FES Bike - Designing a Smart Pedal

Team members:

Vivien Alves Passing

Samuel Karet

Mya-Rae Lord

Nestoras Neofytou

William Paterson

Livia Robic

Mohammed Shahr Abadi

Nicholas Ustaran-Anderegg

Nathanael Tan

Samuel Yeadon

Supervisor:

Dr Ian Radcliffe

June 2019

Abstract

Functional Electrical Stimulation (FES) is a relatively new technology with much potential in rehabilitation and mobility. One such application is its use to produce a pedaling motion of the legs, as used by Johnny Beer-Timms, an athlete with paraplegia, who uses a recumbent bicycle adapted for FES cycling. Just as professional cyclists optimise every aspect of their physical, mental and mechanical performance, Johnny wishes to push his performance to its limit.

However, while much research has been focused on improving the FES technology itself, comparatively little has been put into the mechanical design of these bicycles. For example, the pedals continue to be heavy and unwieldy, and without consideration of the needs of specific users they are unable to provide the sufficient stabilization for efficient pedaling. Furthermore, Johnny lacks any system that can provide him with meaningful feedback, making it difficult to quantify changes to his performance.

In this project, the team designed and manufactured a pedal that addresses these concerns, while adhering to the regulations laid out by the Cybathlon organisers. The requirements of the design were developed by close collaboration with the user on what data the pedals should measure, how they were to be displayed, the issues he had with his original pedals how he envisioned the pedal in general.

The final product adapts a snowboard binding and bicycle pedal to create an adjustable base, with an aluminium calf support of the team's own design and an ITB strap to secure the le.g. The performance feedback system was composed of a microcontroller on each pedal measuring force and cadence, which report this data to a central processor for display and storage. However, it did not meet some of the objectives for physical stiffness and weight. However, testing and evaluation provided valuable insights to the sources and possible solutions to the issue. It is likely that given further resources and development time, the product would be able to meet all requirements laid out.

Acknowledgements

We would like to thank all those who have advised us throughout this project:

Our supervisor, Dr. Ian Radcliffe for his continuing guidance;

Mr. Paschal Egan and Mr. Niraj Kanabar, for their on advice on the electrical and electronics aspects of the project;

Ms. Ji Young Yoon, for her assistance and support of the mechanical aspects and manufacture;

Dr. Frank Gommer for his expertise on composite materials;

Johnny Beer-Timms, for the chance to work with him on this project and his continued feedback and support.

Contents

1 Introduction

1.1 Background

Johnny Beer-Timms is a competitive athlete and cyclist unlike most others - after incurring a spinal injury during a trampoline practice in 2011, Johnny lost the entire mobility of his legs and partial mobility of his arms and hands. He now cycles on a recumbent bicycle using Functional Electrical Stimulation (FES), which passes electrical impulses to his leg muscles through two patches placed on his thighs, stimulating a pedalling motion.

Johnny adjusts the frequency of the electrical pulses that are being sent but the challenge lies in optimising his cycling speed while limiting the fatigue that can rapidly build up in his legs without him feeling it.

Cybathlon is a unique championship hosted every four years by ETH Zurich during which people with physical disabilities compete against each other by completing everyday tasks using state-of-the-art technical assistance systems. In 2016, Johnny participated in the bicycle race and brought back the silver medal. He intends to win the gold medal in the Cybathlon 2020, and reached out to Imperial College London with some suggestions on how his performance can be improved.

1.2 Aims

The aims of the project are as follows:

- To improve the ergonomics of his bike pedal, ensuring that it holds his leg in place securely;
- To provide an improved pedal base that allows for adjusting the position of where the pedal spindle meets the pedal base;
- To install a feedback system that provides live and stored feedback for Johnny to evaluate changes in his performance, and fine-tune his FES parameters and cycling techniques.

1.3 Objectives

It was decided early on that these objectives could be pursued in parallel. The project group was therefore distributed into separate Mechanical and Electrical teams, with the Mechanical team focusing on the redesign of the bike pedal, and the Electrical team working on the feedback system and incorporation of sensors into the pedal. To further facilitate project management

and division of labour, the product itself was further separated into several components, allowing for performance objectives to each part to be defined.

For the mechanical team these were:

- The calf support that holds the cyclists upper leg in place, which needs to be strong and rigid enough to prevent lateral straying of his legs.
- The straps that secure the upper leg to the calf support. These need to hold the leg securely to the support while remaining comfortable and safe for extended use.
- The pedal base, including the adjustable spindle position, which is adjustable to allow Johnny to find the best position for power transfer.

The electrical team dealt with the feedback system, which was broken down into:

- The circuit design and choice of sensors to measure force and cadence, integrated on the pedal. A variety of performance metrics can be obtained from knowing the force and cadence (pedal cycles per unit time).
- The transmission relay that streams data from these sensors to a central processor. The data needs to be transmitted to a central unit to be displayed. The main design parameters are where along the chain the data will be processed, and the transmission protocol to be used.
- The central processing system which processes, stores, and displays the data from the sensors. The graphical interface for live feedback and the data interface for stored information (i.e. the file type and formatting, storage and transfer) will all need to be considered in the programming of the operating system.

Both teams generally followed a similar process of research and design, manufacture and evaluation.

Research and Design.
Possible methods, materials and media that could be used to achieve the objectives of the respective groups were investigated. The primary concerns for the Mechanical parts are their design, choice of materials, and methods of manufacture. Design of the electrical components involved the choice between different sensors, programming languages and data transmission protocols.

Manufacture:

A prototype pedal for end-user testing was manufactured, i.e. for active use by Johnny. As such, the prototype contained enough functionality for feedback and iteration, without having to be a final version.

Evaluation:

The manufactured product was compared to product specification and objectives as well as Johnnys feedback on how they meet his requirements.

Each of these steps will be discussed in more detail in subsequent sections.

2 Requirements definition

2.1 User requirements

Upon commencement of the project the team visited Johnnys residence in Aylesbury to better understand his requirements. Johnny explained that he had been training up his physical fitness and tuning FES parameters, but also wanted to optimise the performance of the bike.

The initial project brief was to design and build a longer crank for the bicycle, but this was quickly discovered to be trivial as he had simply purchased one. The team identified a number of areas for improvement based on discussion with Johnny and examination of the bicycle itself:

1. Johnny wanted to be able to adjust the parameters of the FES control box while cycling, which was not possible as his hands were strapped to the handles;
2. He was hoping to reduce the overall weight of the bike;
3. He was unsatisfied with how his legs tended to stray laterally while pedaling at high speeds, suggesting that either the straps that secured his leg to the calf support were not tight enough, or that the support itself lacked rigidity;
4. The spindle connects to the pedal underneath the ball of the foot in Johnnys current pedal. While this is beneficial for power transfer in traditional cyclists, the team believed that he might be able to transfer more power through his heel.
5. In order to optimise the timing of the FES program and quantify improvements to his performance, Johnny required some form of feedback.

Johnny was pursuing the first point with Rick Berkelman, the designer and retailer of the recumbent bicycle that he uses. Since points two to five could be addressed by redesigning the pedal, it was decided that the team would design an improved pedal with an integrated performance feedback system.

Following background research (detailed in the following section) and further discussion with Johnny, the team established a set of user requirements for the pedals:

- It must provide rigid axial support to his foot, preventing any power from being lost to unnecessary movement in his lower leg and foot.
- It must bind his legs tightly enough to achieve the above while also being safe and comfortable for extended periods of use. This is paramount as Johnny has reduced sensation in his legs and may not be able to sense any impingement, friction or chafing until they lead to significant pain and injury.
- The position of the spindle relative to the foot should be adjustable to allow Johnny to find the most suitable position.
- The pedal should have a reduced weight, and a profile that does not cause the bike to exceed maximum dimensions allowed by Cybathlon.

The feedback system should provide Johnny with access to the following:

- The cadence (pedal revolutions per minute), power delivered to pedal, and power phase. The power phase is defined as the position along the pedaling cycle where maximum power is delivered.
- The aforementioned data delivered while cycling, clear and legible on a monitor attached to the handle bars.
- The aforementioned data, also stored for comparison of long-term changes in performance.
- A user interface to initiate and terminate data recording and display, and to systematically save session data for his access.

It is also worth noting that Johnny is assisted by a carer who straps him in and out of the bike, and is therefore always present when he trains. Regardless, for Johnny's convenience and self-reliance, as much of the system should be accessible to him without assistance as possible.

2.2 Background research

2.2.1 Cybathlon regulations

It was vital first to establish what modifications to the bicycle were permitted under Cybathlon regulations. The rule book for Cybathlon 2020 [Zurich (2019)] specifies the general rules, race task, and regulations for all events. A few clauses relevant to the design were identified, namely:

GR-9 *It is allowed to use commercially available devices. Competitors are permitted to modify them to optimise function. Alternatively, prototypes and research devices are also eligible.*
This clause simply confirms that modifications to the bike are permitted. This is generally known, as many devices used in Cybathlon are themselves prototypes.

GR-13 *Communication (wired or wireless) between the device and any third-party stationary site is not allowed, i.e. remote connection to control the device by any person other than the pilot is forbidden, except for emergency stop and data monitoring.*
This clause confirms that on-board data collection can be relayed to a device off the the bike, even during the race. This option was considered but eventually not used.

3.2.2 Technology discusses regulations pertaining directly to FES bicycles, and includes the following sub-clauses of interest:

- *Only passive cycling devices without actuation are allowed. The only actuation is provided through the FES-stimulated legs of the pilot.*
  This clause clearly states that actuation is disallowed in the design of the bike, and by extension the pedals. Any reduction of stray movement would therefore have to be achieved purely by means of rigidity rather than active damping.

- *The maximum width of the cycling device is limited to 900 mm to enable proper use on the ramps and in the lanes of the race track. The cycling device must fit on the start ramp behind the start gate (total length 2000 mm)*
  This specifies the maximum dimensions of the bike, which limits how much the pedals can protrude from the bike.

Overall, the regulations of Cybathlon were not found to be particularly restrictive. The main restrictions of note were to the final size of the bicycle, and the prohibition of actuation. To ensure unambiguous compliance, however, the team decided that the electrical components should be removable to affirm that the pedals did not contain any active power source.

2.2.2 Currently available products

The pedals that Johnny was using were manufactured by Hase, a company specializing in adaptable bicycles. Hase has updated the design of their pedals to include a highly adjustable footplate, quick release strap, and redesigned calf support.

However, the position of the axle connecting to the spindle remained at the ball of the foot, and is not adjustable. This, in addition to the fact that the Hase pedal is designed for a wide range of disabilities and bicycle configurations, affirmed the team's decision that designing a pedal specifically for Johnny remained worthwhile.

Furthermore, the team was unable to find evidence of a sensor suite designed specifically for recumbent bicycles, with the vast majority of commercial cycling computers and performance sensors designed for typical cyclists.

It was therefore agreed that despite the existence of both cycling monitors and high-performance recumbent bike pedals, the group's objective to develop and build a combination of the two would still be novel.

2.2.3 Relevant biomechanics

One of the objectives of the pedal design is to provide an adjustable position for the pedal axle, in order to allow Johnny to find the optimal position for power transfer. A typical cyclist's optimal axle position is typically accepted to be at the front of the foot [Fonda & Sarabon (2010)], which is presumably why Johnny's current pedals are positioned this way.

However, Johnny's FES bicycle is two degrees removed from a conventional bicycle; firstly due to its configuration as a recumbent bicycle, and secondly due to his different muscle function. The team's intuition was that these differences may mean that the optimal spindle position relative to the foot is different from that used conventionally. Research into literature on the biomechanics of cycling and FES was therefore needed to validate this intuition and the provision of an adjustable axle position.

Difference in configuration – Telli et al. (2017) compares various biomechanical properties (such as body centre of mass, muscle tension and length, joint angles) between recumbent and upright cycling for typical cyclists. They conclude that only small differences are appreciable

between the two postures, with small benefits to aerodynamics and external work applied in the case of recumbent bicycles. In other words, the different posture of recumbent cycling appears not to have a major effect on the biomechanics in terms of how power is applied and transferred, and therefore conclusions drawn for upright cycling are applicable to recumbent cycling.

Muscle activity due to FE stimulation – Rebecca Martin et al. (2012) discuss the major differences between FES-initiated muscle contraction and physiological contraction. Most importantly, motor unit recruitment is based on size and proximity to the electrode, progressing from large to small muscles - an inverse of voluntary contractions.

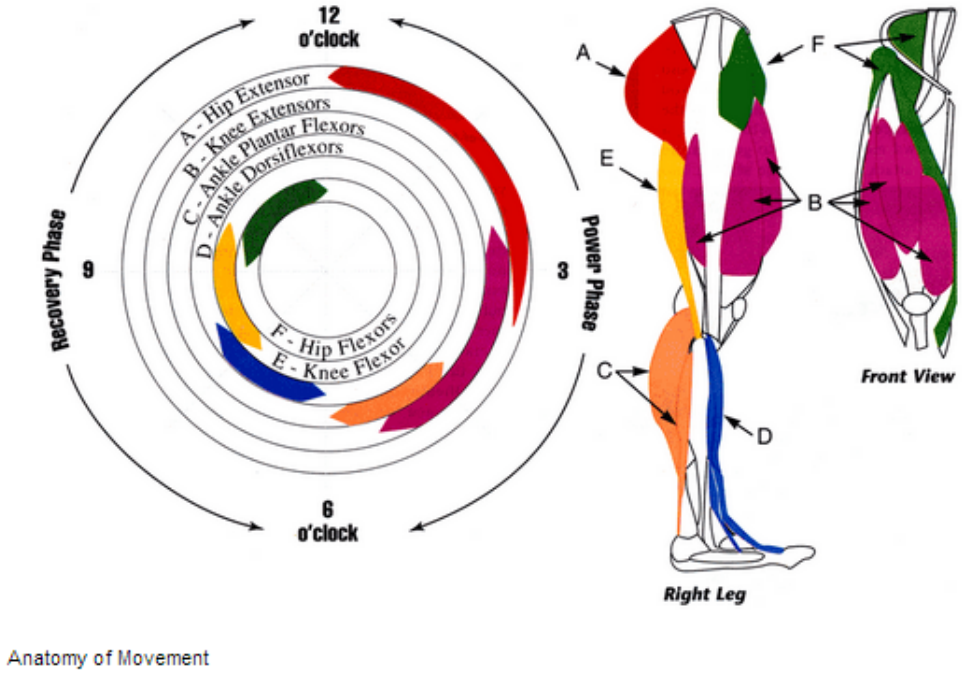


Figure 1: Muscle engagement during pedal cycle. [Physiopedia (2019)]

Figure 1 outlines the muscle engagement in a typical pedal cycle. The key implication of Johnny's cycling motion being FES stimulated is that he does not have electrodes placed at his lower leg, and therefore the ankle plantar flexors and ankle dorsiflexors (C & D in Fig.**??**) are not recruited when he pedals.

The ankle plantar flexor exerts a downward force and is responsible for stabilising the the ankle in the downward stroke of the power phase, while the dorsal flexor stabilises the foot while in the upward motion of the recovery phase. Lacking command of these two muscles, Johnny can

neither control the flexion of his ankles nor produce additional power from his calves.

The team hypothesises that this neutralises the benefits traditionally provided by positioning the spindle beneath the forefoot, since he can neither balance nor stabilize the foot in the manner required to provide leverage to apply force through the ball of the foot to the pedal spindle. The ankle is stabilized instead via seizure by the pedal assembly, but this is likely to affect the balance point, moving it closer to the heel.

