Performing Kick Detection on a Soccer Player: a Classification Approach

TU Delft - IN4398: Internet of Things Seminar - final report

E. Camellini
4494164
E.Camellini@student.tudelft.nl

K. Dreef
4107861
K.Dreef@student.tudelft.nl

ABSTRACT

In sports nowadays it is getting more common to gather data on how a player performs during a match in real-time, and to show this information to coaches and fans. In soccer this is less present: data is used to measure performance, but often this is done only to provide information to coaches and trainers. Furthermore, current systems don't say anything about certain motions of the player like kicking. We propose a system that is able to sense the acceleration of soccer players' legs and advertise it in real-time to a PC using Bluetooth Low Energy. On the PC side we implemented a classification algorithm, using k-Nearest Neighbors, that is able to receive the sensed data and use it to classify the player's movement into three different classes, namely: standing still, running, and kicking. As features it uses the maximum acceleration magnitude reached during a fixed time window, for each of the legs. The results show that the system can classify the data accurately with a misclassification rate of 1.25%, on average. Further testing also indicates that using these features allows the training set of a player to be used for the classification on any other player.

1. INTRODUCTION

Nowadays soccer teams are using more and more tools to gather information on the performance of the players during trainings and matches. Most of the collected data is analysed over-time and used to improve the players' skills. It would be interesting to have players' data available in real-time during a game and to be able to present this data not only to coaches and trainers, but also to the fans. For example it could be useful to measure the heart rate of a player during a penalty kick, or to detect when and how fast a player kicks the ball and provide these data to the fans and to the team staff through a screen in real-time. This is already done in other sports, like tennis (e.g. speed of serving), but not in soccer.

We noticed that the devices that are currently available for soccer focus on the physical conditions of the players, but there are no tools available that focus on providing data in real-time to the fans, and in particular none of the available tools can be used to collect data on the movement of the legs or on the kicking statistics during a game. A system to be used to gather and provide this kind of data should consist of a small sensing device, that must be attached to the player's leg without influencing the playing performance, and should be able to send the gathered data in real-time in order to provide useful information.

The goal of this work is to design a system that can classify when a soccer player is running and kicking in real-time, and to implement the classification algorithm for the kick detection. The main research challenges are:

- Determining a way to classify the state of the player (running, kicking or none of the two) using the data of a leg motion sensor.
- Finding a small-enough sensing device equipped with a motion sensor, a BLE module and a battery that can last for at least 45 minutes (one half of a football match) while gathering and sending data at a high enough frequency in order to guarantee an accurate classification;

Sections 1.1, 1.2 and 1.3 focus on describing in a clearer way our motivations, the related state of the art and the research contribution of this work. Section 2 describes the methods we followed, Section 3 focuses on the experimental results and on their analysis. Ultimately, we state our conclusions in Section 4 .

All the code we wrote is publicly available[1]. In the resources of the PC code repository we also published all the data that we gathered, and the data set that we used for the experiments.

1.1 Motivation and use cases

Having motion data from the players' legs in real-time and being able to detect when a player kicks could add innovation to a soccer game experience. Examples of use cases could be:

- **Real-time statistics:** the motion data gathered from the legs of a player could be used to determine some

[1] `https://bitbucket.org/in4398iot/`

information about a kick: its power or the acceleration of the ball. These data can be useful to the coach and can be directly shown to the fans.

- **User interaction:** properly designed apps on smart devices could use these real-time data to perform some actions that depend on the state of the player (e.g. running, kicking). For example, smart devices of people at home or at the stadium could vibrate with an intensity that follows the running speed and the kicks of the player: this is what nowadays the joystick does while playing a soccer video game, for example.

Gathering more data with the same devices could enable more use cases (e.g. another sensor of the same kind could keep track of the heart-rate profile of the player, and this could be correlated with the kick information).

1.2 Related work

In this field there has already been done a lot of research and some popular devices are already available. The main professionally used system is the Viper Pod [2]: this system is used by coaches and trainers to determine the player and team condition live, but it gathers no data related to the leg motion. For other sports there are sensors like Zepp[3], which focuses on collecting data on certain motions of the player, but this system is not available for soccer and its usage in other sports is limited to one sensor for each player at the moment. To our knowledge there are also no other systems that could be used to provide real-time data to soccer fans.

