



DGMD S-14: Project Report

Happy Greens: Golf putt detection and prediction

Mikhail Stukalo

Contents

Introduction	3
Hardware and Equipment.....	3
System requirements.....	3
Wearable device	4
Golf equipment.....	4
Data.....	5
Data collection	5
Data cleaning	6
Modelling	8
Feature engineering and selection	8
Model results	9
Implementation	10
Further improvements.....	11
Conclusion.....	11
Links	12

Introduction

The use of wearable devices for sports training seems to pick up popularity among professional athletes and “weekend warriors”. When some of the sports applications like digital watches (e.g. Fitbit, Apple Watch) are specifically designed to track overall fitness activity, there is a place for more niche products specifically designed for one sport.

For this capstone project, golf is a sport of choice. I recently started playing golf and discovered that a so called “short game”, i.e. approach shots and putting, is probably a more important determinant of the performance than large swings. Therefore, I chose to use SensorTile as an aid for golf putt training.

In the course of the working on the project, I assembled a wearable, or more precisely an “attachable” device, that collects data of the movement of a putting golf club, and using CatBoost classification algorithm predicts the outcome of the putt. Currently, the performance of the model is decent, although still far from a production quality. Nevertheless, this attempt can be viewed as a first step towards developing a device specifically aimed at analyzing of a putt.

In this paper, we will follow the process of the development and the deployment of a full-stack prediction algorithm: starting from data collection to a deployment of a beta version on a Linux-based system.

The rest of the paper follows the pipeline of the project development:

- Hardware and equipment used for data collection. System requirements
- Data collection and pre-processing
- Feature engineering and model training
- Implementation/deployment

For the reference, you may refer to the Github repo containing the code for all stages of the project

<https://github.com/MikeStukalo/HappyGreen>

Hardware and Equipment

System requirements

- a SensorTile from STMicroelectronics kit together with a small cradle, a battery and a case (STEVAL-STLKT01V1)
- an expansion board that comes with SensorTile for debugging
- a Nucleo board to debug and flash the SensorTile
- a Linux-based computer. Raspberry Pi or BeagleBone can be used for data collection. However, as far as I understand, the deployment of a trained CatBoost model on Raspberry Pi is impossible. Therefore, for the deployment I used a laptop working under Linux Ubuntu 18.04.

Wearable device

Strictly speaking, the device I used for the data collection and the project implementation was not so much wearable as “attachable”. We will talk more about the placement of the device later in this chapter.

After debugging and flashing the SensorTile with FP-SNS-ALLMEMS1 function pack (Tutorial 8. See ‘Links’), I assembled the tile in on a smaller cradle. The assembly required some soldering. However, there is an option to keep the tile on an expansion board, although the device will look bulkier and less professional in that case.

Figure 1 Assembled SensorTile



Once assembled and flashed, the SensorTile communicates with a computer by means of Bluetooth Low Energy (BLE) signal.

Golf equipment

The project was conducted in a controlled environment. I performed putts using the same golf putting club and an artificial putting green.

Figure 2 Golf equipment



To ensure accurate distance measurements, I placed 2-, 3-, 4-, and 5-foot marks measured from the center of the hole.

