated.

Looking back at the selecting and stopping the vibrations use case scenario. Once the user opens Bio Protech's mobile application, it should display the Home Screen, where the user is offered the options to start and stop the vibration and select the frequency. All the user's input must be sent from the app and received by Park Med via Bluetooth. At this point, the app and Park Med are programmed and able to execute a test based on this scenario.

The test will be executed by one of the Bio Protech's members acting like a normal user. Circuit Playground Bluefruit along with five motors and transistors were connected using a breadboard and the mobile application was installed on the smartphone of the member. Once open, the mobile application displayed the Home Screen, the hand on Bio Protech's logo was not green showing that Park Med was not connected. The connection was established by clicking on the connect button in the menu. When the Bluetooth connection was successful, the hand immediately turned green.



A system with Circuit Playground Bluefruit, five motor discs and five transistors. On the smartphone, it shows the Home Screen of Bio Protech's mobile application.

After the button "START" was pressed two things happened. First, the text seen on the button changed to "STOP". Afterwards, all the five motors started vibrating in a nearly unnoticed intensity, even though the frequency displayed on the app was 120Hz and the slider was at the maximum, which are the default ones. This behaviour was noted as a minor issue to be fixed.

The user held one of the motors between the thumb and the index finger to have tactile feedback regarding the change of frequency. The first frequency tested was 80Hz, the jump of intensity was noticed around 1 second after new frequency was selected using the slider. The second jump of intensity was also experienced when the maximum frequency of 180Hz was selected. At 180Hz, the noise generated by all five motors is clearly audible and it may even result in some sort of discomfort to the users. After a few seconds vibrating at 180Hz, two of the motors got disconnected from the breadboard and stopped vibrating, this behaviour demonstrated that all connections must be strong enough to handle constant vibration regardless of the intensity. When the "STOP" button was pressed, the vibrations came to a complete halt in the three remaining motors and the text of the button was changed back to "START".

The two motors that got disconnected from the circuit were reconnected to the breadboard and all the connections were reinforced using pieces of tapes. The user proceeds again to start the vibration and change the frequency to 180Hz, this time the connections handled the strong intensity of the vibrations. Different frequencies were selected by the user and the changes could be felt by both tactile feedback and noise.

The test based on the given use case scenario was completed with success. In addition, it revealed that some of the minimum requirements were met such as:

- The Bio-Protech App should send the user's input to the Bio-Band.
- The Park Med should receive and read the user's input, then vibrate according to the frequency selected by the user

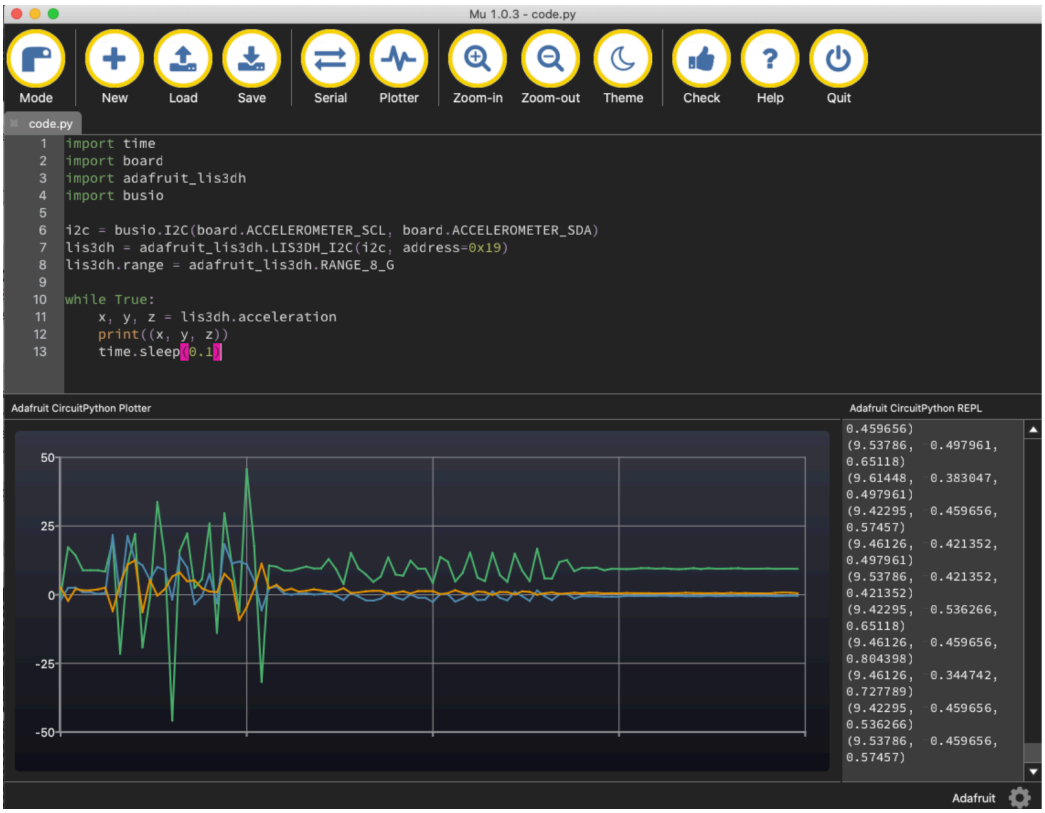
H. Plotting Graph for Tremor Analyses

Bio Protech is aiming to develop a medical device that is not only capable of reducing hand-tremor of Parkinsonians patients by applying vibration to the wrist, but also provide a system that allows users to monitor how the vibrations affects the hand-tremor. It is known that if users are using Park Med's vibration, they will essentially feel the change of the tremor's intensity. However, a deeper analysis would be possible by providing a graphic representation with numerical data. In addition, if a user decides to use Park Med with the assistance of a doctor to find the best frequency, the doctor will need consistent data to undergo a series of tests until a suitable frequency is found. For that reason, Bio Protech will use the built-in Park Med's accelerometer sensor to collect the hand-tremor's data and plot it into a graph which will be displayed in our mobile application in real-time. There will be also an option to save the data.

An accelerometer is a piece of electromechanical equipment that measures acceleration forces. These forces could be static, like the force of gravity, or they may be dynamic, caused by moving or vibrating the accelerometer. Park Med's built-in accelerometer sensor can measure how fast the device is moving by calculating its acceleration according to the Earth's gravity plus the acceleration resulting from the movement on a three dimensional axis.

Graph with Accelerometer Data

The software will identify the 3-Axis Accelerometer and local a default data-collection setup.
In the code below, it was applied the concept of the 3 Axis accelerometer on Circuit Playground Bluefruit. The 3-Axis represents the acceleration on X, Y and Z. This acceleration is achieved by using the gravity's force, whether the board is stopped or not, one of the axes will mark at least 9.8m/s, which is the average force of gravity.



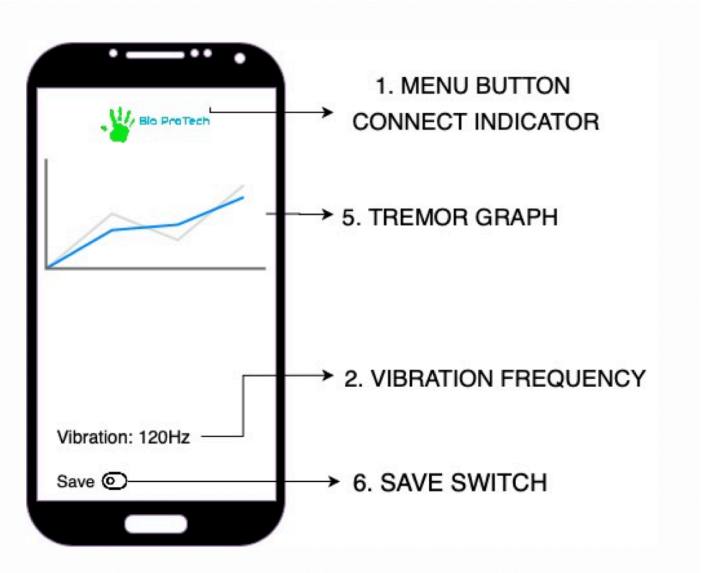
There are some modules imported in this code, such as import time, board, adafruit_lis3dh, and busio. On the while loop, the parameters x, y, z are assigned to be the motion values (Based on the 3 Axis Accelerometers' concept) which is then printed to the console. It was added into the code the time.sleep(0.1) that slows down the readings a little, otherwise it would run too fast to read.

The graph shown in the screenshot represents Park Med's microcontroller moving around to see the plotter responding in real-time. CPB has an accelerometer in the centre of the board, depending on which direction the board is moved, the x, y or z values will change.

In the second approach, a small code has been implemented, which plots a graph with only one parameter, that is the data we are aiming to analyse. For this particular implementation and better performance, the graph has been reduced from 3 components acceleration to only 1. The reduction was calculated using the net acceleration formula, which is given by the square root of the sum of the squares of the three axes accelerations.

Despite using the net acceleration to plot the graph, we are aware this is not an optimum solution to gather data from the hand-tremor, due to the fact the readings will be interfered by the rotation of the sensor, by any other movement the user does and also by the gravity factor, which is a common issue in most accelerometer's sensor. A genuinely optimum algorithm for consistent tremor-data is beyond the scope of the project. However, the main focus of this graph is to collect data during the interaction between the patient and the Park Med. Users are still able to use Bio Protech's graph plotter to analyse how the invoked vibrations affect their hand-tremor.

I. Android Application Graph Activity



The Graph Activity is where the user will be given the option to analyse the consequences of the invoked vibrations in their hand-tremor. The analyses will be done using a graph plotted in real-time with data coming from the accelerometer presented on Circuit Playground Bluefruit. In addition, the user will be able to see the frequency of the vibrations and also an option to save all the data into the internal memory of the device.

We can start building the Graph Activity by adding the menu button and the menu panel. They are both similar to the ones on Main Activity, and it will allow us to reuse the layout and the java code and adapt them to the newest activity. There are two ways to copy the components needed. However, the easiest one is to open the Main Activity XML file and copy the ImageView representing the menu button and the Linear Layout of the menu panel along with the five buttons inside of it. Due to the ConstraintLayout used to determine the position and spacing of the components in the main layout, which are already defined, the settings will remain once the components are copied and pasted. In other words, there is no need to reposition the menu and the menu button on the Graph Activity.

The only changes that must be done are on the java code. If the user is in the home page and decides to open the menu and press the Home Button, this event hides the menu panel. Meanwhile, in the Graph Activity, the menu should be hidden once the Graph Button is clicked, also if the user presses the Home Button, it should be sent back to the Home Page.

It is good to remind that to open new activities we are using Intents. When Intents are used, we need to pass two values as parameters, the current activity and the activity that will be started. The drawback of this technique is that the newest activity opens on top of the previous activity. Even though the previous activity is not displayed, it remains open on the background. The advantage is that we can manually close the newest activity using the finish() method and the previous one will be seen again. The use of the finish() method is also important to not allow the user to stack several activities on top of each other.

The rule that we defined to close activities as the user interacts with the application is simple. The only activity allowed to remains open in the background is the Main Activity, any other activity will be closed before a new one starts.

```
homeButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //The Home Button will close the Graph Activity
        //and return the user to the Main Activity
        //which is running in the background
        GraphActivity.this.finish();
    }});
```

Screenshot from Android Studio: Once the Home Button is pressed, the Graph Activity will close, and the user will be sent back to the Main Activity.

```
graphButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //The graph button and the clickableArea have the same function
        //We can simulate a click in the clickableArea instead of copying the code
        clickableArea.performClick();
    }
});
```

Screenshot from Android Studio: Once the Graph Button is pressed, the menu will be hidden.

Now that the menu button and the menu panel are working, the next component of the Graph Activity is the actual graph. Android Studio does not offer tools to plot graphs and display in the screen by default. The solution is to choose from a variety of options available online. Due to the Bio Protech group not been familiar with graph plotting in Android or any other software, we have researched for the easiest tool available in an open-source distribution. The search returned a link to a question asked on the Quora website, 'What is the best open-source graphing library available for Android?'. This question was answered by several Quora's users, in the first answer with most votes, which means most people agreed to it, Victor Horta, a computer engineering student, suggested using Androidplot and gave some reasons such as simple and straightforward, it is fully customizable and supports dynamic plotting.

In more in-depth research on the Androidplot website, we have found out that the Androidplot is an open-source library kept under the Apache 2.0 license, which allows anyone to use, modify and distribute the code. The library is maintained by Nick Fellows with help from contributors on Github. The website also shows all the features the Androidplot library supports, including line charts and real-time plotting, both features are needed to plot a graph using accelerometer data coming from Park Med. Besides, on their Github page, it is offered a quick start tutorial and examples of how to use different types of graph, one of the examples provided is OrientationSensorExampleActivity.

In this example the graph is plotted using data coming from the accelerometer of the smartphone, which is similar to what Bio Protech wants to achieve, the only difference is that we want to plot the graph using the data from the Park Med device. The supported features and the guidance offered by Androidplot are the reasons for using it for our project.

As mentioned in the quick start tutorial, in order to use the library it is needed to add the dependency to our project and to include the graph to the Graph Activity's layout, it is needed to add an XML element of tag com.android.xy.XYPlot with height, width, title and because we are adding it to a ConstraintLayout, we need to define the constraints, as shown below.

```
dependencies {
    api 'com.android.support:support-v4:28.0.0'
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:design:28.0.0'
    implementation "com.androidplot:androidplot-core:1.5.7"
}
```

Screenshot from Android Studio: The Androidplot dependency was added to the Gradle.build file on Bio Protech's Android project.

```
<com.androidplot.xy.XYPlot
    android:id="@+id/plot"
    style="@style/APDefacto.Dark"
    android:layout_width="420dp"
    android:layout_height="500dp"
    android:layout_marginEnd="8dp"
    ap:layout_constraintEnd_toEndOf="parent"
    ap:layout_constraintHorizontal_bias="0.956"
    ap:layout_constraintStart_toEndOf="@+id/menu"
    ap:layout_constraintTop_toBottomOf="@id/menuButton"
    ap:title="Tremor Analyses" />
```

Screenshot from Android Studio: An XML element of tag XYPlot was added to the Graph Activity layout. The height and width were defined so that the XYPlot covers most of the screen. The constraints defined will position it below the Menu Button.

By adding the XYPlot element, it shows a grey area that represents where the graph will be displayed to the user. Despite not having anything attached to the XYPlot, the application could not be launched to the smartphone. Android Studio displayed an error saying that the newest dependency could not be found. After some research and trying different approaches to solve the issue, the only thing that surprisingly worked was found in the section of "Known issues" on the Developer Android website. In the article, it is suggested to restart the IDE, wait for the Gradle builder process to finish and run the application once again.

Due to the familiarity of the OrientationSensorExampleActivity with our goal, we will be using the source code as a guide to creating our graph plotter along with the Dynamic Plot tutorial found on Nick Fellows Github page (Fellows, 2016). It is important to mention that graph used by Bio Protech's app is an adaptation of an existing source code created by Androidplot's founder Nick Fellows, the Apache 2.0 license agreement will be followed and referenced on the Graph Activity java class.

The best approach to redesign a graph plotter as per requirements is first to understand how the Androidplot library works and the only way to truly understand it is by checking what each method, variable or object does.

The first thing noticed is the four types of variables declared in the global scope of the class. Among the variables, we could see an XYPlot, which is the java equivalent to the XYPlot XML element added to the layout. The second is SimpleXYSeries, this is the variable responsible for storing the numbers plotted into the graph. There is also a static final variable called HISTORY_SIZE, which is the length of the X-axis. Furthermore, the last one is a Redrawer object, this object is what makes the graph dynamic, therefore allowing us to make a graph plotted in real-time. The four types of variables were added to the GraphActivity java class, outside the onCreate method, so that it will be globally accessible.

```
//History_Size represents the length of the graph
private static final int HISTORY_SIZE = 120;
// XYPlot is the x and y axis
// SimpleXYSeries is the graph plotted into x and y
// FrequencyBar will be displayed as a dynamic Bar Chart
private XYPlot FrequencyBar = null;
private SimpleXYSeries FrequencyBarPlot = null;
//FrequencyLine will be displayed as a dynamic line.
private SimpleXYSeries FrequencyLinePlot = null;
private XYPlot FrequencyLine = null;
//The Redrawer allows us to create a dynamic graph
private Redrawer redrawer;
```

Screenshot from Android Studio: XYPlot, SimpleXYSeries, HISTORY_SIZE and Redrawer were added to the GraphActivity.java.

Inside of the onCreate() method, we can link the XYPlot java object to the XYPlot view using the findViewById method, FrequencyBar and FrequencyLine were linked to the same view because we want that both graphs to be displayed in the same space. In other words, one graph will overlap the other giving the impression that there is only one XY axes and two data representation. Also, we need to instantiate new SimpleXYSeries, although its constructor requires a String title as a parameter, we can leave it blank because the title of our graph is already written as part of the XYPlot view.

```
//Tag FrequencyBar and FrequencyLine to the same view
//So it will be visible as only one graph
FrequencyBar = (XYPlot) findViewById(R.id.plot);
FrequencyLine = (XYPlot) findViewById(R.id.plot);
//Instantiate a new SimpleXYSeries,
//The title is blank because it's already defined in the XML file
FrequencyBarPlot = new SimpleXYSeries( title: "");
FrequencyLinePlot = new SimpleXYSeries( title: "");
```

Screenshot from Android Studio: XYPlot java objects were linked to the same XYPlot view. The XYPlot view is the XML element added to the layout.

In the Androidplot library, the x axis is called the domain and the y axis is called range. In order to define the limit of both axes, it can be used two methods called setDomainBondaries and setRangeBoundaries. In both methods, we need to pass three parameters, the lower boundary, which is the value that we want the axis to start, the upper boundary, which is the value that the axis can reach, and the third parameter is an ENUM value determining whether the axis can grow or not.

