Assignment 3 - The Fitness Tracker System

Internet of Things, Security and the Cloud, University of Edinburgh

Ábel Kocsis
University of Edinburgh
abelkcss@sms.ed.ac.uk

Contents

1 INTRODUCTION

These days, more and more people work from home while sitting in front of their computer each day. However, not doing enough exercises can have serious health consequences in the long term, thus, making ourselves go out and do some exercises have never been so important as nowadays. One of the possible factors which can motivate people to exercise more is wearable devices [1], which aim to measure the different activities of a human based on their motions.

The main challenge of human activity recognition is twofold. Firstly, a system needs to collect different properties of the movements, thus, it needs to be decided which factors to use in order to distinguish between activities. Based on the current literature, most wearable fitness devices on the market measure 3-axis accelerometer readings [2, 3], however, the values of gyroscope and magnetometer can also improve the results of a system [4, 5], as it was also mentioned in our proposal document [6]. Thus, during designing a system, it must be defined which of these sensors should be used.

Secondly, an algorithm must be developed which aims to distinguish between these exercises. Nowadays, most of the activity trackers use machine learning to do so [7]. Some of them use more sophisticated methods [8, 9], while the others use Artificial Neural Network (ANN) [10] or K-Nearest Neighbours (KNN) [11–13]. The developed model must be robust enough to work with high accuracy even in case of a high noise as well as general enough to work on as many people as possible.

In this document, we present a fitness tracker system prototype. The system contains an application that can be run on an Android phone, an IoT device, which must be worn by the user and a back-end server on the cloud. Based on our current results, the system accomplishes 92% accuracy using a KNN machine learning model.

The rest of this document is divided into 7 sections. Firstly, the high-level design of the system is presented (Section 2) which is then followed by the discussion of each part of the system (Section 3-5). After that, the security aspects of the project is discussed (Section 6) and then the evaluation of the system is presented (Section 7). This is followed by possible future work (Section 8).

2 HIGH-LEVEL OVERVIEW

The main aim of the system is firstly, to distinguish between idle, walking and running activities successfully based on the readings of a wearable device and secondly, to provide information about the current activity as well as useful statistics to the user using a mobile application. The application should be able to do the following: displaying current activity, displaying milestones and levels of each activity, displaying statistics about the previous 7 days and receiving notifications when the application is not active.

The high-level overview of the system can be seen in Figure 1. First of all, an STMicroelectronics B-L475E-IOT01A1 Discovery IoT node is worn by the user on their hip. The device collects different data and sends these data to the Pelion Device Management System [14]. In that way, the system ensures that no time-consuming operation is performed on the end device. Thus, the only task of the wearable device is to collect and broadcast the data to the server.

Secondly, a Virtual Machine instance is deployed on the Google Cloud [15], which runs a python flask server [16]. This server gathers the data from the Pelion Cloud, stores the data in a database, runs a machine learning algorithm and communicates with the Application and the Firebase. Thus, it is ensured that all the time- and memory consuming operations are done in this, specific server.

Thirdly, to display the data to the user, an Android application is developed. This application communicates with the server running on the Virtual Machine and displays different information which was mentioned above. In order to be able to send push notifications to the application, Firebase [17] must be used, since this is the only way to perform push notifications to an Android phone.

In the next sections, all parts of the system are presented in details. Additionally, an overview of the design decisions, challenges and current limitations are also provided.

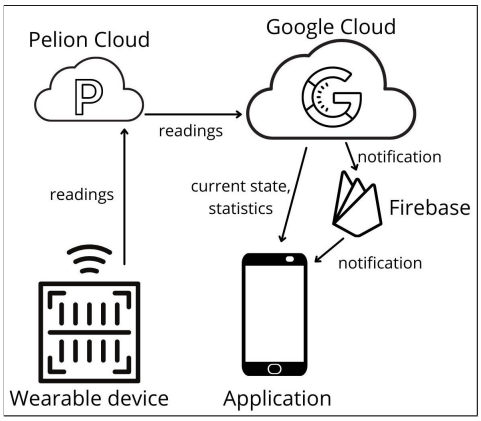


Figure 1: High-level overview of the system

3 THE WEARABLE DEVICE

This section presents the work regarding the wearable device, thus, the STMicroelectronics B-L475E- IOT01A1 Discovery IoT node. Firstly, the method of device wearing is presented, which is followed by a brief overview of the embedded program.

3.1 Wearing the device

A very important aspect of wearable devices is how and where to wear them since it can have an effect on the sensor readings as well as on the precision of the system.

As it was also mentioned in our proposal document [6], Evenson *et al.* [3] suggested that a wearable fitness tracker should be worn in the exact same place every time, close to the body. Research has shown that devices placed on the hip can outperform other devices [18, 19]. Thus, we decided to place our device on the hip as well, as can be seen in Figure 2.

