Elaine Kristant
DGMD S14
Final Project

Pedal Stroke Analysis for Cycling

Abstract

Cycling computers that attach to bicycles typically give you data related to trip time, speeds, and sometimes cadence, but they don't provide any data on the mechanics of the rider unless you're willing to spend over $1000. This project aims to use a set of sensors to analyze the pedal and foot movement to increase efficiency of the rider with no modifications to the bicycle and at a lower price point. With the use of machine learning, the pedal stroke could be characterized with just the use of an IMU. It was determined that a greater than 90% accuracy can be obtained in differentiating between different stroke techniques using NN classification.

1. Introduction

1.1. Background

Sports analytics has become an increasing market with how easy it is to collect data on a small scale. In the next few years, the market is expected to grow by about $215M USD to about $850M USD.[1] Athletes of all levels are always looking for ways to improve their performance by either hiring trainers to coach them, or devices that give them feedback on their technique, or a combination of both. For road cycling, world class athletes have a team of analysts to help trim every millisecond off of their times, but a typical amateur cyclist cannot afford those services.

For amateur cyclists, the most common analytical tools they have are a combination of bike computer, smartphone, and/or smartwatch. These devices are great at getting basic information such as cadence, speed, heart rate, etc. However, none of them take a look at the rider's form and technique. A few companies have started making smart cranks and pedals that have force sensors in them to provide the rider with more data . These devices are

Figure 1: Garmin Rally™ RS200 Dual-sensing Power Meter. These smart pedals measure the force applied and communicate with a smartphone, smartwatch, or Garmin cycling computer. MSRP $1099.99 USD [2]

expensive and require the cyclist to replace parts of their bike with these components. For this project, the type of analytics of interest is pedaling form. When children learn to bike, they typically start with platform pedals. With this style of pedal, the force can only be applied in a downward direction so only one leg is doing work at a time. In figure 2, on platform pedals, force is only able to be applied from the top dead center (TDC) position to the bottom dead center (BDC) position. To get more out of each stroke and to have power coming from both legs throughout the stroke, more advanced riders have pedals that all them to also pull up on the pedal. The two main types of pedals that allow this are toe-clip style pedals that hook over the front of the shoe and what are known as clipless pedals that have a connector on the bottom of the shoe. Having a very stiff cycling shoe also improves on the force transfer from the foot to the pedal.

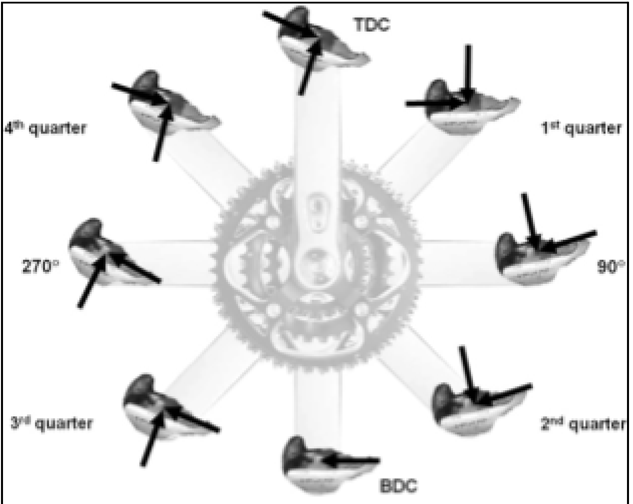


Figure 2: “Ideal” normal and anterior–posterior force component application on the pedal to optimize force effectiveness. [3]

Looking again at figure 2, you can also see that the forces stay normal to the top of the pedal (due to the foot being attached), but the pedal rotates slightly towards the axis of the crank rotation. This rotation allows for a smoother transfer of force from the push to the pull and a less jerky pedal motion. The slight angle change also allows the activation of different muscle groups within the leg. [3]

Another way to look at the stroke is from the animated GIFs found at the website *Perfect Conditioning Ltd.*[4] On the site, it describe five different types of strokes: “perfect” for flat terrain, flat with no ankle movement at all, a flat stroke with only a little ankle movement (seen mostly on platform pedals), toes pointing down with no ankle movement, and lastly “perfect” for uphill. Of the five different strokes, only two are optimal, the two “perfect” ones.

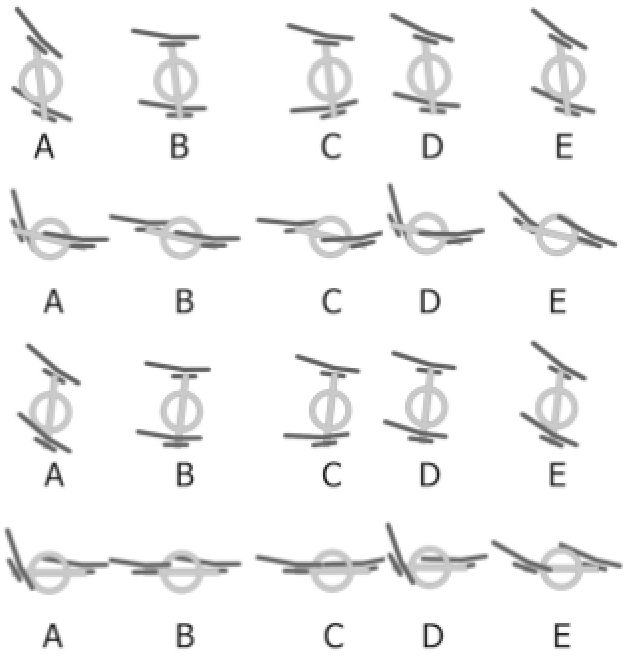


Figure 3: Screenshots of pedal motions for the different classes at the same crank angle. A: “Perfect” on flat terrain, B: Flat with no ankle movement, C: Flat with some ankle movement, D: “Perfect” for hills, E: Toes always pointing down with no ankle movement.