For the domain axis, the FrequencyBar received 0 as lowerBoundary because we want to display the bar on the 0 position of the X-axis. The upperBoundary was set to 1 because there is no need to declare a higher number since the bar will occupy only one space. The BoundaryMode was defined as fixed to forbid the axis to change value or grow. On the other hand, the upper boundary for the FrequencyLine's domain axis was defined as the same value of the HISTORY_SIZE. Considering that Park Med will send new data every 0.1 seconds, it will take around 5 seconds to reach the end of the X-axis.

For the range axis, both FrequencyBar and FrequencyLine were set with the same boundaries, 0 as the lower since we will not get negative net acceleration. The upper boundary was set to 120 because this was the maximum frequency observed while we were using the mu-editor's plotter. However, the BoundaryMode was set to auto to allow any change of boundaries according to what is needed.

```
//The method setDomainBoundaries is used to define the limits of the X axis
FrequencyBar.setDomainBoundaries( lowerBoundary: 0,  upperBoundary: 1, BoundaryMode.FIXED);
FrequencyLine.setDomainBoundaries( lowerBoundary: 0, HISTORY_SIZE, BoundaryMode.FIXED);
//The method setRangeBoundaries is used to define the limits of the Y axis
FrequencyBar.setRangeBoundaries( lowerBoundary: 0,  upperBoundary: 120, BoundaryMode.FIXED);
FrequencyLine.setRangeBoundaries( lowerBoundary: 0,  upperBoundary: 120, BoundaryMode.AUTO);
```

Screenshot from Android Studio: The limits of the X and Y axes were set using the methods setDomainBoundaries and setRangeBoundaries.

Now we need to add the graph type. The graph type represents how the data will be displayed to the user. In this case, it will be displayed in the form of a bar and a sequential line. The method used to define the graph type is called addSeries, which receives two parameters, a SimpleXYSeries object and a Formatter object. We already have two SimpleXYSeries objects, so we need to create new Formatters, one for the bar and one for the line.

The bar can be created using the BarFormatter class included in the Androidplot library. It requires two colours as parameters, one to fill the bar and one for the border. The border will remain black, but the colour used to fill the bar is the same turquoise colour used in Bio Protech's logo, the RGB value for the turquoise is 0, 200, 200.

The line is created using another class called LineAndPointFormatter, this class allows to create a line with data points. However, since we do not want to display data points and only display the line, we just need to pass the colour of the line as a parameter, all the rest referred to the points can be left as null.

```
//The addSeries method is responsible for linking the XY axis to the type of graph needed
FrequencyBar.addSeries(FrequencyBarPlot,
        // The BarFormatter creates a bar chart
        new BarFormatter(
                Color.rgb( red: 0,  green: 200,  blue: 200),
                Color.rgb( red: 000,  green: 000,  blue: 000)
        ));
FrequencyLine.addSeries(FrequencyLinePlot,
        // The LineAndPointFormatter creates a line
        new LineAndPointFormatter(
                Color.rgb( red: 0,  green: 200,  blue: 200),
                 vertexColor: null,  fillColor: null,  plf: null));
```

Screenshot from Android Studio: The bar and line graph types were added to the plotting area, the colour given to them is the same turquoise used in other parts of the application.

Androidplot also allows for redesigning the graph area, including adding labels, subdivisions for the axes and the format of the numbers displayed. Regarding the x-axis, there is no reason to make subdivisions because the x value is not used to plot the graph. For the Y-axis, it was added nine subdivisions, for each subdivision the number will be incremented by 15. In order to format the number so that it will not show any decimal value, it was used the DecimalFormat method passing a '#' as the parameter.

```
//The setDomainStepMode defines how many divisions for the x axis
//StepMode.SUBDIVIDE divides the axis in n parts
FrequencyLine.setDomainStepMode(StepMode.SUBDIVIDE);
//The setDomainStepValue is the n number of divisions
FrequencyLine.setDomainStepValue(0);
//The setDomainStepMode defines how many divisions for the y axis
FrequencyLine.setRangeStepValue(9);
//The setRangeLabel is the label seen on the Y axis
FrequencyLine.setRangeLabel("Frequency");
// Do not display the decimal value for the labels
FrequencyBar.getGraph().getLineLabelStyle(XYGraphWidget.Edge.LEFT).
        setFormat(new DecimalFormat( pattern: "#"));
```

Screenshot from Android Studio: The labels, number of subdivisions and the format of the number were defined.

According to Nick Fellows source code, two elements are vital to make the graph be plotted in real-time. For the line, we need to define that the x value is not used to plot the graph, it can be done by adding the method useImplicitXVals. For better clarification, the x value is not used because the y values are plotted in continual sequence, it can go backwards in the x-axis, only forward. Each y value is plotted after the previous one. The second element is the redraw object, which is responsible for updating the graph after any change is applied to the graph.

```
// The useImplicitXVals allows us to plot the line without an x value
FrequencyLinePlot.useImplicitXVals();
// Initialise the Redrawer object
redrawer = new Redrawer(
        Arrays.asList(new Plot[]{FrequencyLine, FrequencyBar}),
         maxRefreshRate: 1000,  startImmediately: false);
```

Screenshot from Android Studio: The two crucial elements for dynamic plotting were added.

Now that the graph is already created, it is needed to get the data from the accelerometer sensor integrated to the Park Med via Bluetooth. The class used to handle the wireless communication to Park Med is called BroadcastReceiver and for the GraphActivity, the only thing that BroadcastReceiver needs to check if there is a new stream coming from the Bluetooth adapter. Once a new stream arrives with new data, we need to convert it to a String with UTF-8 format because it comes in an array of bytes.

A second conversion must be done before plotting the new data, parsing the String to an Integer value. The next step is to plot the data into the bar graph, then it is needed to check if there is space in the x axis, if there is no space we need delete the first y value of the line, then add the new data to the last position.

```
// The BroadcastReceiver is the class responsible for bluetooth communication
private final BroadcastReceiver UARTStatusChangeReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //The action ACTIO_DATA_AVAILABLE
        // is triggered when there is new stream coming from the bluetooth
        if (action.equals(UartService.ACTION_DATA_AVAILABLE)) {
            //Add the extra data to an array of bytes
            final byte[] txValue = intent.getByteArrayExtra(UartService.EXTRA_DATA);
            runOnUiThread(() -> {
                    try {
                        //Convert it to a string with UTF-8 format
                        String text = new String(txValue, charsetName: "UTF-8");
                        //Parse it to a integer
                        int tremor = Integer.valueOf(text);
                        // Update the bar with the newes data:
                        FrequencyBarPlot.setModel(Arrays.asList(
                                new Number[]{tremor}),
                                SimpleXYSeries.ArrayFormat.Y_VALS_ONLY);

                        // If the x-axis is already full, remove the first Y value of the line
                        if (FrequencyLinePlot.size() > HISTORY_SIZE) {
                            FrequencyLinePlot.removeFirst();
                        }
                        // And the the newest data to the last position of the line
                        FrequencyLinePlot.addLast( x: null, tremor);
                    } catch (Exception e) {
                        Log.e(TAG, e.toString());
                    }
            });
        }
    }
};
```

Screenshot from Android Studio: The data coming from the Bluetooth is captured by the BloadcastReceiver, converted to String, parse to an Integer value and plotted to the graph.

Now it is needed to send the command to the Park Med device requesting the data from the accelerometer sensor. The activation of the command will follow the same method seen in the chapter Starting Vibration and Changing Frequency. In this chapter, CPB was programmed to validate three types of commands such as start, stop and change the frequency of the vibrations.

For the graph, once the user presses the Graph Button on the Menu, the application will transmit over Bluetooth a “GRAPH” text to the device. It was observed repetition of the code needed to send text over Bluetooth, instead of repeating the same code for each command, a method called sendMessage() was created, this method receives a String as a parameter, which is the text meant to be sent. In addition, due to the fact that both Park Med and Bio Protech’s mobile application are getting more sophisticated, we have added a Java thread that will run inside of the sendMessage method, the reason for that is to make sure that the command sent will be validated by Park Med. The solution elaborated is sending a standard text 1.5 second before the command, it will prepare the device to the upcoming message, any text would work, but we have chosen “PKG” shorten for package.

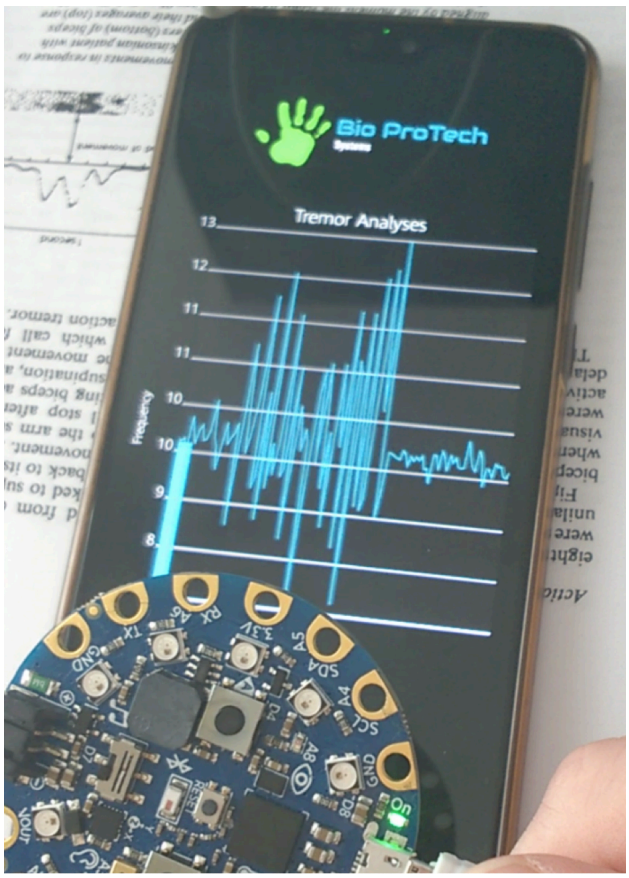
```
public static void sendMessage(String text) {
    final String message = text;
    new Thread((Runnable) () → {
            byte[] value;
            try {
                //Send a PKG string to prepare Park Med to the command
                value = "PKG".getBytes( charsetName: "UTF-8");
                mService.writeRXCharacteristic(value);
                //Sleep for 1 second
                Thread.sleep( millis: 1500);
                //Convert the message to a array of bytes
                value = message.getBytes();
                //Send it over bluetooth
                mService.writeRXCharacteristic(value);
            } catch (UnsupportedEncodingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
    }).start();
}
```

Screenshot from Android Studio: Implementation of the sendMessage method. This method is responsible for a more sophisticated data transmission over Bluetooth.

Park Med must be programmed to validate the newest command, once it is validated a method graph() will be called. The code created in the previous chapter will be applied inside of the method, with the addition of sending the data to the mobile application and also keep checking for new streams from the Bluetooth module. At this point, Circuit Playground Bluefruit has been only programmed to read data the Bluetooth module using the readLine method, but now we need to write the data. According to CircuitPython documentation provided by Adafruit Industries, the method needed to write data is called write() and belongs to the uart_service library, which is the same library of the readLine method.

```
59  i2c = busio.I2C(board.ACCELEROMETER_SCL, board.ACCELEROMETER_SDA)
60  lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c, address=0x19)
61  lis3dh.range = adafruit_lis3dh.RANGE_8_G
62
63  def graph():
64      while True:
65          x, y, z = lis3dh.acceleration
66          tremor = math.sqrt((x * x + y * y + z * z))
67          time.sleep(0.1)
68          uart_service.write(str(tremor))
```

Screenshot from Mu-editor: The implementation of the graph method, which is called once the “GRAPH” message is validated. The write() method requires a String as a parameter.



Simulation of the hand-tremor graph with data coming from Circuit Playground Bluefruit via Bluetooth. The graph was plotted in real-time.

After the newest version of the mobile application was installed and the new configuration of Park Med was saved. The real-time graph was tested while one of Bio Protech's member was holding Circuit Playground Bluefruit and simulating hand-tremor. The test was considered a success, the graph showed the hand-tremor in real-time and the y axis grew as much as it was required. The data collected from the accelerometer sensor and plotted into the graph will allow a deeper analysis of how the invoked vibration delivered by Park Med's motor discs affects the hand-tremor of Parkinson's disease patients.

Going back to the Graph Activity, the next component is a TextView intended to display the frequency the motors are vibrating. This frequency is displayed on the Main Activity and it was already configured to be passed to the Graph Activity via intent.putExtra method. Check the Menu View's implementation for more details. Now, it is just needed to retrieve the data and set it to the TextView. The process to add the TextView to the layout was simple, first the ID was defined as frequencyTF, the height and width were set to wrap-content, the text was set to size 24sp and the colour white. The code below was used to get the frequency passed from the MainActivity and display it in the screen if the motors are vibrating.

J. Saving Tremor Data

The last component on the graph activity is a Switch button, which will allow the user to save the same data used to plot the graph. This feature was included in the project intending to help doctors who may want to assist a patient with Parkinson diseases to regulate the Park Med device according to their needs. By collecting the hand-tremor while different levels of vibration are being applied to the wrist, the doctor will have enough data before recommending a specific range. Since Bio Protech's app does not have an option to retrieve the data and show it on the screen; instead, we will use an external website to show the saved graph.

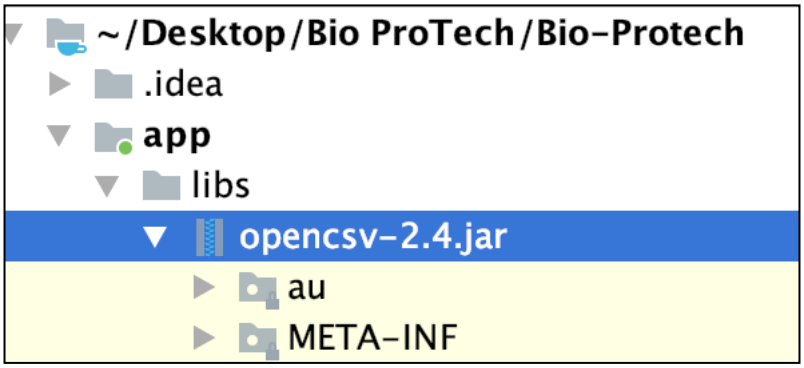
First, it is needed to place the Switch Button on the layout and constrain it to the bottom of the screen, then to enable the ON/OFF feature, we need first to set the onClickListener and when the user presses the Switch, the onClick method will be in charge of checking whether the switch is selected or not.

```
//The switch button to save the graph data
save.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Check if the switch is ON or OFF
        if(save.isChecked()){
            //Starting recording the data
        }else{
            //Stop recording and save the file
        }
    }
});
```

Screenshot from Android Studio: The implementation of the Switch's onClick method.

The first attempt to save the data was to record the graph in a video format, the idea was to develop a screenrecorder tool that would only register the Graph View. However, after seeking assistance from CCT lecturers and trying various different approaches to achieve the desired result, it was decided to abandon the idea of a video. Instead, the graph will be saved in a CSV (Comma-Separated Values) format. This type of file is usually displayed in a table format, where each line is a row of the table and the columns values are divided by comma. For the tremor analyses, two parameters will be save the frequency of the tremor and a counter incremented by 0.1 for each row, the reason is that Park Med sends new data every 0.1 second.

Although the implementation was straightforward, two tutorials had to be followed, Ahmed Jubayer provided the first one on Code Project website(2012). In this short tutorial, Ahmed gives a brief explanation of how to write a CVS file. Moris provided the second tutorial at AndroidWave(2018). It was used to create the name of the file with a timestamp on it, plus the frequency of the vibration. The saving feature starts with downloading an extra Java library called opencsvw and adding it to the /libs folder of the project.



Screenshot from Android Studio: Opencsv.jar was added to the libs folder. The file is available at: http://opencsv.sourceforge.net

Differently from the tutorial where the data was hardcoded and written to the file, for our project we needed to create an ArrayList to store each new accelerometer data coming from the Bluetooth. However, this process will only happen if the Save Switch is on, the solution to create a globally accessible boolean value and insert it into the Switch's onClick method where it will be changed based on the Switch's status.

```
save.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(save.isChecked()){
            //Start saving
            isSavingGraph = true;
        }else{
            //Stop saving
            isSavingGraph = false;
        }
    }
});
```

Screenshot from Android Studio: Change the value of the boolean whether the user wants to save the graph or not.

Now that it is possible to verify if the user wants to save the graph, we can change the BroadcastReceiver method to not only plot the graph but also to add the same values to the ArrayList.

```
            // If the x-axis is already full, remove the first Y v
            if (FrequencyLinePlot.size() > HISTORY_SIZE) {
                FrequencyLinePlot.removeFirst();
            }
            // And the the newest data to the last position of the
            FrequencyLinePlot.addLast( x: null, tremor);
            // Verify if the data needs to be saved.
            if(isSavingGraph){
                // If so, add it to the array
                frequencyArray.add(String.valueOf(tremor));
            }
        } catch (Exception e) {
            Log.e(TAG, e.toString());
        }
});
```

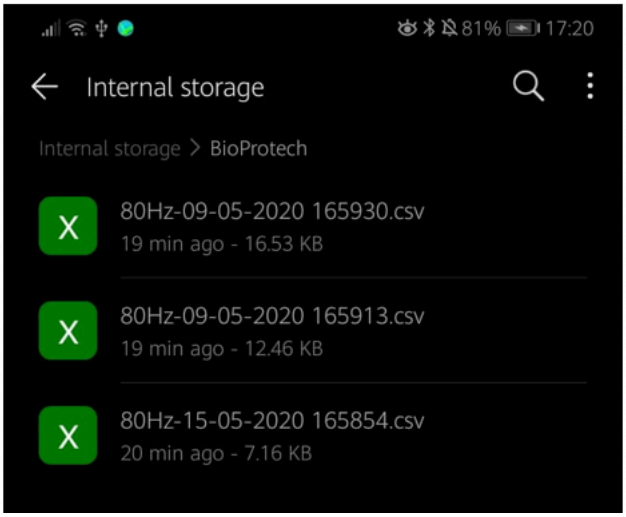
Screenshot from Android Studio: The BroadcastReceiver method of the GraphActivity was changed to enable the graph saving feature.