For what concerns the kick detection, [5] shows a method to do it using visual data gathered from a smartphone camera, but as far as we know no work has been done in this field using motion data gathered directly from the legs of a soccer player.

1.3 Contribution

The main contributions of this work with respect to the state of the art in this field are the research performed on the kick detection, the idea of sensing the players' motion by applying sensors directly on the legs, and the idea of doing classification on data gathered in real-time using only BLE advertising, without the need for the sensing devices to run a connectable Bluetooth service. More details on each one of these innovations can be found in the Sections 2, 3 and 4.

We also contribute to the future research in this field by publishing all the code that we wrote and especially the data set that we built, together with the raw sensor readings: these can be used to perform other experiments in this field.

2. METHODS AND IMPLEMENTATION

2.1 The system

The envisioned system should be player-friendly and easy-to-use, meaning that the motion sensors need to be easy to place onto a player's leg without hindering the player's performance. The data gathered by the sensors should be sent to a PC in real-time. The PC runs software that is able to process the data and output the results to a screen. In this work, as stated in Section 1, the goal is to create a program that focuses on detecting when the player kicks, meaning that we need to solve a classification problem.

To perform the kick detection using the motion data of the leg there are two choices: the player can be equipped with one sensor on the kicking leg or with two sensors, one on each leg. We gathered data with both the approaches and after some exploration we decided to adopt the second one, so to use two sensors (see Section 2.4 for details).

The sensing board we used to design the system is an nRF51 board from Nordic[4] equipped with an LIS3DSH motion sensor and a BLE module.

We decided to send the sensor data in real-time from the sensing boards to the PC using Bluetooth Low Energy (BLE) advertising packets, in order to avoid the need to establish a BLE connection between the two devices. The PC puts the received data in a fixed-size buffer: the size of this buffer is the size of a batch of data needed to output a classification result, and we will call it batch size. When the buffer is full the PC flushes the data that it contains, passing them to the classification algorithm.

The output of the classification system can then be used for the use cases that we described in Section 1.1.

A high level architecture of the system design is shown in Figure 1.

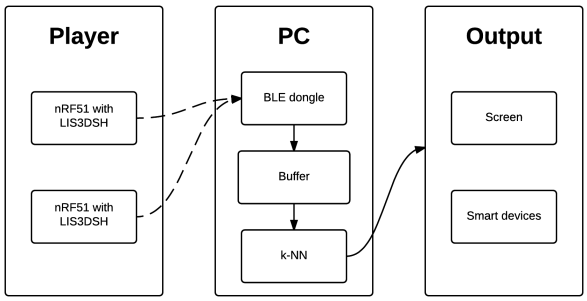


Figure 1: System overview

For the classification we decided to use a supervised machine learning approach: the k-Nearest Neighbors (k-NN) algorithm. The data are classified in three categories:

1. Standing still: the player is standing still or making negligible leg movements;
2. Running: the player is running;
3. Kicking: the player is kicking, one leg stays still while the kicking leg accelerates.

The last category is of course the one we are most interested in.

[2]http://statsports.com/technology/viper-pod/
[3]http://www.zepp.com/

Since k-NN is a supervised machine learning algorithm it is important to collect pre-labeled data for every class and to test if different training sets have an impact on the precision of the classification. This aspect is important, because depending on the results that we obtain with these tests we can check if the system needs to be retrained for every player or if a single training set can be used for almost all players.

We decided to implement the PC side components using the Python programming language.

2.2 Board and communication

The nRF51 board is programmed to perform two main functions:

- Gather accelerometer data from the leg movement of the player;
- Send the sensed data to the PC side of the system.

In order to send the data in the most easy and low-power way we decided to embed them in the Bluetooth advertising packets, so that there is no need to run a Bluetooth service on the board. In this way the sensed data can also be retrieved by the PC without the need to establish a Bluetooth connection: it is sufficient to extract them from the advertising packets that are received periodically.

The two main constraints that the Nordic SDK, the Nordic SoftDevice [3] and the BLE specification impose on advertising data through BLE are:

- A maximum advertising frequency of $50Hz$ [3];
- A maximum payload of 31 bytes, that must contain also the device name and the header for every field in it [1].