Each of these stroke styles were used to identify the classes for machine learning. Obviously, the perfect stroke was used as a class and what the cyclist would be aiming to achieve. The other styles were attempted on a stationary bike and it was determined that two of them were able to be replicated: the flat stroke with some ankle movement and the toes down with no ankle movement. The flat stroke with no ankle movement was not physiologically possible to replicate. The perfect stroke for uphill was difficult to replicate on a stationary bike since it was on a flat surface.

1.2. Data collection and processing

1.2.1. Setup

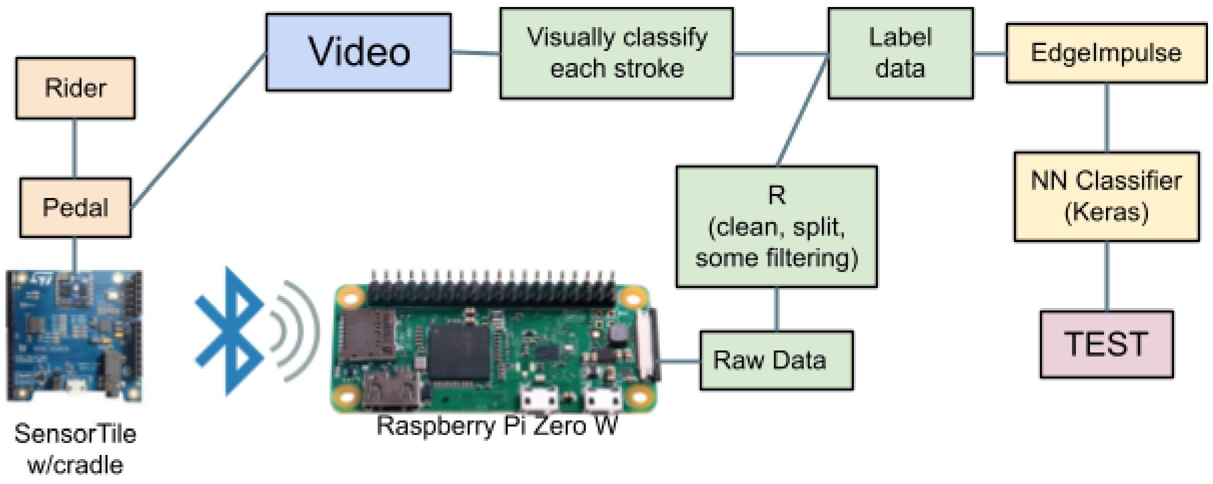


Figure 4: System diagram for collecting and analyzing data from the sensor tile. The colors correspond to the different aspect of the system. Orange is the setup, blue is the data collection, green is the data processing, yellow is analysis, and red is confirmation.

Bike

A stationary bike was used for testing. It is a standard commuter style bike mounted to a resistance bike trainer. The pedals on the bike were toe-clip style where the foot slips into the cage at the front. The toe-clips don't lock the foot as tightly to the pedal as clipless pedals, however they were what was available for testing. Initially, running shoes were worn for testing, but then a switch was made to cycling shoes for the final data collection to reduce variation from flex that the running shoes produced. The resistance on the trainer was set to a low level to simulate pedaling on flat terrain.

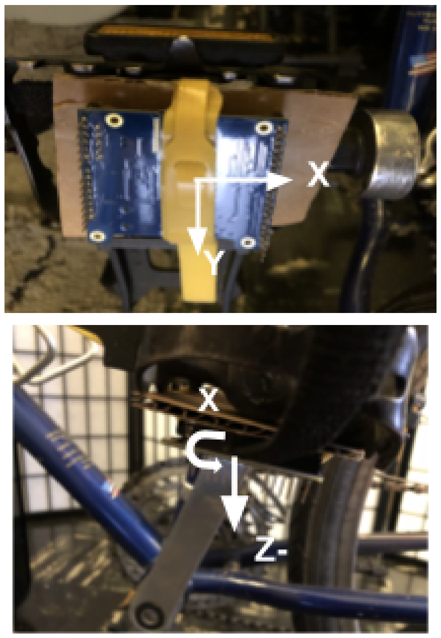


Figure 5: Mounting of the SensorTile to the pedal. Top photo shows the bottom view of the pedal and the acceleration axis orientation. Bottom photo is a side view showing the gyrometer axis (X) and the acceleration Z axis orientation

SensorTile

Initially, it was proposed to mount the SensorTile to the cycling shoe so the system could be easily attached, however it was determined that the data would be more consistent and the axis of the gyrometer could be better aligned with the axis of rotation of the pedal. The SensorTile with the cradle expansion board was mounted to the bottom of the pedal. The system would work the same whether the sensor was on either pedal, but the left pedal was chosen and kept consistent throughout all of the tests.

The SensorTile was mounted such that the accelerometer Z axis was vertical and the X axis was along the pedal's axis of rotation and the Y axis was pointing towards the toe. As such, the gyrometer axis of interest, the one around the pedal's axis of rotation, was the X axis. The sensor tile was powered through the USB port using a wall adapter.

1.2.2. Data Collection

RaspberryPi

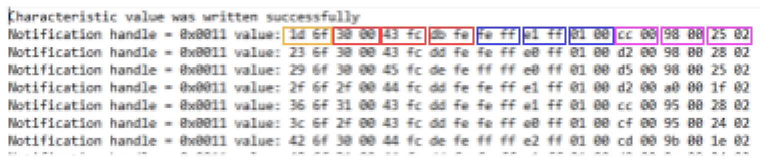
The RaspberryPi was not physically connected to the SensorTile. All of the communication between the Pi and the SensorTile was done through bluetooth low energy (BLE). Data was collected from the SensorTile to the Pi and saved using `gatttool`. The data from the sensors is A WiFi connection was made between the RaspberryPi and a PC and commands were sent to the Pi from an SSH connection. Data was transferred from the RaspberryPi to a PC using `scp`.

Video

In order to be able to identify the technique of each of the pedal strokes, videos of the trials were recorded. The camera was arranged perpendicular to the pedal so the entire stroke could be captured.