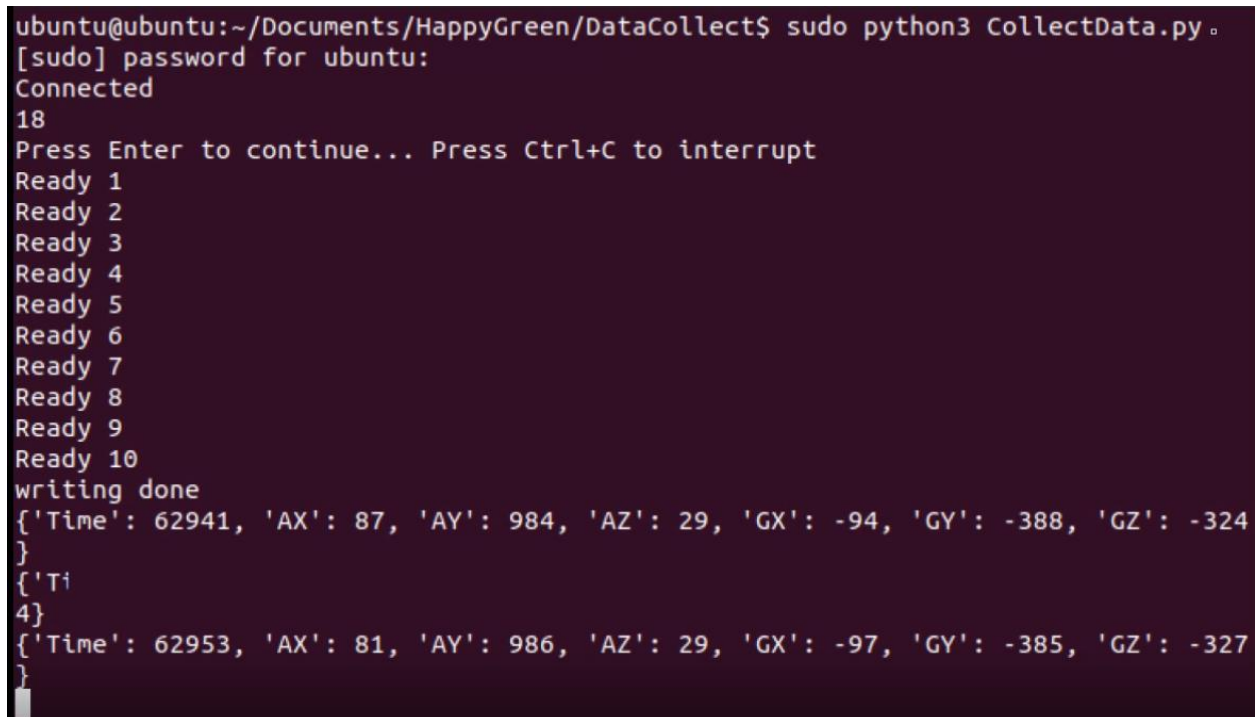
Data

Data collection

In order to collect raw data via BLE protocol, I wrote a python script utilizing Linux 'bluez' library and Python 'bluepy' package that allows to write GATT characteristic values and receive notifications (please refer to the Github repo for the script. See 'Links'). The data collection scripts was deployed on S.B.E. (Raspberry Pi 4).

From each of 5 distances I performed 50 putts, totaling 250 observations. The script recorded data as a pandas data frame and saved as a .csv file. The spotter who helped me with the data collection recorded the results of the putts. I wanted to make data collection practical. Therefore, instead of recording each putt separately, I recorded series of 20-30 putts with the goal of developing a putt identification algorithm.