The code needed to create the file is simple. First, we need to get access to the smartphone's directory and check if there is a folder called Bio Protech, if there is not, the app should create one. The next step is to create the actual file, where the data will be stored. In this case, the file names will contain the current vibration frequency and a timestamp, as mentioned earlier. The opencsv library has a class called CSVWriter. This class is the most crucial part of the process. It extracts the data from an Array of Strings which is then converted to a new line in the file using the CSV format.

```
}else{
    isSavingGraph = false;
    showMessage( msg: "Saving");
    //Get access to the directory of the smartphone and try to get the /Bio Protech folder
    File directory = new File( parent: Environment.getExternalStorageDirectory()+"/BioProtech",  child: "");
    //If the folder doesn't exist create one
    if (!directory.exists()) {
        directory.mkdirs();
    }
    //Create a date format as 31-05-2020 114034
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy HHmmss");
    //Create the file name with the timestamp and the frequency of the vibrations
    //Eg. 120Hz-dd-MM-yyyy HH:mm:ss.csv
    String fileName = frequency+ "Hz-" + dateFormat.format(new Date()) + ".csv";
    //The file class receives the directory folder and the file name
    File file = new File(directory, fileName);
    try {
        //Start the new file
        file.createNewFile();
        //Use a CSVWriter Library to convert the plain text file
        //to a CSV format
        CSVWriter csvWrite = new CSVWriter(new FileWriter(file));
        //Start the counter
        timer = 0;
        //The first row of the file works as a table header
        String title[] = {"Frequency", "Time"};
        //Write the header to the file
        csvWrite.writeNext(title);
        for(String item : frequencyArray) {
            //Each row contains a frequency and the timer counter
            String row[] = {item, String.valueOf(timer)};
            csvWrite.writeNext(row); // Write the newest row to the file
            timer+= 0.1; //increment the timer by 0.1
        }
        csvWrite.close();
    }
```

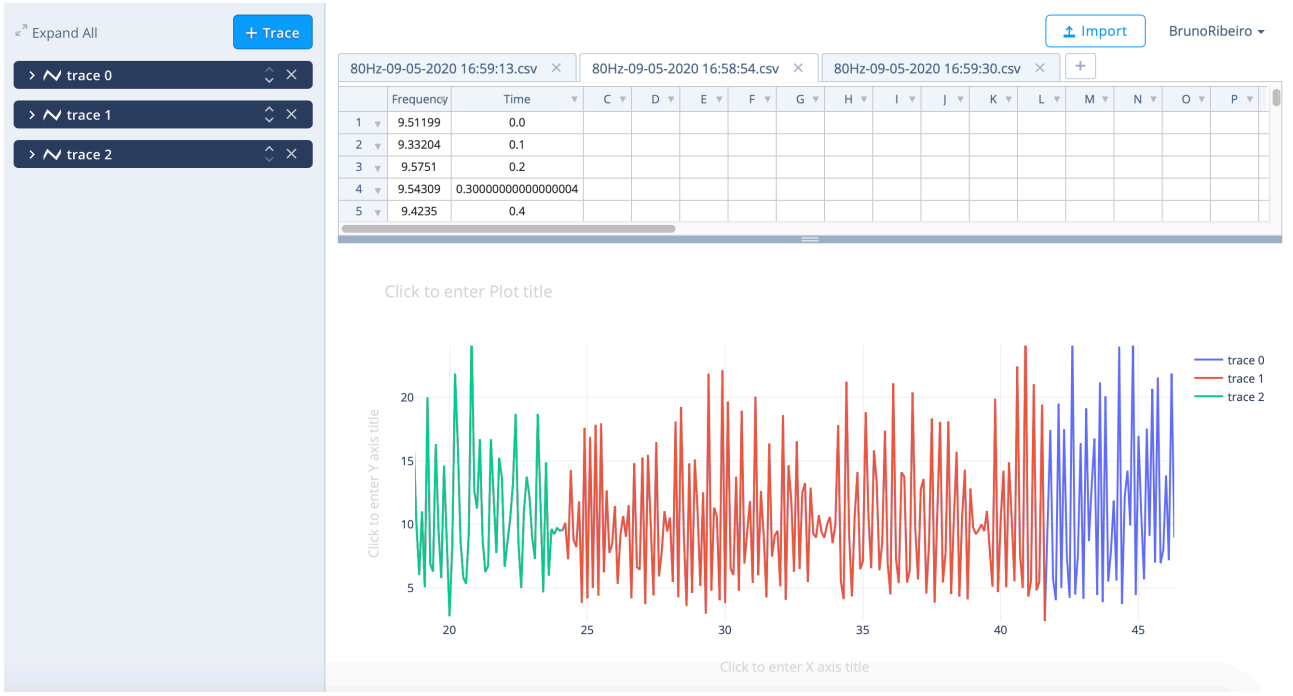
Screenshot from Android Studio: The onClick method of the Switch saves the file once it is deactivated

Three short tests were executed by a Bio Protech member. These tests were not intended to check how the vibration interfered in the wrist movement, but to check if the newest feature could be used by a doctor to run as many tests as needed. While the graph was being plotted and the team member was simulating hand-tremor, the Switch button was activated and deactivated three times. The result can be seen in the following picture.



Screenshot from the Smartphone: Three new CSV files were created inside of the BioProtech folder.

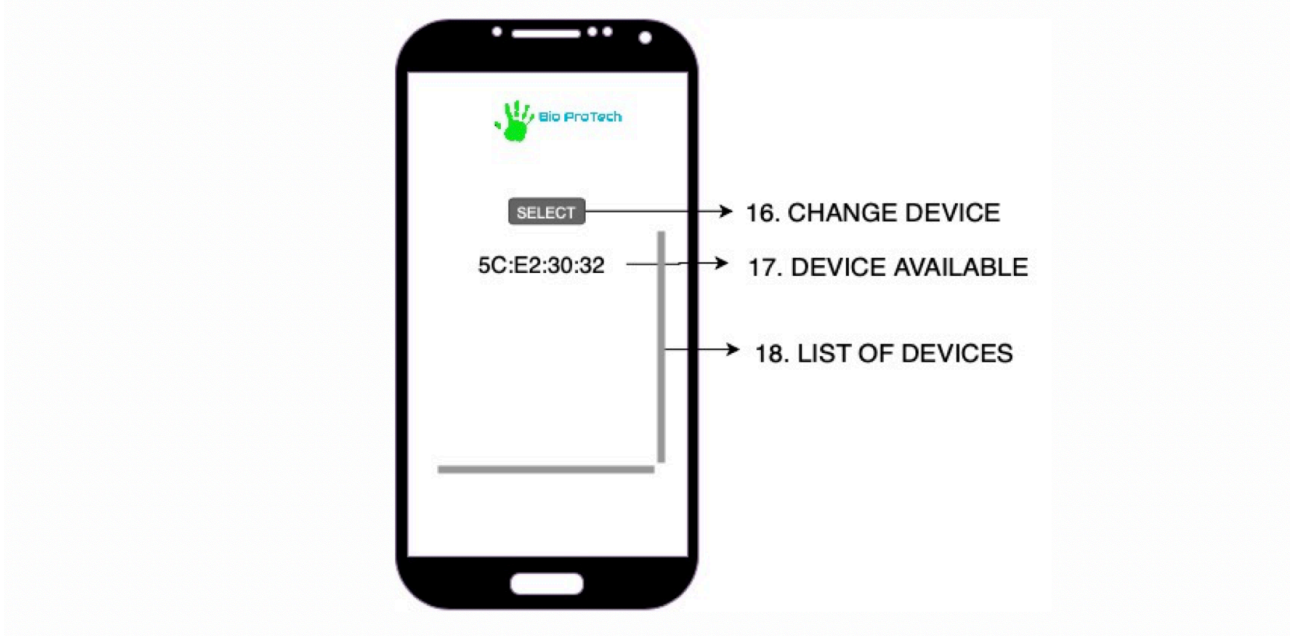
The Bio Protech's app will not be able to plot the saved data back in a graph, even though the saving feature is working as expected, an external tool needs to be used to visualise the data. It is recommended using the Online Graph Maker Plotly website. The site allows to upload a CSV file and create a graph with the data on it. It also has tutorials on how to use the several plotting tools available for free. However, uploading the files to create a graph is quick and straightforward which discard the need to go through the tutorials. It is important to mention that to retrieve the files varies from smartphone to smartphone, in the smartphone used in this case, the file could be easily accessed by connecting the phone to a desktop computer via USB and opening the BioProtech folder. It is possible to use Plotly on the smartphone, but for better visualisation, this procedure will be done over the desktop version.



Screenshot from the Plotly dashboard: The CSV files were imported to the website and the stored data was used to plot the graph. 1. The tutorials on how to use the website can be found at: https://plotly.com/chart-studio-help/make-a-line-graph/ - 2. The graph can be seen at: https://plotly.com/~BrunoRibeiro/13/

In a real-world situation, each colour would represent a different vibration frequency, allowing doctors a better visualisation of how the induced vibrations applied to the wrist affect the patient's hand-tremor. It is believed with that information would be easier to find the vibration with better results.

K. Settings Activity



In the Settings Activity, the user will be given the option to select a different Park Med device to connect. Up to this point, the MAC address of our Circuit Playground Bluefruit was hard-coded on the mobile application, therefore the user would not be able to change it in case the Park Med stops working. In the Settings Activity, the user will be able to scan for new Park Med devices and select one, once a new device is selected, its MAC address will be saved in the smartphone's internal memory. When the user tries to connect to the Park Med, this MAC address is then retrieved.

The first thing to be done in the Settings Activity is adding the Menu Button and the Menu View. This process was already described in the Graph Activity implementation. However, the button that will hide the Menu View is the Settings Button and all the other buttons will redirect the user to the respective activities.

The next component of the layout is a button, this button will trigger the method to scan for new Bluetooth devices, the button's ID was defined as scanButton and the implementation in the SettingsActivity java class was done following the same procedures described in earlier chapters. The code used to scan for devices was created following a tutorial provided by Mathew Kim at Evatotuts (2015) and the suggestion from a StackOverflow user identified as Vingtoft (2016). Mathew describes step-by-step of how to create an Android application that scans for Bluetooth devices, but the only thing that can be adapted to our application is the implementation of the BroadcastReceiver. First, we need to declare BluetoothAdapter and BroadcastReceiver to be globally accessible.

```
public class SettingsActivity extends Activity {

    private ImageView menuButton, bluetoothButton, graphButton, settingButton, homeButton, infoButton;
    private FrameLayout clickableArea;
    private LinearLayout menu;
    private Button scanButton;
    private TextView selectTF;
    //Declaring Bluetooth Adapter and BroadcastReceiver
    private BluetoothAdapter mBluetoothAdapter;
    private BroadcastReceiver mReceiver;
```

Screenshot from Android Studio: Declaring BluetoothAdapter and BroadcastReceiver.

Inside of the scanButton's onClick method, we need to set the Bluetooth adapter to start the scanning process, the method used is called startDiscovery(). Because the Bluetooth adapter is scanning, we need to instantiate the broadcast receiver. As mentioned in previous steps, the broadcast receiver handles different aspects of Bluetooth connectivity, find new devices is one of them. For each new device found, we need to create Bluetooth devices using the java class called BluetoothDevice. This class stores attributes such as device name and address. In order to test the scanner, it was added a print statement.

```
scanButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
         //Setting the bluetooth adapter to start discovery
        mBluetoothAdapter.startDiscovery();
        // Instantiating a new BroadcastReceiver
        mReceiver = new BroadcastReceiver() {
            public void onReceive(Context context, Intent intent) {
                String scanner = intent.getAction();
                // If the broadcast finds a new device
                if (BluetoothDevice.ACTION_FOUND.equals(scanner)) {
                    // create a bluetooth device object
                    BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                    //print the name and the address
                    System.out.println(device.getName() + "\n" + device.getAddress());
            }
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(mReceiver, filter);
    }
```

Screenshot from Android Studio: Implementation of the onClick method of the scanButton.

The mobile application was launched to the smartphone on debug mode and Circuit Playground Bluefruit (CPB) was powered up with the latest code. After opening the Setting Activity, the Select a device button was pressed and a few seconds later, devices' name and address were printed on Android Studio's debug console, including CPB.

```
W/libEGL: EGLNativeWindowType 0x6f2ef88010 disconnect failed
I/System.out: CIRCUITPY2a29
    C4:51:8C:F2:2A:29
I/System.out: null
    57:6F:23:B6:56:62
I/System.out: null
I/System.out: 57:6F:23:B6:56:62
I/System.out: null
    57:6F:23:B6:56:62
I/System.out: null
    57:6F:23:B6:56:62
I/System.out: null
    57:6F:23:B6:56:62
I/System.out: null
    57:6F:23:B6:56:62
I/System.out: null
```

Screenshot from Android Studio's debug console: Names and addresses of Bluetooth devices found were printed to the console. CPB can be seen at the top named as CIRCUITPY2a29.

By default, the name advertised by CPB boards is CIRCUITPY plus the last four digits of the MAC address. For our CPB, the name is CIRCUITPY2a29, which is not pleasant to display to the user. Going back to the CircuitPython, we have found out that it is possible to change the name advertised using the BLE Radio library. The new configuration was added to CPB's code.

```
17  # Defining a new bluetooth low energy service
18  ble = BLERadio()
19  # Setting the advertisement name to Park Med
20  ble.name = "Park Med"
```

Screenshot from Mu-editor: CPB receives a new advertisement name.

Now that it is possible to scan for devices and CPB has been given a better advertisement name, we need to display the device's info to the user instead of printing to the console and because we are only interested in showing Park Med devices, we can add a filter to do it. Android Studio has a component called ListView, this type of component will allow us to create a dynamic list of Bluetooth devices and display to the user. To populate the ListView, we followed a tutorial on the website tutorialspoint.com (2020). We can start by adding a ListView component to the Settings Activity layout. It was placed below the button that triggers the scanning process and the ID was defined as bluetooth_list.

```
<ListView
    android:id="@+id/bluetooth_list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/scan"/>
```

Screenshot from Android Studio: Adding the ListView to the layout.

Because the ListView holds multiple items, the layout of the items also needs to be implemented. A new XML file called bluetooth_item was created inside of the layout folder, and inside of this file, we need to add the item's layout, which in this case it is only a TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/item"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal"
    android:padding="10dip"
    android:textColor="#FDFDFD"
    android:textSize="26dp"
    android:textStyle="bold"/>
```

Screenshot from Android Studio: A TextView was added to the newly created bluetooth_item.xml file.

In the SettingsActivity java class, the ListView needs to be implemented. We can start by creating a ListView java object and an ArrayList that will hold the device's information.

```
// ListView to display the bluetooth devices
private ListView bluetoothList;
// ArrayList to store device name and mac address
final ArrayList<String> devicesInfo = new ArrayList<>();
```

Screenshot from Android Studio: In the SettingsActivity.java class, a ListView and ArrayList were added in the global scope of the class.

Now when a new Bluetooth device is found, it will be checked if the device has a name and if the name is Park Med, if those two conditions are met, the device's name and mac address are added to the ArrayList. In order to populate the ListView, we will set it with an ArrayAdapter. According to the tutorial, the ArrayAdapter is responsible for extracting data from a source and automatically insert it to the ListView.

```
String action = intent.getAction();
// if the broadcast find a new device
if (BluetoothDevice.ACTION_FOUND.equals(action)) {
    // create a new BluetoothDevice object
    BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    // Check if the device found is named Park Med
    if (device.getName() != null) {
        if (device.getName().equals("Park Med")) {
            // If so, add name and address to the devicesInfo array
            devicesInfo.add(device.getName() + "\n" + device.getAddress());
        }
    }
    // Link the bluetoothList to the ListView
    bluetoothList = (ListView) findViewById(R.id.bluetooth_list);
    // Create an ArrayAdapter and set it to the bluetoothList
    // The arguments passed are the current activty, the layout of the item and the arraylist of devices
    ArrayAdapter adapter = new ArrayAdapter<String>( context: SettingsActivity.this, R.layout.bluetooth_item, devicesInfo);
    // Set the adapter to the ListView
    bluetoothList.setAdapter(adapter);
```

Screenshot from Android Studio: Using an ArrayAdpater to populate the ListView with data extracted from an ArrayList.

After repeating the process to scan for devices, Park Med was found, validated against the conditions, added to the ListView and displayed in the screen. In order to make the selection of a new Park Med device more presentable, an invisible TextView was added on top of the ListView, the text defined was “Select one of the devices available”. This TextView will set to visible only when the scanning process starts and finds new devices.

Screenshot from Bio Protech’s app: Park Med’s name and MAC address were displayed on the screen after scanning for new Bluetooth devices.

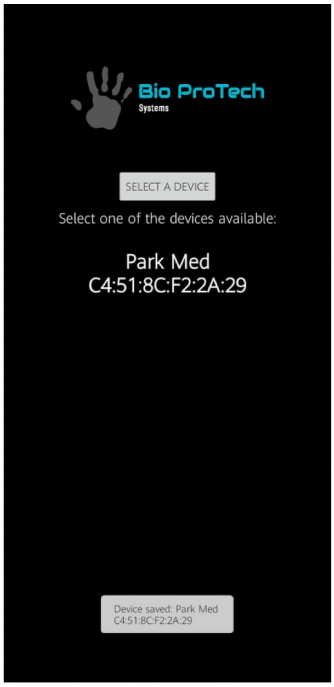
The tutorial also shows how to implement the onItemClickListener method. When the user clicks on an item, two things must happen. First, we need to display a message saying that a new Park Med was selected. After that, we need to store the device’s MAC address into the internal memory of the smartphone. According to the Android Developers website, one way to save key-value data is by using SharedPreferences (2019). In the website, there is a guide to use it to save and retrieve data; the code provided was adapted to our code. Because we want to save the MAC address, a new ArrayList called devicesMACAdress was created, and it is being populated with MAC address for each new Park Med scanned.