1.2.3. Data Processing

Hexadecimal to decimal



```
Characteristic value was written successfully
Notification handle = 0x0011 value: 1d 6f 30 00 43 fc db fe fe ff e1 ff 01 00 cc 00 98 00 25 02
Notification handle = 0x0011 value: 23 6f 30 00 43 fc dd fe fe ff e0 ff 01 00 d2 00 98 00 28 02
Notification handle = 0x0011 value: 29 6f 30 00 45 fc de fe ff ff e0 ff 01 00 d5 00 98 00 25 02
Notification handle = 0x0011 value: 2f 6f 2f 00 44 fc dd fe fe ff e1 ff 01 00 d2 00 a0 00 1f 02
Notification handle = 0x0011 value: 36 6f 31 00 43 fc dd fe fe ff e1 ff 01 00 cc 00 95 00 28 02
Notification handle = 0x0011 value: 3c 6f 2f 00 43 fc dd fe fe ff e0 ff 01 00 cf 00 95 00 24 02
Notification handle = 0x0011 value: 42 6f 30 00 44 fc de fe ff ff e2 ff 01 00 cd 00 9b 00 1e 02
```

Figure 6: Representative data received from the SensorTile. The data correspond to the following: orange is the timestamp, red is the accelerometer data in the order x, y, z, blue is the gyrometer data x, y, z, and magenta is the magnetometer data x, y, and z.

The data coming from the SensorTile is a string of hex values that correspond to each of the sensors and a timestamp. Figure 6 shows how the string is arranged and how it corresponds to each of the sensor axes. In order to be human readable and analyzed by the machine learning algorithm, each set of data was converted to a decimal value and the data was split by each axis. During the processing, the acceleration Z axis was subtracted by 1000 milli-g to normalize with the other acceleration axes. The 1000 milli-g corresponds to the force on the accelerometer due to gravity. This way of hex encoding also uses two's complement, so any values greater than 32737 actually represent negative values and need to have 65536 subtracted to get the accurate value.

The data coming from the sensors had some, but not a lot of noise. To smooth out the data somewhat and have a more even shape, a simple moving average was applied. The data was then cleaned to remove any erroneous data. In some instances, a bad reading was sent as can be seen in figure 7 by the two horizontal lines in the raw data that stretch past the range. The timestamp was over 100000 for that data point which was not logically part of the data and was removed. In some cases the stroke had mixed data where it didn't make sense for the sensor to switch between negative and positive values in quick succession, so that whole stroke was removed.

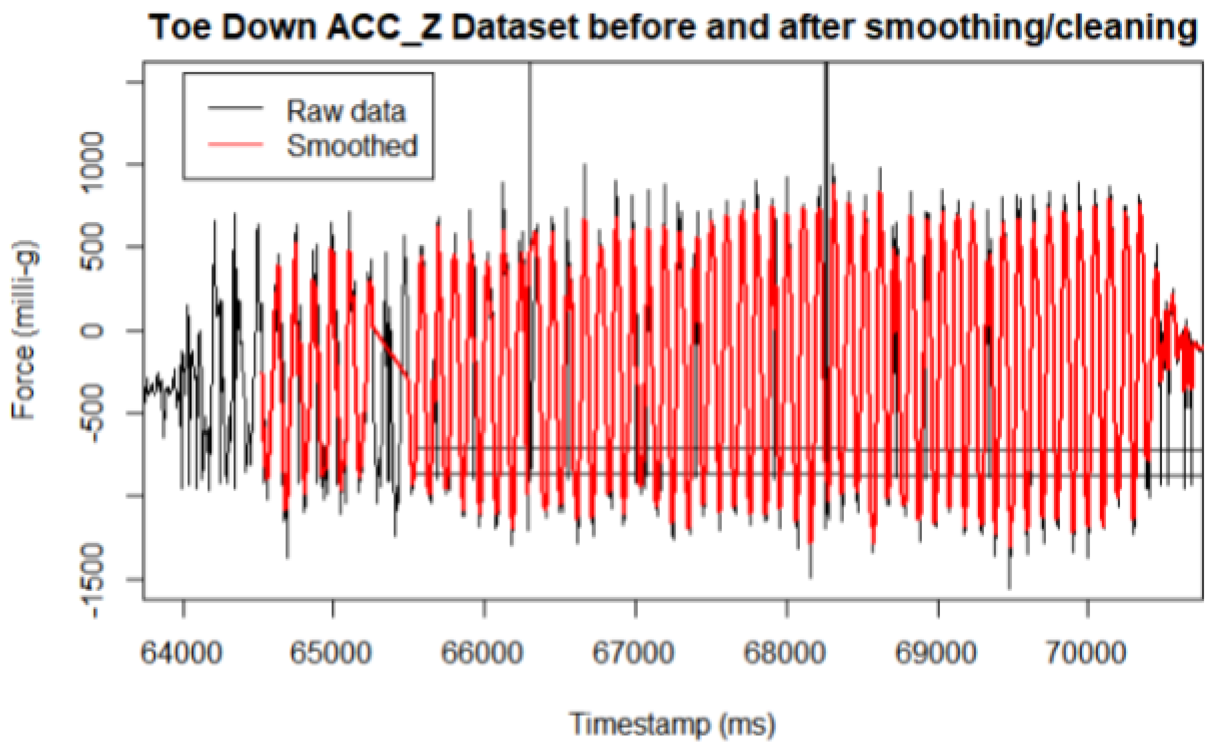


Figure 7: Example data received from the Z axis of the accelerometer before and after smoothing and cleaning.

Data segmentation

Even though pedaling is a very cyclical motion and could be analyzed as a waveform, for the type of analysis that this project aimed to do, each pedal stroke needed to be looked at individually. The data could have been collected one stroke at a time, but it would have provided poor results. Looking back at figure 7, the first few strokes are not consistent until the bike (in this case the rear wheel) has come to a constant speed due to the increased load to get the bike moving. Since the data was collected over a couple of minutes, a lot of samples could be collected at once but then needed to be segmented. The method of segmentation was to iterate through the data, look for two sign changes, and then separate it out. The separated data was written out to a CSV file.