Figure 3 Data collection interface

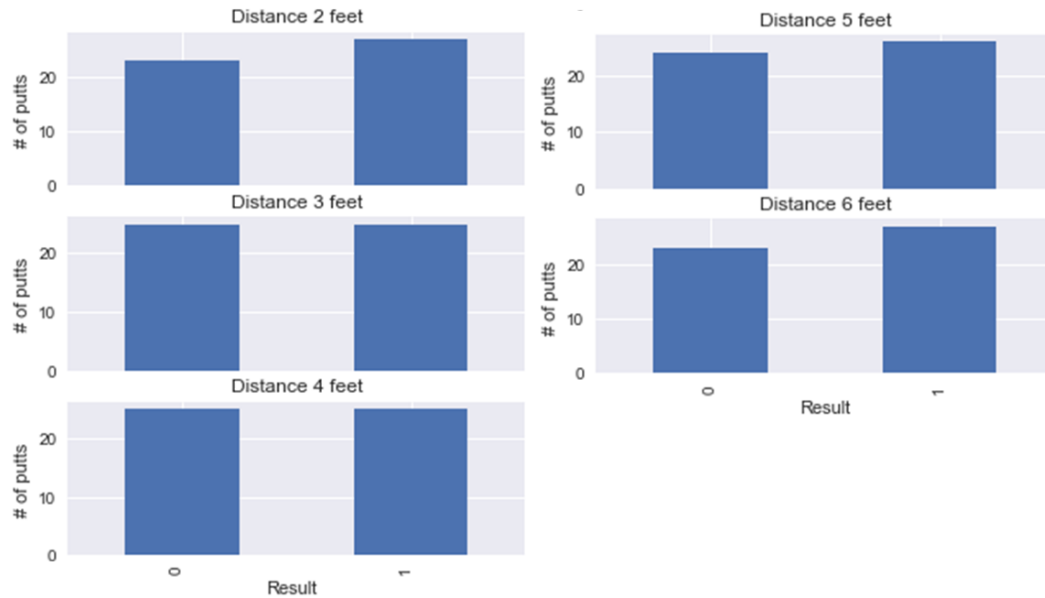


```
ubuntu@ubuntu:~/Documents/HappyGreen/DataCollect$ sudo python3 CollectData.py
[sudo] password for ubuntu:
Connected
18
Press Enter to continue... Press Ctrl+C to interrupt
Ready 1
Ready 2
Ready 3
Ready 4
Ready 5
Ready 6
Ready 7
Ready 8
Ready 9
Ready 10
writing done
{'Time': 62941, 'AX': 87, 'AY': 984, 'AZ': 29, 'GX': -94, 'GY': -388, 'GZ': -324}
{'Time': 62953, 'AX': 81, 'AY': 986, 'AZ': 29, 'GX': -97, 'GY': -385, 'GZ': -327}
```

When collecting the data, I strived for ensuring that the dataset is well balanced. I collected data on 50 putts from each distance aiming to have 25 successful and 25 unsuccessful putts in each series.

The data was collected with the frequency of 1 observation each 30 milliseconds. This parameter was set by default in **FP-SNS-ALLMEMS1** function pack. In hindsight, I believe that the precision of the model could have been improved if the data had higher resolution (e.g. each 10 ms). Nevertheless, even with low resolution I managed to build an efficient putt-detection algorithm and achieve reasonably high model accuracy.

Figure 4 Distribution of successful (1) and unsuccessful (2) putts.

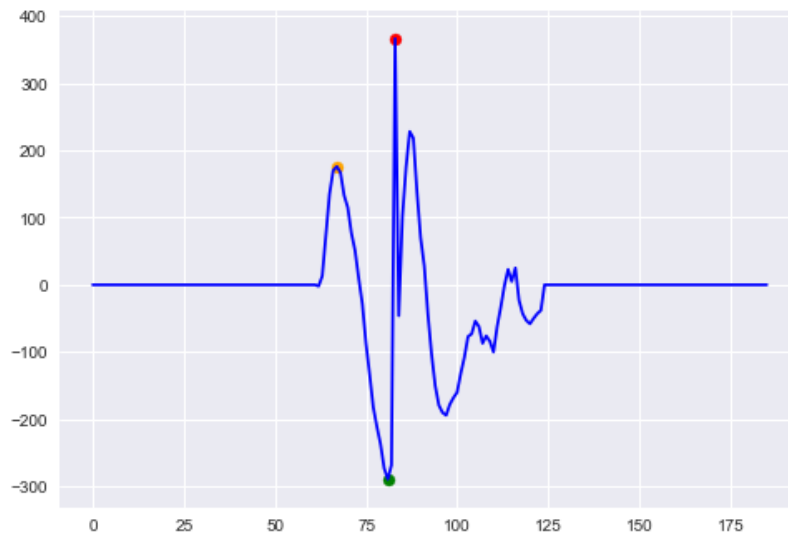


Data cleaning

The key step in cleaning the data was developing of a putt-detection algorithm. This algorithm was further used in the implementation of the prototype.