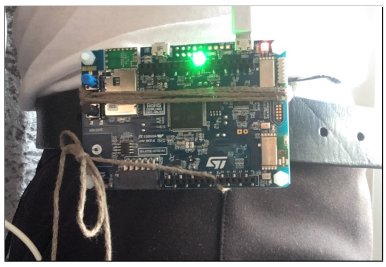


Figure 2: The device should be worn on the left side of the hip

3.2 The firmware

The firmware, thus, the embedded program runs on the IoT device, which collects the data and sends it to the Pelion Device Management System. The embedded program works as follows: firstly, the device is registered to the Pelion system. Then, the device collects the necessary data every 40 milliseconds. This time was determined based on previous studies and will be justified in Section 4. After that, the program sends the values to the cloud.

During the development, MBed Studio [20] was used since it provides an easy way to develop embedded applications. The mbed-os-example-pelion project [21] was used as a basis. The communication is done using resources during the Machine-to-Machine (M2M) communication. When the resources updated, the Pelion system is notified, which then can get these values.

In the proposal of our system [6], we proposed a method that aims to distinguish between activities based on the values of the accelerometer, gyroscope and magnetometer. However, since each sensor has three axes, it is (3*3*25 =) 225 values per second, which would not be sufficient to send one by one. In our interim report [22], we proposed a solution that created an object of the consecutive values and sent it together (9 values a time). However, as it found out, it was still not enough improvement, and in addition, the Pelion Device Management SDK for Python [23] could not deal with these encoded values. Thus, another way was developed in order to solve this issue.

In order to communicate the values efficiently, the embedded program converts all data from the sensors into strings, stores 4 readings, and sends these readings together to the Pelion system (4*3*3 = 36 readings a time) in the following format: sensor values are separated by ";" and different readings are separated by "|". The number of readings sent together (4) was determined based on experience: sending 5 data sometimes resulted in a buffer overflow. In that way, the data is broadcasted to the server using only one resource in an efficient way.

4 THE CLOUD

The design of the server can be divided into three parts. Firstly, an overview of the cloud computing design is presented. This is followed by the machine learning model, which contains not only the used algorithm but the challenges of data gathering as well. After that, the design of the Flask application is described, which contains the main threads in the code.

In order to provide fast computing and accessibility, our system uses a Virtual Machine Instance (VM) in the Google Cloud [15]. This VM Instance is running a Debian operating system on an e2-medium machine that uses 2 vCPUs and 4GB memory. These properties were used due to the credit limitation of the project and our previous knowledge.

The high-level overview of the VM instance as follows. A mySQL [24] server is installed, which is used to store the data as well as the statistics of the user; and a Python Flask server [16] is running the machine learning algorithm and communicates with the Pelion Server as well as with the application and the Firebase.

The next subsection presents the design choices during developing a machine learning algorithm for activity recognition. Note that the basis of the codes used to train the ML model and develop the server is based on the tutorial files of the course.

4.1 The machine learning algorithm

As it was proposed in our Proposal document [6] and further explained in our Interim report [22], we chose to use a K-Nearest Neighbours (KNN) machine learning algorithm because it can be implemented easily [11] and can be used efficiently for human activity recognition [11–13]. In our proposal document, we aimed to

distinguish between five activities (idle, walking, running, climbing stairs and using an elevator), which turned out to be a much more difficult task than we expected. Thus, in this document, we present an algorithm which can be used to distinguish between idle, walking and running activities.

First of all, the examined features must be determined. As it was already mentioned, the device gathers the accelerometer, gyroscope and magnetometer values of the device. However, as it turned out during our experiments, the values of these three sensors resulted in very low efficiency. However, after excluding the examination of magnetometer values, the distinguisher started working. We would like to note that this result could have been expected: based on the East-West, West-East running and walking training data, the algorithm has learnt that when users look to North or South, they are most likely to be sitting. Thus, because of these reasons, the magnetometer values were excluded. Please note, that at the moment, the magnetometer values are sent by the device and stored by the cloud, but not used for the ML algorithm. Fully excluding them remains a future work due to time limitation.

After defined which sensors to use, we also have to decide how often we should measure the data and which features we should use. Based on examining other datasets [25], we defined the sampling rate to be 25Hz.

We must also determine the "size of the windows", thus, how many data to be used to calculate our KNN features. We decided to use 50 consecutive data samples each time: based on our experiments, it was a long interval enough to recognise the activity, but short enough to recognise it as soon as possible. Note that 50 data samples with 25Hz means that our machine learning algorithm examines the data from the last 2 seconds.