```
bluetoothButton.setOnClickListener((v) → {
        //Check if is disconnected
        if (mState == UART_PROFILE_DISCONNECTED) {
            // Create a new SharedPreferences to retrieve the mac address
            SharedPreferences sharedPref = getSharedPreferences( name: "DEVICE", Context.MODE_PRIVATE);
            String macAddress = sharedPref.getString( key: "Mac",  defValue: "");
            //Connect to the saved Mac Address
            mDevice = BluetoothAdapter.getDefaultAdapter().getRemoteDevice(macAddress);
            mService.connect(macAddress);
        } else {
            //If is connected
            if (mDevice != null) {
                mService.disconnect(); }}
});
```

Screenshot from Android Studio: Using SharedPreferences to save the MAC address.

```
// Set the adapter to the ListView
bluetoothList.setAdapter(adapter);
// senOnItemClickListener handles clicks for each item using their position
bluetoothList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> arg0, View v, int position, long arg3) {
        //Text with name and mac address of the device in the position
        String selectedDevice = devicesInfo.get(position);
        //MAC address of the device in the given position
        String mac = devicesMACAddress.get(position);
        //Create a private SharedPreference named DEVICE
        SharedPreferences sharedPref = getSharedPreferences( name: "DEVICE", Context.MODE_PRIVATE);
        // Uses the editor tool to edit the SharedPreference
        SharedPreferences.Editor editor = sharedPref.edit();
        editor.putString("Mac", mac); //Put a new String 'Mac' with the mac address value
        editor.commit(); //Save
        //Display a Toast message
        showMessage( msg: "Device saved: " + selectedDevice);
    }
});
```

Screenshot from Android Studio: Using SharedPreferences to retrieve the MAC address and use to establish a Bluetooth connection once the Bluetooth Button is pressed.



Screenshot from Bio Protech's app: When Park Med information was pressed, a Toast message was displayed saying that the device was saved.

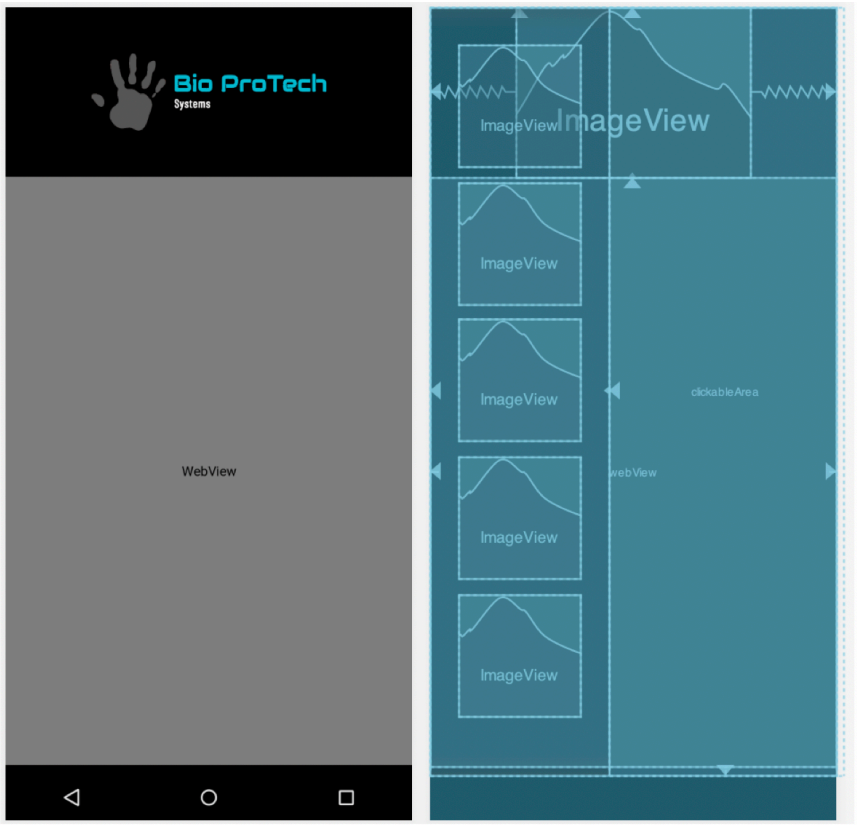
After the item was pressed, a Toast message was displayed. It showed that the onItemClickListener was working. The Bluetooth connection using the saved and retrieved MAC address was also successful. Now if it happens that Park Med stops working, the user will be able to select and connect to a new Park Med device. This feature will be handy if Bio Protech group chooses to build and distribute more devices, but for this project, only one is being created.

L. Info Activity

In the Info Activity, the user will be given a simple user manual of how to use Park Med and the Bio Protech's App, this manual will contain texts and pictures. The Menu Panel and the Menu Button were implemented following the same procedures done in earlier chapters. Even though we are going to simplify the manual as much as possible, it will be long and will contain lots of image resources. One option to create it on the Info Activity is by adding TextViews and ImageViews inside of a ScrollView. After a few minutes trying this approach, it was realised it would be extremely demanding to complete the full manual.

For the best of our knowledge, the best option to create a formatted structure intended to be displayed is by using HTML (HyperText Markup Language). While researching if Android applications can render HTML files, it was found a tutorial provided by Abhishek at AbishekStudio's website (2016). Abhishek described how to render local HTML files inside of a WebView, which is a layout component already included on Android Studio. The implementation provided is much simpler than the previous approach.

The first thing to be done is to add a WebView component in the info activity's layout, this WebView will cover the rest of the screen below the Menu button.



Screenshot from Android Studio: A WebView component was added to the Info Activity layout.

After that, we need to add a folder called assets inside of the Bio Protech app's project, inside of this folder is where the HTML file and any other resource need to be added. For now, only two pictures were included along with an HTML file called manual.html.

Inside of manual.html, we have created a short version of the manual. The content will be changed in a later stage after the Park Med is ready.

```
<!DOCTYPE html>
<html>

<head></head>
<body>
<h1 style="text-align: center;">Welcome to Bio Protech's mobile application</h1>
<h3 style="text-align: center;">How to use the app:</h3>
<p>
    First, make sure your Park Med device is on.
    The green LED will indicate if the device is powered up.
    The red LED will indicate if Park Med is prepared to connect the our app.
    <br>
</p>
<p style="text-align: center;">
    <img src="./IMG_20200509_134623.jpg" height="200px" width="200px">
</p>
<p>
    If the red LED is on, the connection to your Park Med can be established by
    clicking on the Bluetooth button inside of the Menu Panel.
    <br>
</p>
<p style="text-align: center;">
    <img src="button_bluetoothconnect.png" height="200px" width="200px">
</p>
</body>
</html>
```

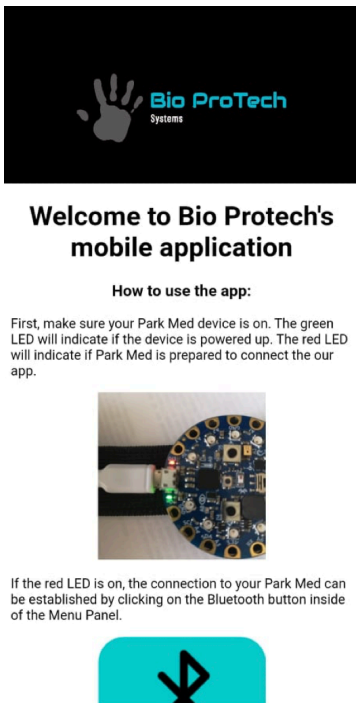
Screenshot from Android Studio: Inside of the manual.html file, a simple manual with two paragraphs and two pictures.

In order to render the HTML into the WebView, it is just needed three steps. First, we need to create a new WebView java object inside of the InfoActivity class. After that, we need to link the java object to the view. Finally, we can render the HTML file into the WebView using the loadUrl method passing the file path as the parameter.

```
private   WebView webView;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_info);

    // Linking to the WebView
    webView =(WebView)findViewById(R.id.webView);
    // Loading the file manual.html
    webView.loadUrl("file:///android_asset/manual.html");
```

Screenshot from Android Studio: Opening the manual.html using the loadUrl method.



Screenshot from Bio Protech's app: Info Activity showing the manual.html inside of a WebView.

A well explained manual will allow anyone to understand all the functionalities offered by Park Med integrated with Bio Protech's mobile application. Fully understanding how a system works is crucial for a save and enjoyable experience.

M. Troubleshooting

1. Leaking ServiceConnection

While we were transitioning between screens, it was noticed that an error referring to the Bluetooth connectivity was happening when the GraphActivty closes. This specific error was saying that the service connection had leaked.

```
E/ActivityThread: Activity com.bioprotech.GraphActivity has leaked ServiceConnection com.bioprotech.MainActivity$11@3051a1a that was originally bound here
    android.app.ServiceConnectionLeaked: Activity com.bioprotech.GraphActivity has leaked ServiceConnection com.bioprotech.MainActivity$11@3051a1a that was originally bound here
        at android.app.LoadedApk$ServiceDispatcher.<init>(LoadedApk.java:1804)
        at android.app.LoadedApk.getServiceDispatcher(LoadedApk.java:1695)
        at android.app.ContextImpl.bindServiceCommon(ContextImpl.java:1892)
        at android.app.ContextImpl.bindService(ContextImpl.java:1845)
        at android.content.ContextWrapper.bindService(ContextWrapper.java:698)
        at com.bioprotech.MainActivity.service_init(MainActivity.java:464)
        at com.bioprotech.MainActivity.onCreate(MainActivity.java:127)
        at com.bioprotech.GraphActivity.onCreate(GraphActivity.java:83)
        at android.app.Activity.performCreate(Activity.java:7458)
        at android.app.Activity.performCreate(Activity.java:7448)
        at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1286)
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3409)
        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3614)
```

Screenshot from Android Studio's console: Error message saying that the GraphActivity has leaked ServiceConnection.

The problem was happening because we were not closing the Bluetooth service used on the GraphActivity. The code below shows how we stopped it when the user goes to another page.