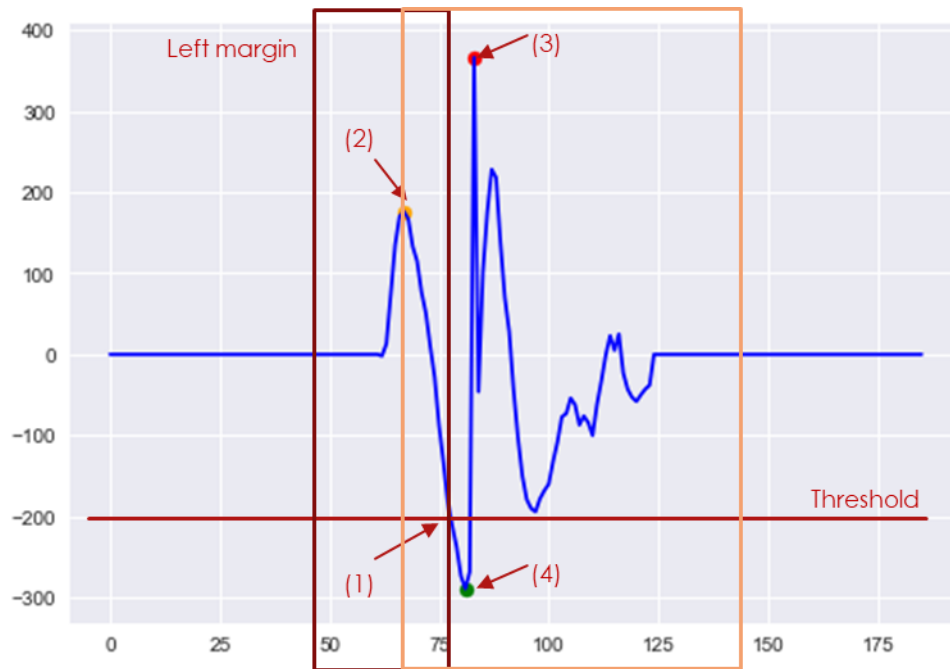
To detect a putt, I focused my attention on the accelerometer data along Z-axis. A typical putt data looks like the one depicted on Figure 5. The dots on the figure mark the important stages of a putt that I wanted to capture in the algorithm: the orange dot shows the beginning of the swing back, the green dot is the beginning of the swing forward, and the red dot depicts the end of the put.

Figure 5 Putt data. Accelerometer Z-axis



To implement the detection algorithm, I used the following heuristics, best described using notations on Figure 6.