Timestamp normalization

After uploading the initial data to EdgeImpulse, it was discovered that the code there did not work well when the timestamp shifted. On occasion, the data sampling would shift by one millisecond which would cause all of the samples to not be the exact same frequency. Essentially, in most cases the rate was one sample per six milliseconds, but sometimes it would be seven milliseconds. EdgeImpulse did not like this. Since I did not think it would affect the data that much, code was written to even out all of the samples to be 6ms apart.

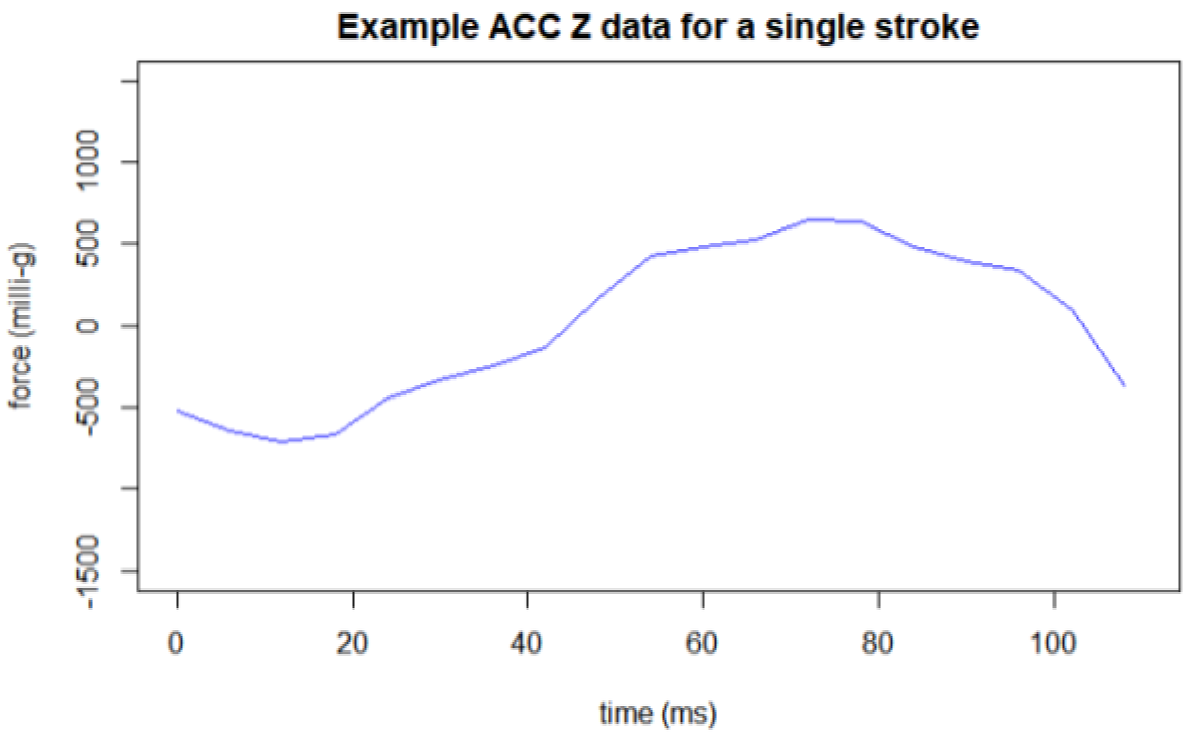


Figure 8: Example segmented data for one stroke.

Labeling data

Once the data was segmented, it needed to be labeled for the class it was. To identify the class, the video recordings were reviewed and compared to the animated GIFs previously mentioned. A few features were used to identify each of the classes. First, the pedal was looked at to see how the angle changed as the stroke went around. Since it was sometimes difficult to interpret, the heel of the pedaling foot was used as a way to identify the technique.

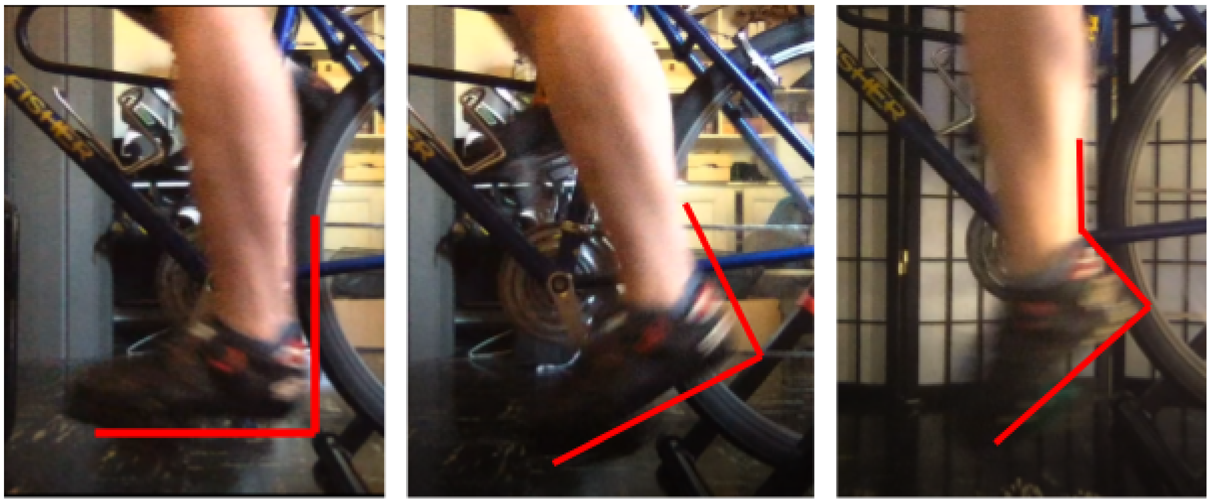


Figure 9: Difference in foot and ankle position used for identifying classes. From left to right: flat pedaling, correct pedaling, toe down pedaling.