The firmware we programmed emits Bluetooth advertising packets with a payload that contains six 16-bits integer values and it uses a 50 Hz advertising frequency. A sensor sample is composed by three 16-bits values (i.e. x, y, z accelerations), so the described setup corresponds to two samples sent every 20 ms (i.e. 100Hz sampling frequency, 50Hz advertising frequency).

For what concerns the battery, the board is equipped with a supercapacitor as power source. Upon fully charged this capacitor can deliver 2.5V, which will decrease upon time. The board (nRF51) needs at least 1.8V to continue to operate, meaning it can supply 1.944 mAh.

2.2.1 BLE on the PC side

The focus of this work, for what concerns topic that are not related to the classification, is not to build a real-time prototype (as described in Section 1 and for the reasons described in Section 2.3), but to design the system, so to test and motivate all our decisions: for this reason we also implemented a simple version of the module that can be used for the BLE communication on the PC side.

It uses the linux BlueZ protocol stack[4] tools to read the advertised values in real time: the commands used are hcitool lescan (low energy scan) to read the advertising packets and hcidump to see their byte representation and to extract the needed byets from the advertising payload. In a working prorotype, this program could easily pass the sensor samples to the trained classifier in real-time in order to obtain the corresponding classification.

2.3 Motion sensing

For the leg acceleration sensing we chose to use the LIS3DH sensor equipped on the nRF51, as stated in Section 2.1.

The board contains, next to a nRF51 SoC, the LIS3DH [2], which is a very low-power accelerometer. This accelerometer is hooked up to the nRF51 in a master-slave configuration in which it is the slave. It can then be programmed to send an interrupt to the nRF51 every time the sensor values change. The values that should be written to the register to set it up with the correct settings can be found in Table 1.

Register	Value
CTRL_REG1	0x57
CTRL_REG3	0x40
CTRL_REG4	0x80
CTRL_REG5	0x0C
INT1_CFG	0x7F

Table 1: Register values to initialize the LIS3DH accelerometer.

Ultimately, because of time constraints, and lack of resources and documentation we weren't able to get this setup to work. The LIS3DH needs an extra input signal (*CS*) to be able to read/write from/to the registers using SPI, and it isn't standardized in the nRF51 SDK. There is only a *SS* signal left in the SDK which you can connect *CS* to, but it has a different purpose than *CS*. This problem emerged while trying to create a connection between the accelerometer and the nRF51 in a standard master-slave configuration.

Due to the described problems related to the data gathering phase through the board, we decided to adopt a different solution that still allowed us to perform the classification experiments on real-data: we used an Android application to gather motion data from the players. This application senses the accelerometer data for the x, y and z axes from the sensor included in the smart device on which it is installed, and saves these data into a file, together with a label and a time-stamp. The label must be pre-set from the user interface and can be 'S', 'R' or 'K' (standing still, running or kicking respectively): when it is set the app starts logging the sensors data with a frequency of $100Hz$.

We decided not to modify the system setup that we designed (see Section 2.1), so we equipped the soccer player with two devices (i.e. Smartphones): one on each leg.

2.3.1 The data set

Merging the data from the two separate log files into a single data set that contains the label and both left and right

[4]http://www.bluez.org/

accelerometer data, as required by the k-NN classifier, was done through a Matlab script. Because of small time differences between the sample timestamps of the two log files the data needed to be interpolated, in order to align them and at to make sure that at every line in the final data set contains data from both the sensors.

The data also needed some pre-processing before it was ready to be used for the classification. The kicking data as can be seen in Figure 2c contains many parts where nothing happens. This is data collected in between the various kicks, and is of course not needed as training data for the kicking, so we had to remove some useless sensor samples. Furthermore, the beginning and ending parts of the data needed to be removed to gain a good training set, because they contain noise due to equipping the player with the sensing devices after having started the application and having pressed the correct button.

After the described pre-processing is done the data set is ready to be used to train or test the classifier. Its composition (i.e. columns) can be seen in Table 2.

For more details on what data we gathered for every experiment see Section 3.

x-left	y-left	z-left	x-right	y-right	z-right	label

Table 2: Header of the built data set: the first three columns are the acceleration values of the left leg, for the three axes, then the consequent columns are the right leg values and the last one is the label ('S', 'R' or 'K').

