

detectBiklio - Detecting Bicycle Usage with a Smartphone App

William Janoti
University of Oslo
Oslo, Norway
wcjanoti@uio.no

Paulo Ferreira
University of Oslo
Oslo, Norway
paulofe@uio.no

ABSTRACT

The importance of cycling as an alternative mean of transportation and leisure is ever-increasing. In addition, the current ubiquity of mobile computing made it possible to be used to detect the transportation used. Thus, we present three possible solutions to detect cycling activity using a smartphone which are based on: Google Activity Recognition API, BLE sensors, and a Machine Learning classifier. An innovative and common aspect to all these solutions is that they do not require network connection to run, i.e., they execute solely on the smartphone. We present their architecture and evaluate them according to a set of requirements for the mobile app. Based on their evaluation, we concluded that the most interesting solution is based on machine learning given its balanced respect for the app requirements and possible evolution.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing design and evaluation methods**;

KEYWORDS

mobile app, smartphone, cycling detection

ACM Reference Format:
William Janoti and Paulo Ferreira. 2021. detectBiklio - Detecting Bicycle Usage with a Smartphone App. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3412841.3441965

1 INTRODUCTION

In today's sprawling cities, the importance of cycling as an alternative mean of transportation and leisure is ever-increasing. The adoption of bicycles for urban transportation may improve the mobility of citizens, help to minimize traffic issues and promote a healthier and greener lifestyle [5, 23].

The current ubiquity of mobile computing and the speed at which smartphones evolved have created unprecedented opportunities for using these technologies in innovative ways [16]. Modern mobile devices have powerful embedded sensors such as GPS, accelerometers, magnetometers, gyroscopes, etc., that are able to provide a vast amount of valuable data in a cheap and easily accessible way. Data generated by those sensors can be interpreted and used in many ways (e.g., to detect a user's activity).

Mobile applications that rely on such embedded sensor's data are no novelty and have become an integral part of people's daily lives, to help avoid traffic (e.g. Waze [29]), to track a fitness activity (e.g., Strava [25]) and even to play some games (e.g., Pokémon GO [22]); they use location and other sensors to offer new experiences based on what the user is doing.

In this context, Biklio [6] is a system including a smartphone application (app for short) that takes advantage of such technologies to provide benefits and rewards to those who cycle. Biklio aims at helping more people to take up cycling by offering them rewards based on their cycling activity. It was developed under the TRACE project [26] (European Union's Horizon 2020 research and innovation programme under grant agreement N° 635266) as one approach to encourage and promote cycling in cities. The data generated by Biklio's users can also be used to help cities better understand citizen mobility and plan transport policies.

The whole Biklio system consists of a smartphone app available for both iOS and Android and a website (Biklio.com) where shop owners are able to register their businesses and add the benefits that they would like to offer. The main objective of the smartphone app is essentially enabling users to connect to partner shops that are offering benefits to those who cycle to them.

By using the Biklio app, a user is able to start tracking a bicycle ride to a selected participating shop; once she/he gets there, she/he can claim a reward that was previously defined by the shop owner in Biklio's website. Rewards can also be given to users who cycle a certain distance, not necessarily going to a particular shop. The objective is to encourage a more frequent bicycle usage in the cities covered by the app and also to help local businesses to increase their potential customer base and visibility. Biklio has been tested and is in use in cities in Portugal, Italy, Sweden, Luxembourg, Bulgaria, United Kingdom, and in the Netherlands.

In short, Biklio's main feature is rewarding those who have effectively ridden their bicycles to a given shop or for a predetermined distance. Therefore, it is essential that the Biklio app is able to determine and track accurately and precisely that a user has been in fact cycling. Correct detection of a user's transportation mode is paramount in order to guarantee a fair rewarding of honest users.

Thus, the goal of this work is to successfully design, implement and evaluate three different solutions for cycling activity detection (suitable for use in apps such as Biklio) using:[1]

- the latest version of Google's Activity Recognition API, that allows detecting user current activities or transitions between activities [11];

[1]Note that the tracking of the trip itself is already successfully implemented by Biklio.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441965

- Bluetooth Low Energy (BLE) sensors attached to a bicycle, allowing a smartphone to pair with them and thus detect that a user is effectively cycling [9]; and,
- a machine learning (ML) classifier based on an algorithm adapted to resource-constrained devices (i.e., Random Forest) [2] .

Given that the cycling detection module is meant to be executed by an application running exclusively on a mobile device, some fundamental functional and non-functional requirements regarding accuracy, usability, speed, and battery usage must be satisfied by the proposed solutions. **HA** (*High accuracy*) - The accuracy of the cycling activity detection is by far the most important requirement. In Biklio's use case, it ensures that users are properly rewarded when cycling, and keeps the shops from giving out benefits for false rides. Ideally, accuracy should be as high as possible (i.e., 100%). **RT** (*Soft real-time*) - Given the nature of the app (Biklio), a user should not wait too long after finishing a cycling trip before being able to claim a reward. Therefore, the detection process should reach an outcome (determining if the trip was done using a bicycle) in a sensible time frame, at most a few seconds after the ride has finished. **OFF** (*Offline*) - All data processing and gathering necessary to detect the transportation mode of a ride should be done **locally**, i.e., on the device. Server communication or any kind of network connection should not be necessary. **EE** (*Energy efficient*) - The activity detection module should not have a noticeable negative impact on the device's battery by consuming too much power when compared with a user's typical device usage. **UF** (*Easiness of use*) - The activity detection process in the app should be totally transparent to the user and easily integrated into existing apps. **AD** (*Android*) - The solution should run on Android-based smartphones.