2. Related Work

From the background research performed, it seems that most similar analysis is performed with the use of force sensors instead of accelerometers. Since a lot of the publications are from research labs or high-end sports analysis companies where the cyclist is in a motion capture lab hooked up to an array of sensors and video recorded, the acceleration and speed data was typically done through video. Since this device is ultimately meant to be portable, video analysis is not feasible.

For the couple of companies that make the force sensing pedals and cranksets, it seems that it’s up to the user to determine what is wrong with their stroke. For instance, the Garmin pedals will tell the user the balance of force between left and right, where their power is on the stroke, and it can tell the difference between sitting and standing. [2] For a professional athlete, they might be able to look at the data and know how to adjust their foot to fix their technique, but for an amateur rider, they might not.

For the motion analysis itself, the tutorials from UCLA on using the SensorTile were heavily used as reference, specifically, tutorials 1-9. [5] For the machine learning elements, the analysis was performed similarly to the EdgeImpulse demo for gesture recognition. [6] More information will be described later for the specifics on the analysis.

3. Team organization

This project was completed solely.

4. Software and Developing Tools

SensorTile software

The software package used on the SensorTile was a modified version of the ALLMEMS1_V3 provided by STMicroelectronics. The ALLMEMS software reports out data for all of the sensors on the SensorTile. Different commands can be sent to the board to collect the data from grouping of the sensors, for example, the command for environmental data will send the temperature, humidity, and barometric pressure. For this project, the motion data, which is a combination of the accelerometer, gyrometer, and magnetometer, was sent from the SensorTile to the RaspberryPi.

RaspberryPi software

The OS installed on the RaspberryPi OS was Raspbian, which is a form of Linux based on Debian. The RaspberryPi was primarily used as an interface between the SensorTile and a PC. Since all of the analysis was performed offline, only smaller software tools such as gatttool and the bluetooth utility were used. An attempt was made to integrate the Raspberry with the EdgeImpulse CLI, however the installation was causing problems and not all of the required packages worked with the specific RaspberryPi used (RaspberryPi Zero W).

Data processing software

Since I have minimal experience with Python and this was a solo project, I chose a language I was familiar with to perform the data processing. If it was a group project, I would have had more time to work in Python, but instead I chose to use R. All of the programming was done in RStudio. From my understanding, Pandas is similar to R, and in the future the functions and code created could be transferred to Python without much work. A full copy of the code is provided in the supporting documentation in GitHub, but an excerpt for the segmentation is provided below as an example.

```r
divide.data.function <- function(df, filename){
  state <- 0
  count <- 0
  current.timestamp <- c(0)
  timestamp.zero <- df[1,1]
  index <- 0
```

```r
 loop.count <- 0

 for( index in 1:nrow(df)){
  current.timestamp[index] <- count*6

  if ((state == 0) && (df[index, 4] < 0)){
  state <- 0
  count <- count + 1
  index <- index + 1
  }

  else if((state == 0) && (df[index, 4] >= 0)){
  state <- 1
  count <- count + 1
  index <- index + 1
  }

  else if((state == 1) && (df[index, 4] >= 0)){
  state <- 1
  count <- count + 1
  index <- index + 1
  }

  else if ((state == 1) && (df[index, 4] < 0)){
  state <- 0

  output.data.frame <- data.frame(
        timestamp = current.timestamp[c((index-count):index)],
        acc_x = df$acc_x_smoothed[c((index-count):index)],
        acc_y = df$acc_y_smoothed[c((index-count):index)],
        acc_z = df$acc_z_smoothed[c((index-count):index)],
        gyr_x = df$gyr_x_smoothed[c((index-count):index)],
        gyr_x = df$gyr_y_smoothed[c((index-count):index)],
        gyr_x = df$gyr_z_smoothed[c((index-count):index)],
        mag_x = df$mag_x_smoothed[c((index-count):index)],
        mag_y = df$mag_y_smoothed[c((index-count):index)],
        mag_z = df$mag_z_smoothed[c((index-count):index)]
  )

  loop.count <- loop.count+1
  write.csv(output.data.frame, file = paste(filename, loop.count,
".csv"), row.names = FALSE)
```

```
            index <- index + 1
            count <- 0
            }
        }
    }
```

Machine learning software

The software used for machine learning was EdgeImpulse, which is a web base GUI for setting up machine learning. It uses the same packages for common ML projects (Keras and tensorflow) but it was easier to use for someone not familiar with machine learning or Python prior to this course.

5. List of Milestones

Week 1. Receive equipment, install environments, and run through tutorial 1.

Week 2. Work through tutorials 2 & 3 to see what the capabilities of the sensor Tile are. Come up with project idea.

Week 3. Short week (4th of July), finalize project idea and try to recruit a partner.

Week 4. Work on tutorials 4 & 5. Finish project proposal and start collecting data from a stationary bike.

Week 5. Finish collecting data and start processing it for uploading into EdgeImpulse. Finish identifying classes and label data.

Week 6: Finish data processing and complete analysis. Collect more data if needed. Work on final presentation and final report.

Week 7: Final presentations.

6. Results and analysis

Initial analysis

Prior to uploading the data into EdgeImpulse, the data was looked at to see if the hypothesis of using the IMU to identify stroke technique was valid. Random samples were chosen, plotted, and interpreted to see if there was a visual difference in the profiles. Figure 10 below shows the visualization and differences can be noticed. For the “good” stroke, the waveform looks symmetric across the x-axis which shows a nice even pace throughout the cycle. For the “toes down” sample, the wave looks sinusoidal, however it is skewed towards the negative values, implying a greater force pushing than pulling. For the “flat” sample, it has a much more linear pattern which translates to an uneven stroke. This data looked promising prior to uploading to EdgeImpulse.