2.4 Classification

As mentioned in Section 2.1 the classification is done using the supervised machine learning algorithm called K Nearest Neighbours (k-NN). This uses a training set that contains data from the sensors with a predetermined label. From this data, features are extracted and then used to classify new incoming instances. In this section we describe how we implemented the algorithm, which features we decided to use and why.

The k-NN training consists in saving the features computed for the training set instances, together with the corresponding known label. A list of n computed features can be seen as a point in an n-dimensional space: from now on we will call distance the distance between two of these points.

The classification of new data is then performed in a few steps: First, the features are extracted from the incoming data to obtain a new feature list. Second, the distances are calculated between the computed feature list and all the saved training feature lists. Third, with the computed distances it is possible to figure out which instances in the training set are the most similar (the nearest) to the one to classify.

Finally, the instance will be classified with the most common label in the K closest similar training instances (neighbours).

In our implementation of the k-NN the features are not extracted for every single line in the data set, but for batches of them. A batch is a fixed-size set of consequent lines that have the same label. The training set is simply split in batches, and for every batch the features are computed and stored as described. Before a label transition (e.g. when the player starts running) it could happen that the last samples of the considered label are not enough to fill a batch: in this case they are discarded (the same is done with the testing sets). In a real prototype, the sensor samples collected in real-time should be also grouped in batches: this could be done using a buffer as described in Section 2.1.

The features that we decided to use for the classification are the maximum magnitudes of the acceleration of the left and the right leg in the considered batch of data. Equation 1 shows how the magnitude is computed on a single sensor sample of a single leg, then the maximum value obtained over a batch is extracted as feature.

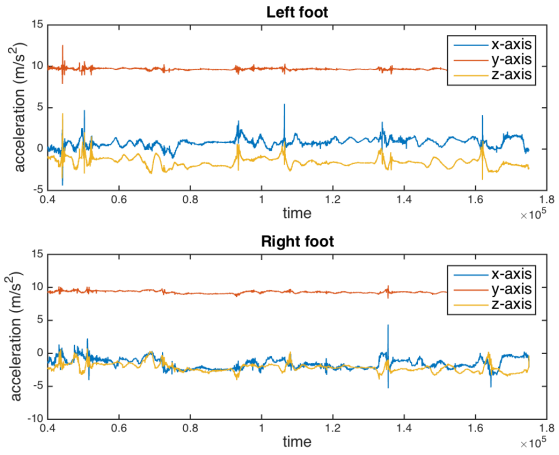
The decision to use two sensors was based on the data shown in Figure 2. The data that are shown here are the sensor data for each leg during the different classes. It can be seen that the magnitude of the two legs differ significantly depending on the activity, and because of that we concluded that the magnitude of the acceleration of both legs could be an interesting feature for classifying the data. The intuition is: by using two sensors it should be clear that during a kick one magnitude is relatively small in comparison with the other one (the one of the kicking leg). This would suggest that the movement should be classified as a kick. Using one sensor it would be problematic to determine the difference between kicking and running, because there is no way of knowing what the other foot does. We performed some tests and the outcomes suggest that the difference in the magnitude of the kicking leg acceleration while running or during a kick is not significant enough.

What we just described is also the reason why we need to use batches and to select the correct batch size: the features we use must be computed on a time window that includes the whole movement of a kick, and the movement of both legs while running.

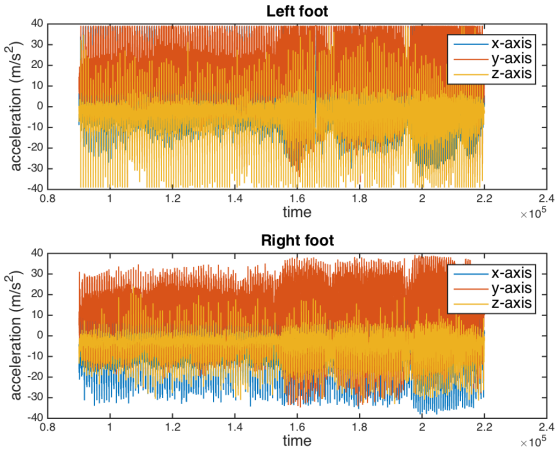
$$magnitude = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

Another feature that we tested is the frequency with which the acceleration oscillates: we tried to compute the Fourier transform of the acceleration profile for each leg. We did it because the running movement is periodic, while kicking and standing still movements are not, so this distinction could have helped in the classification. The problem with Fourier is that it needs a considerably larger window size to give accurate results (approximately 4 to 5 times the time that a kick takes). Because of that, the kick detection would be difficult to be performed in real time, so we discarded this approach.