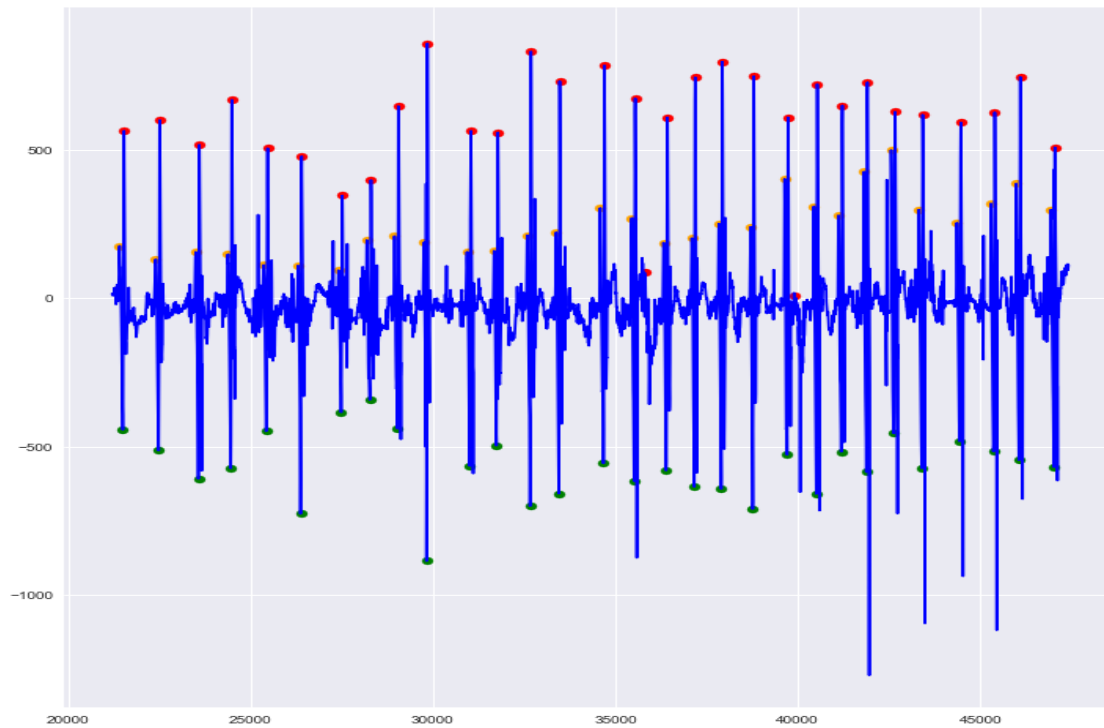
Figure 6 Putt detection



-
1. Move along timeline until an observation below the threshold (1)
 2. From observation (1) step back the timeline by a predefined number of steps (left margin depicted by a maroon square) and find a local maximum (2).
 3. From point (2) establish a predefined window (depicted as an orange square) and find a local maximum in that window (3).
 4. Finally, find local minimum in the range of observations between point (2) and (3).
-

Such heuristics yielded very precise putt detection algorithm, that was able to correctly identify all 250 putts in the collected data, with just a handful of false positives that were easy to spot and exclude from the analysis.

Figure 7 Detection results for the series of 30 putts



Modelling

Feature engineering and selection

It is intuitively clear that the most important stage of the putt is a swing forward. This part of the putt affects the movement of the ball and ultimately dictates the result of the putt. Therefore, after detecting all the putts in the dataset, I created features for the subsequent modelling using only the last stage of the move (the curve between points (4) and (3) on Figure 6).

After experimenting with small number of features and simpler models (Logistic Regression, SVM), I realized that the granularity (frequency) of the collected data was probably not enough to get away with using a simplistic approach to feature engineering.

As the result, for each axis data of the accelerometer and the gyroscope I created a set of derivative features attributed to each observation. The features included: minimum, maximum value and the last value, the range of values (i.e. the difference between minimum and maximum), the mean, and the standard deviation of values for the given putt, as well as the mean and the standard deviation of observations re-centered by the minimum value. I also calculated the slope of the line between the extreme points of the putt (last value minus first value divided by the time delta). Also, I calculated vector sizes (Euclidean distances) for all three axes of the accelerometer and the gyroscope and the derivative features listed above for those vector size.

It is clear that many of the features will exhibit multicollinearity just by design. Therefore, I chose a gradient boosting model which is known for high tolerance to multicollinearity.

The features were merged with the distance feature and true labels both of which I entered manually using the record from the data collection experiment.

I divided the resulting dataset in the proportion 30%/70% (or 75 observation to 175 observation) into a test and train sets. The division used stratification approach, i.e. the resulting sets were randomly selected, however, the randomization algorithm ensured that the sets end up having balanced observations in terms of putt distances and the true labels.