While it is beyond the scope of this design project to provide a full biomechanical analysis and recommend the ideal balance point, the above analysis serves as proof of concept that it will be worthwhile for the user to experiment with various positions to find the point of balance suited to them. The feedback system further facilitates this by allowing him to quantify changes in performance.

2.2.4 Force sensors

There are three main technologies available for force sensing - **load cells**, **strain gauges**, and **force-sensitive resistors**. The principle of operation, benefits and drawbacks of each are outlined below:

Strain gauges are lengths of resistive wire bonded to the object of interest. When force is applied to the object, the gauge experiences a change in length (i.e. a strain) and the accompanying change of resistance is measured. They can be accurate if well-implemented, and their size makes them an appealing option. However, strain gauges must be well bonded to the material to be effective, and are prone to breaking if exposed and improperly handled. They were therefore considered unsuitable for use in the prototype, as robustness of the sensors was desired.

Load cells are sensors which transduce force to an electric signal using various means, such as piezoelectric and hydraulic effects. Several load cells actually use strain gauges for transduction, circumventing the issues described above by enclosing the gauge within the cell. They are also convenient to use in that the strain gauge is pre-connected to a differential bridge circuit. However, their form factor is unsuitable to be integrated onto the pedal, which requires a flat, thin, wide surface of measurement.

Force-sensitive resistors (henceforth FSRs) are made of conductive polymer which decreases in resistance when a force is applied to it. FSRs are reasonably robust but also come in thin

films, making them highly suited for the needs of the design. However, the resistance is known to be nonlinear with respect to force applied.

Table 1: Comparison of force-sensing technologies. **o** denotes suitability, **x** denotes unsuitability, and – denotes neutral suitability (not ideal, but can be designed around).

Technology	**Robustness**	**Form factor**	**Accuracy**
Strain gauge	x	o	–
Load cells	o	x	o
Force-sensitive resistors	o	o	–

From the comparison in **Table 1**, it is evident that FSRs are the most suitable for the requirements of the design, and were selected for use in the final design.

2.2.5 Cadence sensing

To measure cadence, there are several different options available, including a Hall effect sensor or a gyroscope/accelerometer-based system. Ultimately, the team decided on the bipolar Hall effect sensor because it involves a less complicated and more efficient method with regards to coding and setting up. This was particularly important due to the time constraints imposed by the project. The Hall effect itself is discussed in **Appendix F**.

2.2.6 Wireless transmission

Any form of data transmission from the pedals would have to be wireless due to the continuous rotational motion of the pedals. For this, several methods were considered during the design process: Wi-Fi, Bluetooth and ANT+. The final decision was to use Bluetooth Low Energy (BLE) because the Raspberry Pi and Bluno Nano have inbuilt hardware, which allow for Bluetooth connectivity. BLE is energy efficient and has a lower latency than the other methods during connection and data transmission.

2.3 Technical requirements

The technical requirements the product should fulfill are outlined below. **M** and **D** indicate **mandatory** and **desired** outcomes respectively.

Table 2: Table summarising the technical requirements of the product.

No.	Requirement	Priority
1	**Functionality and Performance**	
1.1	Pedal must be able to transfer a force of 600N from the underside of the foot to the end of the spindle without flexing more than 2 degrees in all directions. In practice must withstand 1200N (using a safety factor of 2).	M
1.2	Calf support must not allow the lower leg to move laterally more than 5mm in all transverse directions under a 50N load.	M
1.3	The bindings must secure the foot to the pedal surface tightly, so that there is no more than 3mm of movement of the foot in relation to the pedal in all directions.	M
2	**Size and Weight**	
2.1	Electronic box embedded in the pedals must be no more than 10% of the total mass on the pedals. Small monitor must be no more than 0.3kg.	D
2.2	Each pedal must be lighter than the existing pedal at 1.2 kg.	D
2.3	The dimensions of the main pedal assembly should conform to the following:	
2.3.1	Area of base of pedal must fit UK male size 11-12 shoe (i.e. 250mm length and width ranging from 85mm - 120m).	M
2.3.2	To comply with the starting gate lane width at Cybathlon, the pedals must not cause the overall width of the bike to exceed 900mm in width and 2000mm in length.	M
2.3.3	The calf support should attach to the upper calf 50-60mm below the knee.	M
2.3.4	Calf support to be 100mm to 150mm long and have a circumference of 320mm to firmly hold the calf and knees in place.	D
2.3.5	Any additional straps used to be at least 40mm wide to securely hold the leg in place during use.	D
2.3.6	Bolt connecting pedal to the bike spindle must be an M12 bolt size.	D
2.4	The pedal feedback sensors must be sized as to securely fit the pedals, that is, no more than 120mm wide and no more than 200mm long. These sensors must be as thin as possible, preferably less than 10mm. Small monitor to display real time data must be smaller than 100mm x 200mm.	M
3	**Usability & ergonomics**	

Table 2: Table summarising the technical requirements of the product.

No.	Requirement	Priority
3.1	Water resistance - electronic systems must not be affected by light rain, and mechanical components should not rust easily from occasionally cleaning or light rain.	D
3.2	As Cybathlon and various other competitions take place overseas, the pedals should be detachable or otherwise easily transported.	D
4	**Safety**	
4.1	The final product parts must not interfere with the braking system so that the user will be able to slow down and stop as before.	M
4.2	Electrical components (such as sensors) must be insulated so as to not come into contact with water.	M
4.3	The pedals should not contain any sharp edges upon which the user could potentially hurt himself.	M
5	**Life, reliability and Maintenance**	
5.1	Must at least last until Cybathlon 2020 and withstand at least 365 hours of sustained use (based on 1 hour of training every 2 days).	M
5.2	The materials chosen need a high fatigue strength, about 130 MPa.	M
5.3	Parts can be attached/detached separately and can be attached using screws or bolts. Parts must also be easily accessible in case maintenance is required.	D

3 Final design

The final design of the pedal system consists of:

- An aluminium plate that mounts the assembly to the pedal spindle, forming the base,
- Snowboard bindings for the foot to strap in, mounted onto the aluminum plate,
- An aluminum calf support mounted onto the heel of the snowboard binding,
- An ITB strap to securely fasten the leg into place,
- A Bluno Nano on each pedal which communicates wirelessly with a Raspberry Pi display.

Figure 2: The pedal presented to Johnny on demonstration day.

3.1 Hardware

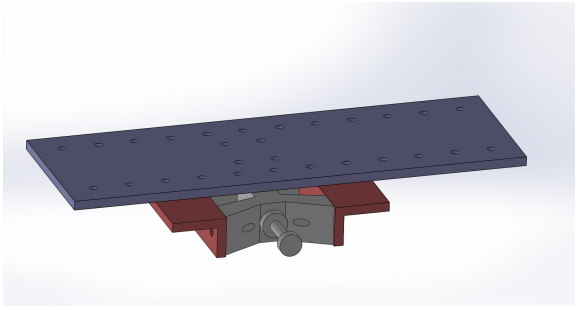
The mechanical side of the pedal design begins with a pair of **MKS Espirit alloy pedals.** These were chosen because they have M5 tapped holes to which the rest of the pedal can be attached. The pedal axle itself is complicated to design, and doing so well would not be novel - hence the use of an existing component.

A highly adjustable **aluminium base plate** was then created and mounted on top of the MKS pedal by two 74 mm long aluminium angle brackets. Two rows of 9 holes were drilled onto a 3mm thick, 260 mm by 71 mm plate, which allow for the adjustability that allows Johnny to change his foot position and experiment to find the optimum arrangement for peak cycling

performance. **Figure 3b** depicts the resulting arrangement.

The underside of this plate also provides an attachment point for small laser cut boxes that contain the microcontrollers and cadence measuring equipment. These boxes each contain a small LED on the side to indicate that the electrical system is working.

(a) The MKS Espirit alloy pedals.



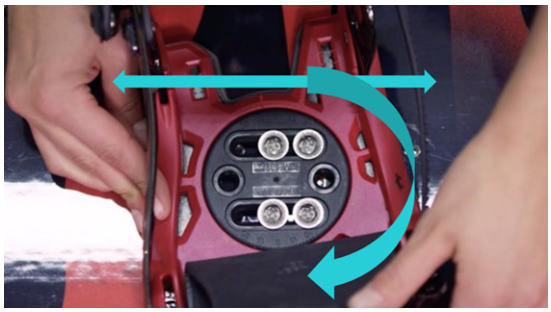
(b) A SolidWorks rendering of the aluminum base.

Figure 3: The MKS pedals and a SolidWorks rendering demonstrating the base plate assembly.

On top of the aluminium base plate, a **snowboard binding** was used as the mechanism to keep Johnnys foot secured in place. This was selected for its comfort and ease of use when fastening or unfastening, but also for its high degree of adjustability. The toe and ankle bindings firmly secure the user's feet to the binding base, while the padded straps minimise the risk of pressure sores developing. Salomon Rhythm bindings were chosen in particular because they have a toe strap which wraps around the front of the foot, rather than on top of the toes, meaning that they would be more comfortable for use with shoes, as the user would not be wearing snowboard boots.

The binding is attached to the base plate via an **indexable disc**, which allows rotation of the foot if necessary, to keep the leg in line. The slotted holes allow lateral movement of the binding with respect to the pedal base to maintain clearance between the binding and pedal spindle. This adjustability is illustrated in **Figure 4b**. This further adjustability means that Johnny is really able to find the best configuration that works for him.

(a) The Salomon Rhythm bindings.



(b) The indexable disc.

Figure 4: The snowboard bindings used, and the indexing disc allowing for foot angle adjustment.

The highback of the snowboard binding was replaced by a **calf support** of the team's design that extends upwards behind the calf to support the lower leg. This part was cut from an aluminium sheet and then rolled into a semi-circular shape of similar radius to Johnnys calf; room to grow was allowed. Foam padding was added at the top of the calf support for comfort.

Originally designed for iliotibial band (ITB) injuries, a **Mueller ITB strap** was used to secure the leg below the knee, as shown in **Figure 5c**, at the top of the calf support. This was threaded through slots made in the calf support visible in **Figures 5a & 5b** easily. The Velcro binding allows adjustment and good mechanical strength, resisting both normal and shear forces. The plastic buckle provides additional mechanical strength by halving the force. A nylon strap was sewn onto the ITB strap to provide more strength and minimise flexibility so that Johnny stays completely secure.

(a) Front view.

(b) Rear view.

(c) Proposed placement of strap.

Figure 5: The manufactured calf support installed on the snowboard bindings, with slot for ITB strap circled.

3.2 Software and electronics

The aim of the electrical components integrated with the bike pedal is to provide the user with a live and post-training performance feedback, to keep track of progress and determine if any other adjustments (i.e. crank length, wheel size), made on the bicycle improve its performance.

Two data collecting devices (one on each pedal) feed the main unit where the processing and the presentation of the data takes place. The main unit has a user-friendly touch screen and can be mounted either on the handlebars, or on a side stand. All the data is exported as a spreadsheet to a flash memory drive at the end of the session for the user to get a more detailed post training analysis on his personal computer.

Figure 6 provides a visual overview of this system, while the full circuit diagram is available in **Appendix H**.

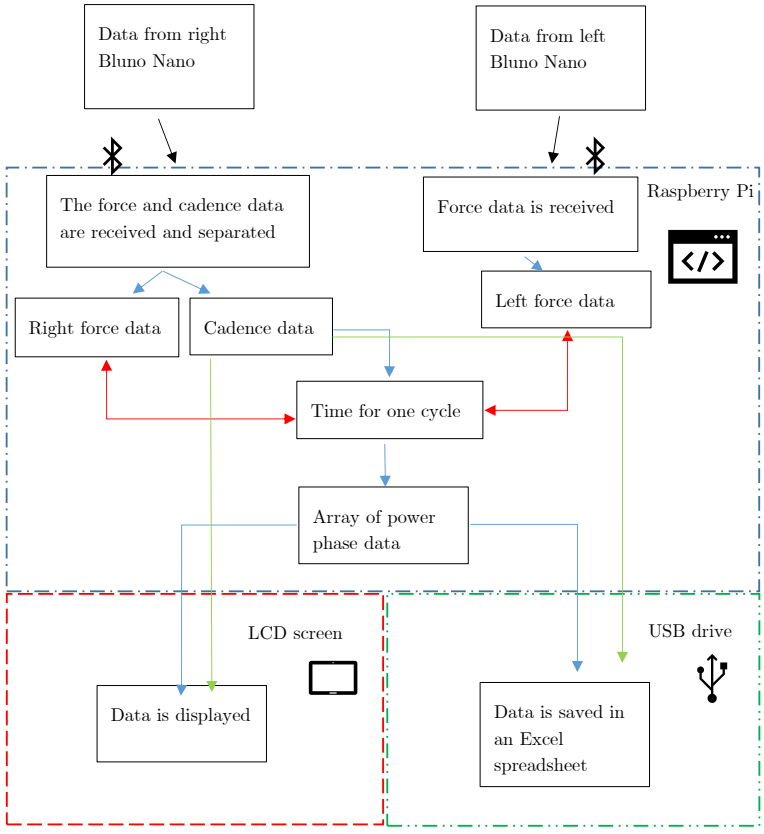


Figure 6: Flowchart of the input, processing and output of data through the system

3.2.1 Sensors

In this section, the design of the sensors built in to the pedals is discussed.

Bluno Nano

Each pedal feeds the processing unit with force measurements, but only one of them also provides cadence. Due to the continuous rotational motion of the pedals the connection needs to be wireless. The most energy efficient way to create a small distance fast data transferring connection of two peripheral devices on a parental device was by Bluetooth Low Energy (BLE). Therefore, the selected micro controller was the **Bluno Nano by DFrobot** (henceforth simply Bluno Nano). Its small size (53 x 19 x 12 mm), the integrated BLE module and the 30 hours of running time out of a 9V battery made it well suited to the requirements. Programs for these controllers are written in the Arduino programming language - hence any reference subsequent reference to Arduino refers to the programming language, while Bluno Nano refers to the physical device.

Force-sensitive resistor

Each FSR is connected in series with a resistor as shown in **Figure 7**, forming a potential divider. The Blunos measure the voltage drop across the fixed resistor, to calculate the voltage across the FSR. The conductance of the FSR is calculated, and the corresponding force is found. The 70kΩ value was selected as this allows the measured voltage to range from 0.4 V to 4.1 V for forces up 400 N. Since there are two FSRs on the pedal, the FSRs are together able to measure up to 800 N.

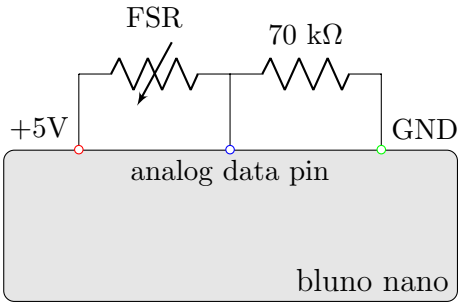


Figure 7: Simplified circuit diagram depicting FSR sensor connection

Two FSRs are placed on each pedal, connected to the same Bluno Nano. They are situated at the heel and the metatarsus - the two points where pressure from the feet is applied. In order to improve the contact area and force concentration, the FSRs are placed between flat plates that fit the shape of the binding area on one side, and the shape of the FSR on the other. These 'pucks' were then laminated onto the ski binding. In addition to increasing the accuracy and sensitivity of the FSRs, they also protect the sensor from shear forces mostly occurring when the user gets his feet on to and off of the pedal.