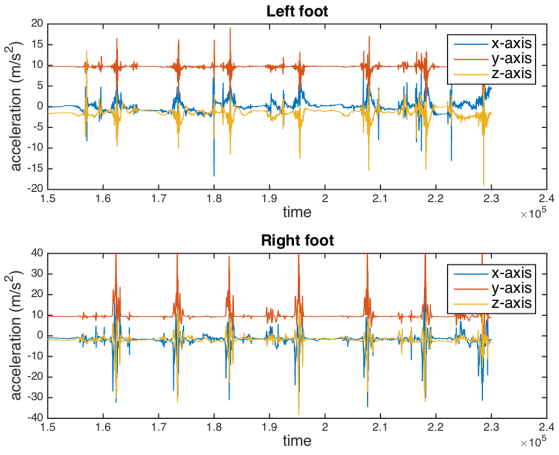
For the described reasons, we decided to use only the two magnitudes as features, and in this way it is also easy to plot the instances in a two-dimensional space in order to check if the features we selected are really useful to distinguish the three clusters (i.e. three classes). Because of that, we



(a) Standing still data



(b) Running data



(c) Kicking data

Figure 2: Different sorts of training data.

selected to use the Euclidean Distance as distance measure between couple of features: it works good in a situation with two features like this one.

In this implementation of the k-NN it is possible to tune the algorithm by adjusting the batch size and the number of neighbors to consider (the 'k' parameter). A smaller batch size gives the end user a system that gives an updated classification more frequently, but it also means less data for every batch, which can directly result in a decrease in accuracy. The optimal batch size is in the end a trade off between classification frequency and accuracy, and it should include the whole kicking movement.

To evaluate the accuracy of the classification we adopted the k-folds (10-folds in particular) and leave-one-out cross validation techniques, and also the classic training-testing method on different data sets. By applying these techniques it is possible to see how accurate the system is in classifying the data without falling into an overfitting scenario. See Section 3 for more details on how we evaluated the accuracy of the system in every experiment, and on the results that we obtained.

3. EVALUATION

In this section we show how we tested and evaluated the classification system described in Section 2.4.

3.1 Experiment 1: features and cross-validation

This first experiment consists in gathering data from two players to build two distinct data sets. K-folds and leave-one-out cross-validation techniques are then used to evaluate the classification results on the single data sets. In this way the k-NN is trained and tested on the same player.

3.1.1 Experiment design

For this experiment we gathered data from two different players running the sensors for one minute for every class. After the data pre-processing, as described in Section 2.3, we obtained two data sets (one for each player) composed as follows:

- 130000 standing still (i.e. 'S' labeled) samples;
- 130000 running (i.e. 'R' labeled) samples;
- 40000 kicking (i.e. 'K' labeled) samples;

We also created a second, normalized (or "fair"), version of the data set for each player where every label has 40000 samples, in order to avoid to over-train some of them with respect to the other ones.

We first trained the classifier on both the data sets and plotted the features to see how distinct are the three classes. Eventually we applied k-folds (with 4 and 10 folds) and leave-one-out cross-validation techniques on each data set, trying to tune the batch size. We chose a K value of three for the k-NN: this number of neighbours is sufficient to determine the correct class in this situation, where the clusters are distinct enough, and it is the value that resulted in the best accuracy.

3.1.2 *Results*

The plot of the features after the training on the first player is shown in Figure 3c. We can see that using a time window of 100 samples (i.e. 100 samples per batch) makes the classes more distinct: the more the batch size is shrink, the less distinct are the three classes (see Figures 3b and 3a) and this can result in wrong classifications, but it also mean that the system can classify faster in a real-time scenario because it needs a smaller amount of samples to output a classification. We obtain the same results for the second player, as shown in Figure 4. For all these plots we used the normalized versions of the data sets, with the same number of batches for every class.