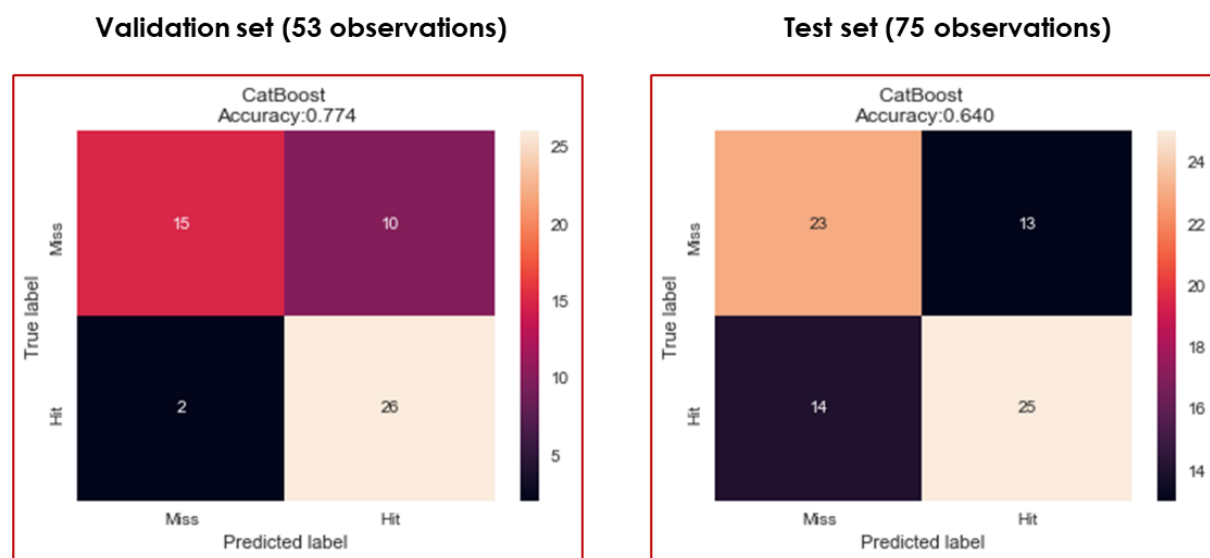
Model results

The implemented model uses CatBoost classification algorithm developed by Yandex (<https://catboost.ai/>). This algorithm is well known for high accuracy “out of the box” and for the ability to automatically prevent overfitting by using built-in cross-validation. To enable the cross-validation I further sub-divided the train set into the purely train set (122 observations or 70% of the initial train set) and the validation set (53 observations).

Nevertheless, the large number of features (72) with relatively small number of observations (122 for the train set) led to severe overfitting of the model. I made a decision to exclude all features related to X-axis (corresponding to moving golf club left/right) and vector sizes, ending up with 37 features.

The result of the model training was decently high with accuracy around 0.7 and without any apparent bias in predictions.