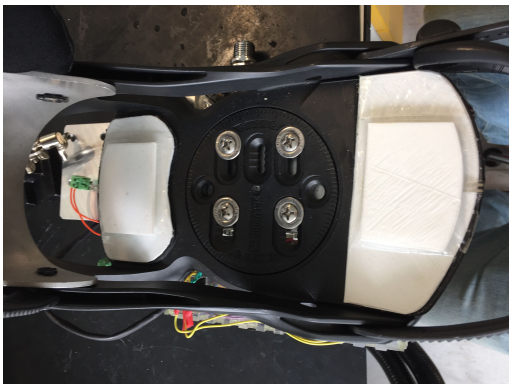


Figure 8: The placement of the FSRs on the pedal, beneath pucks 3D-printed in white.

Cadence

Cadence is measured using a bipolar Hall effect sensor connected to the Bluno Nano. The Hall effect sensor itself is mounted under the pedal, and a 10kg-pull magnet is mounted to the frame of the bike using Velcro.

Housing

A detachable transparent housing containing the circuitry, the 9V battery and the micro controller was constructed by laser cutting its side panels out of acrylic Panel. In order to reduce weight and ensure compliance with regulations, the housing was designed to be easily detached from the pedal by removal of two screws on its underside. The power button, alongside an LED to indicate that the device is on, are located on the outer side of the housing.

3.2.2 Software

Microcontroller-level data processing

Force measurements are taken at a fixed rate of 10 per second. Knowing the period of a single pedaling cycle from the cadence, and approximating the angular velocity to be constant, the position at which each force measurement was taken can be approximated.

Since the Bluno Nanos communicate with the Raspberry Pi over a Bluetooth serial channel, the cadence and force data is combined in a string. They must be separated by a buffer character so that the Raspberry Pi can differentiate the two. Since only the right pedal measures cadence, however, a time delay was introduced in the code of the left one so that the serial communication of the two pedals to the Pi would be synchronized.

Bluetooth communication

The use of BLE on the Raspberry Pi was facilitated using the `bluepy` module in Python 3. This module allows the main Python script to initiate a connection to both Bluno Nanos and receive the transmitted data. This was one of the crucial stepping stones to unifying the entire feedback system. The Bluno Nanos were then formatted so that the corresponding serial monitor would be transmitted via Bluetooth.

Data handling in the Raspberry Pi

The Raspberry Pi runs a Python script written to process, store, and display the data. Data from the left and right Bluno Nanos were received simultaneously via two parallelized processes defined within the Python script, allowing for the seamless transmission of data without any lost data due to clashes (this is also why the delay in the left Bluno Nano was necessary in sending data to the Raspberry Pi).

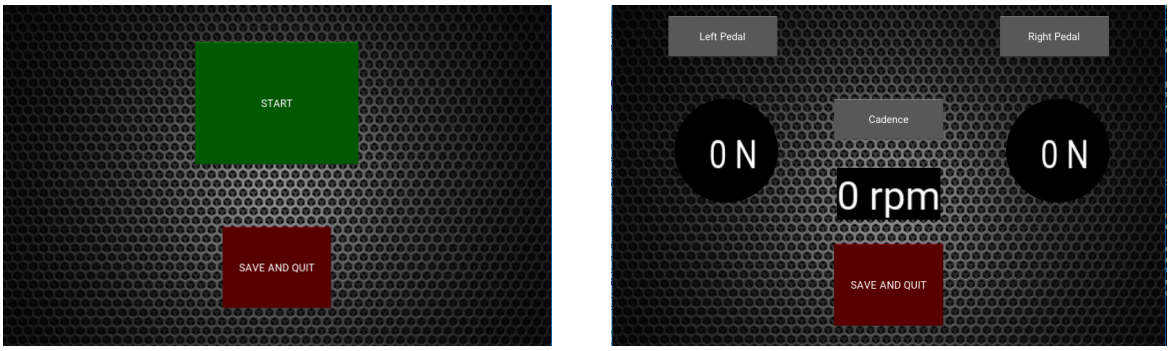
The Raspberry Pi sorts the data using buffer characters in the serial string. Once sorted, the data was prepared to be displayed and recorded on the Excel workbook. To format force readings into power phase readings, the time of each force reading was correlated with the period of the pedaling cycle, which was known from the cadence data.

Once this calculation was made, the angle of the pedal when the force reading was taken could be estimated from the position of the magnet on the bicycle frame. The force readings from the left pedal were assumed to be lagging by 180°, i.e. half a cycle compared to readings from the right pedal.

User interface and data storage

A user interface was created as part of the main python code using the `Kivy` module. This module allows for clean and user-friendly design as well as simple animations, making it ideal for the interface. Once started, the Raspberry Pi boots into its operating system and runs the main Python code, which displays the start screen in **Figure 9a**.

Upon pressing the start button, the Raspberry Pi connects to the Bluno Nanos and receives the data from the pedals. This data was processed as discussed in the above section, and was then displayed as in **Figure 9b**.



(a) The start screen of the feedback system.

(b) The live feedback during a session.

Figure 9: The two main screens shown to the user.

When the ride is complete, all the recorded data is saved by pressing the quit button. The user interface then returns to the start screen. Recorded data is saved in an Excel workbook and to a USB drive connected to the Raspberry Pi. This was facilitated using the `Xlsxwriter` module in Python 3. This module allows for the main Python script to record all the data received onto an Excel spreadsheet, and create graphs from this data automatically. Each workbook contains 2 spreadsheets, one for the cadence data and the other for the power phase data. **Figure 10** is an example of such a spreadsheet.

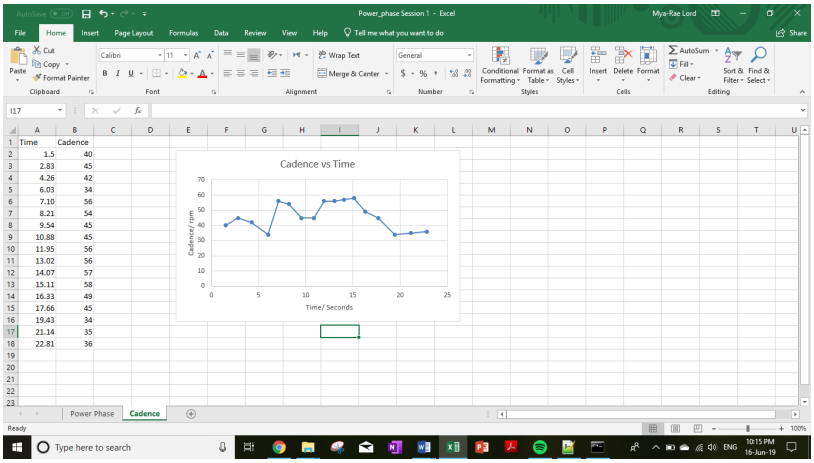


Figure 10: A sample of the Excel sheet produced by the Raspberry Pi using `Xlsxwriter`.

4 Testing and evaluation

The feedback system and physical pedal were evaluated separately to see if they met the technical requirements.

4.1 Feedback system

During the development process, the wireless communication system was continuously tested. Each part of the system was tested both individually and also in conjunction with the already existing parts as they were developed.
The parts of the system that were individually tested were:

1. The Bluno Nanos ability to send data via Bluetooth.
2. The ability to form a Bluetooth connection between individual Bluno Nanos and the Pi.
3. The ability to simultaneously connect both Bluno Nanos to the Raspberry Pi.
4. The Raspberry Pi's ability to receive data via Bluetooth.

The code written to test these and provide relevant feedback can be found in **Appendix I and J**.

During the development process, it was not uncommon for parts of the wireless communication system to fail these tests, but each failure provided valuable feedback that allowed for further improvement. The final product passed each of these tests, and the wireless communication system was in working order when the user tested the final product.

4.2 Sensors

4.2.1 Force-sensitive resistors

The accuracy of the force sensor was tested by applying a series of known forces, and comparing the voltage reading across the known resistor from both a Multimeter, and the serial port reading of the Bluno Nano. The force application was done with the aid of the column force tester that could hold a constant force.

The multimeter and Bluno Nano's voltage readings were plotted against each other, resulting in **Figure 11**.

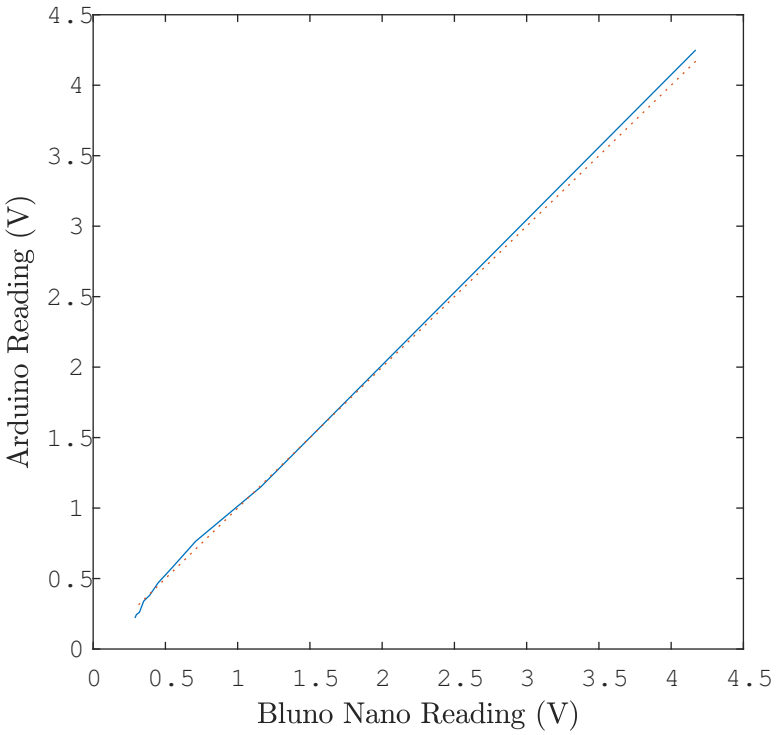


Figure 11: The graph of Bluno Nano voltage against multimeter voltage measurement

Comparing the data, represented by the blue line, to the dotted $y = x$ line suggests that the Bluno Nano readings are very accurate, i.e. they follow the multimeter readings closely.

4.2.2 Cadence

To test the maximum distance at which the Hall effect circuit can be consistently activated, a magnet suspended by an adjustable height pendulum was set up above the Hall effect sensor. As the pendulum swings, the magnets passed above the Hall effect sensor with a constant period of half a second.

The distance between the sensor and the magnet was measured and then the pendulum was set to oscillate freely. Activations were observed as spikes on the serial plotter in the Arduino software.

Spikes were only observed at distances closer than 1.0 cm, but the sensor was triggered inconsistently despite the pendulum oscillating freely with a constant period. Consistent activation of the Hall effect sensor was only observed at 0.5 cm.

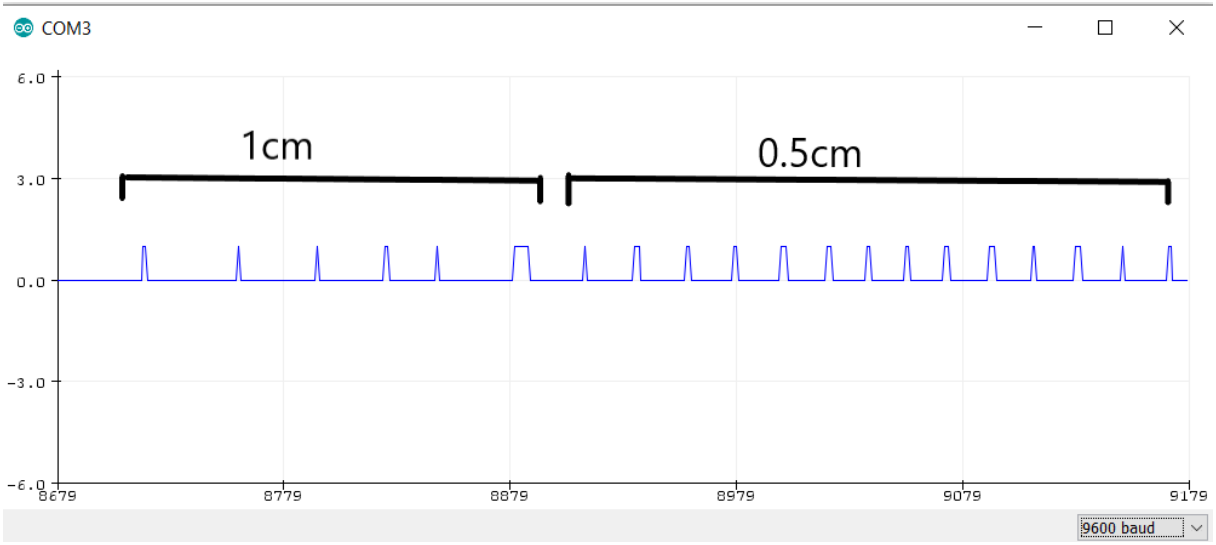


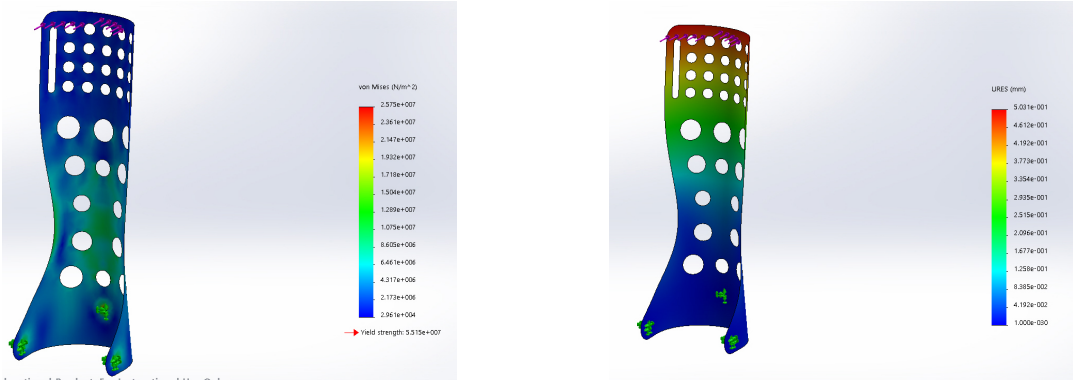
Figure 12: Capture from the serial plotter of distances of 1.0 cm and 0.5 cm. Note the differing spike patterns, despite identical pendulum motion.

4.3 Mechanical testing

4.3.1 Finite element analysis

Since holes were to be drilled into the calf support, it was analysed in SolidWorks using finite element analysis (FEA) to ensure it was not compromised by stress concentrations in the aluminium part. With the part initially being modeled in SolidWorks, FEA was easily performed on it using the following procedure:

1. Anchors were placed at the fixing holes in the bottom of the part.
2. A 50N load was placed at the top edge of the part, firstly in the lateral direction (side to side), then longitudinal (front to back).
3. The simulation was then meshed and run.

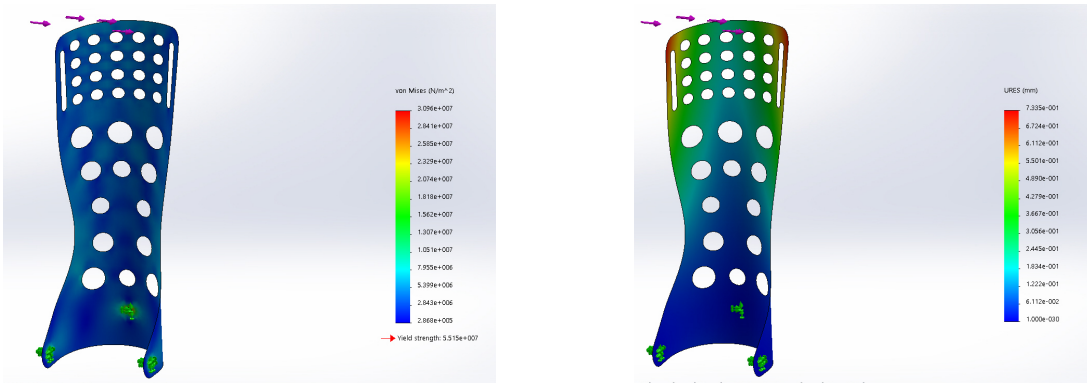


(a) Heatmap of stress.