The features of the model is based on the paper of Duarte *et al.* [13], as it was also mentioned in our Interim report [22], thus, the following were used: mean vector (2 sensors * 3 axes = 6), standard deviation (2 sensors * 3 axes = 6), Euclidean norm of mean vectors (2 sensors = 2), Euclidean norm of the standard deviation (2 sensors), correlations of each axes (2 sensors * 3 pairs ((x,y), (x,z), (y,z)) = 6), signal magnitude area (2 sensors) and entropy of each band after a Fast Fourier Transform (FFT) (2 sensors * 3 axes = 6). Thus, overall, 30 features are used per window.

As it was mentioned in our interim report [22], the `preprocessing.StandardScaler` scaler of the sklearn package [26] is used to scale the data before training our ML model. This scaler uses the $x_{i,j} = \frac{x_{i,j} - mean(x_i)}{standard\_dev(x_i)}$ equation to normalise the data, where $x_{i,j}$ is the j-th reading of sensor i. In that way, it is ensured that the model does not overfit in the readings of a few sensors.

Even though we intended to use a pre-collected dataset [22] to distinguish between the above-mentioned five activities, unfortunately, this aim could not be resolved. During our tests, we tried to use the dataset [25] to train our model. After that, we collected numerous values of our device, fitted a new `StandardScaler` class to our data, and tried to use this scaler to scale the new data. We also tried to filter the features and find which one is the reason for our extremely low accuracy. However, these efforts were unsuccessful and the device worked with very low accuracy (under 50%). Note that that dataset [25] cannot be used to distinguish between idle, running and walking since the data of running activity is very under-represented.

After the unsuccessful usage of the dataset, we decided to distinguish between only idle, walking and running and train our model with data gathered by ourselves. During the data collection, we turned off the data scaling. The device was trained in one person (the author). Since the data can be different if we place the device into our body in a slightly different way, we trained the data performing more training stages, and it was ensured that we took off and on the device between every stage. Overall, the device was trained in 7 stages: five 1-minute-long, and two 7.5 minutes long stages. Thus, each activity was recorded for 20 minutes.

After collecting the data, a scaler was fitted to each column, the columns were scaled and the machine learning model was trained based on the above-mentioned strategies. This scaler was used later to scale our new data. One of the challenges of this task was how to be able to not re-train the algorithm and re-fit the scaler every time we restart the server. It was solved by using the joblib [27] package of Python which provides us with an easy way to save both instances for later usage.

The value of k in the KNN method was determined similarly to our Interim report [22]: KNN algorithms were trained using different k values between 1 and 50. The efficiency of the methods are presented in Figure 3, using cross validation [28] in order to prevent overfitting. In the end, the k value was chosen to 7 based on the results of this experiment and our experimental results.

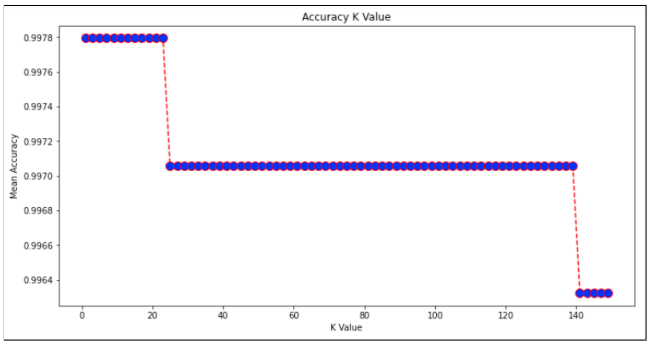


Figure 3: Finding the optimal k value for the model

4.2 The server

As it was mentioned earlier, a Python Flask server runs on the VM instance on the cloud. Please note that since Flask is for developer purposes only, we would replace Flask with another service such as Gunicorn [29] in the real product.

The server is running four main threads, which are the following: receiving data, calling the machine learning model, calling notifications and responding to the application. The threads are called using the Thread package. This subsection presents each of these threads.

4.2.1 Receiving data from the Pelion System. First of all, a thread is running (`receiveData()` function) to connect to the Pelion System and receive the readings of the wearable device using the Pelion Python SDK [23]. In order to do so, the thread subscribes to get a notification when the resource value of the device is updated. If a resource is updated, thus, a new package is broadcasted by the wearable device, a subscription handler is called. In case of disconnection, this thread tries to reconnect every three seconds. This part ensures that the server does not have to be restarted every time the wearable device restarts.

The subscription handler (`_subscription_handler()` function) aims to perform the real data processing in this thread. The handler firstly rescales the data based on the scaler fitted during the training data collection and then saves the data to the database where the values of the device are stored (table `d1`). However, based on our experiment, a huge delay occurred when all of these operations (rescaling the data and updating the database) were done in the same thread. This delay was mitigated in the following way: the subscription handler calls a new thread every time it is called. In this way, the scaling and database updating is done in a new thread every time, and the original thread is ready to receive new data. The increasing delay was successfully mitigated.