We applied the K-folds cross validation technique with 4 and 10 folds and a batch size of 100 on the normalized data set, we counted the wrong classifications over the total ones (i.e. misclassification rate) and we obtained the following results for both the players:

- Standing still is always classified correctly;
- Kicking and running are always classified correctly for one player, and for the other there is only one misclassification for both the classes over 40 classified batches. This result states that misclassifications, on the data we gathered, happen on average in 1.25% of the cases (1 time over 80).

We obtained the same results (in terms of misclassification percentage) using the not normalized data set, so having a bigger amount of samples for some of the classes seems to not affect the classification.

Shrinking the batch to 50 samples results in doubling the number of batches to classify over the same amount of data, and this also reflects on the accuracy: kicking is misclassified 4 times and running 2 times (both over 160 classified batches), so misclassifications happen in the 2.5% of the cases.

This effect even is more evident when we shrink the batch size to 25: the misclassifications of the kicks happen in more than 5% of the cases.

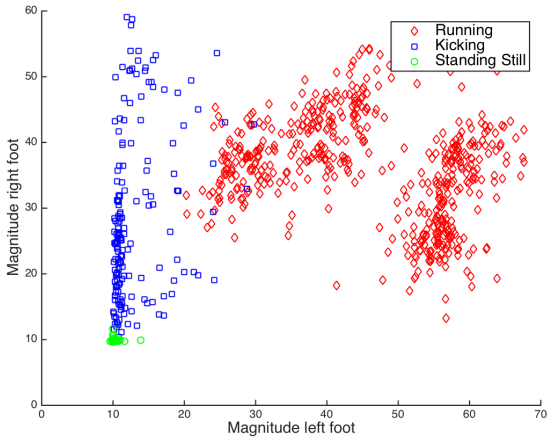
Since the algorithm is fast we also tried to apply the leave-one-out cross-validation technique and the results are the same.

3.2 Experiment 2: training and testing on different players

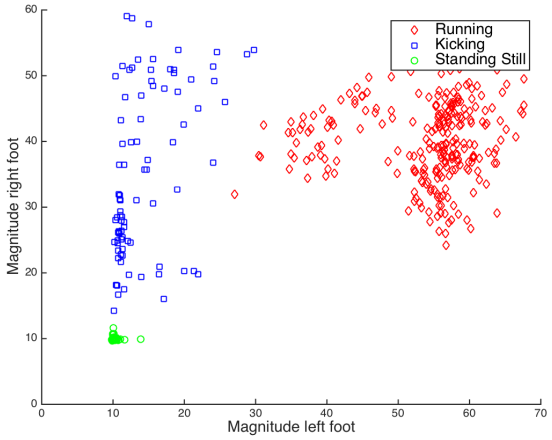
With this experiment we wanted to prove that the classification accuracy is not significantly influenced by the player on which the system is trained, so that it is possible to train the classifier in a general way in order to obtain a model that can be used for anyone.

3.2.1 *Experiment design*

The data sets we used for this experiments are the same described in Section 3.1.1. To prove that is possible to use the same model for many players we trained the classifier on the data set of the first player and used the data set of the second one for the testing, then we inverted the roles of the two sets. If the data set can be used for all players the



(a) Features with batch size 25 - player 1.



(b) Features with batch size 50 - player 1.

(c) Features with batch size 100 - player 1.

Figure 3: Different sorts of training data.

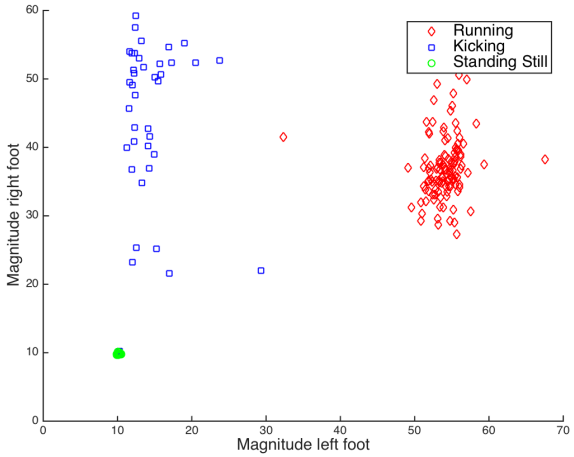


Figure 4: Features with batch size 100 - player 2.

results should be similar.