(b) Heatmap of displacement.

Figure 13: FEA of stress and displacement under 50 N **lateral** force.

The lateral simulation shows a maximum stress of about 12MPa (**Figure 13a**), and a max displacement of 0.5mm (**Figure 13b**). This small deflection reflects the large thickness of material of 3mm, and large Youngs Modulus of the material of about 69GPa [ASM (1998)].



(a) Heatmap of stress.

(b) Heatmap of displacement.

Figure 14: FEA of stress and displacement under 50 N **longitudinal** force.

The longitudinal simulation shows a maximum stress of about 10MPa (**Figure 14a**) in the main body of the part and a maximum displacement of 0.7mm (**Figure 14b**), again reflective of the material thickness and elasticity.

4.3.2 Lateral force-displacement testing

The lateral stiffness requirement (**Table 2, 1.2**) is that the pedal should not allow more than 5mm of lateral movement of the lower leg in relation to the pedal under a 50N load. The procedure to test this was as follows:

1. A simple 45x45mm Rexroth^{TM} fixture was made to support the pedal.
2. The pedal was mounted in a Mecmesin force tester as in **Figure 15**. This configuration results in maximum lateral bending moment that could be applied by the user. A 0 1000N force sensor was used to give a 0.7N resolution.
3. A program was run that records the force needed to reach a 15mm displacement, with the force sensor just touching top of the calf support.
4. The results were exported to MATLAB for analysis, where the displacement was plotted against transverse force.

Figure 15: The setup used for the bend test with the Mecmesin force tester.

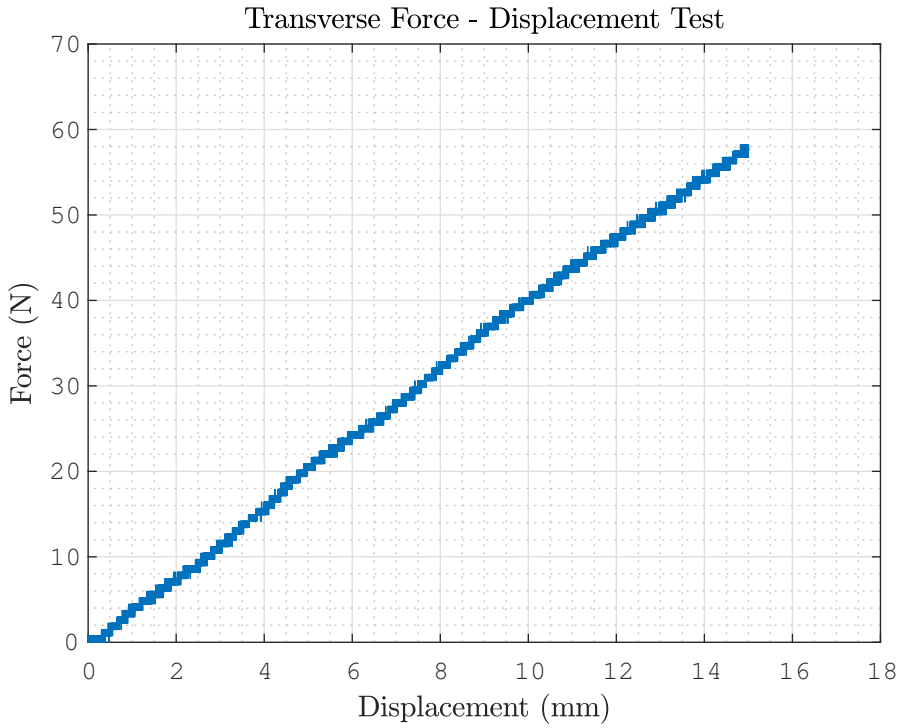


Figure 16: The results of the bending test - force (N) against displacement (mm).

Figure 16 demonstrates that at the target load of 50N, the pedal displacement is **12.2mm**, much larger than the desired 5mm maximum deflection. Whilst performing the test, however, it was observed was that much of the deflection was coming from torsional strain over the length of the snowboard binding, and not a moment over the calf support, which remained quite straight. This was verified by holding a straight rule along its length to gauge deflection.

4.4 Qualitative prototype review

During this phase of testing, the prototype pedals were fitted to the users assistive bicycle and were tested whilst on a rolling road. The ensuing feedback is a combination of the user's response to a questionnaire after testing (in quotation marks), and the team's own observations.

Pedal installation was smooth; the pedals screwed into the bikes cranks very easily. There were no fitment or clearance issues between the bike and the pedals.

The calf strap material was "too stretchy", and so did not prevent lateral deflection of the legs. The area the Velcro attached to was too small and kept coming undone. The user felt that ratchet type straps (as with the ankle binding) might be more suitable.

The aluminium calf support "was lightweight and looked grea", but did not restrict the movement of the lower leg enough, partly due of the stretch in the calf strap. The user reported feeling a "[slight lateral] mismatch prevent[ing] any long-term use." This was likely a result of the manufacturing method used.

The attachment of the calf support to the binding "was very secure, but some adjustability in the angle between calf support and binding would have been good."

Snowboard binding strapping – The user found that "everything with regards to the snowboard bindings worked fine and the strapping was very secure".

The adjustability of the snowboard binding in relation to the pedal axle – "The swivel action function built into the snowboard binding was useful and helped realign the rotational positioning of the foot. The adjustment of the pedal forwards and backwards was also very useful" The team noted it could have be further improved if the baseplate could be loosed and slid along, rather than having to be fully taken off the pedal for every adjustment.

Conclusion – "In my opinion, the cycling motion was smoother, although the calf support's [slight lateral] mismatch prevented any long-term use. The overall concept of the design was good, but would have been better if the calf support did indeed support the calf better."

4.5 Evaluation matrix

An evaluation matrix was created that compares the requirements outlined in **Table 2** to the results that were obtained. High-priority outcomes not achieved are highlighted in **red**.

Table 3: Evaluation matrix comparing the technical requirements to the outcome of testing.

No.	Requirement	Test results	Achieved?
1	**Functionality and Performance**		
1.1	Pedal must be able to transfer a force of 600N spindle without flexing more than 2 degrees in all directions (1200N with a safety factor of 2).	Not tested specifically, though SolidWorks FEA showed overly adequate safety factor.	Yes
1.2	Calf support must not allow the lower leg to move laterally more than 5mm in all transverse directions under a 50N load.	Force testing indicated deflection of **12.2 mm** under 50N load, corroborated by user.	No
1.3	The bindings must secure the foot to the pedal surface tightly, so that there is no more than 3mm of movement of the foot in relation to the pedal in all directions.	Not tested specifically, though bindings were reported to be firm but comfortable.	Yes
2	**Size and Weight**		
2.1	Electronic box embedded in the pedals must be no more than 10% of the total mass on the pedals. Small monitor must be no more than 0.3kg.	The electronic box weighs 155 g. The Raspberry Pi screen and case weighed 241 g in total.	Yes
2.2	Each pedal must be lighter than the existing pedal at 1.2 kg.	Each pedal weighs 1.618 kg.	No
2.3	The dimensions of the main pedal assembly should conform to **2.3.1 - 2.3.6** in **Table 2**	The pedals and bicycle were measured during user tests and fit all the requirements.	Yes
2.4	The pedal feedback sensors must be sized as to securely fit the pedals, that is, no more than 120mm wide and no more than 200mm long. These sensors must be as thin as possible, preferably less than 10mm. Small monitor to display real time data must be smaller than 100mm x 200mm.	The FSRs and puck assembly is 5mm thick. The monitor has dimensions of 178mm x 229 mm but could be comfortably mounted on to the handlebars.	No
3	**Usability & ergonomics**		

Table 3: Evaluation matrix comparing the technical requirements to the outcome of testing.

No.	Requirement	Test results	Achieved?
3.1	Water resistance - electronic systems must not be affected by light rain, and mechanical components should not rust easily from occasionally cleaning or light rain.	The box that houses the electronics was not manufactured to be waterproof. Endurance to rusting will only be known with long-term use.	No
3.2	As Cybathlon and various other competitions take place overseas, the pedals should be detachable or otherwise easily transported.	The pedals were successfully attached/detached several times to the bike	Yes
4	**Safety**		
4.1	The final product parts must not interfere with the braking system so that the user will be able to slow down and stop as before.	There remained large clearance between the brakes to the pedals when pedals were fitted during user tests.	
4.2	Electrical components (such as sensors) must be insulated so as to not come into contact with water.	The FSRs were laminated onto the pedal and were thus waterproof.	Yes
4.3	The pedals should not contain any sharp edges upon which the user could potentially hurt himself.	Any sharp edges were filed down and neoprene was used to line the aluminium calf support	Yes
5	**Life, reliability and Maintenance**		
5.1	Must at least last until Cybathlon 2020 and withstand at least 365 hours of sustained use (based on 1 hour of training every 2 days).	Requires long-term testing	Unknown
5.2	The materials chosen need a high fatigue strength, about 130 MPa.	Requires long-term testing	Unknown
5.3	Parts can be attached/detached separately and can be attached using screws or bolts. Parts must also be easily accessible in case maintenance is required.	All parts are fixed using screws and bolts and have been tested to be easily accessible for the users carer	Yes

5 Discussion

From the evaluation matrix, it is clear that the pedals meet the majority of the requirements, with a few exceptions. However, testing and evaluation have given rise to a number of recommendations as to how these can be remedied, and other functionality that could benefit the user:

Reducing lateral knee movement further – In tests with the user, it was found that the current strapping still allows too much lateral knee movement. This can be reduced in two ways:

- The straps need to be more rigid and have a larger contact area to securely fasten the calf.
- The calf support could encircle the leg more completely. However, it should be borne in mind that the users calf still needs to be inserted into the support, so the calf support cannot enclose too much.

Increasing the rigidity of the pedal assembly – From the results **Section 4.3.2** it is clear that the pedal assembly is not as rigid as required. However, this appears to to be mostly due to a lack of torsional rigidity of the snowboard binding, rather than flexibility in the calf support. This is supported by the results of the FEA, particularly **Figure 13b**. This design flaw can be overcome if the calf support was directly connected to the baseplate instead of the snowboard binding, so that any force placed on the calf support is not transmitted through the snowboard binding.

Reducing the weight of the pedal – Currently, the entire pedal weighs 1618 g, despite the holes which were introduced in the calf support to reduce the weight. The calf support, or even the entire support and base, could be made from carbon fibre, which would make it significantly lighter without compromising on stiffness.

Even if carbon fibre cannot be used, the FEA in **Section 4.3.1** suggests that the calf support well exceeds the current strength and rigidity requirements. Therefore, the calf support could be made thinner, without compromising on an adequate safety factor.

Improving the ergonomics of the calf support – The calf support is lined with a thick layer of neoprene that is glued on. While more comfortable than raw aluminium, the user found it to not be very breathable. A soft foam that is detachable, washable and breathable would be preferred. This could also be contoured for an even better fit. The design of night splints could

be adapted for further improvements - they are fitted on the same parts of the calf and foot, and provide a simple solution to issues frequently associated with bracing options.

Figure 17: Two night splint designs that could improve comfort of the calf support. [ssur UK (2019)]

Comparing previous force measurements more easily – While the user has access to previous data of various sessions, he still needs to compare them himself to see if there has been improvement to his performance over time. It would be helpful if the Raspberry Pi made a graph displaying the performance over time, such as average force or cadence. This would make it much easier to Johnny to visualise trends and greatly improves the user experience.

Use multiple and stronger magnets to improve cadence measurements – Using a stronger magnet allows more distance between the bike frame and the pedal, ensuring the cadence sensor will be functional even if the user experiments with different crank lengths. Furthermore, in the current design it is assumed that the angular velocity of each pedal stroke is uniform. This is not *necessarily* true, and having more magnets along the circular travel of the pedal gives a more accurate pedal position. This would improve the accuracy of the cadence measurement because the average of the two recordings could be taken.

Waterproof electrical component housing – The current laser-cut box was not tested for any waterproofing standards, as the pedal system was designed for indoor use only. To allow Johnny to use the electronics in outdoor environments, it should be waterproofed from light rain and puddles. This corresponds to an IP rating of at least IPX4 - protection from water sprayed from all directions [MPL (2019)].

Integration with the FES control system – Johnny currently controls his FES parameters manually to manage the fatigue in his legs while cycling. It may be possible to integrate the user feedback system as a closed-looped control for the FES stimulator. This appears to be a

reasonably common strategy in the FES cycling event, and is explicitly permitted within the Cybathlon regulations [Zurich (2019)], Clause 3.2.2: *The FES stimulators may apply closed-loop control strategies using sensors applied to the pilots or the bike. It is also allowed to manually trigger the stimulator.*

6 Conclusion

The new pedal design and feedback system adequately addresses the issues of the current pedal setup that were raised by the user. The feedback system provides accurate measurements of cadence as well as force, and the user interface that goes with it is very intuitive and allows for use with limited arm and finger mobility. The improved design of the pedal base was largely a success, as it allows the user to find the ideal foot positioning to deliver the maximum force through the pedals. In addition, the pedal design adheres to the Cybathlon regulations, as the electronic components can be completely removed for the race.

Whilst the electrical aspects of the project have met all of the aims set out at the beginning of the project, the mechanical design was not able to provide the desired stabilisation, and a significant reduction in weight was not made. However, evaluation and testing has identified various practical improvements that could be made to address these issues. The possibility of using of carbon fibre to improve the mechanical rigidity, ergonomics and weight of the pedal proved particularly promising, but was not possible to implement in this iteration due to time and resource constraints (the mould that would be needed proved too difficult to manufacture).

It is likely that further iterations of the design with more resources and time would produce a design that Johnny could use in his training. Furthermore, it is also possible that he may manage to find a particular configuration that is considerably more comfortable and efficient than any other. In this case, the adjustability built in to the design may no longer become necessary, and a bespoke pedal with streamlined ergonomics and minimal weight can be achieved. In this sense, the adjustable pedal that the team has designed will always be a prelude to a more personalised product that will truly optimise Johnny's perfomance in Cybathlon 2020 and beyond.

References

ASM (1998), *Metals Handbook Desk Edition*, 2nd edn.

ElectronicsTutorials (2019), 'Hall effect sensor'.
URL: *https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html*

Fonda, B. & Sarabon, N. (2010), 'Biomechanics of cycling: literature review', *Sport Sci Rev* **19**, 131–163.

MPL (2019), 'Ip ratings ingress protection'.
URL: *https://www.mpl.ch/info/IPratings.html*

Physiopedia (2019), 'Cycling biomechanics — physiopedia,'. [Online; accessed 14-June-2019].
URL: *https://www.physio-pedia.com/index.php?title=Cycling_Biomechanics&oldid=206615*

Rebecca Martin, OTR/L, O., Cristina Sadowsky, M., Kimberly Obst, OTR/L, M., Brooke Meyer, PT, D. & John McDonald, MD, P. (2012), 'Functional electrical stimulation in spinal cord injury: From theory to practice', *Top Spinal Cord Inj Rehabil.* .