During processing the data, a global `lastTime` variable is also updated, which saves the last time a data was received. This variable is used to check if the wearable device is active.

4.2.2 Calling the Machine Learning model. Keeping track of the statistics of the user could be done in two different ways. It could be done by saving all the data of the device and running the machine learning model several times when the statistics are requested. However, this solution would be very time and memory consuming in the long term, thus, another solution is proposed.

In this thread (`callML()` function), it is ensured that the machine learning model (which was pre-trained before and loaded at the beginning of the program) is called every second. After calculating the current activity of the user, the necessary tables are updated (`increase()` function, `activitySum` and `dailyActivity` tables) based on this activity.

A challenge of this thread was to ensure that the model is called exactly once in every second without a delay. In order to mitigate any delay, we measure the time of these time-consuming operations (ML calling and table updating, 0.4-08 second). At the beginning of the eternal loop, 1-timeOfPreviousOperations is calculated and a `sleep()` is called for the calculated time.

4.2.3 Calling a notification. The third thread (`callUpdateAndNotify()`) basically an "others" thread, which does everything which is not important to happen in a restricted time. Note that time was a very important factor in the previous two threads, thus, operations that have a lower priority are performed in this thread.

Firstly, this thread ensures that the `isActive` global variable is updated based on the fact if there were received data in the last three seconds.

Note, that the scaled data of the device is stored in a database (table `d1`), which is used by the machine learning algorithm. However, the machine learning algorithm uses only the last 50 rows of the database every time. Thus, the second task of the current thread is to ensure that unnecessary rows are deleted (`delete_not_important()` function): it deletes all the rows except the last 50. In that way, we highly decrease the storage requirement of the server.

Lastly, this thread calculates the current levels of the user regarding each activity. If any of these levels changes, it sends a push notification (`notify()` function) to the device using the Firebase API. An example notification can be seen in Figure 4.

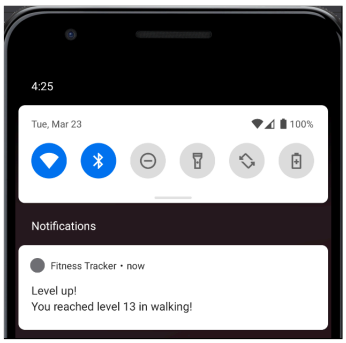


Figure 4: Receiving a Notification

4.2.4 Responding to the application. Last but not least, the Flask server needs to answer the android application. This task is ensured by listening in the 2333 port using different routes.

The `getSum` route returns the overall summary of each activity in seconds. The `getPrediction` route provides a current activity prediction which can be either IDLE, RUN or WALK. The `isActive` route returns `true` if the wearable device is active. The `getNextMilestonesAndLevels` root responds with the level, previous milestone and next milestone (in seconds) for each activity. In that way, it is ensured that the milestones can be controlled on the server-side.

In order to send notification through Firebase to the app, the server needs to know the token of the device, which can change over time. The `replaceNotificationToken` route ensures that the android application can replace the token of the device on the server by sending this route to its new token as a query parameter.

The `getDailyStats` route expects an activity as a query parameter and returns with the correspondent activity seconds of the last seven days.

These listeners ensure that the android application can query every information it needs from the server.

5 THE ANDROID APPLICATION

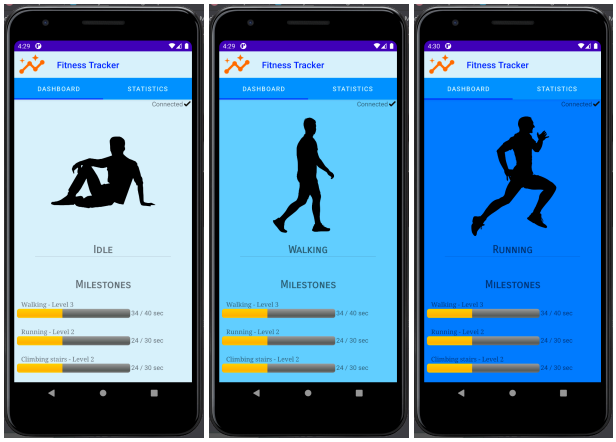
Lastly, an Android application is developed using Android Studio [30] with Java programming language. The application is based on the Tabbed Activity project.

The developed application, which is named Fitness Tracker App, contains three main segments. After initialising the application in the `MainActivity` class, the `DashboardFragment` is displayed in the default tab, which is the first segment. The user can change the tabs and make the application display the other tab, which is controlled by the `StatisticsFragment`. If the application is closed,

the `MyFirebaseMessagingService` service ensures that the device can receive notifications from the server. The networking was done using the Fast Android Networking Library [31]. This section presents each segment of the application.