```
@Override
public void onDestroy() {
    mService.stopSelf();
    mService = null;
    redrawer.finish();
    super.onDestroy();
}
```

Screenshot from Android Studio: The onDestroy method is called when the current activity closes.

2. Connect Button

The Connect Button added to the InfoActivity, and GraphActivity was causing a null pointer exception and crashing the application. The reason for that is that both activities do not have access to the Bluetooth connection methods of the uart_service class, which is the class responsible for Bluetooth connectivity and communication provided by Nordic Semiconductors. We have tried to create a static method called connectBluetooth inside of the MainActivity, the idea of a static method is to allow other classes to use it without the need of instantiating a new MainActivity object. However, the implementation did not work.

The second approach tried was the same implemented on the GraphActivity, where the class was extending the MainActivity class. This approach worked as intended, but we have realised that it caused the repetition of unnecessary code throughout all the classes. Therefore, we decided to remove the option to establish Bluetooth connection from the InfoActivity and also from the SettingsActivity. It does not compromise the functionality of the mobile application in any way. It is important to mention that even though the SettingsActivity has a Bluetooth scanner, it does not provide a Bluetooth connection.

Nevertheless, the user will be given the possibility to connect to Park Med from the MainActivity (Home Page) and the GraphActivity.

IV. Park Med Design Process

After several months and going through a lot of researches, tests, and references, we are about to design Park Med. As we do not have a considerable finance resource to be invested, the material used to build the prototype had to be accessible and easy to find. The idea is to use different pieces of equipment and combine them to make a fully functional Park Med. This document has been produced to explain the step-by-step of the Park Med's design process and the required materials used.

As the device is worn around the wrist, it is similar in design to the standard wristwatches, but it is not in functionality. There is no display or any analogic interface instead, on the top will be placed the Circuit Playground Bluefruit (CPB). The five motor discs will be positioned around the wrist between gaps and connected by jumpers to the CPB board. All connections will be hidden internally with straps and externally with the wristband itself.

A. Materials

The material includes wires, CPB, vibrating motor discs, transistors, straps, glue, a recycled package lid and a flexible wristband which will give us the option to regulate according to the size of the wrist. Most materials can be easily found on the market, which makes the design creation accessible for the members of the group.

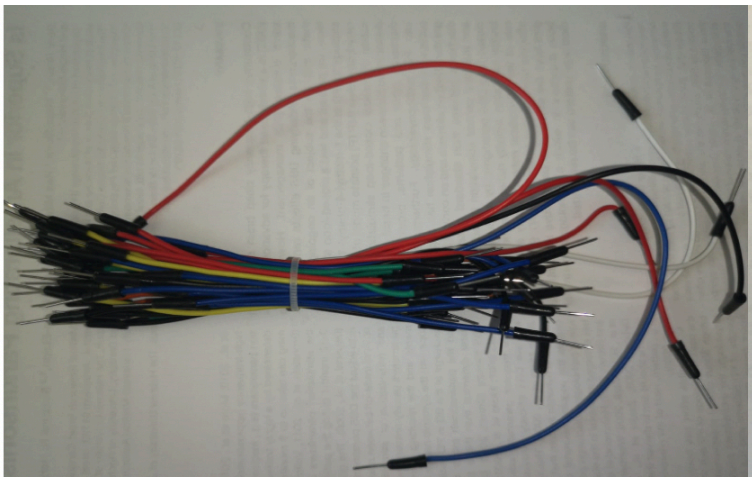
1. The wristband was bought on Amazon.

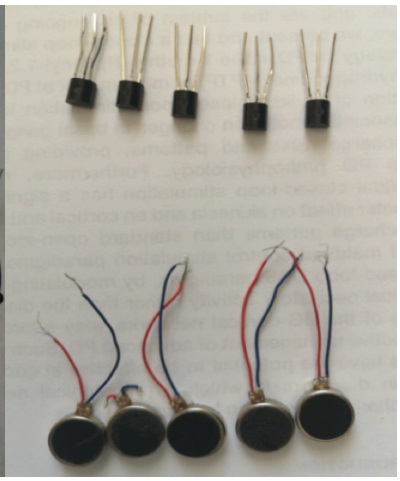
Available at: https://www.amazon.co.uk/dp/B004QC9IEM/ref=cm_sw_em_r_mt_dp_U_YpuVEbN3GRAHF

2. The glue, insulation tape, velcro straps and large coin velcro adhesives were bought at one Dealz store.

The glue will be used to stick the motors discs and cables in the internal part of the wristband, the straps will help to hide the electronics equipment around the wrist and the large coins velcro with an adhesive side will be placed on the bottom of the lid to attach it to the wristband.

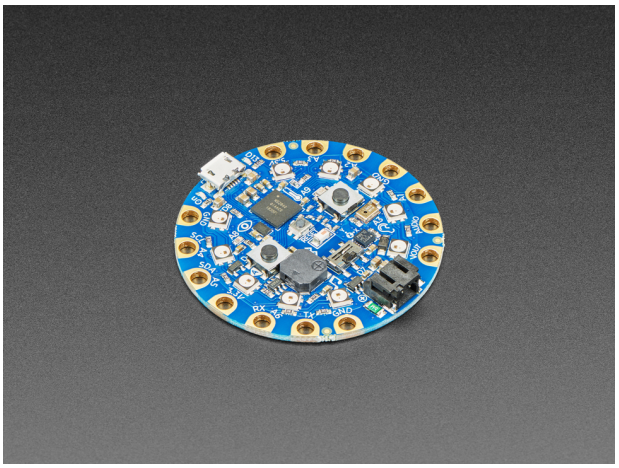
3. The wires, motors and transistors are part of a Raspberry Pi Development Kit bought on Amazon.





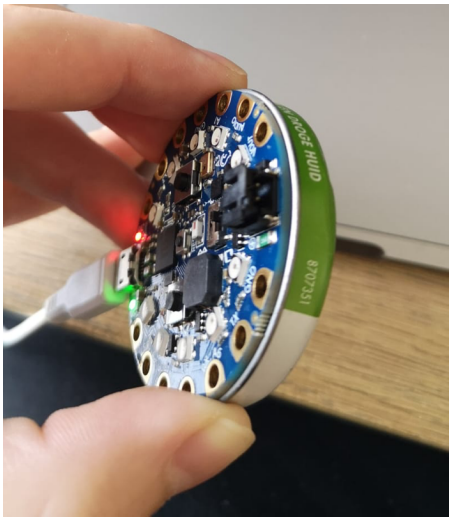
Available at: https://www.amazon.co.uk/dp/B06VTH7L28/ref=cm_sw_em_r_mt_dp_U_uOuVEbHCF9AZY

4. The CPB board was bought on Adafruit Industries's website.



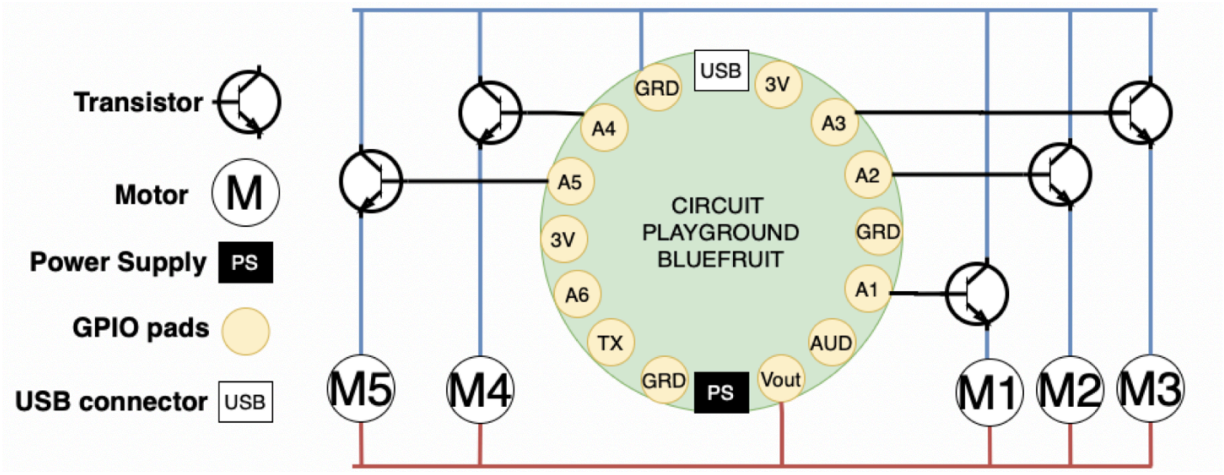
https://www.adafruit.com/product/4333

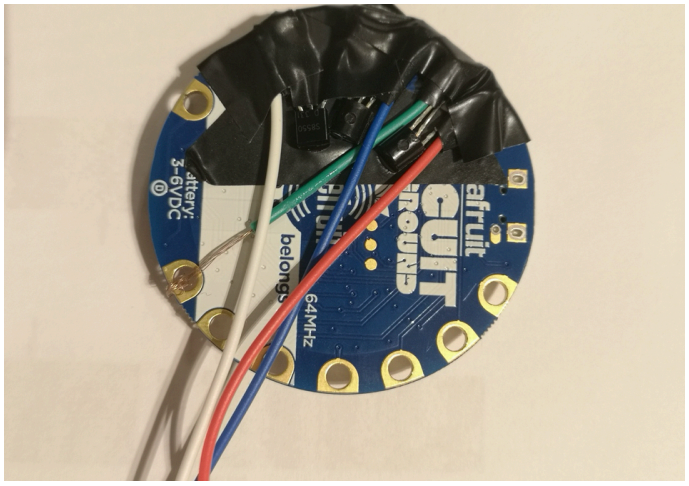
5. The two aluminium lids used to cover the board was recycled from a product package.



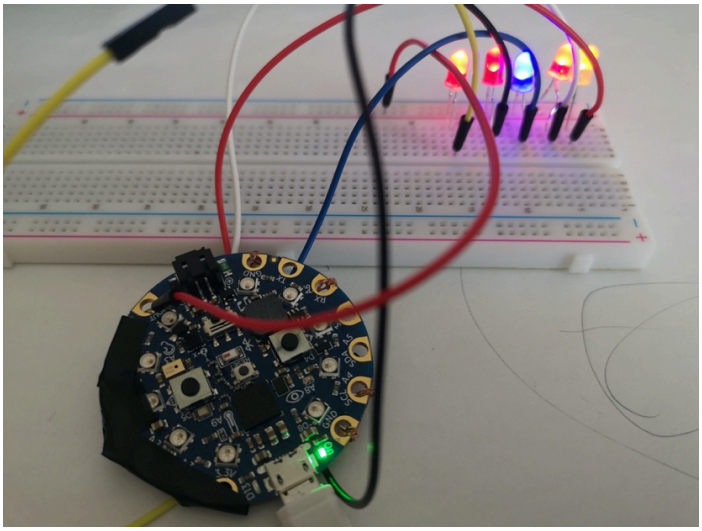
B. Building Process

We can start by connecting the transistors following the diagram below. The connections were made by wrapping the cables around the transistors' legs and the CPB's GPIOs pads. Ideally, it would be necessary to solder all the electronics components, but Bio Protech members could not have access to soldering equipment. Consequently, the connections were secured using only insulating tape.

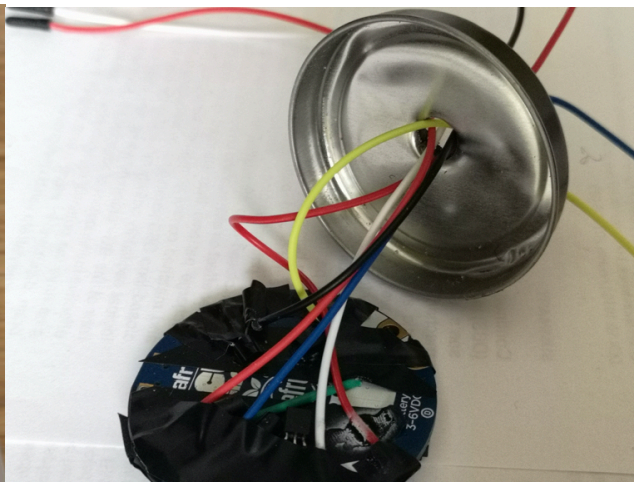




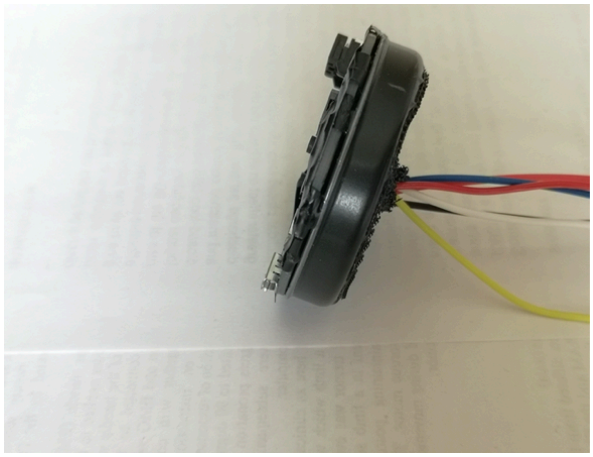
After connecting all the cables and before going any step further, we used five LEDs and one breadboard to run a test to check if all the cables were connected correctly. As the image below shows, the five LEDs turned on demonstrating that electric circuit was working as expected.

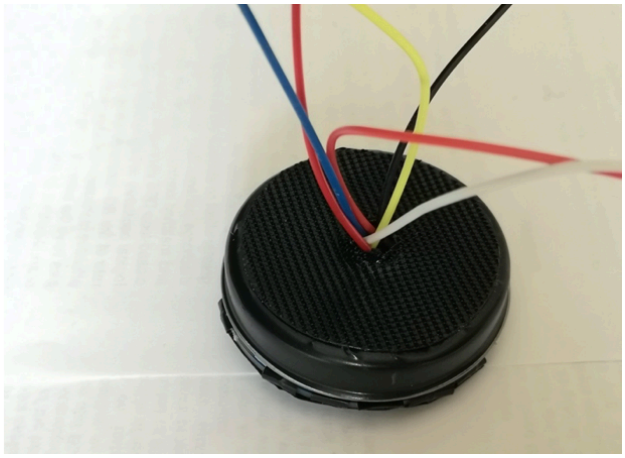


The aluminium lids were entirely covered with insulating tape. The idea behind this was not only to avoid short-circuiting the GPIO pads but also to hide the brand. In better conditions, those lids would be 3D printed with plastic material, but at the time the Park Med was built, the group could not have access to a 3D printer. The coin velcro adhesive was attached to the bottom of one of the lids and using a drill we could make a whole to put the cables through. The second lid will be used on the top of the board.

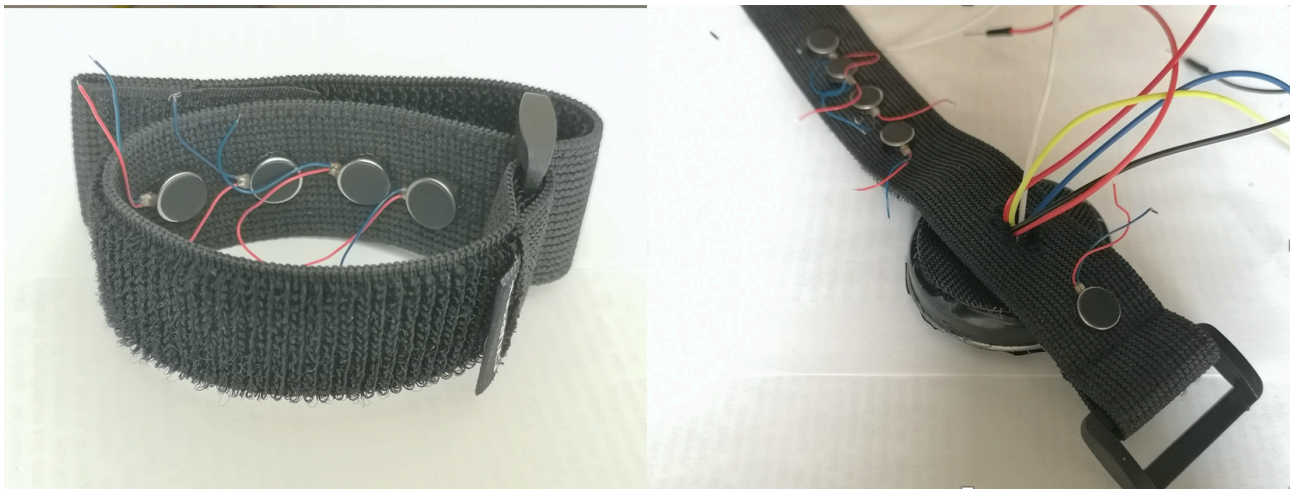


We used a small amount of glue to attach the CPB board to the lid because we did not want to put clue directed on the board. Instead, the glue was placed over the insulating tape used to secure the GPIO pads connections

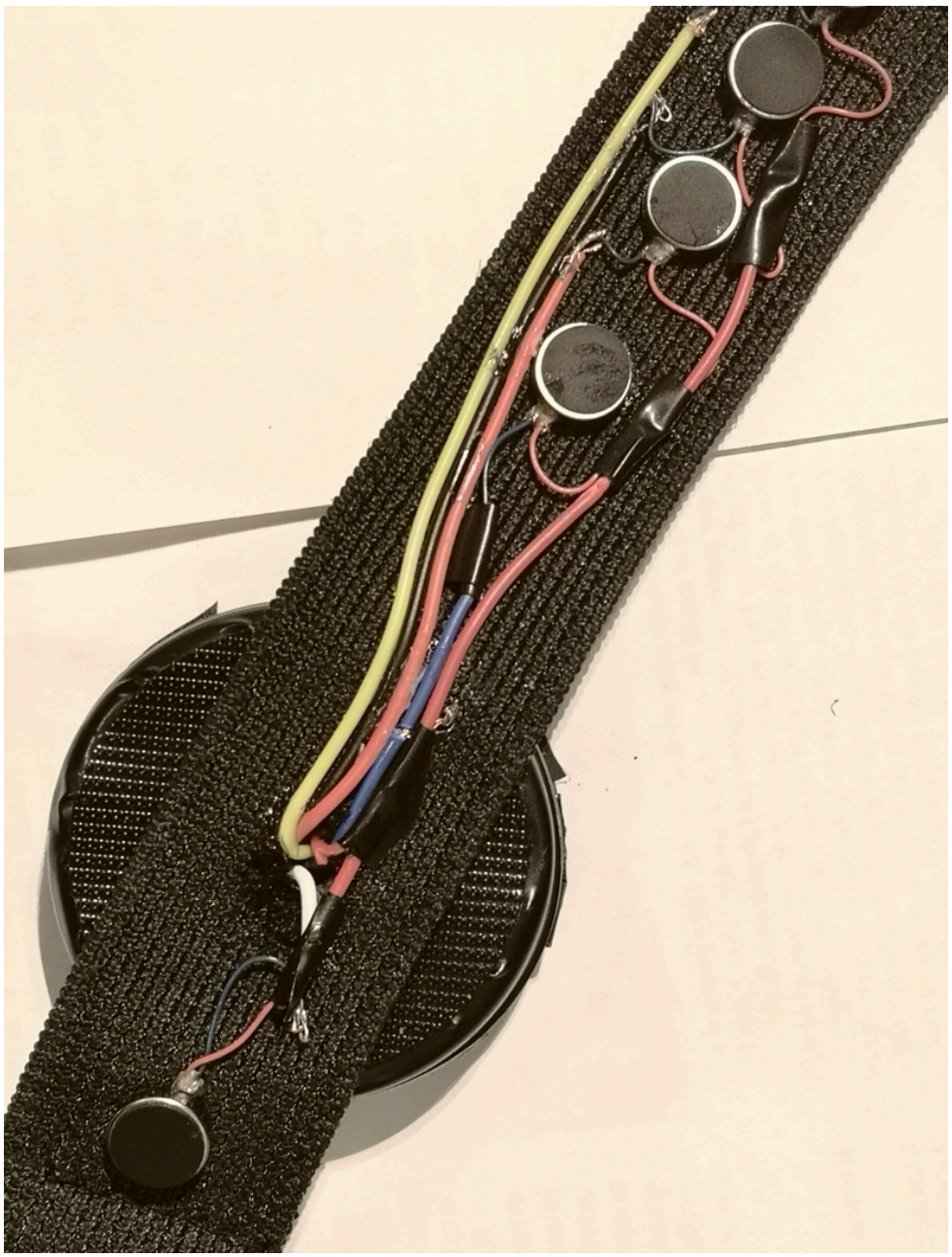




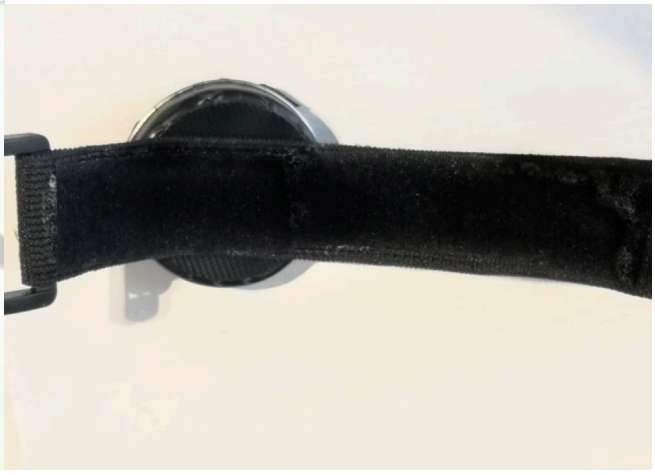
The next step is to attach the motors to the wristband with glue. The correct position was determined following the medical evidence described in the *Medical Resources* chapter. The positioning of the motors is critical to have an effective result. In the same wristband, we made a whole to pass the cables coming from the CPB. As mentioned earlier, the board will be on the top of the wrist.



The next step is to connect the cables coming from the board to the motors. The colours of the cables were crucial to connect them in the right place. The connections were secured using glue and insulating tape. Due to the small space and the extremely thin cables seen on the motors, this step was the longest and hardest to complete. However, the result was satisfactory to see.



After making all connections, it is time to cover it using the straps. This cover will not only offer a more pleasant and comfortable experience to the user while wearing the Park Med, but it also works as a protection to the device itself. The glue was placed on the edges of the strap and the wristband.



After making the wristband safe and comfortable, we have concluded the development of Bio Protech's Park Med device and Bio Protech's mobile application.

C. Bio Protech presents Park Med

6. Testing and Evaluation

I. Final Testing

The first test that has been done with the Park Med and the Bio Protech App. It was executed by one of the team members who **do not have Parkinson's disease**. The aim of this test is limited to demonstrate that both the wearable device Park Med and the Bio Protech application work according to the expectations and will be recorded on a video format. Please note, the presented images is to express and demonstrate what is written in this section, but for audio and motion, the video can be found at https://1drv.ms/v/s!AreC_jhDXjcq73vBw8BJVGCumOBC.

A. Bluetooth Connectivity Test

Once the Bio Protech app is running in the smartphone and the person is wearing the Park Med was needed to establish the Bluetooth connection manually. The connection was established successfully, a LED in the Park Med is the first proof of it as we could see in images 1.0 and 1.1

Images 1.0 and 1.1 proving Bluetooth connectivity

B. Vibrations Frequency Test

After establishing the Bluetooth connection, it was the time to test the frequency of the vibrations that the user can input through the mobile app and if they are received by the Park Med to activate them. With a range from 20hz to 180hz and control to modulate that changes on frequency. Our team member was able to test this process successfully (See image 2.0 and 2.1), one of the most significant proves is the sound that the Park-Med emits depending on the strength of the vibrations.

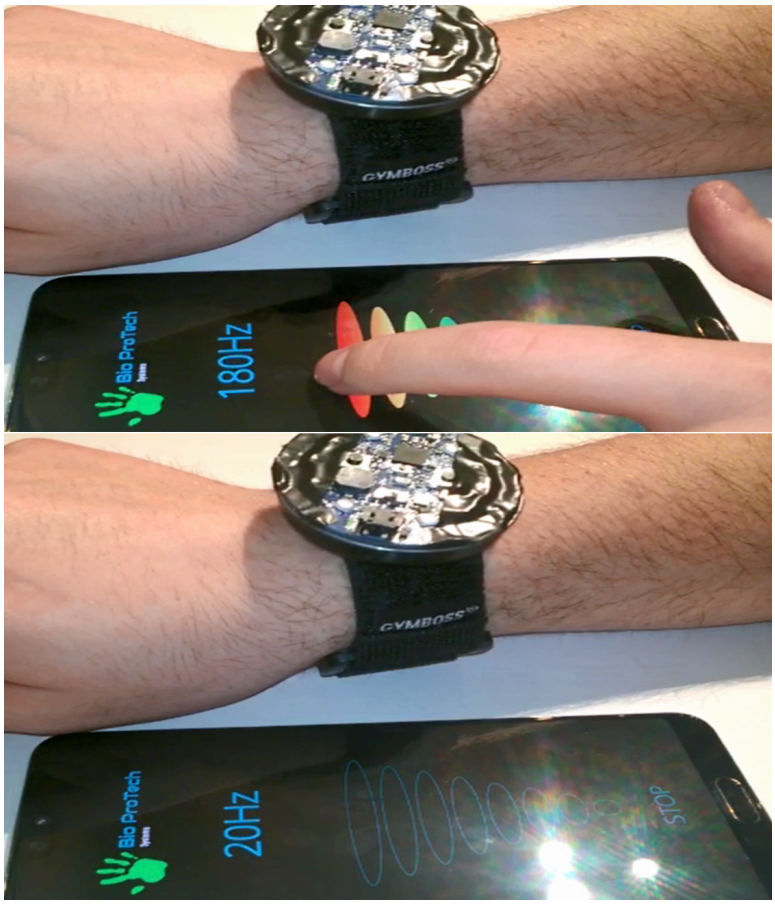


Image 2.0 and 2.1 proving control of vibrations

C. The Measure of the Tremors from the User

The next step was the tremors measuring testing, according to the System Analysis and Design chapter, the Park Med can monitor the tremors that the user experiences, which are expected to be reduced while wearing the Park Med. The accelerometer data is sent through the Bluetooth connection to the Bio Protech App and displayed on the screen in a graph format. Our team member was able to recreate the mentioned functions by shaking his hand and those shakes where expressed on the accelerometer graph built in the Bio Protech App. Please follow image 3.0.

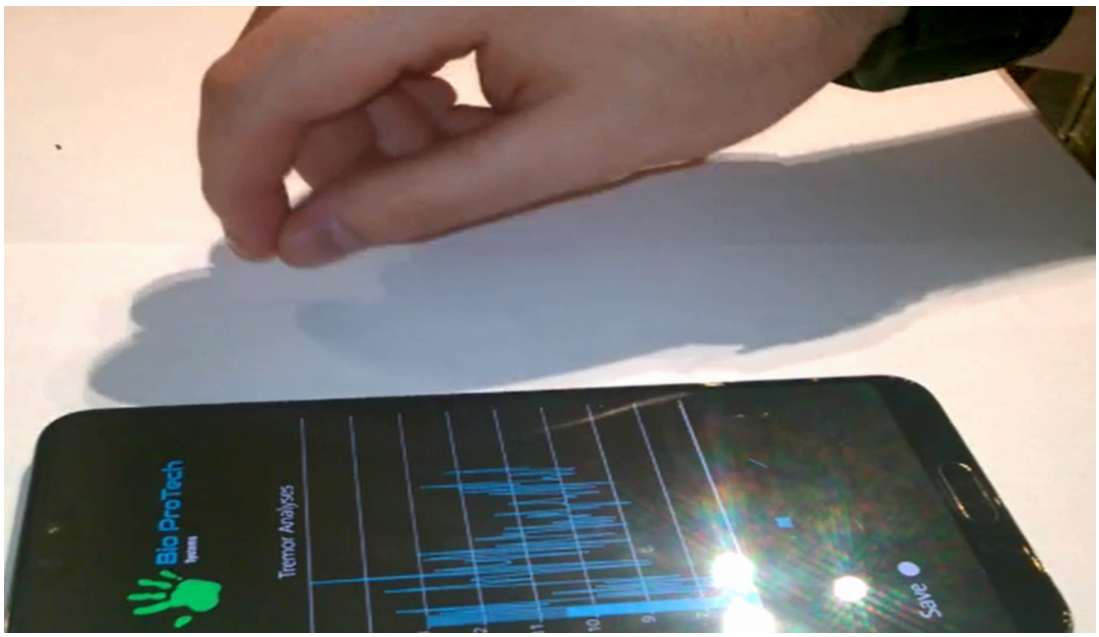


Image 3.0 displaying tremors activity on Accelerometer Graph

II. Human Testing Implications

When testing a medical device or any kind of medical treatment on humans, there are crucial factors that come in to play in order to protect the person who is submitted to testing. Such things as laws, procedures, facilities and appropriate environment, trained staff or medical expertise among many others are required in order to guarantee the health, safety, and well-being of the individual subjected to the possible test.

Bio Protech team is working on developing a wearable wristband called Park Med which will produce vibration controlled by the app called Bio Protech App. This project aims to counter-react the tremors manifested in the hands of Parkinson's disease patients by wearing the Park Med. This is a matter of medical concerns that might be and are out of the scope from Bio Protech team members and adding what is stated in the previous paragraph, makes no viable the possibility to test the device in a person who is diagnosed with the disease.

Also, the Covid-19 outbreak makes it impossible to access to medical opinion due to the current situation. The group hopes in a not far away future we could bring the Park Med to an institution an evaluate the possibility to test it according to the protocols established on a real Parkinson's disease case.

Nevertheless, it is important to state that the development of the Park Med and Bio Protech App is strongly supported by proved medical research and information gathered and exposed to this report.

Besides, what it is inside of the scope of Bio Protech group is the information technology knowledge that has been applied on the design, creation, programming and development of the device and app in order to communicate to each other and be able to work as expected firmly based on proved medical research.

7. Conclusion

This project was intended to develop a smart medical device to benefit from the concept of involuntary mechanical vibration absorbers used to reduce the undesired hand-tremor and research/ analyse their effect in suppressing the random shake. A vast study has been done to confirm the performance of the vibration absorber in reducing this specific symptom of Parkinson's Disease (PD).

Based on earlier similar projects derived from wrist medical devices, it brought a firm decision that the Park Med is a way to provide regulation of random shaking in Parkinson's Disease. Park Med is a suitable vibration absorber designed and configured to performs reduction in the tremor's motion at the wrist joint. As the group involved in this project was not entitled to perform tests on patients, it cannot be confirmed practical results that the Park Med medical device reduces the tremors in the human hand.

However, based on all the scientific experiments that have been done during this project, Bio Protech strongly believes it can reach similar results to the medical experiments. Those medical experiments have supported all the functionalities developed for the project, including the various vibration frequency generated by the Park Med. Besides, since the Bio Protech's wearable device can be fully controlled through the mobile application, it could be used by a doctor working independently from the group to test its efficiency.

I. Learning Outcomes

One of the significant points of choosing this medical device as the final year project was because it would bring a vast technical knowledge to all group members involved. As expected, the technologies required for the successful development of a smart medical device was beyond the ones acquired throughout the academic qualification from Bachelor of Science in Information Technology.

The group had the confidence to learn a new programming language in order to program the Park Med device. Two members of the group completed a 35 hours course on Udemy called 'Complete Python Bootcamp: Go from zero to hero in Python 3'. The knowledge acquired from this course was vital for the implementation of the project. A third member was responsible for researching different wireless connectivity technologies, the information gathered on Bluetooth connectivity, and its different protocols allowed the Park Med to communicate with the Bio Protech's mobile application. Another member had to learn different aspects of Android development, including design guidelines and data transmission.

As the project was closely connected to the medical area, the group had to find the scientific medical proof that the project is not only a wristband that vibrates. The only way was to deep dive in medical research, scientific journals and academic articles. The vast research in a non-related field to information technology allowed us to improve our research skills.

As a result of a pandemic outbreak, the group had to change the planning schedule for the final phase of the development process. It demonstrated that the group is able to readapt to unexpected events. Besides, as the group could not meet in person, the last two months of the implementation was done via online means showing that all group members successfully implemented the project remotely.

The group is satisfied with the skills and knowledge gained throughout the project. By the end of the project, the group had a better understanding of technologies such as:

- Python Programming Language
- Raspberry Pi
- Internet of Things
- Android Development
- Wireless Connectivity
- Hardware and Electronics Components
- Research Methodologies

II. Future Considerations

1. Due to the fact we could not test the Park Med in a person, in the future, the group hopes to seek the assistance of a doctor to undergo a series of tests in different patients.

2. The development of a second device with a more powerful microcontroller board would allow us to implement more features such as a fall detection system that would emit an alert to someone who could give immediate assistance to the user.

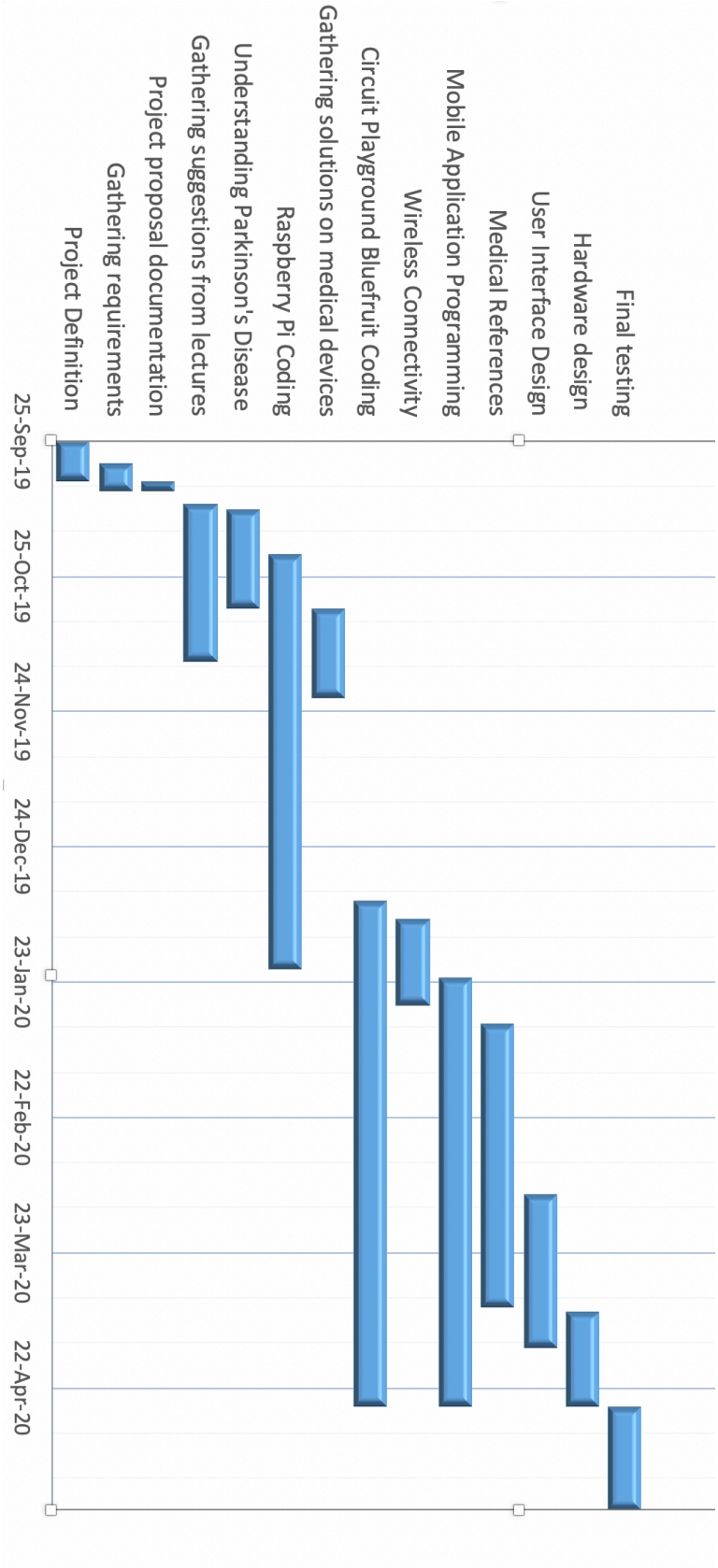
3. A more pleasant Park Med design would also be implemented in a second device, using better materials and colours.

4. As most Parkinson's disease patients take medication to reduce the symptoms, the Bio Protech's App could be used as a reminder of the correct hour to take each medicine by emitting an alarm sound.

5. As the frequency of the vibration that Park Med needs to generate is based on the hand-tremor intensity, we could use the accelerometer data and machine learning to adjust the frequency for each individual user.

8. Appendix

I. GANTT Chart