3.2.2 Results

We show the results of the experiment applied using a batch size of 100. The differences in the accuracy due to changing the batch size are the same as described in Section 3.1.2.

Training on the first player's data set and testing on the second one resulted in no misclassifications, while the opposite setup resulted in 1 misclassification for the kicking class and one for the running one, both over 40 classified batches. This means that misclassifications happen in 1 over 80 cases for both the classes, so in the 1.25% of the cases.

The result obtained is the same that we had with the first experiment, as described in Section 3.1.2, so the accuracy of the classifier is not influenced by the players on which the model is trained and tested.

3.3 Experiment 3: left foot kicking

With this experiment we wanted to prove that the classifier works also for a left-footed player if trained on data gathered from other left-footed players.

3.3.1 Experiment design

For the standing still and running classes we used the same data set of the first player that we already described in Section 3.1.1, while for the kicking we gathered 2500 new samples from the same player kicking with the left foot. We also normalized this new data set in order to have the same number of features for every class.

We first trained the classifier and plotted the features to see how distinct are the three classes in the left-kicking case. Eventually we applied k-folds and leave-one-out cross-validation techniques on it.

3.3.2 Results

We show the results of the experiment applied using a batch size of 100. The differences due to changing the batch size or to using the not normalized data set are the same as described in Section 3.1.2.

The plot of the features after the training is shown in Figure 5 (the two axes are inverted in order to make it consistent with the right kicking plots shown in Section 3.1.2).

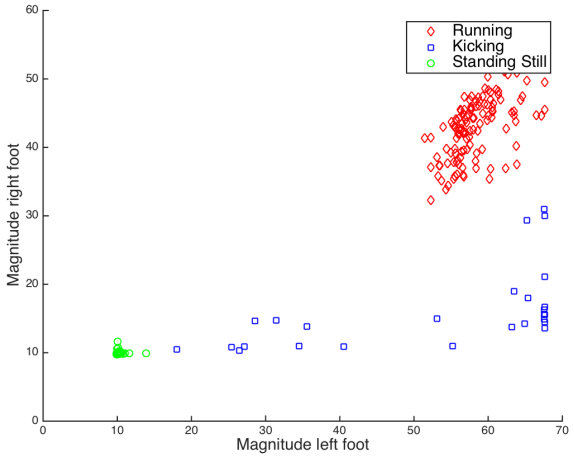


Figure 5: Features with batch size 100 using other foot to shoot - player 1.

We applied the K-folds (with 4 and 10 folds) and the leave-one-out cross validation techniques on the normalized data set, and we obtained only 1 misclassification over 25 classified batches in the kicking situation.

This data set is of course small with respect to the other ones, but the result is enough to prove that the classifier works also for left kicking scenarios.

3.4 Experiment 4: power consumption

We decided to perform tests on the nRF51 board although we didn't use it for gathering the data and we didn't build a working prototype. The purpose of this experiment is to check if the power capabilities of the board are sufficient to support the system in a real-scenario: to satisfy this requirement the battery should last at least 45 minutes (the half-time of a football game).

3.4.1 Experiment design

The board is equipped with the firmware described in Section 2.2, so it advertises packets at a 50Hz frequency through Bluetooth Low Energy, and every packet contains two accelerometer samples of three 16-bit integer values (i.e. one for each axis). Because of the reasons describe in Section 2.3, the values sent are not real sensor readings, meaning that the accelerometer is also not used in this test case.

The experiment consists in starting the firmware execution when the battery is fully charged and timing how much time it lasts. We repeated this procedure three times.

3.4.2 Results

With this configuration the battery of the board lasted for an average of 38 minutes over the three attempts with a

standard deviation of around 1 minute. This amount of time is still not enough to make the system run for the whole half-time of a football match.

In this experiment the power consumption due to the accelerometer was left out, meaning that the power consumption of the board could be in reality a bit higher. Ultimately, a small improvement in the battery size and in the board power efficiency could make possible to have the board running for half a game (45 minutes). We think that the battery life should be improved of nearly the 20% to last enough.