5.1 The Dashboard

The Dashboard aims to provide basic information to the user in a light an easy-to-understand way. The layout of the dashboard can be seen in Figure 5. Four information can be seen in the dashboard: connectivity status, current activity, the current level of the activities and the progress of each activity.



(a) (b) (c)

Figure 5: The dashboard of the Fitness Tracker App during idle (a), walking (b) and running (c)

The logic of the program works as follows. When the layout is created, two function is registered to run after every second: the `update()` and the `updateBarsSometimes()` functions. The algorithm of the `update()` updates the connectivity and activity segments, however, it calls only the necessary functions: if the first request regarding the connectivity is unsuccessful it can be assumed that the server, the phone or the wearable device is offline. In all three ways, it does not make sense to get a prediction, thus the prediction is not called. It is also true that if the device is active, but the current activity is idle, then it is not possible that the progress bars are increasing, thus, they do not have to be updated. In this way, it is ensured to communicate as less with the server as possible. If the current activity is not idle, it calls the `updateBars()` function.

The `updateBars()` function does the following: it calls the server (`getNextMilestonesAndLevels` and `getSum` API-s), calculates the current progresses and updates the progress bars. It is called from the `update()` function or from the `updateBarsSometimes()` function. The `updateBars()` function is also called when the dashboard fragment is created.

The progress bars should be updated when the device connects to the internet as well. This case is solved using the `updateBarsSometimes()` function, which is also called in every second. This algorithm ensures that if the device looks inactive, the bars are updated every five seconds, but if `isActive` has just changed to `True`, thus, the connection is restored, the bars are updated immediately.

5.2 The Statistics tab

The other tab of the application is the Statistics tab. The layout of the tab can be seen in Figure 6. As can be seen, the user can decide which activity they would like to see. At the moment, the activities are presented in seconds, but they can be easily changed to minutes or hours in the real product. Please, note that the application is ready to receive data regarding climbing stairs and using the elevator as well, which is not yet included on the server-side.

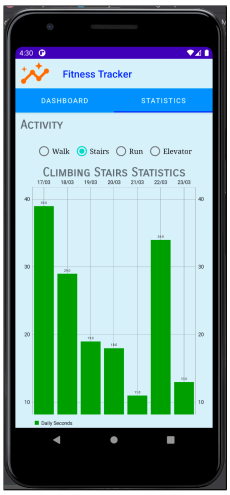


Figure 6: The Statistics tab of the application

The logic of the `StatisticsFragment` class as follows: if the fragment is created or one of the radio buttons is pushed, then the app requests the relevant statistics from the server (`getDailyStats` route), which then are displayed in a graph. The MPAndoirdChart by Philipp Jahoda [32] is used, which provides an easy and nice way to display graphs.

At the moment, only the last seven days are displayed, but it can be increased easily both in the app and on the server-side.

5.3 Communicating with Firebase

As it was mentioned earlier, the application is able to receive notifications, even when it is not running. It is ensured by Firebase and the Cloud Messaging Service. In order to make this work, a `MyFireBaseMessagingService` service is added based on the official example [33].

This class ensures that when the app is not active, the server can push notifications to the phone. One notification as an example can be seen in Figure 4.

6 SECURITY

Regarding the security of the system, we would like to note that at the moment, there is no significant thread in an attacker learning our test activity results. However, in order to present how we

would ensure security in the real world, different security measurements were taken.

First of all, the wearable device and the server both need a Pelion API key in order to be able to communicate with the Pelion cloud, thus, without learning these keys, an adversary could not send to or receive device data from the Pelion system.

In order to make our server more secure, we generated and added a hardcoded API key to the server as well. In this way, without knowing this key, an attacker should not be able to get requests from our Flask server. This key is hardcoded in the android application, which is not a threat since, at the moment, the application is not available on the internet. If we intend to publish it, we should use a way that ensures that it cannot be reverse engineered, which can be done by using the Android NDK [34].

The MySQL server on the VM instance is also secured by adding a user and a password which is used by the Flask server, thus, only this can access the values. Moreover, SQL injection is mitigated by checking the user query input when it is necessary in the `getDailyStats()` function.

There are several strategies that should be used to make our system more secure before deploying in the real world. For example, the Flask server should not be used as it is only for developing purposes, thus, another server should be used such as Gunicorn [29]. In addition, the communication should use HTTPS instead of HTTP, by generating a trusted certificate for our server.

7 EVALUATION

The evaluation of the system contains 3 main parts. Firstly, the developed machine learning model is evaluated. Secondly, the fitness tracker system is evaluated based on the precision of the measurements. Lastly, other important aspects such as internet usage of the application and the size of the codebase are presented.

7.1 Evaluating the machine learning model

The new ML model is evaluated in the same way as it was in our interim report [22]: the dataset is divided into training and test sets (80-20%), and then the confusion matrix is calculated, which can be seen in Table 1. The precision, recall and f1-score values are 1.