Telli, R., Seminati, E., Pavei, G. & Minetti, A. E. (2017), 'Recumbent vs. upright bicycles: 3d trajectory of body centre of mass, limb mechanical work, and operative range of propulsive muscles.', *J Sports Sci.* .

Zurich, C. E. (2019), *Race Task Description Cybathlon 2020*, v 2019 03-22 edn, ETH Zurich.

ssur UK (2019), 'Night splints'.
URL: *https://www.ossur.co.uk/injury-solutions/products/foot-and-ankle/night-splints*

Appendix

A Project management

While the design itself was organized by separation of mechanical and electrical components, the team itself was only loosely organized along these lines. Team members also performed a variety of other tasks to ensure completion of the project. The following is a summary of the roles and responsibilities undertaken by each team member:

Nicholas Ustaran-Anderegg

- Project manager
- Organisation of regular team meetings
- Management of individual tasks
- Liaison between team and user
- Programmer for wireless communication by Bluno Nanos and data processing software on Raspberry Pi
- Developer for user interface
- Manufacture of complete pedal

William Paterson

- Head of manufacture
- Design and manufacture of baseplate and angle brackets
- CAD design, FEA and manufacture of calf support
- Manufacture of bend test fixture and bend testing

Samuel Yeadon

- Procurement manager - budget and acquisitions management
- Liaison with Bioeng orders
- Selection of materials
- Aided in manufacture of various components

Vivien Alves Passing

- Lead presentation designer
- Co-lead poster designer
- Research and design of calf straps
- Research, design and manufacture of calf support lining
- Proofreading of final report

Nathanael Tan

- Research into biomechanics
- Electronic design of Hall effect circuit
- Design and manufacture of calf straps
- Design and manufacture of peripheral components (Pi mount, electronics caseworks)
- Lead poster designer; creation of layout and visual elements of poster
- Report lead for LaTeXformatting and compilation

Mya-Rae Lord

- Lead programmer for the data processing software on the Raspberry Pi
- Wireless communication software development
- User interface development
- Testing of the FSRs

Nestoras Neofytou

- Research on force measuring devices and selection of FSR
- Design and implementation of the FSRs
- Arduino programming

- Set-up of serial BLE connection

- Overall design and performance testing of the feedback system

Samuel Karet

- Creation of Gantt Chart

- Secondary programmer for wireless protocols

- Secondary programmer for microcontrollers

- Preparatory formatting of data for wireless communication

- Performance testing for software, communication and feedback system

Livia Robic

- Worked on Electronic branch of the design

- Research on circuit design for force measuring devices

- Implementation of FSRs and BLE Bluno Nanos

- Power phase animation

- Secondary Python programmer

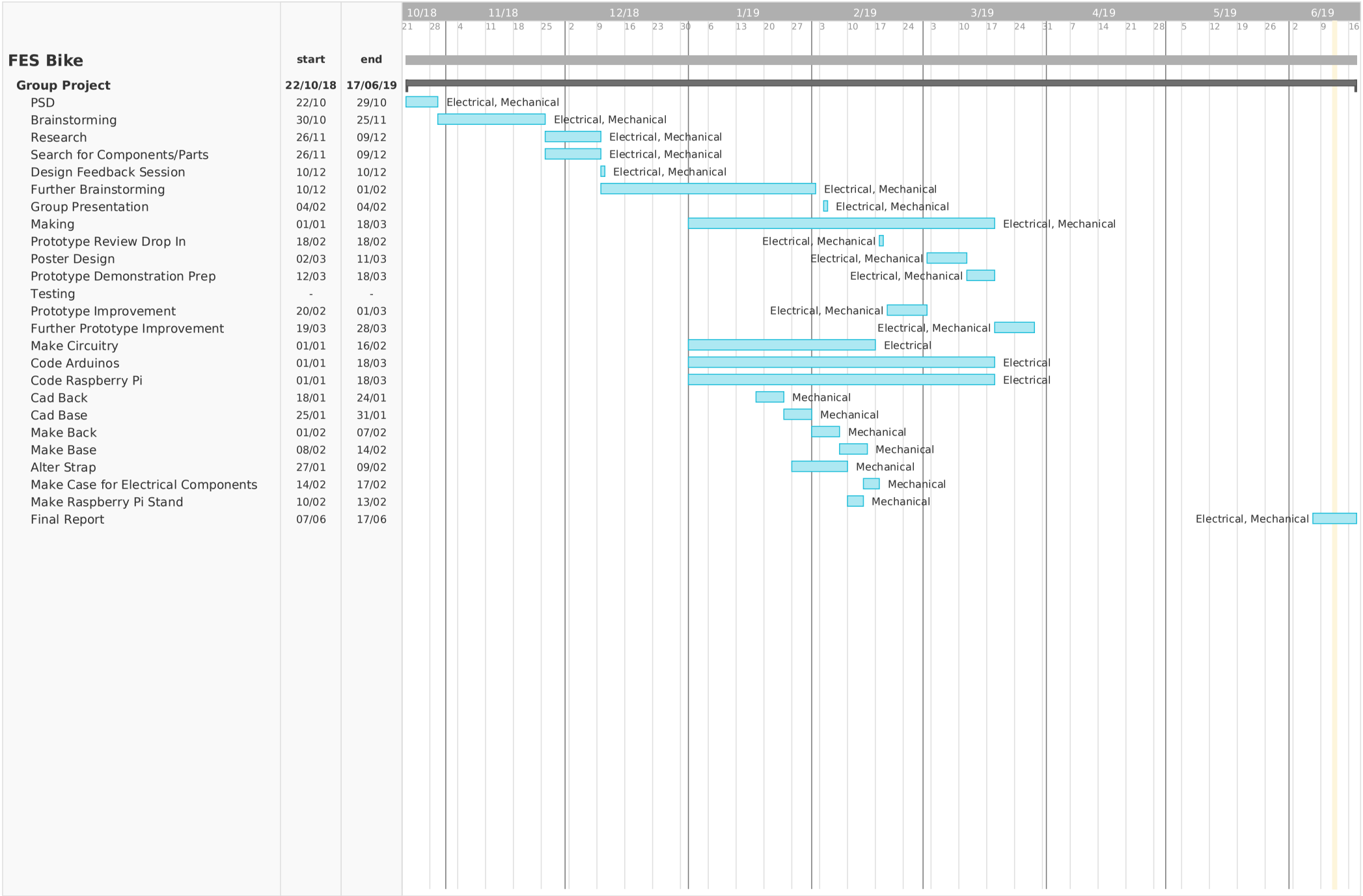
Mohammed Shahr Abadi

- Primary development and implementation of Hall effect cadence sensor

- Secondary Arduino programmer

A Gantt chart was also developed to keep track of project deliverables, deadlines and internal milestones:

Printed: 12/06/2019



FES Bike	start	end
Group Project	**22/10/18**	**17/06/19**
PSD	22/10	29/10
Brainstorming	30/10	25/11
Research	26/11	09/12
Search for Components/Parts	26/11	09/12
Design Feedback Session	10/12	10/12
Further Brainstorming	10/12	01/02
Group Presentation	04/02	04/02
Making	01/01	18/03
Prototype Review Drop In	18/02	18/02
Poster Design	02/03	11/03
Prototype Demonstration Prep	12/03	18/03
Testing	-	-
Prototype Improvement	20/02	01/03
Further Prototype Improvement	19/03	28/03
Make Circuitry	01/01	16/02
Code Arduinos	01/01	18/03
Code Raspberry Pi	01/01	18/03
Cad Back	18/01	24/01
Cad Base	25/01	31/01
Make Back	01/02	07/02
Make Base	08/02	14/02
Alter Strap	27/01	09/02
Make Case for Electrical Components	14/02	17/02
Make Raspberry Pi Stand	10/02	13/02
Final Report	07/06	17/06

B Risk management

A risk analysis using the **Risk Prevention Number** (RPN) matrix was performed in order to identify potential risks to the user before and after mitigation. The most important considerations to the risk were:

1. The user's physical condition. In addition to his paraplegia, Johnny will also have his hands and legs strapped into the bike.
2. The typical setting and conditions of use. Johnny primarily trains on a roller at his residence, but occasionally takes the unit to cycle around his neighbourhood.
3. The pedals will also be used in a competitive setting, where Johnny has reported he will be using the bike for extended periods of time for technical checks in addition to the actual races.
4. The specificity of our user, i.e. the fact that Johnny Beer-Timms is the only user the pedal is designed for.

While points **1, 2**, and **3** are considered risk magnifying factors, point **4** is a major mitigating factor, in that countermeasures can be tailored specifically for him, and warnings and advice on safe use can largely be relied on to be heeded.
The team therefore continues to recommend an RPN threshold of **75**. Note also that the risk analysis only accounts for failures involving the pedals, and not failure of the bike and FES system as a whole as these are not within the purview of the project. Numbers in red and green represent ratings before and after mitigation respectively.

Assembly	Failure & Effect	Preventive measures	S	O	D	**RPN**
Calf strap	Over tightening may cause pressure sores and chafing.	When tightening visually check straps are not too tight.	2 2	8 5	8 4	**128** **40**
	Insufficient tightening causing too much movement, resulting in gashes and fractures from impact with the bicycle during motion.	When fastening, check by for looseness by running finger along inside of strap.	5 5	8 4	8 2	**320** **40**
Base binding straps	Over tightening may cause pressure sores and chafing.	Mark points on ratchet strap with suitable pressure to ensure consistent, appropriate tightening.	2 2	7 2	8 1	**112** **4**
Sharp edges on calf support	Sharp edges could lead to pressure sores and cuts.	All edges are filed and the calf support is lined with neoprene to ensure all edges are blunt and padded.	2 2	7 2	8 1	**112** **4**
Electrical circuits	Possibility of mild electrical shock from water contact with circuit.	Circuit enclosed in containers so more resistant to water and operating at low voltages so shock will be minimal.	1 -	2 -	5 -	**10** **-**
	Electrical components could overheat, causing burns to user.	There is no direct contact between the electrical components and the user.	4 1	3 2	2 2	**24** **4**

C Ethics

The first set of considerations pertain to adherence to the Cybathlon regulations. As discussed in **Section 2, Requirements**, the team has ensured that the design and capabilities of the pedal are in agreement with the rule *and* the spirit of the Cybathlon event.

The second and arguably more pertinent ethical matters pertain to the user himself, Johnny-Beer Timms. Johnny has been very vocal about his desire to improve and takes his training very seriously. The team has therefore taken care to manage his expectations with regard to what is possible and achievable in the time span of the EDP.

Another aspect that needs to be taken into account is confidentiality. Since Johnny will compete in the Cybathlon against other athletes, information on his performance, training and equipment should remain confidential within the project group. Hence, the team ensured that Johnny will not be at a disadvantage when it comes to the racing day because the competitors have gathered information on his bike or training sessions, for example by giving data to third parties.

D Bill of materials

Table 4: Bill of materials.

Component	**Price ()**
Aluminium sheet	42.41
Aluminium angle & smaller plate	48.41
Clear Perspex acrylic	7.49
Neoprene foam	23.17
Magnet	5.11
GorillaPod	32.00
(2X) ITB strap	29.70
32 GB SD card	8.40
Nylon strap	5.69
Super slim power bank	10.99
Raspberry Pi 3	34.11
Raspberry Pi 7-inch touch screen	61.72
2pcs ADS1115 16 Byte 4 Channel IIC Analog-to-Digital ADC PGA Converter	6.99
9v batteries	8.99
Velcro strap	10.39
(2X) Bluno Nano Arduino Module with BLE	51.48
Snowboard Bindings	105.00
Force sensitive resistors	104.00
Cycle Pedals	36.49
Use of water-jet to cut aluminium back plate	100.00
TOTAL	**732.54**

E Manufacture

Appropriate details regarding manufacture of the final design have been discussed in **Section 2.1, Hardware**. Further technical drawings & CAD renderings are provided in this section.

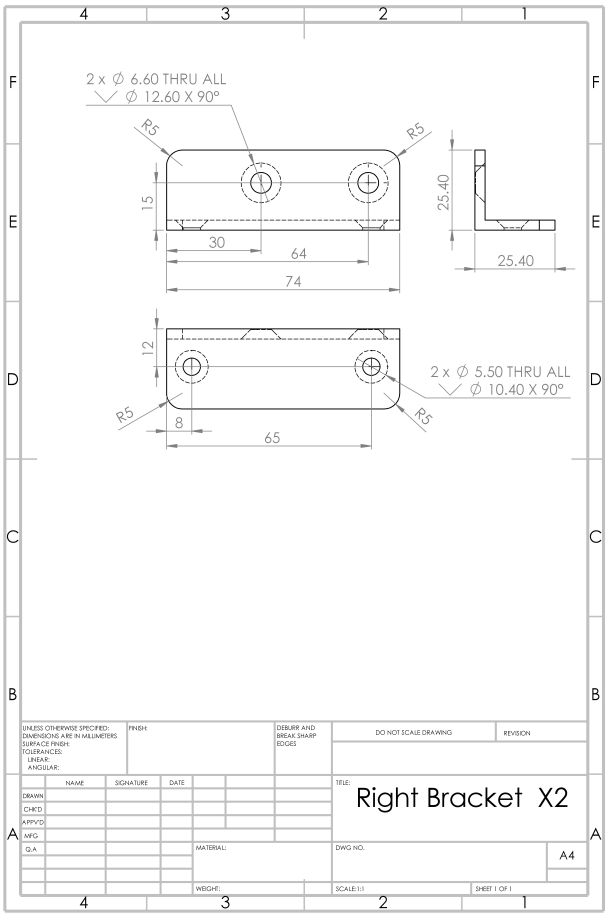


Figure 18: Technical drawing of angle bracket for mounting to pedal.

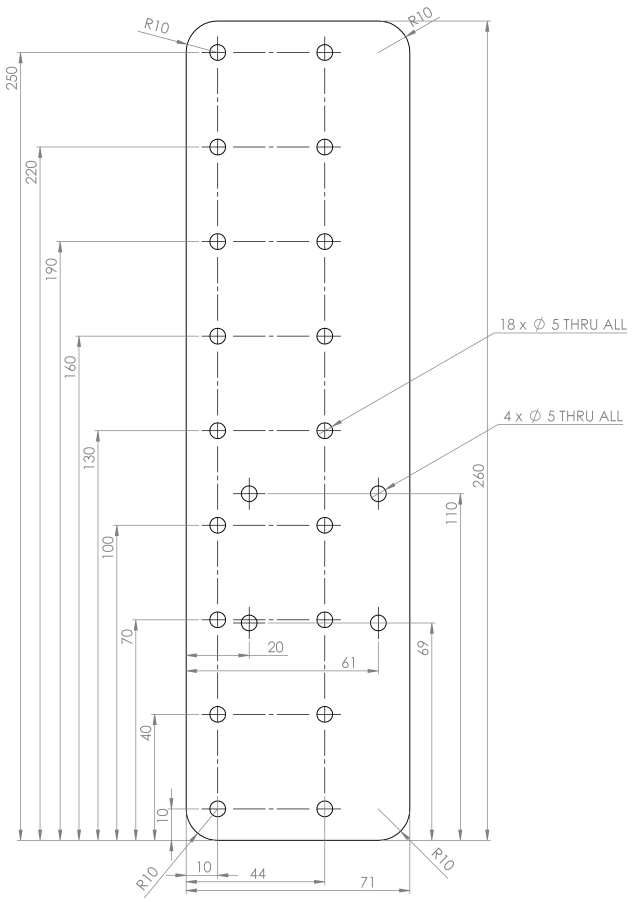


Figure 19: Technical drawing of base plate with dimensions for manufacture.