II. Project Planning

Task	Start date	End date	Duration
Project Definition	25-Sep-19	04-Oct-19	9
Gathering requirements	30-Sep-19	06-Oct-19	6
Project proposal documentation	04-Oct-19	06-Oct-19	2
Gathering suggestions from lectures	09-Oct-19	13-Nov-19	35
Understanding Parkinson's Disease	10-Oct-19	01-Nov-19	22
Raspberry Pi Coding	20-Oct-19	20-Jan-20	92
Gathering solutions on medical devices	01-Nov-19	21-Nov-19	20
Circuit Playground Bluefruit Coding	05-Jan-20	26-Apr-20	112
Wireless Connectivity	09-Jan-20	28-Jan-20	19
Mobile Application Programming	22-Jan-20	26-Apr-20	95
Medical References	01-Feb-20	04-Apr-20	63
User Interface Design	10-Mar-20	13-Apr-20	34
Hardware design	05-Apr-20	26-Apr-20	21
Final testing	26-Apr-20	19-May-20	23

III. Individual Contributions

A. Rodrigo Aguiar

This report is created to describe some activities that were carried by me Rodrigo Aguiar in this project. It has the simple purpose of showing my contribution to this group which I have had a pleasure to work with. The group is formed by Bruno Ribeiro, Leopoldo Medeiros, Juan Carlos, and me.

The idea of this project came from our classmate Leopoldo Medeiros. The project was inspired by “Emma Watch” developed by Microsoft which its functionality is to decrease the shakiness of the person who is suffering from Parkinson's disease. Once we decided to choose this subject related to Parkinson’s disease we started doing some research and tried to understand a little bit more about this disease. At the beginning we realised that were some topics to be learned and we should have to go for it in order to implement the functionalities in our device.

After planning we decided what would be necessary to give the first steps in this project. Some tasks started to be delivered for the members of the group. As long as the development process of the project was happening more tasks started to appearing.

At this point for now I will try to show the process of some activities that I was in charged in this project, when it was made and how long it took to be achieved just to have some idea of this journey working in this group called Bio Protech.

Python Course

In October I started a python course called " Complete Python Bootcamp: Go from zero to hero in Python 3 available on Udemy, where gives you many courses for a cheap. Then a took a second course that is available on YouTube called "Python Tutorial for Beginners [Full Course] Learn Python for Web Development".

During these courses, I had to keep searching about Parkinson's disease and its effects. I also search on internet kinds of treatments available for the patients.

While I was taken the course I also tried to understand more about the project we were about to do. The "Emma Watch" developed by Microsoft does not explain too much about the device and there are no comments from users because the device is not been sold to the customers. It was designed for a specific person which is the watch's name, "Emma".

After 3 months in this course, the group asked to do a small activity which was annexed in the project where a motor coin should vibrate through some command lines using the Phyton language. This activity was delivered to Leopoldo and me. Each one had to develop this small program and create a report explaining what Phyton features we were using and how it would be beneficial to our project.

Using the knowledge of these courses help me to develop this small experience and gave me the principle of using a programming language to develop something real that could represent a relief to someone.

These courses went until December 2019 and the small experience was upload on basecamp in February with the name "Rodrigo report.docx"

Medical Device Report

Some of the first reports I did was in November 2019. Each member was required to do research and to create documentation about the chosen topic. My part was to research about portable medical devices. I did not know about this topic at all. At the end of this document I just mentioned our idea in producing a prototype that would use a Raspberry-pi which was the first idea at the beginning.

In this report, I mentioned the relation between electronics and medical devices nowadays in the market and how they are contributing to customize old, larger and uncomfortable devices in a new, light and easy to use the device.

It took around 3 weeks to get it done and was upload on basecamp with the name “Medical device updated.”.

Experience with Raspberry-Pi

My experience with Raspberry-Pi was short. I spent around 25 days using it. While I was developing the coin motor experience I wanted to put it in practice by switching on and off the led installed on the Raspberry-Pi. I asked Bruno and he borrowed me his Raspberry-Pi. I started some tests and I got some results but suddenly it stopped working. My smartphone was no longer having access to his Raspberry-Pi. I thought that was just reset the device and everything would be ok but the device did not switch on anymore. Later Bruno did some tests to check what was wrong but the did not work anymore. I could not go further after that using the Raspberry-Pi.

Scientific Medic Results

This particular role was the one that asked more from me of part of my work in this group. I started in January until the beginning of April. The medical field is complex and huge with many expressions and words that are complicated to understand. In the beginning, Leopoldo helped me researching some medical stuff on the internet but himself had to work in another part of the project. I had to register myself on many medical websites to have access to some medical reports. Many of them were not available for students and there was a price to pay for just some pages. Bruno helped me with some articles that I did not have access to.

My main goal was to find as much information about induction vibration and Parkinson's disease as I could. While my colleagues were developing different parts of the project my focus was to find information that could show me where the vibrations were been applied and which frequency these vibrations were using.

Here are some medical websites that I have been through: https://link.springer.com/, https://www.sciencedirect.com/, https://academic.oup.com/, https://jnnp.bmj.com/, https://www.researchgate.net/,https://science.sciencemag.org/, https://www.ncbi.nlm.nih.gov/.

After a lot of research, I found some interesting articles but the one that I found more valuable information to add to our project was blocked. I had to pay to have access to 7 pages of experiments. It was worth! With this medical article called " Vibratory proprioceptive stimulation affects Parkinsonian tremor" from Elsevier medical journal and another I created my first report called "Using induced vibration to decrease tremors in people with Parkinson disease" where I upload on basecamp as "vibration test (3).docx".

With this report, Bruno started to reproduce some vibration tests based on this scientific article in his lab environment. Keeping working through more evidence about induction vibration related to Parkinson's disease I found more experiments on the internet and created a second report. The report called "Using induced vibration to improve movements in people with tremors", where two experiments are showed in different periods and vibrations were applied to the wrist in different ranges.

Device's Design

Bruno and I could work together in this process. With his suggestions, I could buy the material needed to build the device. I went to some cheap stores in the city center in Dublin to buy straps, glue, and a specific Velcro in a circular shape. Bruno also suggests me buying a wristband from Amazon which was used to wrap the wrist.

My focus was to create a step by step way to assembly the device in order to help Bruno. As this pandemic situation is occurring we could not work together. I sent part of the material by post so he could work in his home. He was able to use part of this document that I created to connect the pieces but during the implementation process, he also had to modify and create another part of this document to perfect connect all pieces.

B. Leopoldo Medeiros

This documentation aims to provide a clear explanation regarding what I've been covered in the Bio Protech project. As a student of Computer Science, the main focus of all of the team members was trying not to be a developer but problem solvers.

The main idea for this project came from me, inspired and concerned about those people who suffer from any sort of disease. Then I faced a particular project on the Internet about "Emma Watch Project", which is a medical device developed to help Parkinioans people.
After that, for many and many days, I wrote down huge notes regarding what I could do to help people. It was a big challenge because I didn't have enough knowledge and skills to do such a big project. By concerning all these boundaries I brought all my papers and doubts to the professor Amilcar a year before the project starts. In total, this process had from 2 to 3 meetings with the professor Amilcar.
He helped me to be clear about myself and my ideas, all taught me all the steps I should take to success such a hard project.
Then when I was clear and confident about what I wanted, I talked to my team members (Bruno Ribeiro and Rodrigo Aguiar). I sent then everything I had at that moment explaining a bit about Parkinson's Disease, Biomedical Engineering and Medical Devices. They got excited about the idea and agreed to start a group for the future project.

What made me choose such field to this project was the passion for the mix between Health and Technology. As I have a background in Biomedical Technician it was the start point to think about how we could use our knowledge to help people in the real-life, in special those people who suffer from a disease.
By thinking about it we got inspired by the "Emma Watch project" developed by Haiyan Zhang, Innovation Director at Microsoft Research in Cambridge.
Emma Watch as the name says itself is a watch developed exclusively to stabilize involuntary handshaking of people who suffers Parkinson's Disease.
Once the agreement of all of the team member of the project by choosing this field for the project, the group had a lot of meetings to have a brainstorm. The group broke down in different pieces about what a project like this would cover to develop it.This demanded new knowledge, and a deep research from all of team members to start an accurate plan before it's started, such as Python Programming Language, Biology, Neurology and so on.

I have been covering the topics listed below, that will be broke down:

9. Homestasis
10. Python
11. Motor Control Vibration
12. Simulation Motor Vibration in Python
13. RPi Experiment
14. Graph Accelerometer

1. Homeostasis

November 3th, 2019 - In this stage, I did research a topic which is mentioned in the Emma Watch Project, something that I should know related to PD (Parkinson's Disease), it was when I faced at Google Scholar about Homeostasis.
Based on huge research that I have done, I could understand that Homeostasis refers to the balance with a system that keeps it operating with a range of conditions.
In other words, Homeostasis is an organism's process of maintaining an internal environment. In the human body Homeostasis cover:

- Chemical levels
- Body Temperature
- Ratios of water and minerals

In the Academic Research, I added a study about the importance and the relation between our project and Homeostasis. This study shows how Homeostasis is relevant to know before we take any further step in the project.

2. Python Course

September 24th, 2019 - One of the most tools we would need to know before taking any further steps was to learn Python Programming Language. The reason for it is that all of the application we decided to use even in Raspberry Pi or Circuit Playground Bluefruit both are based on Python.

As we don't learn this particular Programming Language at College, I had to study a full Python course available at Udemy website called "Complete Python Bootcamp: Go from zero to hero in Python 3" which is a quite long tutorial that took 2 months to learn all content of this.

Also, in order to practise more Python skills with practical examples exercises, I study the tutorial available on Youtube called "Python Tutorial for Beginners [Full Course] - Python for Web Development" in the channel Programming with Mosh. This was a shorter tutorial with a duration of 6 hours.

Once I learned Python I could apply the tools required to develop many parts of the project and the experiments I have done during all the process of creation of our medical device.

3. Motor control Vibrations

March 8th, 2020 - The research about this topic covers a study about motor control which means the features of the movement of the human body and how the motor system to control it.
It brings us the path to understand more effectively how movements happen and plan how we could achieve the aim of the project which is to develop a device to control those involuntary movements from Parkiniosian people.

4. Simulation Motor Vibration in Python

February 6th, 2020 - This part of the project I have done a small experiment by implementing a simple code in Python to see how a tiny coin motor would act in a future application or development tool.
I have implemented a function called menu with some options on it, and the second option of the menu there is command that enable the program to "Vibrates", which would be the coin motor.
It runs in an interval of 1 second (function time.sleep( )). At this stage I was struggle on how to stop the while loop by tipping any key on the keyboard.
This experiment has been done only by using an IDE with no hardware added on that matter. It was only an overview to understand the principle that the future application would demand to.

5. Raspberry Pi Experiment

April 14th, 2020 - Differently of the previous experiment, now I applied my knowledges by using hardware and software.
As in that stage I didn't have a tiny coin motor available, then I have done a simple experiment by applying a function in Python that runs a while loop in different intervals of time on LED's connected on Raspberry Pi.
I could brush my knowledge about Programming, Electronics and Linux operating System, which would be needed in the next steps in the projects.

6. Graph Accelerometer

May 6th, 2020 - A long the process of development our project we have decided to swap that was being used. So we migrated from Raspberry Pi to Circuit Playground Bluefruit board. The reason of this change was basically the more compact size of this hardware, as we were trying to develop a such watch, the CPB(Circuit Playground Bulefruit) fits perfectly to our needs.

As part of the application that our Team Manager Bruno Ribeiro has created, this would need a Graph Accelerometer to show the frequency acting in the Park Med App. I helped Bruno Ribeiro to implement this Graph.

The Graph accelerometer will be used to collect data, it means that this collects the frequency of the user's hand shaking. In other words, once a user shaking his/her hand involuntary, the frequency in the application oscillates.
The implementation of the Graph accelerometer has been done by using the Mu Editor which is a Python code editor, then I connected the CPB board on my computer and implemented the code.

C. Juan Carlos

At the beginning of the project, Bio Protech team decided the create a medical device, a wearable wristband Called “Park Med” aiming to help to control tremors in hands for people that suffer Parkinson’s disease. This wristband will be controlled from a smart device application called “Bio Protech App” through Bluetooth Low-Energy (BLE). But a bridge was needed, a small low-cost computer and the answer at that moment was the implementation of the Raspberry PI, a credit-card-size computer which will work as a bridge connection from the Bio Protech App to the Park-Med and provide it with the functionality needed.