	Idle	**Walk**	**Run**
Idle	104	0	0
Walk	0	75	0
Run	0	0	93

Table 1: Confusion matrix of the ML model

However, please note that the too high accuracy is understandable since in that case, the training subject and the test subject was the author. Thus, another evaluation is necessary, which will be done by evaluating the whole system.

7.2 Evaluating the system based on precision

Evaluating the precision of the whole system means that we investigate the results which are displayed to the user in the application in the statistics and milestones part.

Firstly, we evaluated the system by doing different exercises for 10 minutes without stopping. The experiment was conducted by two people: the author (Subject 1) and another one (Subject 2). Table 2 shows the results of this experiment.

Subject	Activity	Real time	Measured walking	Measured running
1	walking	600 sec	587 sec	0 sec
2	walking	600 sec	586 sec	7 sec
1	running	600 sec	3 sec	595 sec
2	running	600 sec	5 sec	594 sec
1	idle (sit)	300 sec	0 sec	0 sec
1	idle (stand)	300 sec	7 sec	0 sec
2	idle (sit)	600 sec	249 sec	0 sec

Table 2: The result of the first experiment

As can be seen, the system reached very high accuracy in most cases. The overall accuracy is 97.75% in terms of walking and 99.07% in terms of running. Regarding measuring the idle position the system did very well when it was used in the author (98.83%), but not well with the other subject (58.5%), which resulted in an overall 78.67% precision. It is worth noting that in the last experiment the subject worked on a computer in a different chair that was used to train the model. The overall accuracy of the system based on this experiment is 91.83%.

We would argue that the results of the experiments are promising since the system was able to reach high accuracy in terms of walking and running in a subject who was not used to train the system. Thus, our system could be useable in terms of other users as well.

However, it is also clear that these results only tell us that the system works well when someone keeps running or walking. In order to learn more about the corner cases, we aimed to ”stress test” the system, thus, measure how it works in extra cases.

First of all, new users are likely to test the system by checking in the application how fast it is to recognise their movements. Thus, in our first ”stress test” experiment, we measured how fastly the app displayed a new activity. Table 3 presents the results of the test. In the first column, the first letter of each activity is presented, where the first is the starting activity and the second one is the goal activity.

As it was expected, it took much time to change from running to idle (5.94 seconds on average), and it was very fast to change from idle to walking (2.15 seconds). Note that our machine learning classifier works based on 2-seconds-long windows, thus, it was

Activity	**1**	**2**	**3**	**4**	**5**	**Avg.**
I-W	1.72	2.45	2.28	2.06	2.23	2.15
I-R	4.36	3.93	4.42	4.76	4.89	4.47
W-R	3.59	3.53	4.42	5.29	4.48	4.26
R-W	2.32	3.96	3.65	4.19	2.95	3.41
R-I	5.40	6.60	6.38	5.53	5.80	5.94
W-I	4.92	6.00	4.98	4.56	3.86	4.86

Table 3: Time to display the next activity (seconds)

not expected to have lower results than 2 seconds. Furthermore, note that the experiments were done as follows: the starting activity was performed for at least 5 seconds, and then, suddenly the second activity started and a timer was started. Thus, there was almost no "middle" case.

As it was clear that it takes some time for our system to recognise the next activity, we were also interested in the effects this imprecision has on the overall statistics. Thus, secondly, another experiment was conducted as follows: a specific activity (walking or running) was performed in a specific way (slow, medium or fast) for 20 seconds. After that, the test subject stopped moving. In the experiment, we did not measure how much time did it take to change to idle, but how many seconds are the statistics increased overall, measured at the time when the activity is changed to idle.

The results of this experiment can be seen in table 4. As it can be seen, 20 seconds walking resulted in 22-23 seconds walk increasing while 20 seconds running resulted in 4-9 seconds walk increasing and 13-20 seconds running increasing. These results could be expected since the ML works based on 2 second-long windows and if the user stops after running it usually measures a few walking before recognises that the user has stopped.

Act	1	2	3	4	5	Avg
W-S	(22, 0)	(22, 0)	(22, 0)	(22, 0)	(22, 0)	(22, 0)
W-M	(23, 0)	(23, 0)	(23, 0)	(23, 0)	(23, 0)	(23, 0)
W-F	(23, 0)	(23, 0)	(23, 0)	(23, 0)	(23, 0)	(23, 0)
R-S	(10, 14)	(7,16)	(10, 14)	(16, 8)	(5, 16)	(9.6, 13.6)
R-M	(4, 19)	(4, 19)	(4, 20)	(3, 20)	(5, 19)	(4, 19.4)
R-F	(4, 21)	(5, 20)	(4, 20)	(3, 20)	(5, 21)	(4.2, 20.4)

Table 4: Exercising 20 minutes and measuring the overall increase in the statistics. (W - walking; R - Running; S - Slow; M - Medium; F - Fast; Results (w, r), where i is the increase in walking and r is the increase in running, both in seconds)

Please, note that all the experiments were measured in the application which is due to the limitations of the system can measure time only second by second. We would argue that this is not a problem since in the real world, these values will be measured in minutes rather than seconds. Please, also note that the stress test experiments were conducted by the author alone and using a manual timer, which means that it was difficult to measure everything with high precision. The fact that it was tested by the author who was used to train the model also makes it clear that the machine learning model could be overfitted to this person.