Figure 20: CAD model of calf support designed in SolidWorks.

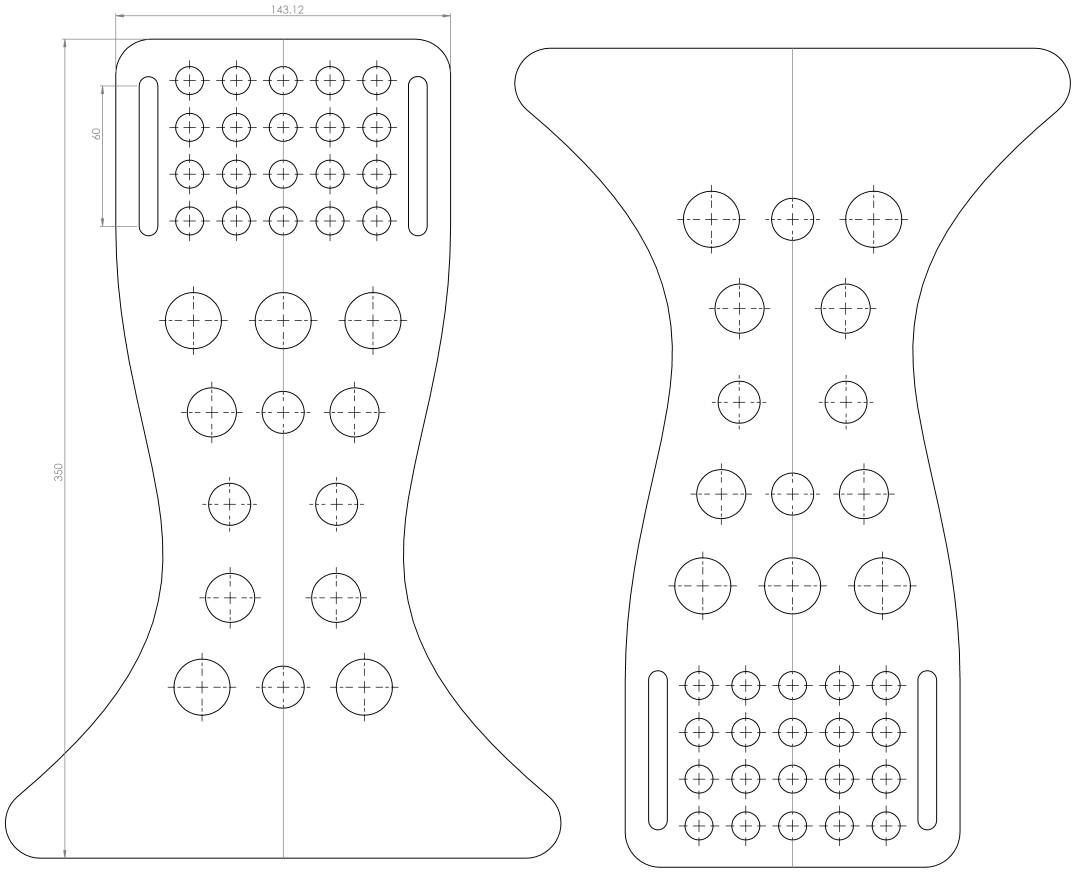


Figure 21: Technical drawing of ‘unfolded’ calf support for water-jet cutting.

F Hall effect and cadence sensor programming

The Hall effect sensor consists of a rectangular p-type semiconductor with a current flowing through it. When it enters a magnetic field, a force is exerted on the charge carriers in the semiconductor, deflecting them to the other side. This build up of charge carriers produces a potential difference. This particular Hall effect sensor is latching, which means it goes HIGH for the north pole of a magnet and LOW for the south. It is desired, however, for it to produce a single voltage pulse whenever the north pole passes it. This is known as non-latching behaviour, and was mas possible in the code by turning the power to the sensor on and off during every cycle, thereby resetting the sensor.

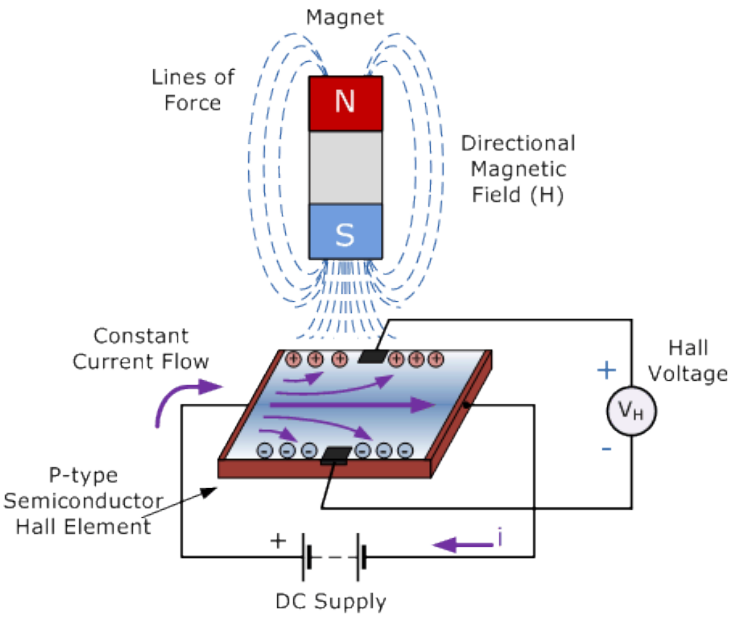


Figure 22: Demonstration of the Hall effect. [ElectronicsTutorials (2019)]

G FSR calibration

In order for the FSRs to be used to measure unknown forces, their force-conductance relationship must first be known. To do this, a series of known forces were applied to the FSR and the resulting conductance was measured. Plotting the force against conductance produced the relation shown in **Figure ??**, which was encoded in the microcontrollers to enable them to process resistance to force applied to the FSRs.

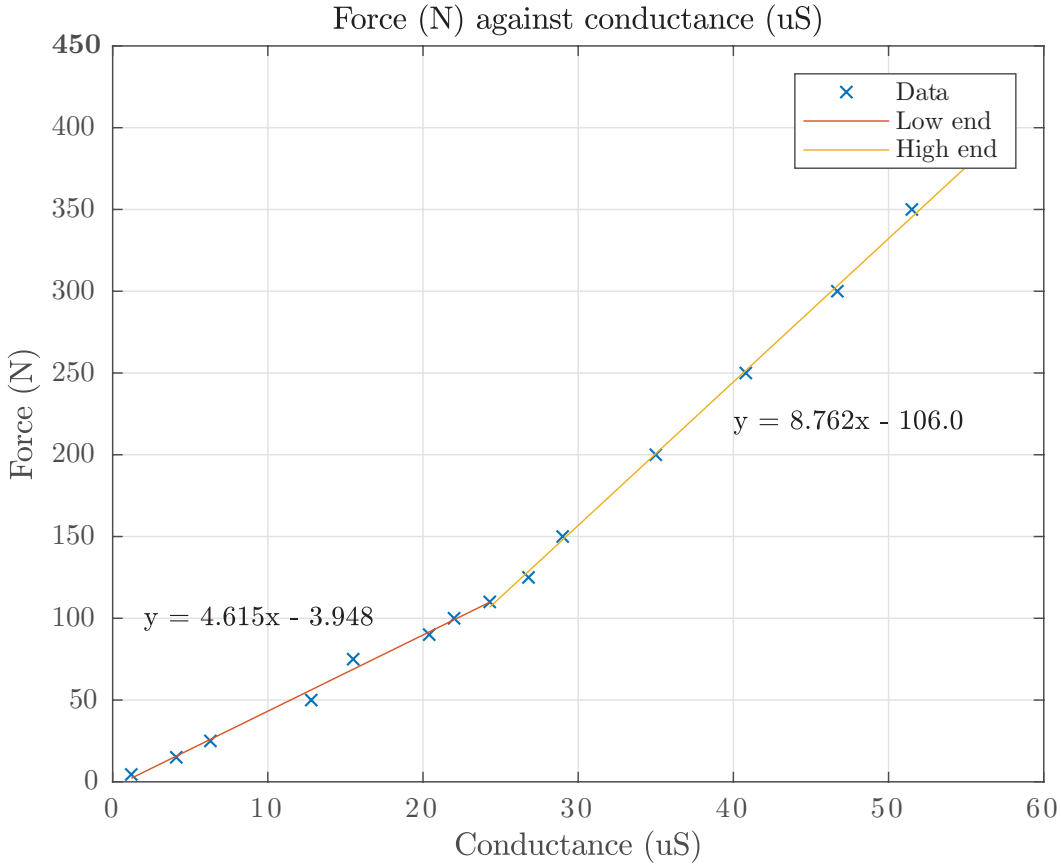


Figure 23: Graph of force against conductance for the calibration experiment

H Microcontroller circuit diagram

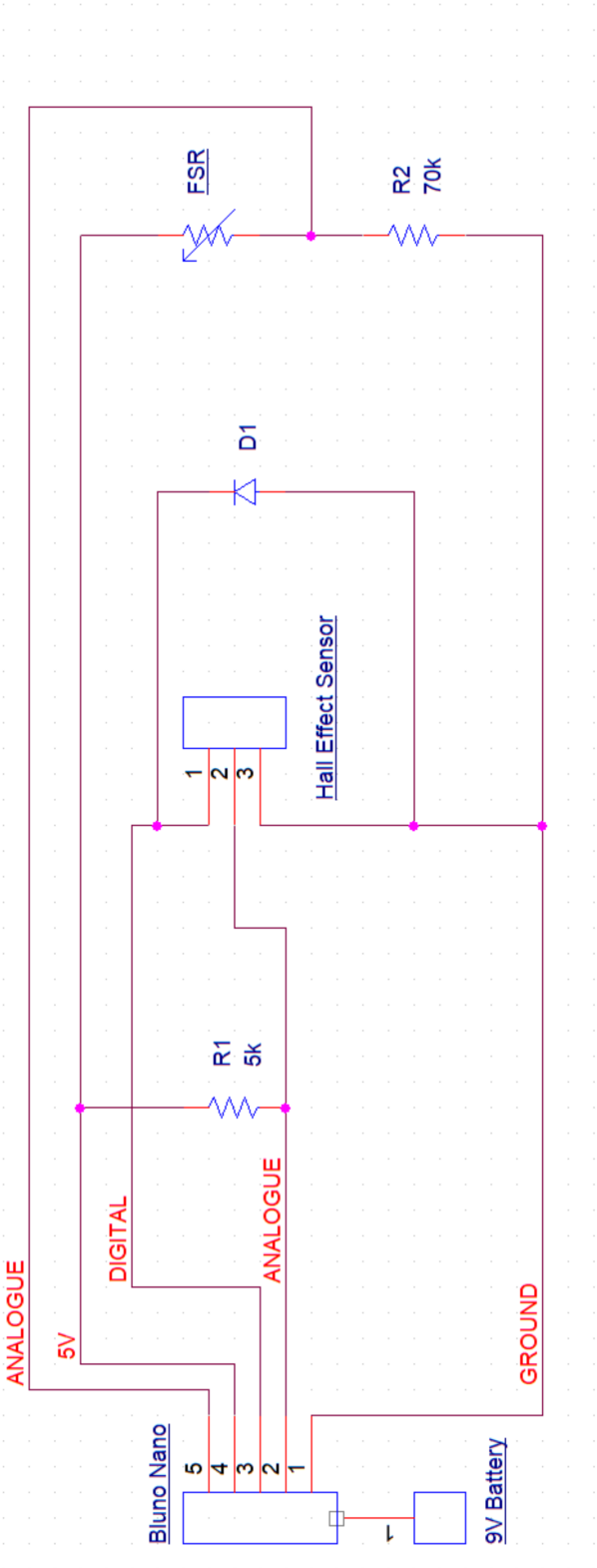


Figure 24: The circuit used to connect the sensors to the right pedal. The left pedal is identical, with the omission of the cadence sensor.

Figure 25 below represents these connections, and those made with the central processor. Data connections represented by bold pointed arrows, while data connects are represented by the rounded, dashed arrows.

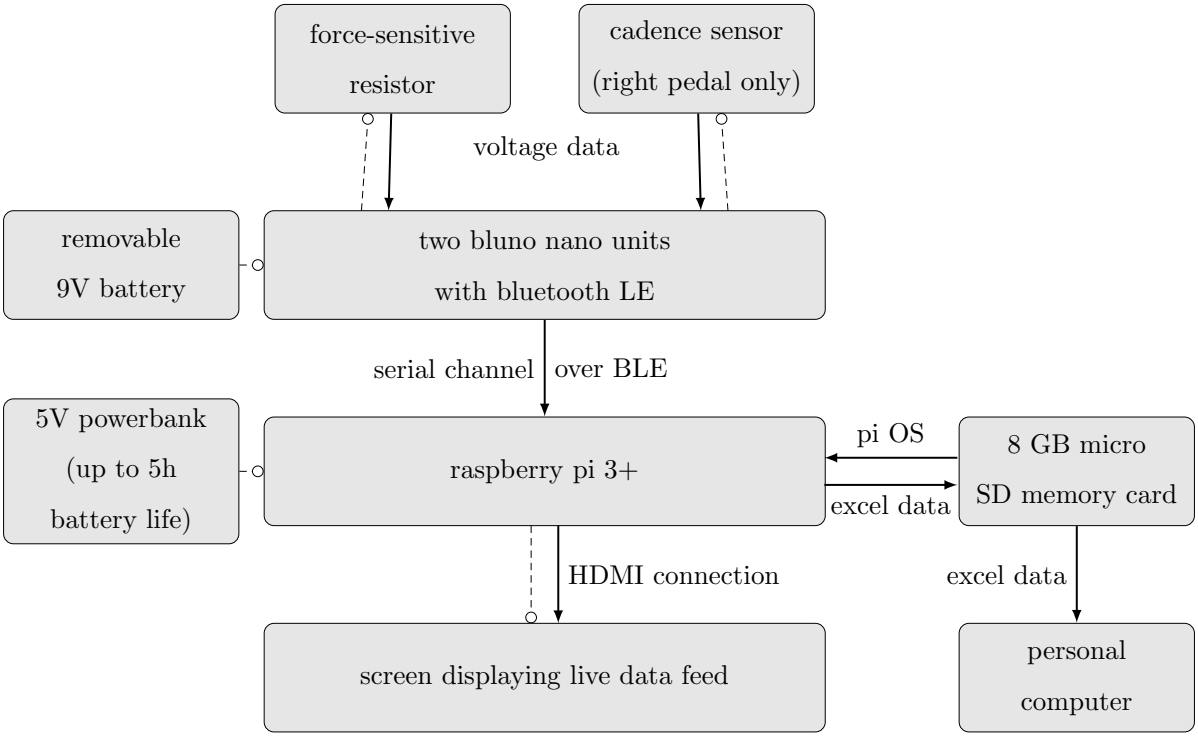


Figure 25: A flowchart of data and power connections in the feedback system.