3.5 Summary of results

The results of the experiments show that, when the right batch size is chosen, the accuracy of the classifier is surprisingly good. The results show that a batch size of 100 is the best choice because of the accuracy obtained, and it can classify the player's state quick enough so it would be good also in a real-time scenario (i.e. one classification per second with a 100Hz sampling rate).

Smaller batch sizes result in a faster classification, but are also less accurate. This is due to the fact that in less than one second it is difficult to gain enough information to determine the state of the player: a kick takes more or less one second and batches that contains samples spread over less than this time could have acceleration profiles that are not meaningful.

The accuracy does not depend from the player on which the system acts and the classification can be used indifferently for left and right-footed players, provided that it has been trained with data from the correct one of the two profiles. A model that mixes features from left and right kicking samples should lead to meaningless results. This is the outcome that we found more surprising, because it shows that the system behaves really good and it is does not show overfitting-related behaviours.

For what concerns the board, we showed that with this configuration it is not capable to last till match half-time, but this goal could be reached by slightly enhancing the capacity of the battery. It could be still usable for the penalty scenario, if activated only during a penalty situation. We were pleasantly surprised that it was possible to send values in real-time at a high-enough frequency (50Hz, two samples per packet) to perform a good classification, while maintaining a fairly good battery life on such a small supercapacitor, but it still needs further optimization and a larger battery.

It needs to be said that, even though the results are surprisingly good, the data set can still be considered small and more widespread tests should be performed.

4. CONCLUSION AND FUTURE WORK

In this work we implemented a software capable of doing kick detection on a soccer player. It uses the k-NN algorithm, adapted to our needs: with a batch size of 100 samples, utilizing the data of 2 sensors (one for each leg), and extracting the maximum acceleration magnitudes over data batches for every leg as k-NN features, it is possible to accurately identify when the player kicks, with a misclassification rate of 1.25%.

We also designed a complete system that includes the described classifier, and that can be used to perform kick detection in real-time. We made tests using an nRF51 sensing board, to proof that is possible to send data through Bluetooth advertising at a high enough frequency in order to perform a good classification. The results show that it is low-power enough to be active for 38 minutes when sending two samples every 20ms, so to support one classification per second when a batch size of 100 is used. With a small improvement it can be used for an entire half of a football match, and charged or substituted during the break. This is possible because the computational intensive part is done on a powerful computing device that receives all the data of the sensors (i.e. the k-NN is run on a PC).

4.1 Future work

While the initial results are promising we need to note that a lot of future work should be done, in order to give more significance to the results that we obtained.

The data set and the test group were small (i.e. 2 players, nearly 200000 sensor samples for each one of them) and because of that the results can not be generalized to all the soccer players. More tests should be done with a larger test group to see if one set of training data would work for all players.

Depending on the outcome of a test with larger test group and data set, it would also be interesting to take a look at an unsupervised learning approach instead of the supervised learning which is used now. With a unsupervised machine learning algorithm like k-means, that is able to automatically determine the cluster that are present in the data, the system would probably adjust better to the specific style of shooting of different players.

This work can be seen as a proof of concept, but we didn't build a working prototype. So other future work could be to build the whole system as designed in Section 2.1, and to implement tools that can use the data obtained from it (e.g. applications for smart devices that can connect to the PC part of the system, reachable through the Web, in order to enable the use cases described in Section 1.1).

5. REFERENCES

[1] BLE Advertising tutorial. `https://devzone.nordicsemi.com/tutorials/5/a-beginners-tutorial-advertising/`. [Online; accessed 17-Nov-2015].

[2] Documentation LIS3DH. `http://www.st.com/web/en/resource/technical/document/datasheet/CD00274221.pdf`. [Online; accessed 17-Nov-2015].

[3] nRF51 SDK 9.0.0. `http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk51.v9.0.0%2Findex.html`. [Online; accessed 17-Nov-2015].

[4] nRF51 Series SoC. `https://www.nordicsemi.com/eng/Products/nRF51-Series-SoC`. [Online; accessed 17-Nov-2015].

[5] T. Han, J. Alexander, A. Karnik, P. Irani, and S. Subramanian. Kick: Investigating the use of kick

gestures for mobile interactions. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 29–32, New York, NY, USA, 2011. ACM.