Overall, it is clear that the system has limitations regarding the speed of the recognition and distinguishing idle from walking, however, it is very promising that the efficiency is very high in terms of running and walking, even when another person is involved in the testing.

7.3 Evaluating the system based on other aspects

Apart from precision, there are many other properties, which describe a system. In this subsection, we briefly present these properties.

The number of code lines is 735 in the cloud, 978 in the firmware and 1639 in the application. Note that these statistics exclude the generated libraries but include the reused code lines which are mostly from the labs of the course.

When the server was running, the python script used around 6.2% memory and 108% CPU on our VM instance (2 vCPUs, 4 GB memory). Thus, the CPU usage is relatively high, which was expected because of the different threads, but the memory usage is low.

During our training and testing, which was done using a mobile phone as a personal hotspot, the application used 62.4MB of data overall, while the wearable device used 47.5MB. It is relatively high data usage compared to the fact that the devices do not download images or videos, thus, the communication could be improved to require fewer data. Unfortunately, the actual usage hours could not be presented, thus, this experiment should be repeated in the long term.

According to the android phone which runs the application, the app requires 6.05MB of storage. This means that this is a very small application at the moment. The average memory use of the application was 55MB, while the maximum was 111MB.

8 FUTURE WORK

During this work, a robust system has been designed and developed which aims to recognise the current activity of the user and provide detailed statistics about running and walking activities. As it was presented, the prototype of the system is reliable, works well and was tested. The system managed to reach overall 91.83% precision and can recognise the activities in at most 6 seconds.

However, there is always a place for improvement. Firstly, more activities such as climbing stairs and using the elevator could be added. Secondly, as it was mentioned above, the system is not reliable in terms of recognising idle activity. In order to improve this, more data could be gathered to train the machine learning model and other machine learning models could be tested to learn more about the strengths and weaknesses of each. Additionally, the system could be tested on more test participants and during different, real-world scenarios.

Optionally, more sophisticated strategies could be used to ensure overall precision. For example, it could be assumed that a participant is running only if it continues for at least three seconds, and activities which was performed less than three seconds could be excluded from the statistics.

We would like to also note that since this is a prototype and we had only one android phone and one wearable device, we developed a system that only deals with a single wearable device and a single phone. In the next step, this could be improved in order to be able to be used by different users at the same time. Moreover, the communication method could be simplified: the API could be further improved in a way to communicate every data in a single message to the application. The frequency of data gathering and app-cloud communication could be also tested an increased to achieve higher accuracy or decreased in order to get lower data usage.

REFERENCES

[1] Thomas Fritz, Elaine May Huang, Gail C Murphy, and Thomas Zimmermann. Persuasive technology in the real world: A study of long-term use of activity sensing devices for fitness. In *Persuasive Technology in the Real World: A Study of Long-Term Use of Activity Sensing Devices for Fitness*. ACM, 2014.

[2] Kanitthika Kaewkannate and Soochan Kim. A comparison of wearable fitness devices. *BMC public health*, 16(1):433–433, 2016.

[3] Kelly R Evenson, Michelle M Goto, and Robert D Furberg. Systematic review of the validity and reliability of consumer-wearable activity trackers. *The international journal of behavioral nutrition and physical activity*, 12(1):159–159, 2015.

[4] G. De Leonardis, S. Rosati, G. Balestra, V. Agostini, E. Panero, L. Gastaldi, and M. Knaflitz. Human activity recognition by wearable sensors : Comparison of different classifiers for real-time applications. In *2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6, 2018.

[5] Ahmad Jalal, Majid Ali Khan Quaid, Sheikh Badar ud din Tahir, and Kibum Kim. A study of accelerometer and gyroscope measurements in physical life-log activities detection systems. *Sensors (Basel, Switzerland)*, 20(22):6670, 2020.

[6] Ábel Kocsis. Assignment 1 - Proposal document, January 2021.

[7] Sreenivasan Ramasamy Ramamurthy and Nirmalya Roy. Recent trends in machine learning for human activity recognition—a survey. *Wiley interdisciplinary reviews. Data mining and knowledge discovery*, 8(4):e1254–n/a, 2018.