I Microcontroller codes

Listing 1: Left microcontroller code

```
#define ARRAY_SIZE 3
//The number of values to keep in the running average
#define TIMEOUT 1500
//Max time without input before turning off LEDs
#define FADE_INCREMENT .1
//Controls the fading speed:adjust with trial & error
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(10, 11); // RX | TX
//Pin Declarations (Setup may require editing)
  const int fsr1pin=A1;
  const int fsr2pin=A2;
  const int magnetSensor = A5;
  const int ledPin = 14; //pin D11
  const int mosSwitch = 9;  //pin D6
//Global Variables
  float Force1;
  float Force2; //final value variable
  float Total;
  //Cadence counter variables
  float times[ARRAY_SIZE];
  //array to hold values for the running average
  bool magnetOn = false;
  bool prevMagOn = false;
  bool primed = false;
  bool arrayEmpty = false;
  float startTime = millis();
  int rpm = 0;
  bool pedaling = false;
  //zero values for FSRs
 float F_zero1;
 float F_zero2;
```

```
//Function Declarations
  float f_fsr(int, float);
  void updateTimes(float);
  int avgArray(float);
  void clearTimes();
  void checkFullArray();
  void f_rpm();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200); //start the serial
  F_zero1= f_fsr(fsr1pin,70000);
  F_zero2= f_fsr(fsr2pin,70000);
  randomSeed(analogRead(A3));
  float Voltage[25];
  for(int i = 0; i < 25; i++){
    Voltage[i] = 0;
  }
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}
void loop() {
  // put your main code here, to run repeatedly:
    Force1 = f_fsr(fsr1pin,70000)-F_zero1;
    Force2 = f_fsr(fsr2pin,70000)-F_zero2;
    Total=Force1+Force2;
  //Concatenates start and end strings with data
    String fString = String(Total);
    String fR_Pi=String("LFstart"+fString+"LFend");
    Serial.print(fR_Pi);
    //f_rpm();
delay(52);
}
//Force finding function
```

```
float f_fsr(int in_pin,float R_ref){
  int fsrPin = in_pin;
  // the FSR and 10K pulldown are connected to a0
  int fsrReading;
  // the analog reading from the FSR resistor divider
  int refVoltage;
  /* the analog reading converted to voltage
  (voltage drop across reference resistor) */
  int fsrVoltage[24];     //voltage drop across fsr
  float fsrI;
  float fsrResistance;
  /* The voltage converted to resistance,
  can be very big so make "long" */
  float fsrReference = R_ref; //reference resistor
  double fsrConductance;
  float fsrForce;
  // Finally, the resistance converted to force
  float voltageErr;


  int i=0;
  int useful=0;
  float AvgVoltage=0;//fsr avg voltage

  for(i=0;i<24;i=i+1);
  {
    fsrReading=analogRead(fsrPin);
    //analog reading from port: in_pin
    refVoltage = map(fsrReading, 0, 1023, 0, 5000);
    //map the reading to corresponding voltage value
    delay(2);

    fsrVoltage[i] = 5000 - refVoltage;
    // fsrVoltage is in millivolts so 5V = 5000mV
```

```
  //error correction
  voltageErr = (0.063*fsrVoltage[i]);
  fsrVoltage[i] = fsrVoltage[i] - voltageErr;
  ;
  if (fsrVoltage[i]<4750 && fsrVoltage[i]>200)
  {
    useful=useful+1;
    AvgVoltage=AvgVoltage+fsrVoltage[i]/24;
  }
}
AvgVoltage=(AvgVoltage*25)/useful;
refVoltage=5000-AvgVoltage;
 /* The voltage = Vcc * R/(R + R_FSR) where
 R = 1M2 and Vcc = 5V(from arduino) so
 R_FSR = ((Vcc - V) * R) / V where R_fsr is
 the resistance of the sensor(infinity to
 300kohms when fully pressed */


  fsrI = refVoltage / fsrReference;
  /* 1M2 resistor but actual value measured with
  multimeter is on input r_ref */
  fsrResistance = AvgVoltage / fsrI;

  fsrConductance = 1000000/ fsrResistance;
  /* conductance is on *10^-6 scale so multiply by
  10^6 to see values
  convert to Force
  using the relation out of experiment */
  if (fsrConductance<1210)
    {fsrForce=0;}
  else if (1210<=fsrConductance<=24315)
    {fsrForce=4.615*fsrConductance*0.001-3.963;}
```

```
    else if (fsrConductance>24315)
    {
      fsrForce=8.702*fsrConductance-10.31;
      fsrForce=fsrForce/1000;
    }
    //for testing
/*
    Serial.print("voltage ");
    Serial.println(AvgVoltage);
    Serial.print("resistance ");
    Serial.println(fsrResistance);
    Serial.print("Conductance ");
    Serial.println(fsrConductance);
    Serial.print("Force ");
    Serial.println(fsrForce);
    Serial.println("---------------------");
 */
    return(fsrForce);
}
//add a new value to the array, moving the rest back
one space and removing the oldest
void updateTimes(float newValue)
{
  for(int i = 0; i < ARRAY_SIZE-1; i++)  {
    times[i] = times[i+1];
  }
  times[ARRAY_SIZE-1] = newValue;
}
//returns an average of the values in the array
int avgArray(float values[])  {
  int total = 0;
  int counted = ARRAY_SIZE;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
    total = total + values[i];
```

```
    if(values[i] == 0)
      counted--;
  }
  return(total/counted);
}
//clear the array
void clearTimes()  {
  for(int i = 0; i < ARRAY_SIZE; i++)
    times[i] = 0;
}
//check if the array is full
void checkFullArray()  {
  arrayEmpty = true;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
    if(times[i] != 0)
      arrayEmpty = false;
  }
}
void f_rpm(){

    //Serial.println("test");
    digitalWrite(mosSwitch,HIGH);
    delay(50); // prepare next data
    digitalWrite(mosSwitch,LOW);

    //Read from the hall effect sensor(analog values)
    int magnetState = digitalRead(magnetSensor);
    // Serial.println(magnetState);

    if(magnetState == HIGH)  {
      magnetOn = false;
    }

    else  {
```

```
      magnetOn = true;
    }

    if(!magnetOn && prevMagOn)  {
      primed = true;
    }

    //timeout
    if(millis()-startTime > TIMEOUT) {
      clearTimes();
      pedaling = false;
    }

    //if(magnetOn &&!prevMagOn)  {
    //if magnet passes sensor once  && primed
    float currentTime = millis();
    float changeTime = (currentTime - startTime);
    //record the time since the last pedal
    startTime = millis();

    if(pedaling){
    //if there's been pedaling since the last timeout
      updateTimes(changeTime);
      //add the time to the running average array
      //Concatenates start and end strings with data
      String cadString = String(rpm);
String cadR_Pi=String("cadStart"+cadString+"cadEnd");
      //Sends data
      Serial.print(cadR_Pi);
    }

    primed = false;
    pedaling = true;
    //}
```

```
    prevMagOn = magnetOn;
    checkFullArray();
    float gap = avgArray(times);
    if(arrayEmpty)  {
      rpm = 0;
    }
    else{
      rpm = 60000/gap;
      //turn millisecond gap value into rpm
    }
    /* light the lowest red LED when the first pedal
    stroke is recorded, since there isn't enough data
    to calculate rpm */
    if(pedaling && rpm == 0){
      rpm = 0;
    }
    return;
}
```

Listing 2: Right microcontroller code

```
#define ARRAY_SIZE 3
//The number of values to keep in the running average
#define TIMEOUT 1500
//Max time without input before turning off LEDs
#define FADE_INCREMENT .1
//Controls the fading speed:adjust with trial & error
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(10, 11); // RX | TX
//Pin Declarations (Setup may dictate editing)
  const int fsr1pin = A1;
  const int fsr2pin = A2;
  const int magnetSensor = A5;
```

```
  const int ledPin = 14; //pin D11
  const int mosSwitch = 9; //pin D6
//Global Variables
  float Force1;
  float Force2; //final value variable
  float Total;
  //Cadence counter variables
  float times[ARRAY_SIZE];
  //array to hold values for the running average
  bool magnetOn = false;
  bool prevMagOn = false;
  bool primed = false;
  bool arrayEmpty = false;
  float startTime = millis();
  int rpm = 0;
  bool pedaling = false;
  //zero values for FSRs
 float F_zero1;
 float F_zero2;
//Function Declarations
  float f_fsr(int, float);
  void updateTimes(float);
  int avgArray(float);
  void clearTimes();
  void checkFullArray();
  void f_rpm();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200); //start the serial
  F_zero1= f_fsr(fsr1pin,70000);
  F_zero2= f_fsr(fsr2pin,70000);
  randomSeed(analogRead(A3));
  float Voltage[25];
```

```
  for(int i = 0; i < 25; i++){
    Voltage[i] = 0;
  }
  pinMode(A5, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}
void loop() {
  // put your main code here, to run repeatedly:
    Force1 = f_fsr(fsr1pin,70000)-F_zero1;
    Force2 = f_fsr(fsr2pin,70000)-F_zero2;
    Total=Force1+Force2;
  //Concatenates start and end strings with data
    String fString = String(Total);
    String fR_Pi = String("RFstart"+fString+"RFend");
    Serial.print(fR_Pi);
    f_rpm();
delay(2);
}
//Force finding function
float f_fsr(int in_pin,float R_ref){
  int fsrPin = in_pin;
  // the FSR and 10K pulldown are connected to a0
  int fsrReading;
  // the analog reading from the FSR resistor divider
  int refVoltage;
  /* the analog reading converted to voltage
  (voltage drop across reference resistor)*/
  int fsrVoltage[24];     //voltage drop across fsr
  float fsrI;
  float fsrResistance;
  /* The voltage converted to resistance, can be very
  big so make "long"*/
  float fsrReference = R_ref; //reference resistor
```

```
double fsrConductance;
float fsrForce;
// Finally, the resistance converted to force
float voltageErr;


int i=0;
int useful=0;
float AvgVoltage=0;//fsr avg voltage

for(i=0;i<24;i=i+1);
{
  fsrReading=analogRead(fsrPin);
  //analog reading from port: in_pin
  refVoltage = map(fsrReading, 0, 1023, 0, 5000);
  //map the reading to the corresponding value
  delay(2);

  fsrVoltage[i] = 5000 - refVoltage;
  // fsrVoltage is in millivolts so 5V = 5000mV

  //error correction
  voltageErr = (0.063*fsrVoltage[i]);
  fsrVoltage[i] = fsrVoltage[i] - voltageErr;
  ;
  if (fsrVoltage[i]<4750 && fsrVoltage[i]>200)
  {
    useful=useful+1;
    AvgVoltage=AvgVoltage+fsrVoltage[i]/24;
  }
}
AvgVoltage=(AvgVoltage*25)/useful;
refVoltage=5000-AvgVoltage;
/* The voltage = Vcc * R / (R + R_FSR) where
```

```
R = 1M2 and Vcc = 5V(fom arduino) so
R_FSR = ((Vcc - V) * R) / V  where R_fsr is
the resistance of the sensor(infinity to
300kohms when fully pressed*/


fsrI = refVoltage / fsrReference;
/* 1M2 resistor but actual value measured with
multimeter is on input r_ref */
fsrResistance = AvgVoltage / fsrI;

fsrConductance = 1000000/ fsrResistance;
/* conductance is on *10^-6 scale so multiply by
10^6 to see values
convert to Force
using the relation out of experiment*/
if (fsrConductance<1210)
    {fsrForce=0;}
else if (1210<=fsrConductance<=24315)
    {fsrForce=4.615*fsrConductance*0.001-3.963;}
else if (fsrConductance>24315)
  {
    fsrForce=8.702*fsrConductance-10.31;
    fsrForce=fsrForce/1000;
  }
//for testing
/*
  Serial.print("voltage ");
  Serial.println(AvgVoltage);
  Serial.print("resistance ");
  Serial.println(fsrResistance);
  Serial.print("Conductance ");
  Serial.println(fsrConductance);
  Serial.print("Force ");
```

```
    Serial.println(fsrForce);
    Serial.println("--------------------");
 */ //code for testing functions
    return(fsrForce);
}
/* add a new value to the array, moving the rest back
one space and removing the oldest */
void updateTimes(float newValue)
{
  for(int i = 0; i < ARRAY_SIZE-1; i++)  {
    times[i] = times[i+1];
  }
  times[ARRAY_SIZE-1] = newValue;
}
//returns an average of the values in the array
int avgArray(float values[])  {
  int total = 0;
  int counted = ARRAY_SIZE;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
    total = total + values[i];
    if(values[i] == 0)
      counted--;
  }
  return(total/counted);
}
//clear the array
void clearTimes()  {
  for(int i = 0; i < ARRAY_SIZE; i++)
    times[i] = 0;
}
//check if the array is full
void checkFullArray()  {
  arrayEmpty = true;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
```

```
    if(times[i] != 0)
      arrayEmpty = false;
  }
}
void f_rpm(){

    //Serial.println("test");
    pinMode(A5, OUTPUT);
    digitalWrite(mosSwitch,LOW);
    digitalWrite(A5, LOW);
    delay(50); // prepare next data
    pinMode(A5, INPUT_PULLUP);
    digitalWrite(mosSwitch,HIGH);

    //Read from the hall effect sensor(analog values)
    int magnetState = digitalRead(magnetSensor);
    // Serial.println(magnetState);

    if(magnetState == HIGH)  {
      magnetOn = false;
    }

    else  {
      magnetOn = true;
    }

    if(!magnetOn && prevMagOn)  {
      primed = true;
    }

    //timeout
    if(millis()-startTime > TIMEOUT) {
      clearTimes();
      pedaling = false;
```

```
    }


    float currentTime = millis();
    float changeTime = (currentTime - startTime);
    //record the time since the last pedal
    startTime = millis();

    if(pedaling){
      //check pedaling since the last timeout
      updateTimes(changeTime);
      //add the time to the running average array
      //Concatenates start/end strings with data
      String cadString = String(rpm);
String cadR_Pi=String("CADstart"+cadString+"CADend");
      //sends data
      Serial.print(cadR_Pi);
    }

    primed = false;
    pedaling = true;
    //}


    prevMagOn = magnetOn;
    checkFullArray();
    float gap = avgArray(times);
    if(arrayEmpty)  {
      rpm = 0;
    }
    else{
      rpm = 60000/gap;
      //turn millisecond gap value into rpm
    }
```

```
    /*light the lowest red LED when the first
    pedal stroke is recorded, since there isn't
    enough data to calculate rpm*/
    if(pedaling && rpm == 0){
      rpm = 0;
    }
    return;
}
```