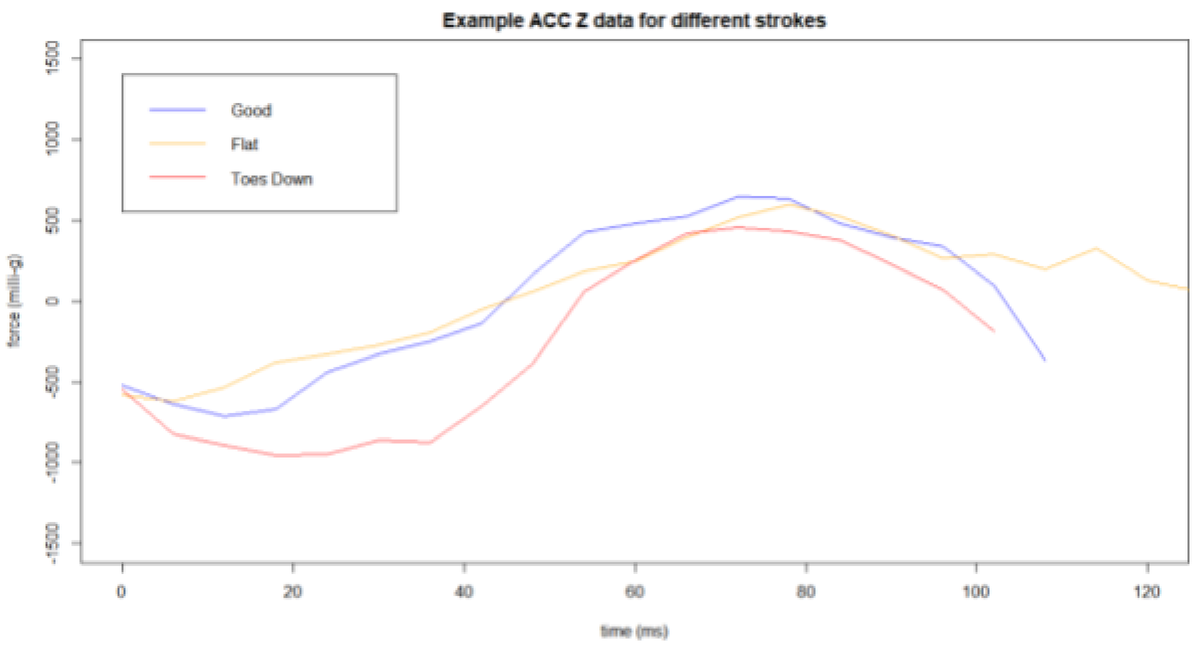


Figure 10: Random samples chosen to visualize the differences between strokes.

The first set of data that was processed and analyzed had a mix of four different techniques, flat, good, toe down, and hill. The breakdown of the dataset was 34, 45, 12, and 5 samples respectively for the training data and 10, 6, 5, and 1 for the testing data. The first thing that was noticed is that the data was fairly unbalanced and I did not have high confidence in my results (about a 66% accuracy). Due to the difficulty in replicating the hill technique, I removed those items from the set. I also collected more data for the toe down technique to bring the number of samples to be closer to the other two classes. There were also issues with the timestamps which I did not understand at the time, and those issues are addressed in the final analysis.

Iteration

After collecting more data, the new data was uploaded and combined with the remaining data from the first analysis. The new datasets had the following number of samples: training/testing: flat 33/11, good 38/11 , toes 36/9. For the EdgeImpulse parameters, the following were chosen: time series data, raw data with all of the axes included (acceleration, gyro, and magnetometer), and Keras for the classification. For the first run, all of the default values were kept except for the window size which was adjusted for the length of data collected.

Even though I had over 100 samples uploaded, the feature generation was only using 26 total samples and there was an error being reported: `WARN: failed to process<file name>:`

`Shape of data does not match earlier data. Expected 97 columns, but got 25. Does all your data have the same frequency?` I did not understand exactly what this error meant since all of the samples were roughly the same length and had the same sampling rate. After doing some research, it was determined that every timestamp needs to be exactly the same length from the previous one. However, with only 26 samples, the accuracy of the training data was 94% with a 1.92 loss but the testing data could not be analyzed due to the same error as above.

Final analysis

After reprocessing the data to standardize the timestep as described in section 1.2, I was able to train and test on the full dataset. I again started with the default settings for the analysis and got about an 80% accuracy for both the training and testing. After looking at the different sensors and axes, I decided to remove the axes that I didn't think had an influence on the stroke identification, such as the accelerometer X axis and the gyroscope Y and Z axes. One interesting thing that was noticed is that the magnetometer did seem to change depending on the type of technique. Logically, I could not understand why the magnetometer should change since the position of the bike never changed (always pointing in the same cardinal direction), but I kept those axes in. If this device is used on a moving bike, those axes should be removed, though. The feature generation looked really promising with the individual classes being clustered as shown below:

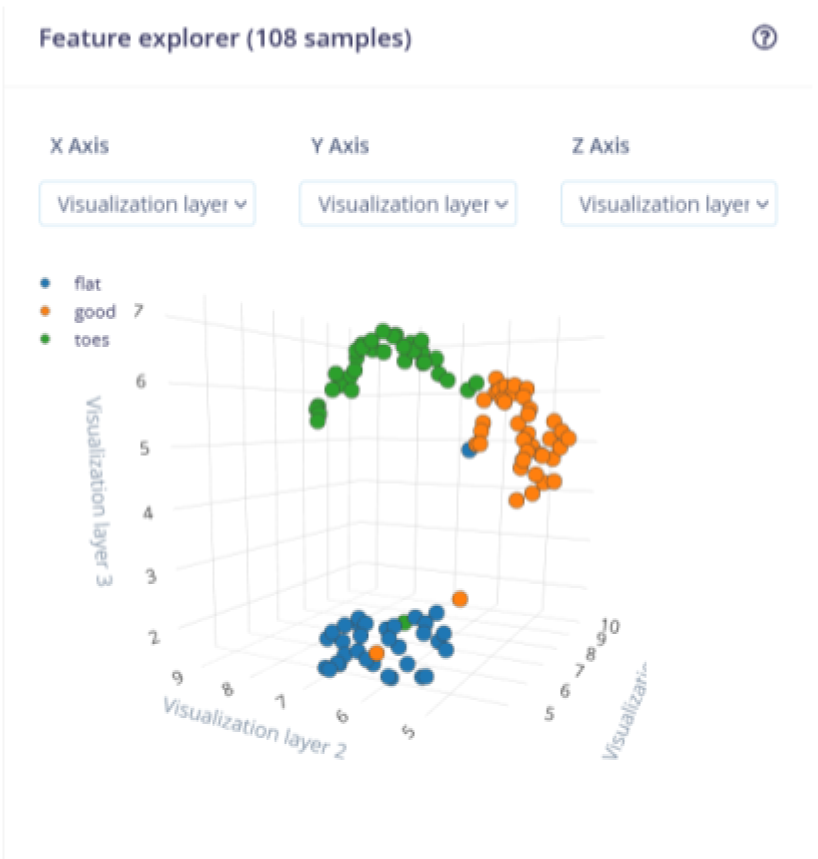
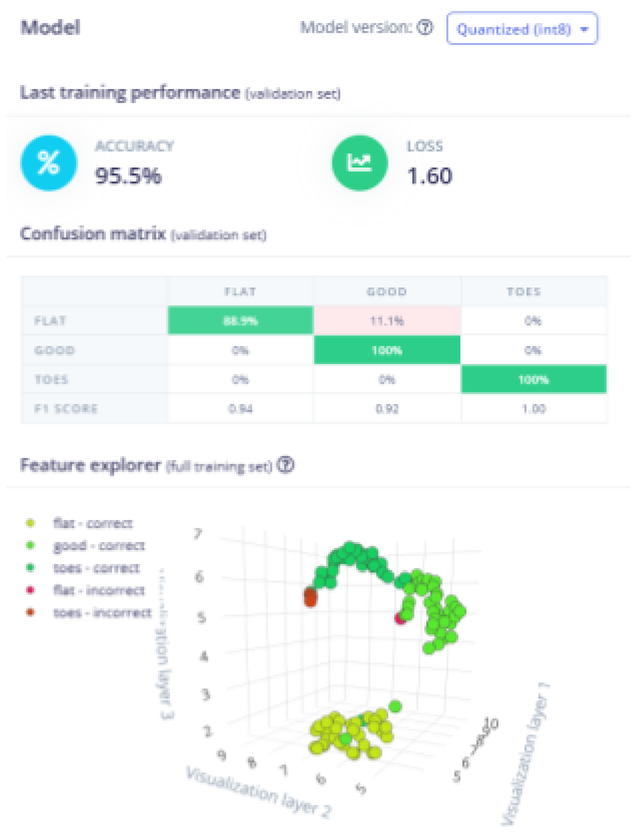


Figure 11: Results of feature generation from EdgeImpulse. There are a few scattered points, but most of the clusters are by class.

The learning rate was then adjusted to find an optimal solution that would increase the accuracy without causing the training data to be overfit. The rate was changed incrementally from 0.001 to 0.04 and the rate that seemed to give the best results was 0.015, which had an accuracy of 95.5% with a loss of 1.6. The testing data was then run and an accuracy of 100% was obtained. The project is public and is located here: https://studio.edgeimpulse.com/public/43059/latest.



Model — Model version: Quantized (int8)

Last training performance (validation set)

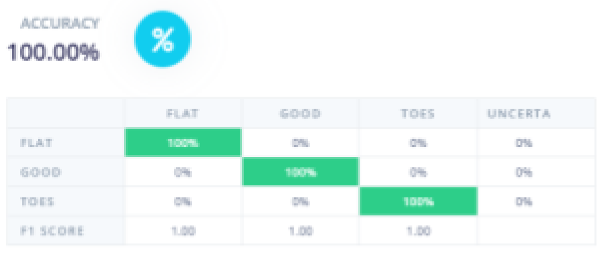
ACCURACY 95.5%

LOSS 1.60

Confusion matrix (validation set)

	FLAT	GOOD	TOES
FLAT	88.9%	11.1%	0%
GOOD	0%	100%	0%
TOES	0%	0%	100%
F1 SCORE	0.94	0.92	1.00

Feature explorer (full training set)