[8] Javier Andreu, Javier Andreu, Plamen Angelov, and Plamen Angelov. An evolving machine learning method for human activity recognition systems. *Journal of ambient intelligence and humanized computing*, 4(2):195–206, 2013.

[9] Yiming Tian, Jie Zhang, Lingling Chen, Yanli Geng, and Xitai Wang. Selective ensemble based on extreme learning machine for sensor-based human activity recognition. *Sensors (Basel, Switzerland)*, 19(16):3468, 2019.

[10] Jozsef Suto, Stefan Oniga, Claudiu Lung, and Ioan Orha. Comparison of offline and real-time human activity recognition results using machine learning techniques. *Neural computing applications*, 32(20):15673–15686, 2020.

[11] Stephen J Preece, John Yannis Goulermas, Laurence P. J Kenney, and David Howard. A comparison of feature extraction methods for the classification of dynamic activities from accelerometer data. *IEEE transactions on biomedical engineering*, 56(3):871–879, 2009.

[12] Anju S.S and Kavitha K.V. Performance evaluation of various machine learning techniques for human activity recognition using smartphone. *International Journal of Computer Sciences and Engineering*, 7(8):316–319, 2019.

[13] Francisco Duarte, André Lourenço, and Arnaldo Abrantes. Classification of physical activities using a smartphone: evaluation study using multiple users. *Procedia Technology*, 17:239–247, 2014.

[14] IoT Device Management. Retrieved from https://pelion.com/product/iot-device-management/ accessed 12 February 2021.

[15] Documentation | Google Cloud. Retrieved from https://cloud.google.com/docs accessed 20 January 2021.

[16] Welcome to Flask — Flask Documentation (1.1.x). Retrieved from https://flask.palletsprojects.com/en/1.1.x/ accessed 24 March 2021.

[17] Documentation | Firebase. Retrieved from https://firebase.google.com/docs accessed 20 January 2021.

[18] Keith M Diaz, David J Krupka, Melinda J Chang, James Peacock, Yao Ma, Jeff Goldsmith, Joseph E Schwartz, and Karina W Davidson. Fitbit®: An accurate and reliable device for wireless physical activity tracking. *International journal of cardiology*, 185:138–140, 2015.

[19] Meredith A Case, Holland A Burwick, Kevin G Volpp, and Mitesh S Patel. Accuracy of smartphone applications and wearable devices for tracking physical activity data. *JAMA : the journal of the American Medical Association*, 313(6):625–626, 2015.

[20] Mbed Studio | Mbed. Retrieved from https://os.mbed.com/studio/ accessed 12 February 2021.

[21] mbed-os-example-pelion - Mbed OS example for Pelion Device Management | Mbed. Retrieved from https://os.mbed.com/teams/mbed-os-examples/code/mbed-os-example-pelion/ accessed 12 February 2021.

[22] Ábel Kocsis. Assignment 2 - Interim report document, March 2021.

[23] Pelion Device Management SDK for Python — Mbed Cloud SDK Python 0.1 documentation. Retrieved from https://developer.pelion.com/docs/device-management/v1.5/mbed-cloud-sdk-python/index.html accessed 24 March 2021.

[24] MySQL. Retrieved from https://www.mysql.com/ accessed 24 March 2021.

[25] Daily and Sports Activities. Retrieved from https://www.kaggle.com/obirgul/daily-and-sports-activities accessed 20 January 2021.

[26] sklearn.preprocessing.StandardScaler — scikit-learn 0.24.1 documentation. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html accessed 12 February 2021.

[27] Gael Varoquaux. joblib: Lightweight pipelining with Python functions. Retrieved from https://joblib.readthedocs.io accessed 24 March 2021.

[28] sklearn.model_selection.cross_val_score — scikit-learn 0.24.1 documentation. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html accessed 12 February 2021.

[29] Gunicorn - Python WSGI HTTP Server for UNIX. Retrieved from https://gunicorn.org/ accessed 24 March 2021.

[30] Download Android Studio and SDK tools. Retrieved from https://developer.android.com/studio accessed 24 March 2021.

[31] AMIT SHEKHAR. amitshekhariitbhu/Fast-Android-Networking, March 2021. Retrieved from https://github.com/amitshekhariitbhu/Fast-Android-Networking accessed 24 March 2021.

[32] Philipp Jahoda. PhilJay/MPAndroidChart, March 2021. Retrieved from https://github.com/PhilJay/MPAndroidChart accessed 24 March 2021.

[33] firebase/quickstart-android. Retrieved from https://github.com/firebase/quickstart-android accessed 24 March 2021.

[34] Securing API Keys using Android NDK (Native Development Kit). Retrieved from https://blog.mindorks.com/securing-api-keys-using-android-ndk accessed 24 March 2021.