Listing 3: Hall effect sensor testing code

```
#define ARRAY_SIZE 5
//The number of values to keep in the running average
#define TIMEOUT 1500
//Max time without input before turning off LEDs
#define FADE_INCREMENT .1
//Controls the fading speed:adjust with trial & error
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(10, 11); // RX | TX
/**PORT SETUP**/
const int magnetSensor = A5;
const int ledPin = 11;
const int mosSwitch = 13;
bool magnetOn = false;
bool prevMagOn = false;
bool primed = false;
bool arrayEmpty = false;
float times[ARRAY_SIZE];
//array to hold values for the running average
float startTime = millis();
int rpm = 0;
bool pedaling = false;
/* add a new value to the array, moving the rest back
one space and removing the oldest */
void updateTimes(float newValue)
```

```
{
  for(int i = 0; i < ARRAY_SIZE-1; i++)  {
    times[i] = times[i+1];
  }
  times[ARRAY_SIZE-1] = newValue;
}
//returns an average of the values in the array
int avgArray(float values[])  {
  int total = 0;
  int counted = ARRAY_SIZE;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
    total = total + values[i];
    if(values[i] == 0)
      counted--;
  }
  return(total/counted);
}
//for debugging
void printValues()  {
  for(int i = 0; i < ARRAY_SIZE - 1; i++)  {
    Serial.print(times[i]);
    Serial.print(", ");
  }
  Serial.println(times[ARRAY_SIZE -1]);
}
//clear the array
void clearTimes()  {
  for(int i = 0; i < ARRAY_SIZE; i++)
    times[i] = 0;
}
//check if the array is full
void checkFullArray()  {
  arrayEmpty = true;
  for(int i = 0; i < ARRAY_SIZE; i++)  {
```

```
    if(times[i] != 0)
      arrayEmpty = false;
  }
}
void setup()  {
  Serial.begin(9600);
   BTSerial.begin(9600);
   pinMode(A5, INPUT_PULLUP);
}
void loop()  {
 // BTSerial.write("test");
 // BTSerial.println("test");
 // Serial.println("test");
  pinMode(A5, OUTPUT);
  digitalWrite(mosSwitch,LOW);
  digitalWrite(A5, LOW);
  delay(50); // prepare next data
  pinMode(A5, INPUT_PULLUP);
  digitalWrite(mosSwitch,HIGH);

  //Read from the hall effect sensor(analog values)
  int magnetState = digitalRead(magnetSensor);
   Serial.println(magnetState);
  if(magnetState == HIGH)  {
    magnetOn = false;
  }
  else  {
    magnetOn = true;
  }
  if(!magnetOn && prevMagOn)  {
    primed = true;
  }
  //timeout0
  if(millis()-startTime > TIMEOUT) {
```

```
    clearTimes();
    pedaling = false;
  }
  if(magnetOn &&!prevMagOn)  {
  //if magnet passes sensor once  && primed
    float currentTime = millis();
    float changeTime = (currentTime - startTime);
    //record the time since the last pedal
    startTime = millis();
    if(pedaling)
    //if there's been pedaling since the last timeout
      updateTimes(changeTime);
      //add the time to the running average array
    primed = false;
    pedaling = true;
  }
  prevMagOn = magnetOn;
  checkFullArray();
  float gap = avgArray(times);
  if(arrayEmpty)  {
    rpm = 0;
  }
  else
    rpm = 60000/gap;
    //turn millisecond gap value into rpm
  /* light the lowest red LED when the first pedal
  stroke is recorded, since there isn't enough data
  to calculate rpm */
  if(pedaling && rpm == 0)
    rpm = 0;
  //Serial.println(rpm);
}
```

J Python codes

Listing 4: Main Python code

```
from kivy.app import App
from kivy.graphics import Color, Rectangle, Ellipse
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.image import AsyncImage
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.core.text import Label as CoreLabel
from kivy.uix.widget import Widget
from kivy.properties import ListProperty
from kivy.core.window import Window
from kivy.animation import Animation
import multiprocessing as mp
from multiprocessing import Process
import binascii
import struct
import time
from bluepy import btle
from bluepy.btle import UUID
import xlsxwriter
from xlsxwriter import*
import os.path
import string
import tkinter as tk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
import math
import time
import random
```

```
#Global variables
dev_R = btle.Peripheral(None)      #Right foot arduino
dev_L = btle.Peripheral(None)      #left foot arduino
bt_power = 0
#Rows in spread sheet1 for data from right foot
r_1R = 1
#Rows in spread sheet1 for data from left foot
r_1L = 1
#Rows in spread sheet2
r_2 = 1
#process variables
p1 = None
p2 = None

#Worksheet
wb = None

#Sheets
sheet1 = None
sheet2 = None



def disconnect():
    global dev_R
    global dev_L
    dev_R.disconnect()
    dev_L.disconnect()
    return

def make_workbook():
    save_path = '/media/pi/LEXAR'
    #Variable used to name file number
    n=1
    #Searches for exixting file of same name
```

```
    match = os.path.exists("/media/pi/LEXAR/Power_phase Session "
        + str(n)+".xlsx")

    while match==True:
        n=n+1
        match = os.path.exists("/media/pi/LEXAR/Power_phase
           Session " + str(n)+".xlsx")

    #Names File
    name_of_file = "Power_phase Session " + str(n)
    completeName = os.path.join(save_path, name_of_file+".xlsx")

    # Workbook is created
    global wb
    wb = xlsxwriter.Workbook(completeName)

    return


def display_data(a, data):
    start.refresh_text(a, data)


def data_processing():
    global bt_power
    bt_power = 1
    def right():
        global dev_R
        dev_R.connect("F4:5E:AB:B0:8E:CC", )
        global bt_power

        class MyDelegate(btle.DefaultDelegate):
            def __init__(self):
                btle.DefaultDelegate.__init__(self)
```

```
    def handleNotification(self, cHandle, data):
        data = str(data)
        if (data[0:9] == "CADstart ")

            data_c = data
            global sheet
            global r_2
            sheet2.write(r_2, 1, data_c)

        else

            data = data[9:13]
            print("A notification was received from right
               : " + str(data))
            data = int(data)
            global sheet1
            global r_1R
            sheet1.write(r_1R, 0, data)

        a = 0
        display_data(a, data)

        r_1R = r_1R + 1
        r_2 = r_2 +1


dev_R.setDelegate( MyDelegate() )
svc_R = dev_R.getServiceByUUID( 0xdfb0 )
ch_R = svc_R.getCharacteristics()[0]
val_R = ch_R.valHandle
print(val_R)

dev_R.writeCharacteristic(ch_R.valHandle+1, b"\x02\x00")
```

```
    while True:
        if (bt_power == 0):
            break

        if dev_R.waitForNotifications(1.0):
        # handleNotification() was called
            continue

        print("Waiting...")
    return

def left():
    global dev_L
    dev_L.connect("F4:5E:AB:B1:14:FB")
    global bt_power

    class MyDelegate(btle.DefaultDelegate):
        def __init__(self):
            btle.DefaultDelegate.__init__(self)

        def handleNotification(self, cHandle, data):
            data = str(data)
            data = data[9:13]
            print("A notification was received from left:" +
               data)
            data = int(data)
            a = 1
            global r_1L
            global sheet1
            sheet1.write(r_1L, 2, data)
            display_data(a, data)

            r_1L = r_1L +1
```

```
        dev_L.setDelegate( MyDelegate() )
        svc_L = dev_L.getServiceByUUID( 0xdfb0 )
        ch_L = svc_L.getCharacteristics()[0]
        val_L = ch_L.valHandle
        print(val_L)

        dev_L.writeCharacteristic(ch_L.valHandle+1, b"\x02\x00")

        while True:
            if (bt_power == 0):
                break

            if dev_L.waitForNotifications(1.0):
                # handleNotification() was called
                continue

            print("Waiting...")
        return

    if __name__ == '__main__':
        global p1
        p1 = Process(target = right)
        p1.start()
        global p2
        p2 = Process(target = left)
        p2.start()


class start(FloatLayout):

    def __init__(self, **kwargs):
        # make sure we aren't overriding any important
           functionality
```

```
        #Window.size = (800 , 480)  #Sets window sixe to size of
           display
        #Window.fullscreen = True
        super(start, self).__init__(**kwargs)
        self.add_widget(
            AsyncImage(
                source="/home/pi/carbon.jpeg",
                size_hint= (1.5, 1.5),
                pos_hint={'center_x':.5, 'center_y':.5}))
        self.startbtn = Button(
                text="START",
                background_color=(0,1,0,1),
                size_hint=(.3, .3),
                pos_hint={'center_x': .5, 'center_y': .7})
        self.startbtn.bind(on_press=self.btn_pressedstart)
        self.add_widget(self.startbtn)

        self.quitbtn = Button(
                text="SAVE AND QUIT",
                background_color=(1,0,0,1),
                size_hint=(.2, .2),
                pos_hint={'center_x': .5, 'center_y': .3})
        self.quitbtn.bind(on_press=self.btn_pressedquit)
        self.add_widget(self.quitbtn)


        with self.canvas.before:
            Color(0, 0, 0, 0)  # green; colors range from 0-1
               instead of 0-255
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self._update_rect, pos=self._update_rect)

    def _update_rect(self, instance, value):
```

```
        self.rect.pos = instance.pos
        self.rect.size = instance.size

    def btn_pressedstart(self, instance):
        self.remove_widget(self.startbtn)
        cadence = Button(
            text='Cadence',
            size_hint=(.2, .1),
            pos_hint={'center_x': .5, 'center_y': .7})
        self.add_widget(cadence)
        right = Button(
            text='Right Pedal',
            size_hint=(.2, .1),
            pos_hint={'center_x': .8, 'center_y': .9})
        self.add_widget(right)
        left = Button(
            text='Left Pedal',
            size_hint=(.2, .1),
            pos_hint={'center_x': .2, 'center_y': .9})
        self.add_widget(left)
        self.draw()
        make_workbook()
        global wb
        global sheet1
        sheet1 = wb.add_worksheet('Power Phase')
        global sheet2
        sheet2 = wb.add_worksheet('Cadence')
        #Adds headers for different values
        sheet1.write(0, 0, 'Right Power')
        sheet1.write(0, 1, 'Right Crank Angle')
        sheet1.write(0, 2, 'Left Power')
        sheet1.write(0, 3, 'Left Crank Angle')
        sheet2.write(0, 0, 'Time')
        sheet2.write(0, 1, 'Cadence')
```

```
    print('Hello')
    data_processing()
    return
def draw(self):

     with self.canvas:

         # Draw right circle
         label_r = CoreLabel(text="0 N", font_size=40)
         label_r.refresh()
         Color(0, 0, 0)
         Ellipse(pos=(570,300), size=(150,150))
         Color(1, 1, 1)
         Rectangle(texture=label_r.texture, pos=(620,335),
            size=(70, 70))

         # Draw left circle
         label_l = CoreLabel(text="0 N", font_size=40)
         label_l.refresh()
         Color(0, 0, 0)
         Ellipse(pos=(90,300), size=(150,150))
         Color(1, 1, 1)
         Rectangle(texture=label_l.texture, pos=(140,335),
            size=(70, 70))

         #Draw cadence rectangle
         label_c = CoreLabel(text="0 rpm", font_size=40)
         label_c.refresh()
         Color(0, 0, 0)
         Rectangle(pos=(325,275), size=(150, 75))
         Color(1, 1, 1)
         Rectangle(texture=label_c.texture, pos=(325,275),
            size=(150, 70))
```

```
    def btn_pressedquit(self, instance):
        global wb
        wb.close()
        global bt_power
        bt_power = 0
        global p1
        p1.terminate()
        global p2
        p2.terminate()
        disconnect()
        print("Bye")
        self.add_widget(self.startbtn)
        MainApp.get_running_app().stop()
        Window.close()


class MainApp(App):

    def build(self):
        root = start()
        return root

if __name__ == '__main__':
    MainApp().run()
```

Listing 5: Initial Bluetooth test code

```
from bluepy import btle
print("Connecting...")
dev = btle.Peripheral("F4:5E:AB:B0:8E:CC")
print("Services...")
for svc in dev.services:
    print (str(svc))
```

```
print("Characteristics..")
for char in dev.getCharacteristics():
    print(str(char))
#data = char.read()
#print (str(ord(data)))
```

Listing 6: Excel Creation Test Code

```
import xlsxwriter
from xlsxwriter import*
import os.path
import string
import tkinter as tk
import numpy as np
import matplotlib.pyplot as plt
def excel():
    #def returnCadence (num):
    save_path = 'D:/EDP Programming'
    #Variable used to name file number
    n=1
    #Searches for exixting file of same name
    match = os.path.exists("D:/EDP Programming/Cycling Session "
       + str(n)+".xlsx")
    while match==True:
        n=n+1
        match = os.path.exists("D:/EDP Programming/Cycling
           Session " + str(n)+".xlsx")
    #Names File
    name_of_file = "Cycling Session " + str(n)
    completeName = os.path.join(save_path, name_of_file+".xlsx")
    # Workbook is created
    wb = xlsxwriter.Workbook(completeName)

    # add_sheet is used to create sheet.
    sheet1 = wb.add_worksheet('Cycling_Data')
```

```
#sheet1.add_table('A1:B12')

sheet1.write(0, 0, 'Time (s)')
sheet1.write(0, 1, 'Cadence (rpm)')
sheet1.write(0, 2, 'Power (W)')
#take update cadence and power values initially

c=5             #cadence value
p=10            #power value
i=1             #time iterator

#write values in sheet
while c!=0:
    sheet1.write(i, 0, i)
    sheet1.write(i, 1, c)
    sheet1.write(i, 2, p)
    i=i+1
    c=0
    #create loop here to update cadence and power


#cadence graph
cadenceChart = wb.add_chart({'type': 'line'})
sheet1.insert_chart('E2', cadenceChart)
cadenceChart.add_series({
    'categories': '=Cycling_Data!$A$2:$A$11',
    'values':     '=Cycling_Data!$B$2:$B$11',
    'line':       {'color': 'blue'},
})
cadenceChart.set_title({ 'name': 'Cadence (time)'})
cadenceChart.set_x_axis({'name': 'Time (s)'})
cadenceChart.set_y_axis({'name': 'Cadence (rpm)'})
cadenceChart.set_legend({'none': True})
```

```
    #power graph
    powerChart = wb.add_chart({'type': 'line'})
    sheet1.insert_chart('M2', powerChart)
    powerChart.add_series({
        'categories': '=Cycling_Data!$A$2:$A$11',
        'values':     '=Cycling_Data!$C$2:$C$11',
        'line':       {'color': 'red'},
    })
    powerChart.set_title({ 'name': 'Power (time)'})
    powerChart.set_x_axis({'name': 'Time (s)'})
    powerChart.set_y_axis({'name': 'Power (W)'})
    powerChart.set_legend({'none': True})
    wb.close()
    #returnCadence (time)

    #   return cadence
class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()
    def create_widgets(self):
        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "START"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")
        self.quit = tk.Button(self, text="QUIT AND SAVE", fg="red
           ",
                              command=self.master.destroy)
        self.quit.pack(side="bottom")
    def say_hi(self):
        V = np.array([[1,1],[-2,2],[4,-7]])
        origin = [0], [0] # origin point
```

```
        fig = plt.figure()
        plt.quiver(*origin, V[:,0], V[:,1], color=['r'], scale
           =21)
        plt.show()
        excel()
root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

Listing 7: Simultaneous Bluetooth test code

```
import multiprocessing as mp
from multiprocessing import Process
import binascii
import struct
import time
from bluepy import btle
from bluepy.btle import UUID
def right():
    dev_R = btle.Peripheral("F4:5E:AB:B0:8E:CC")
    class MyDelegate(btle.DefaultDelegate):
        def __init__(self):
            btle.DefaultDelegate.__init__(self)
        def handleNotification(self, cHandle, data):

            print("Received Notification: %s" %data)

    dev_R.setDelegate( MyDelegate() )
    svc_R = dev_R.getServiceByUUID( 0xdfb0 )
    ch_R = svc_R.getCharacteristics()[0]
    val_R = ch_R.valHandle
    print(val_R)
    dev_R.writeCharacteristic(ch_R.valHandle+1, b"\x02\x00")

    while True:
```

```
        if dev_R.waitForNotifications(1.0):
            # handleNotification() was called
            continue
        print("Waiting...")

def left():
    dev_L = btle.Peripheral("F4:5E:AB:B1:14:FB")
    class MyDelegate(btle.DefaultDelegate):
        def __init__(self):
            btle.DefaultDelegate.__init__(self)
        def handleNotification(self, cHandle, data):

            print("Received Notification: %s" %data)
            #add_to_graph(data)



    dev_L.setDelegate( MyDelegate() )
    svc_L = dev_L.getServiceByUUID( 0xdfb0 )
    ch_L = svc_L.getCharacteristics()[0]
    val_L = ch_L.valHandle
    print(val_L)
    dev_L.writeCharacteristic(ch_L.valHandle+1, b"\x02\x00")

    while True:

        if dev_L.waitForNotifications(1.0):
            # handleNotification() was called
            continue
        print("Waiting...")
if __name__ == '__main__':
    p1 = Process(target = right)
    p1.start()
    p2 = Process(target = left)
```

```
p2.start()
```