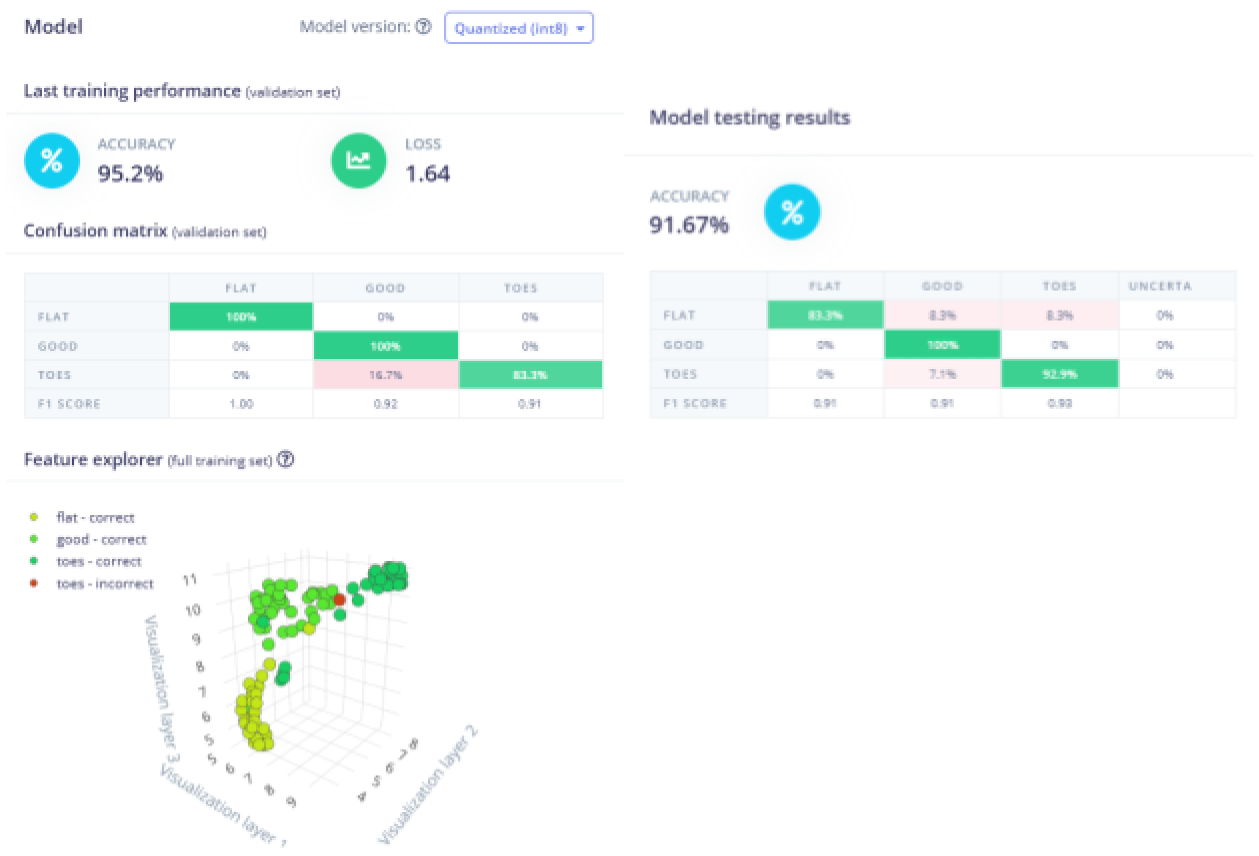
ACCURACY 100.00%

	FLAT	GOOD	TOES	UNCERTA
FLAT	100%	0%	0%	0%
GOOD	0%	100%	0%	0%
TOES	0%	0%	100%	0%
F1 SCORE	1.00	1.00	1.00	

Figure 12: Results from the training data (left) and the testing data (right) for the optimized settings.

Confirmation

To make sure that the testing can be replicated and wasn't just overtrained to the initial training data, the data was rebalanced and rerun with the same parameters. The results were slightly different, but still very good.


Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 95.2%

LOSS 1.64

Confusion matrix (validation set)

	FLAT	GOOD	TOES
FLAT	100%	0%	0%
GOOD	0%	100%	0%
TOES	0%	16.7%	83.3%
F1 SCORE	1.00	0.92	0.91

Feature explorer (full training set)

Model testing results

ACCURACY 91.67%

	FLAT	GOOD	TOES	UNCERTA
FLAT	83.3%	8.3%	8.3%	0%
GOOD	0%	100%	0%	0%
TOES	0%	7.1%	92.9%	0%
F1 SCORE	0.91	0.91	0.93	

Figure 13: Results from the rebalanced data. Left is the training data and right is the testing data.

7. Conclusion

The goal of this project was to determine if it was even possible to differentiate between a good pedal stroke and a poor one, and that goal was achieved. I had a stretch goal of implementing more sensors, such as force sensors, however I underestimated the amount of work it would take to implement the current solution with my level of programming experience. I was also hoping to have a partner to work with and labor could have been divided.

8. Future Work

The most obvious improvement to implement in the future would be to find a way to do live classification. It's one thing to look back at how one did, but it would be much better as a training (in the sports sense) device if the user could get instantaneous feedback. Assuming I could learn Python better, the way I would proceed would be to have all of the calculations done on the RaspberryPi. Instead of segmenting the strokes afterwards, a similar algorithm to what I did could be used, or an even better solution would be to mount a sensor such as a reed switch or hall affect sensor to the crank and a magnet on the bike that triggers once per each revolution. That signal would trigger the start and stop of each stroke and then the data would be pushed off to be processed and classified. A signal would then be sent to a display or smartdevice to let the user know what their last pedal was. For this to work though, it would all have to happen in less than 500 milliseconds due to the typical cadence of a rider. I have high confidence that it is possible, though.

Even without live classification, increasing the variety of riders and stroke styles should be investigated. It would be good to determine if each rider would have to train to their technique, or if a universal classifier could be used. Some analytical devices need a calibration step to personalize it to a user, so it wouldn’t be too bad, but it would be good to know. Also, increasing the total number of samples is always a good practice for machine learning.

9. References

[1] *Sports Analytics Market Size 2021 Global Industry Insights by Global Share, Emerging Trends, Regional Analysis, Segments, Prime Players, Drivers, Growth Factor and Foreseen till 2027 with Top Growth Companies*. (2021, May 21). MarketWatch. https://www.marketwatch.com/press-release/sports-analytics-market-size-2021-global-industry-insights-by-global-share-emerging-trends-regional-analysis-segments-prime-players-drivers-growth-factor-and-foreseen-till-2027-with-top-growth-companies-2021-05-21

[2] G. (n.d.). *Garmin Rally^{TM} RS200 | Power Meter Bike Pedals*. Garmin. Retrieved July 29, 2021, from https://buy.garmin.com/en-US/US/p/658593

[3] Bini, R. R., & Carpes, F. P. (2014). Kinetics and Pedaling Technique. In *Biomechanics of Cycling* (2014th ed., p. 47). Springer.

[4] *Pedal Torque*. (2002). Perfect Condition Ltd. http://www.perfectcondition.ltd.uk/Articles/Pedalling/index.htm

[5] *UCLA - STMicroelectronics SensorTile*. (n.d.). STMicroelectronics SensorTile Internet of Things Curriculum. Retrieved July 30, 2021, from https://sites.google.com/view/ucla-stmicroelectronics-iot/home

[6] *Edge Impulse*. (2020). Edge Impulse. https://www.edgeimpulse.com/