The Raspberry PI (PI) works a Unix-Based computer and Python programming language. Been that said the first challenge was to be able to establish the Bluetooth connectivity between a smartphone (host of the app) and the PI. That was the first main task assigned to me.
The objectives of this task were the following:
-Be able to pair the PI with a Smartphone
-After pairing, establish real Bluetooth connectivity between the PI and the Smartphone.
-Once the connectivity was successfully achieved be able to transfer some data, represented as strings using an Application called “*Serial Bluetooth Terminal”* which is mainly used for Bluetooth connectivity test.

The first challenge found it was the setting up the PI so it could be remotely accessed from a Windows Operative System (OS) environment. For basic control of the PI a television monitor, keyboard and mouse are only needed, but to have more control of it a remote access control is suggested.

Within the PI with Linux open-source Operative System, from the command line perspective, an *Upgrade and Update* was needed to get the most recent updates developed for the OS.
The PI is not set up for Bluetooth connectivity so that was the next challenge. The installation of packages like *bluetoothctl, Bluetooth, bluez and blueman* were needed.
This information was mainly got It from the www.cnet.com website, where an easy readable but full and concrete tutorial is given.

The pair was easy to achieve, the real connection with the smartphone was more challenging in the sense of input the commands in the right order. Finally, in collaboration with the Project Manager Bruno Pereira we were able to transfer the data to the smartphone and displayed it on the Serial Bluetooth Terminal app.
Please find further details on section "Connectivity between RPI and a smart device Test 1 (Via Bluetooth)"

Documentation Chapter 1, 5th semester

Another task that was assigned to me to work on a par with the previous section was to make a report gathering the information collected by that time and create the Documentation Chapter 1 due to December of 2019.

Unfortunately, I found this very confusing to do it, first of all, I have to say that I clearly failed and that affect my colleges as well, also, that in my opinion, I feel there was no congruence between what Lectures were saying in class and what they mark. I felt myself very participant in that class in order to get the ideas and what was expected and sadly it was not enough. This task was after reassigned to Leopoldo Medeiros so it could be recreated and added to new chapters.

Circuit Playground Bluefruit and Coin Motors

At the beginning of the second phase of this project our PM Bruno Pereira found what will be turned in a new path to follow, an even smaller low-cost computer which will simplify users experience, The Circuit Playground Bluefruit (CPB).
This rounded 2-inch size computer has the perfect design for Bio Protech project purposes.
My new task was to research about it, that research should have included what is it, why is useful. The difficult part was to understand what I was looking at and translate that information into my words so and I could capture the ideas in my report. That report also includes information:
-Coin size motors that will be used to recreate vibrations (Previously supported by another report created by Leopoldo Medeiros)
-And the PN2222 transistor used to be able to control the input and output power and therefore control the intensity of the vibrations.
Please find more details in section "Circuit Playground Bluefruit, Coin Vibration motors and PN2222 Transistor"

System Analysis and Design (SaaD)

This new assignment consists in to explain how the whole system (which includes software, hardware and actor) should work and what should be expected to do according to the requirements capture in a report format.
The first subsection you will see in SaaD report is the description of the project as a system, a high-level general idea of what is the system about as an introduction to it.

The next subsection is System Requirements. On this section, I went more into details, still high level, but the introduction of technologies used and technicians start coming to play. Also, in this section, the system is broken down into smaller components of it and it is explained what the purpose is of each one of these components as similar to a guideline.
Later in this report, is shown 2 Use Case Scenarios, which both express the tow main functionalities of the system as their main 2 reasons to exist. These scenarios are presented in both text and Use Case Diagram which represents and explain situations where the system is been used by the potential user.

At the end of this report at a bit lower level, we can find the Class Diagram to express what it is happening internally or in the background while the system is working. How the classes will interact with each other at the moment of the system been used by the user.

Final Video Edition

Alongside PM Bruno Pereira, we both will be working on the final edition of the video where the finished project will be exposed so everyone could see how it works. I will give my voice to guide you through the video explaining what is happening as he recorded the actual video.

D. Bruno Ribeiro

This is the report of how I have contributed to the development of the project. Because this is the final report I am writing to the project, it brings me great joy to go back in time and analyse what my group and I went through in this long journey.

In **September 2019**, my teammate Leopoldo showed me the short BBC video showing the Emma Lewton story and how the watch developed for her changed her life. This was the moment I knew the group should do something that can have a real impact on someone's life. The decision to develop Park Med came after we had brainstorming meetings and discussed the requirements, the time-limitation and also how much this project would demand from us because it is closely connected to the medical field in which we do not have the experience nor enough knowledge.
In those meetings, it was discussed that Park Med should be controlled through a graphic user interface and because I had some previous knowledge in Android development, I suggested a mobile app and mentioned that I could be responsible for this part.

In **October 2019**, when the individual tasks started being given and we had some idea of who would take each implementation part. It was decided I would **develop the mobile app**, which would bring me extreme pleasure because I hadn't developed any app in a long time. My knowledge in this area goes back to 2016 when I held a Google scholarship to a nanodegree in Android Development offered on the Udacity platform.

Although I was only in charge of the app, I also wanted to learn how to use a Raspberry Pi, so in **November 2019**, I decided to buy one. This integration of computing, hardware, mechanics and electronics fascinates me, back in Brazil, I went to a technical secondary school where I got a diploma in Mechatronics Technician. In my Raspberry Pi, I was developing some tests for personal interests, but what I learnt while doing them came in hand in a later stage. In the same time, I was given a task to research a topic related to Parkinson's disease called **Feedback Loop**, by the end of this research I produced a report on what Feedback loop is and how it is related to our project.

In **January 2020**, while searching on the internet for some hardware components that could be used for our project, I found the Adafruit Industries' website. In this website, I saw the **Circuit Playground Bluefruit (CPB)**, which is the microcontroller board currently used in the Park Med, the shape, size and price of the board were the first things noticed, but before making any compulsory decision, I decided to double-check the website. While doing it, I saw all the tutorials, guides and the community channel offered by them. This was a big green light telling me to order one board.

In **February 2020**, Juan Carlos and I worked together to test the best approach to connect Park Med and the smartphone. We managed to connect the Raspberry Pi with his phone via Bluetooth and run a python program to make a simple sum with an input sent from the phone. This was an indication of how I could develop the mobile app to control the Park Med. In the same week, the board ordered on Andrafruit's website was delivered to my home. Because my teammates Leopoldo and Rodrigo were executing a test with the Raspberry Pi and motors, I tried to **reproduce the same test using the newest board**.

It was effortless to do, so I decided to replicate the same connectivity test done with Juan, it was also possible to establish a Bluetooth connection to an app. Moreover, instead of making a simple sum, I decided to put together the motor system with the connection system, the result was that I could power on and off the motors through a mobile app.

Using CPB in only one day, I could **execute an important part of the whole project**, which is controlling the device with a smartphone. So I sent to the group one report suggesting to use CPB instead of the Raspberry Pi. After a few meetings, the group agreed to make the replacement. In one of those meetings, one member of the group suggested me as the new **Project Manager**, after talking with the Leopoldo, the Project Manager at that time, we agreed this change would be beneficial to the group.

In **March 2020**, I started developing Bio Protech's mobile app. The first thing was to make the app connect and transmit data to CPB via Bluetooth. At the same time, because having only one CPB was making the schedule falls behind, and the group could not meet to work together, I started working on the Park Med's code by myself and I implemented the system to give the user **control over the vibration's frequency**.

This was the only way not to delay the implementation part of the project, even though the rest of the group wanted to help, we had only one board and I needed it to test the connection with the app. After I finished those tests and the app was able to **connect and send data to the board**, I handed it over to Leopoldo so that he could finish the last part of the code. And as a Project Manager, I was delivering tasks to the group according to what was needed at that specific time.

In **April 2020,** I implemented the app's design. It took a long time because we did not have a mock-up ready to be followed. Using the Draw.io, I **drew a few mock-ups** and with the help of the group we chose one. After the design was chosen, I started implementing it in the app. This was the time I completed the design of different pages of the app with the exception of the Graph Activity, which could not be tested because Leopoldo had the board.

At that time, the group did not have a draft of the complete documentation, only individual reports done by each member of the group. As the deadline was getting closer and closer, I had to **put all the individual reports together** following a consistent structure that was suggested by professor Amilcar.

In **May 2020**, after the group realised we could not meet before the submission date, I asked Rodrigo and Leopoldo to send me the board and the material by post, so I could finish the implementation and run some **final tests**. As I already had lots of contribution to the project, this was nothing close to an ideal solution but was the only one viable due to the COVID-19 outbreak and the imposed lockdown.

After I received the packages, I finished the Graph page of the app and using the code implemented by Leopoldo, it was possible to make it work. When the app was ready, I was following Rodrigo's report on how the Park Med was intended to be designed and put the parts together to **create the final product.**

In the last week before the submission and following the advice from our Project Supervisor, I thoroughly read the full documentation for a **final review**. I could see that lots of changes needed to be made in order to make it more comprehensive and avoid more significant problems, so I asked the members of the group to remake some parts. However, after reviewing the remade parts, which did not meet the expectations, I had to **apply the changes** by myself. Despite contributing to a massive part of the full implementation by myself, I am fully aware this was the only solution to deliver the project in time.

IV. GitHub Page

The project git repository can be found in the link below. There, it is possible to see the entire Android app code inside of a folder called Bio-Protech-App. The code, libraries and electric diagram used for the Park Med is a folder called Park-Med.

https://github.com/brunobpr/Bio-Protech-Project

15. List of References

1. Encyclopedia Britannica. 2018. *Homeostasis | Definition, Examples, & Facts*. [online] Available at: <https://www.britannica.com/science/homeostasis> [Accessed 14 November 2019].

2. Parkinson's Disease: New Insights for the Healthcare Professional: 2013 Edition [online] Available at: https://books.google.ie/books?id=hQdROY02laoC&pg=PA193&lpg=PA193&dq=homeostasis+and+parkinson%27s+disease&source=bl&ots=bh9F3yuBmo&sig=ACfU3U3kMjcjnEDvs4pifXWcejoySzoNTA&hl=en&sa=X&ved=2ahUKEwjj_f3rhs_lAhUWSBUIHaUDAdc4RhDoATAGegQICRAB#v=onepage&q=homeostasis%20and%20parkinson's%20disease&f= false

3. Project inspiration on “Microsoft Project Emma” [online] Available at: https:// www.youtube.com/watch ?v=DZ6E49B_Pec

4. Introduction, PD’s overview [online] Available at: https://medlineplus.gov/ency/im agepages/ 19515.htm

5. Stopping tremors with vibrations “Action tremors in PD’s” medical document by Dr. Teravainen and Dr. Calne

[online] Available at: https://www.ncbi.nlm.nih.gov/p mc/articles/PMC490518/?page= 1

6. James J. Licari. 1998. Science Direct website.[Accessed 01 November 2019].

[online] Available at:https://www.sciencedirect.com/

7. Magazine Pro Theme on Genesis Framework, 2019. EdgeFX.in website.

[Online]. [Accessed 01 November 2019]. Available at: https://www.edgefx.in/

8. Dr.Yiming Wang. Qsota medical website.

[Online]. [Accessed 03 november 2019]. Available at: https://qsota.com/

9. Scranton Gillette Communications. 2018. DAIC website.

[Online]. [Accessed 04 November 2019]. Available at: https://www.dicardiology.com/

10. Valentina Palladino. 2017.arsTechnica website.

[Online]. [Accessed 03 November 2019]. Available at: https://arstechnica.com/

11. Jeb Su. 2018.Forbes website.

[Online]. [Accessed 09 November 2019]. Available at: https://www.forbes.com/

12. Vanessa Hand Orellana. 2019. Cnet website.

[Online]. [Accessed 10 November 2019]. Available at: https://www.cnet.com/

13. Bayer. 2019. website. [Online]. [Accessed 10 November 2019]. Available at: https://www.thrombosisadviser.com/

14. FDA. 2019. [Online]. [Accessed 30 November 2019]. Available at: https://www.fda.gov/

15. Monir El Azzouzi. 2019. [Online]. [Accessed 30 November 2019]. Available at: https://easymedicaldevice.com/

16. ThermPro. 2019. [Online]. [Accessed 29 November 2019]. Available at: https://buythermopro.com/

17. Evarts, E. V., Teräväinrn, H. T. & Beuchert, D. E., 1979. Pathophysiology of motor performance in Parkinson's disease. In Dopaminergic ergot derivatives and motor function. Pergamon, pp. 45-59.

18. Fatsy, L. M., 2016. Positive and Negative Feedback Loops in Biology, s.l.: Fairchild Wheeler Magnet H.S. Instructor.

19. Teräväinen , H. & Calne, D. B., 1980. Action tremor in Parkinson's disease. Journal of Neurology, Neurosurgery and Psychiatry, pp. 257-263.

20. Zuckerman, B., 2016. Human Population and the Environmental Crisis, s.l.: University of California.