Figure 8 Confusion matrices for the validation and the test set

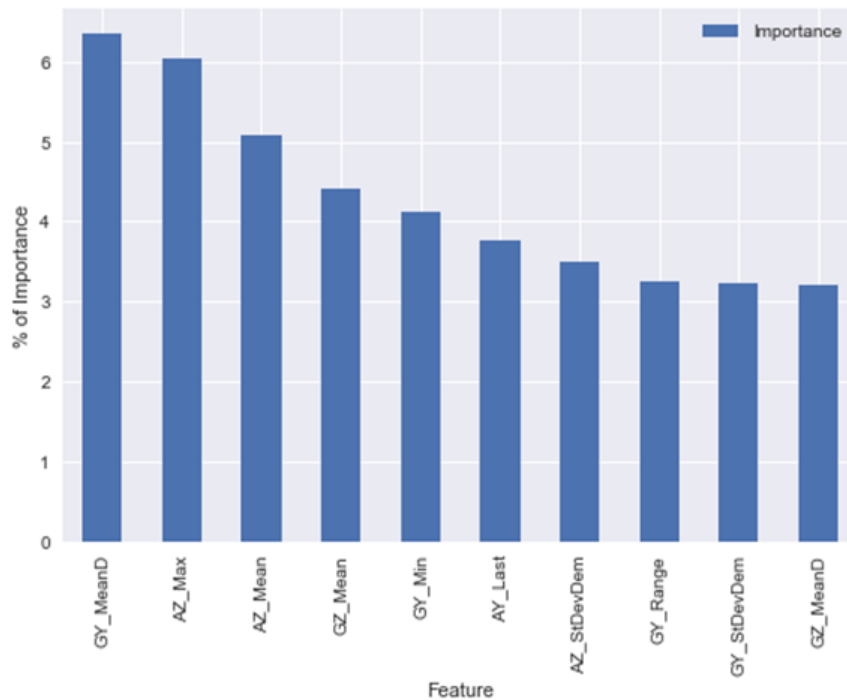


The analysis of feature importance shows that among top features are those related to the accelerometer data along Z-axis (moving the club away and to the ball) and gyroscope data along Y-axis (bringing the club back the ball level). These results correspond well to the intuition behind the data. Interestingly, the distance feature albeit important, did not make it to the list of top-10 features. I tend to attribute it to the fact that the experiment was conducted on a flat surface with some incline built-in the design of the hole. Therefore, most of the misses were due to the putt being not strong enough.

However, once the ball has enough force to reach the hole, probably the difference between the distance of the putt does not matter that much anymore.

Figure 9 Top-10 features by importance

In variable names: G stands for 'Gyroscope', A – 'Accelerometer', Y and Z for Y-axis and Z-axis. 'Dem' for a variable calculated based on observations corrected by the minimum.



Implementation

Once the model was trained, I developed a python script that utilized all previous steps to make a prediction of a putt success. The script:

1. Establishes connection with the SensorTile and collects raw data for 3 seconds
2. Detects a putt (or announces that the putt was not detected if the data pattern differs from the one described in the discussion of putt-detection)
3. Creates features and feeds them into the trained model
4. Announces the prediction.

The video of the working prototype is available on <https://youtu.be/--KsTxibZ40>

I hoped to deploy the prototype on the Raspberry Pi used for the initial data collection. However, it seems like CatBoost does not support Pi architecture and I had difficulties compiling the library from the source. I found some workarounds for this issue by using *piwheels* service, however, it was easier and more practical to deploy the script on a laptop running under Linux.

Further improvements

Although I am satisfied with the accuracy of the model, given an extremely limited time available for the data collection and the project prototyping, I strongly believe that the performance can be significantly improved.

For the further research I suggest collecting more data with higher granularity. Larger dataset and extended sets of observations for each putt, may lead not only to the improvement of the model, but potentially to the significant decrease in the number of features required for modelling. This, in turn, may allow to increase the speed of the algorithm as well as experiment with simpler classification algorithms like Logistic Regression and SVM.

Another improvement may be made by collecting the data from larger range of distances and with higher granularity of the distance variable.

Finally, to make the project even more realistic, the data collected on the further steps of development may include data obtained in field experiments using real golf putting greens with uneven terrain.

Conclusion

This was a fun project that allowed me to explore both the hardware and the software sides of the wearable device designs.

The project and this paper are designed to illustrate the early stages of the development of a wearable device. It should be obvious that the real-life commercial application would require much more work on the accuracy of the model, the proximity of the training data collection to real-life environment of a golf putt as well as require the development of a user-friendly interface for the app.

Nevertheless, by implementing this project I could learn and hopefully demonstrate that the application of wearable devices for such a niche task as a golf putt not only can be accomplished but also can bring a lot of joy and fun.

Links

Project Github repo

<https://github.com/MikeStukalo/HappyGreen>

Full project video

<https://youtu.be/2S7rDOkyQkU>

Video of the prototype demo

<https://youtu.be/--KsTxibZ40>

SensorTile Tutorials

<https://sites.google.com/view/ucla-stmicroelectronics-iot/home>