21. The GyroGlove. [online]. [Accessed 16 March 2020]. Available at: https://gyrogear.co/

22. Parkinson Smartwatch. [online]. [Accessed 13 March 2020]. Available at: https://parkinsonsmartwatch.com/en/

23. iStopshaking. [online]. [Accessed 09 March 2020]. Available at: https://www.youtube.com/watch?v=kgRgcmiS-vI

24. European Journal of Neuroscience. (2012). Temporal features of human tendon vibration illusions. [online] Wiley Online Library. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1111/ejn.12004

[Accessed 03 Apr. 2020].

25. The Free Dictionary by Farlex. (2020). Maximum Voluntary Contraction. [online]. Available at: https://medical-dictionary.thefreedictionary.com/maximum+voluntary+contraction

[Accessed 04 Apr. 2020].

26. Rheumatology. (1953). A Strain-gauge Dynamometer for Measuring the Strength of Muscle Contraction and for Re-Educating Muscles [online]. Oxford Academic. Available at: https://

academic.oup.com/rheumatology/article-abstract/1/5/163/1780320?redirectedFrom=PDF [Accessed 01 Apr. 2020].

27. Scandinavian Journal of Work, Environment & Health. (1993). Contribution of the tonic vibration reflex to muscle stress and muscle fatigue [online]. JSTOR. Available at: https://www.jstor.org/stable/40966107?seq=5#metadata_info_tab_contents [Accessed 02 Apr. 2020].

28. Health Union LLC. (2020). What is Carbidopa/Levodopa Therapy? [online]. Parkinsons Disease.net. Available at: https://parkinsonsdisease.net/medications/carbidopa-levodopa-therapy/ [Accessed 02 Apr. 2020].

29. Kavita R. Gandhi; Abdolreza Saadabadi. (2019). What is Carbidopa/Levodopa Therapy? [online]. National Center of Biotechnology Information. Available at: https://www.ncbi.nlm.nih.gov/books/NBK482140/ [Accessed 01 Apr. 2020].

30. Tatjana Seizova-Cajic ,Janette L. Smith,Janet L. Taylor,Simon C. Gandevia. (2007). Proprioceptive Movement Illusions Due to Prolonged Stimulation: Reversals and Aftereffects [online] Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0001037 [Accessed 31 Mar. 2020].

31. Health Union. (2020). What Are Dopamine Agonists? [online]. ParkinsonsDisiase.net. Available at: https://parkinsonsdisease.net/medications/dopamine-agonists/ [Accessed 02 Apr. 2020].

32. Sinead Greenan, Kim Jackson, Scott Buxton, Oyemi Sillo and Evan Thomas. (2020). Nine-Hole Peg Test [online]. Physiopedia. Available at: https://www.physio-pedia.com/Nine-Hole_Peg_Test [Accessed 02 Apr. 2020].

33. ScienceDirect. (2002). Vibratory proprioceptive stimulation affects Parkinsonian tremor. [online] Elsevier. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1353802001000165/ [Accessed 10 Mar. 2020].

34. Eric Wong. (2020). How to improve wrist mobility. [online] Precision Movement. Available at: https://www.precisionmovement.coach/how-to-improve-wrist-mobility/ [Accessed 9 Mar. 2020].

35. Jose H. Friedman,MD.(2015). Advanced Parkinson's Disease: Shared Treatment Decisions and Patient Advocacy. [online] MedScape. Available at: https://www.medscape.org/viewarticle/839251/ [Accessed 9 Mar. 2020].

36. Niamh Gorman .(2020).Flexor Carpi Ulnaris Muscle. [online] Ken Hub. Available at: https://www.kenhub.com/en/library/anatomy/flexor-carpi-ulnaris-muscle/ [Accessed 9 Mar. 2020].

37 Niamh Gorman .(2020).Extensor Carpi Radialis longus Muscle. [online] Ken Hub. Available at: https://www.kenhub.com/en/library/anatomy/extensor-carpi-radialis-longus-muscle/[Accessed 9 Mar. 2020].

38. Pietro Mazzoni, Britne Shabbott, and Juan Camilo Corte´ s .(2012). Motor Control Abnormalities in Parkinson's Disease. [online] ResearchGate./ Available at: https://

www.researchgate.net/figure/Bradykinesia-and-hypokinesia-manifested-in-finger-tapping-Individuals-were-asked-to-tap_fig2_225276798 [Accessed 23 Feb. 2020].

39. James J. Licari. 1998. Science Direct website. [Online]. [Accessed 01 November 2019]. Available from: https://www.sciencedirect.com/

40. Magazine Pro Theme on Genesis Framework, 2019. EdgeFX.in website. [Online]. [Accessed 01 November 2019]. Available from: https://www.edgefx.in/

41. Dr.Yiming Wang. Qsota medical website. [Online]. [Accessed 03 november 2019]. Available from: https://qsota.com/

Scranton Gillette Communications. 2018. DAIC website. [Online]. [Accessed 04 November 2019]. Available from: https://www.dicardiology.com/

42. Valentina Palladino. 2017.arsTechnica website. [Online]. [Accessed 03 November 2019]. Available from: https://arstechnica.com/

43. Jeb Su. 2018.Forbes website. [Online]. [Accessed 09 November 2019]. Available from: https://www.forbes.com/

44. Vanessa Hand Orellana. 2019. Cnet website. [Online]. [Accessed 10 November 2019]. Available from: https://www.cnet.com/

45. Bayer. 2019. website. [Online]. [Accessed 10 November 2019]. Available from: https://www.thrombosisadviser.com/

46. FDA. 2019. [Online]. [Accessed 30 November 2019]. Available from: https://www.fda.gov/

Monir El Azzouzi. 2019. [Online]. [Accessed 30 November 2019]. Available from: https://easymedicaldevice.com/

47. ThermPro. 2019. [Online]. [Accessed 29 November 2019]. Available from: https://buythermopro.com/

48. Fried, L., 2019. *Adafruit Industries*. [online] Adafruit.com. Available at: <https://www.adafruit.com/about> [Accessed 10 January 2020].

49. Rembor, K., 2019. *Welcome To Circuitpython!*. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/welcome-to-circuitpython/what-is-circuitpython> [Accessed 5 February 2020].

50. Rembor, K., 2019. *Installing Mu-Editor*. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/welcome-to-circuitpython/installing-mu-editor> [Accessed 5 February 2020].

51. Tollervey, N., 2020. *Mu: A Python Code Editor — Mu 1.1.0.Alpha.2 Documentation*. [online] Mu Readthedocs. Available at: <https://mu.readthedocs.io/en/latest/index.html.> [Accessed 6 March 2020].

52. B, Cian., 2019. *UF2 Bootloader: Creating Custom Boards*. [online] Arduino Project Hub. Available at: <https://create.arduino.cc/projecthub/wallarug/uf2-bootloader-creating-custom-boards-c9620c> [Accessed 11 March 2020].

53. CircuitPython. 2020. *Core Modules — Adafruit Circuitpython 0.0.0 Documentation*. [online] Available at: <https://circuitpython.readthedocs.io/en/latest/shared-bindings/index.html> [Accessed 8 March 2020].

54. Rembor, K., 2020. *Creating And Editing Code*. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/welcome-to-circuitpython/creating-and-editing-code> [Accessed 4 January 2020].

55. 2C Info – I2C Bus, Interface and Protocol. (2020). I2C Bus Specification. [online] Available at: https://i2c.info/i2c-bus-specification#data-transfer [Accessed 8 Mar. 2020]

56. Circuit Basics. (2020). Basics of the I2C Communication Protocol. [online] Circuit Basics. Available at: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ [Accessed 8 Mar. 2020].

57. Texas Instruments. (2020). DRV2605L 2- to 5.2-V Haptic Driver for LRA and ERM with Effect Library and Smart-Loop Architecture. [online] Available at: http://www.ti.com/lit/ds/symlink/drv2605l.pdf [Accessed 8 Mar. 2020].

58. Rembor, K. (2020). *Adafruit Circuit Playground Bluefruit*. [online] Learn Adafruit. Available at: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-circuit-playground-bluefruit.pdf [Accessed 7 Mar. 2020].

59. Eames, A. (2013). *Setting up and using outputs with RPi.GPIO*. [online] RasPi.TV. Available at: https://raspi.tv/2013/rpi-gpio-basics-5-setting-up-and-using-outputs-with-rpi-gpio [Accessed 8 Mar. 2020].

60. Kuphaldt, T. (n.d.). *Resistors | Ohm*. [online] Allaboutcircuits.com. Available at: https://www.allaboutcircuits.com/textbook/direct-current/chpt-2/resistors/ [Accessed 8 Mar. 2020].

61. National Instruments. 2019. *What Is A Pulse Width Modulation (PWM) Signal And What Is It Used For? - National Instruments*. [online] Available at: <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU&l=en-IE> [Accessed 09 Mar. 2019].

62. Kesharwani, M., 2020. *Do The SDA And SCL Pins Of I2C Require Pull Up Resistors? If Yes, Why? - Quora*. [online] Quora.com. Available at: <https://www.quora.com/Do-the-SDA-and-SCL-pins-of-I2C-require-pull-up-resistors-If-yes-why> [Accessed 10 Mar. 2020].

63. Resistor's Guide. 2017. Pull Up Resistor / Pull Down Resistor » Resistor Guide. [online] Available at: <http://www.resistorguide.com/pull-up-resistor_pull-down-resistor/> [Accessed 12 March 2020].

64. Raj, A., 2018. What Is PWM: Pulse Width Modulation. [online] Circuitdigest.com. Available at: <https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation> [Accessed 13 March 2020].

65. Tutorialspoint. 2020. Signals Basic Types. [online] Available at: <https://www.tutorialspoint.com/signals_and_systems/signals_basic_types.htm> [Accessed 12 March 2020].

66. Ashrit, L., n.d. Pulse Width Modulation (PWM) - Generation, Applications And Advantages. [online] electricalfundablog.com. Available at: <https://electricalfundablog.com/pulse-width-modulation/> [Accessed 12 March 2020].

67. Ada, L., 2013. Halloween Pumpkin. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/halloween-pumpkin/overview> [Accessed 10 March 2020].

68. Clark, L., 2020. BLE Synth With The Feather Nrf52840 And Circuit Playground Bluefruit. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/ble-synth-with-the-feather-nrf52840-and-circuit-playground-bluefruit> [Accessed 10 March 2020].

69. Android Developers. 2020. *Known Issues With Android Studio And Android Gradle Plugin*. [online] Available at: <https://developer.android.com/studio/known-issues?hl=ro> [Accessed 14 April 2020].

70. Brothers, R., 2020. Circuitpython BLE Controlled Neopixel Hat. [online] Adafruit Learning System. Available at: <https://learn.adafruit.com/circuitpython-feather-ble-neopixel-hat> [Accessed 10 March 2020].

71. W3Resource. 2020. Python Bytes, Bytearray - W3resource. [online] Available at: <https://www.w3resource.com/python/python-bytes.php> [Accessed 11 March 2020].

72. EDUCBA. 2020. String Array In Python | Python Lists | Methods Of String Array In Python. [online] Available at: <https://www.educba.com/string-array-in-python/> [Accessed 11 March 2020].

73. En.wikipedia.org. 2020. *Raspberry Pi*. [online] Available at: <https://en.wikipedia.org/wiki/Raspberry_Pi> [Accessed 7 May 2020].

74. Explain that Stuff. 2020. *Forces And Motion: A Simple Introduction*. [online] Available at: <https://www.explainthatstuff.com/motion.html> [Accessed 5 May 2020].

75. Adafruit Learning System. 2020. *Sensor Plotting With Mu And Circuitpython*. [online] Available at: <https://learn.adafruit.com/sensor-plotting-with-mu-and-circuitpython/motion> [Accessed 5 May 2020].

76. Dominguez, A., 2020. *Why You Need A Monitoring System ? Offer The Best Service For Your Clients*. [online] Pandora FMS - The Monitoring Blog. Available at: <https://pandorafms.com/blog/why-you-need-a-monitoring-system/> [Accessed 5 May 2020].

77. Learn.sparkfun.com. 2020. *Accelerometer Basics - Learn.Sparkfun.Com*. [online] Available at: <https://learn.sparkfun.com/tutorials/accelerometer-basics/all> [Accessed 5 May 2020].

78. Vernier.cz. 2020. [online] Available at: <https://www.vernier.cz/katalog/manualy/en/3d-bta.pdf> [Accessed 5 May 2020].

79. Dimensionengineering.com. 2020. *A Beginner's Guide To Accelerometers*. [online] Available at: <https://www.dimensionengineering.com/info/accelerometers> [Accessed 6 May 2020].

80. Developers, A., 2020. *Start Another Activity*. [online] Android Developers. Available at: <https://developer.android.com/training/basics/firstapp/starting-activity> [Accessed 5 February 2020].

81. Przybyla, D., 2020. *Turquoise Color Psychology And Meaning*. [online] Available at: <https://www.colorpsychology.org/turquoise/> [Accessed 6 February 2020].

82. Castañeda, A., 2020. *The Noun Project*. [online] The Noun Project. Available at: <https://thenounproject.com/term/hand-print/44713/> [Accessed 6 February 2020].

83. StatCounter Global Stats. 2020. *Mobile Operating System Market Share Worldwide | Statcounter Global Stats*. [online] Available at: <https://gs.statcounter.com/os-market-share/mobile/worldwide> [Accessed 15 May 2020].

84.Android Authority. 2020. *The History Of Android OS: Its Name, Origin And More*. [online] Available at: <https://www.androidauthority.com/history-android-os-name-789433/> [Accessed 15 February 2020].

85.Android Authority. 2020. *7 Reasons Why You Should Develop Apps For Android Rather Than Ios*. [online] Available at: <https://www.androidauthority.com/develop-apps-for-android-rather-than-ios-607219/> [Accessed 14 February 2020].

86. Android Authority. 2020. *I Want To Develop Android Apps — What Languages Should I Learn?*. [online] Available at: <https://www.androidauthority.com/develop-android-apps-languages-learn-391008/> [Accessed 14 February 2020].

87. AltexSoft. 2020. *The Good And The Bad Of Android App Development*. [online] Available at: <https://www.altexsoft.com/blog/engineering/pros-and-cons-of-android-app-development/> [Accessed 14 February 2020].

88. Statista. 2020. *Android Versions Market Share 2019 | Statista*. [online] Available at: <https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/> [Accessed 14 February 2020].

89. Google Play. 2008. *Nrf UART 2.0*. [online] Available at: <https://play.google.com/store/apps/details?id=com.nordicsemi.nrfUARTv2&hl=en_US> [Accessed 7 January 2020].

90. GitHub. 2020. *Nordic Semiconductors Android-Nrf-UART*. [online] Available at: <https://github.com/NordicPlayground/Android-nRF-UART> [Accessed 5 January 2020].

91. Docs.gradle.org. 2020. *What Is Gradle?*. [online] Available at: <https://docs.gradle.org/current/userguide/what_is_gradle.html> [Accessed 19 February 2020].

92. Enterin. 2020. Neurodegenerative Disease | Parkinson's Disease. [online] Available at: <https://enterininc.com/parkinsons-disease/> [Accessed 10 March 2020].

93. Encyclopedia Britannica. 2020. *Equilibrium | Physics*. [online] Available at: <https://www.britannica.com/science/equilibrium-physics> [Accessed 15 May 2020].

94. Precisionmicrodrives.com. 2020. *Coin Vibration Motors - Precision Microdrives*. [online] Available at: <https://www.precisionmicrodrives.com/vibration-motors/coin-vibration-motors/> [Accessed 21 February 2020].

95. Udemy. 2019. *Python Bootcamps: Learn Python Programming And Code Training*. [online] Available at: <https://www.udemy.com/course/complete-python-bootcamp/> [Accessed 2 November 2019].

96. Androidplot. 2016. *Adroidplot About*. [online] Available at: <http://androidplot.com/about/> [Accessed 16 April 2020].

97. GitHub. 2016. *Androidplot*. [online] Available at: <https://github.com/halfhp/androidplot/blob/master/docs/quickstart.md> [Accessed 16 April 2020].

98. H. and Gabriel, S., 2015. *How To Scan For Available Bluetooth Devices In Range In Android?*. [online] Vingtoft. Available at: <https://stackoverflow.com/questions/3170805/how-to-scan-for-available-bluetooth-devices-in-range-in-android> [Accessed 17 April 2020].

99. Kim, M., 2015. *Create A Bluetooth Scanner With Android's Bluetooth API*. [online] Code Envato Tuts+. Available at: <https://code.tutsplus.com/tutorials/create-a-bluetooth-scanner-with-androids-bluetooth-api--cms-24084> [Accessed 17 April 2020].

100. Tutorialspoint.com. 2020. *Android List View - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/android/android_list_view.htm> [Accessed 17 April 2020].

101. Android Developers. 2019. *Save Key-Value Data | Android Developers*. [online] Available at: <https://developer.android.com/training/data-storage/shared-preferences> [Accessed 17 April 2020].

102. Shek, A., 2018. *How To Add/Create Local HTML File In Android Studio*. [online] Abhiandroid.com. Available at: <https://abhiandroid.com/androidstudio/add-local-html-file-android-studio.html> [Accessed 20 April 2020].

103. CodeBind.com. 2017. *Android Studio – How To Save A File On Internal Storage (Read / Write)*. [online] Available at: <http://www.codebind.com/android-tutorials-and-examples/ndroid-studio-save-file-internal-storage-read-write/> [Accessed 21 April 2020].