Running on constrained environments brings some challenges related to memory and energy restrictions inherent to mobile devices that must be taken into account. Activity recognition, in particular, can be quite heavy on the device resources, since it relies on data from multiple sensors, including the GPS (which is one of the most energy-demanding components in a modern smartphone). Also, the classification of the activity is usually a computationally expensive task, which can demand a great deal of battery power to run.

The issue of detecting what activity a person is doing by analyzing a smartphone sensors readings has been extensively researched and experimented in the field of Human Activity Recognition (HAR) [18]. Nevertheless, most existing solutions we found do not focus on detecting one specific kind of activity; instead, they try to detect a broad set of possible activities [24, 30]. Furthermore, some solutions [17] rely on processing data in a remote server to classify the transportation mode, not satisfying some requirements proposed here (e.g, OFF).

The architecture of each one of the solutions shares some common points to encourage reuse and simplicity. Each one has a main long-running service responsible for bootstrapping the necessary components and keep the app running at all times, even in the background. Upon detecting that a trip started/stopped, it notifies different components to start/stop tracking the trip. When a trip is over, its relevant data is opportunistically uploaded to a cloud database.

In short, this work brings the following contributions:

- Design and architecture of different cycling detection mechanisms based on: 1) a recognition API from Google, 2) a set of BLE sensors, and 3) a ML classifier.
- Implementation of the above mentioned solutions in Android.
- Evaluation of the above mentioned solutions with a focus on accuracy, energy consumption, and usability.

The rest of this document is structured as follows. The next section gives a more detailed overview of the existing work on the topic of transport mode detection. Then, Section 3 and Section 4 present Biklio's cycling activity detection modules and its overall architecture, and their most relevant implementation details, respectively. Section 5 shows the evaluation of each solution developed. Finally, in the last section, some conclusions are drawn.

2 RELATED WORK

There are several different approaches to identify the most diverse kinds of activities such as standing, running, jumping, etc.; however, we are specifically interested in those that identify the mode of transportation that a person is using [21]. Regarding such systems, there are two main approaches: remote and local. A *remote* approach means that the collected sensor data is sent to a server (typically in the cloud) to perform the activity inference steps and provide an answer (possibly returning the activity classification result to the device); a *local* approach means doing the aforementioned task on the device itself.

Remote solutions provide very good detection accuracy levels in some cases, but they are not entirely adequate for Biklio as a soft real-time response (RT) for the transport modality detection is needed in order to provide a reward to the user in a timely manner. In particular, with a local approach, there is no need for an internet connection (to transfer data to a server) with obvious advantages (e.g., regarding privacy). In fact, all solutions that perform remote transport/activity detection are not adequate solutions since they usually take a long time to process the transport mode, thus not providing a prompt response. It is also worth mentioning that the amount of data and the algorithms needed to train a ML model makes this kind of task computationally expensive; thus, when using a local approach for mobile devices, such training is usually done on a classic computer and then, the generated model is deployed on the mobile device to be used. Also note that some of the existing solutions, both local and remote, do not consider cycling in their activity classification process at all. Nevertheless, as mentioned before, cycling is an increasingly important mode of transportation in many cities and justifies being taken into account.

While there are several solutions that aim to track physical activity, including cycling (e.g., Strava [25], Google Fit [10], Endomondo [8], etc.), they do not meet the requirements presented in Section 1. In fact, these and other similar apps, require the user to manually indicate what kind of activity they will perform beforehand; thus, they do not detect automatically a user's activity.

Ferreira et al. [9] were able to obtain a perfect detection accuracy of the cycling activity (100%); this solution was just an experiment using Biklio's platform, thus not being distributed to users and only available for iOS. It is based on a BLE (Bluetooth Low Energy) beacon attached to the bicycle frame and paired with a mobile (iOS)

	HA	OFF	RT	EE	AD	UF	ML Algorithm	Accuracy
T. Sonderen	*yes*	*yes*	*yes*	*no*	*yes*	*no*	Random Forest, KNN, Decision Trees	94% to 100%
Widhalm et al.	*yes*	*yes*	*no*	*no*	*no*	*no*	DHMM	95%
Reddy et al.	*no*	*yes*	*yes*	*no*	*no*	*no*	Decision Tree & DHMM	45.6% to 87.9%
Woorti	*no*	*yes*	*yes*	*no*	*yes*	*yes*	Random Forest	94.62%

Table 1: Existing solutions and correlations with the requirements

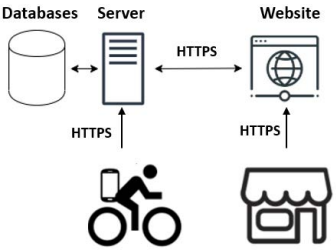


Figure 1: Biklio's high level overview

app. Even though a BLE sensor is fairly easy to install, it is not unreasonable to expect that not everyone will be able or willing to do so, which might affect user-friendliness negatively, even more so in an app like Biklio that is aimed to be used by everyone; thus, in some cases, this solution does not fulfill the UF requirement. In addition, this solution only works for iOS smartphones, not respecting the AD requirement.

Tim Sonderen [24] used several ML-based classification algorithms and compared the detection accuracy: decision tree, random forest, and KNN (K-Nearest Neighbors) [15]. The best results for cycling were obtained using random forest, although CPU usage was high (89%) which might affect the performance in a real setting. The activities classified were walking, running, cycling, and driving. His work managed to achieve high accuracy in most of the activities. This solution used MATLAB [19] to create the model and exported it to be used as a JAR dependency on the mobile app, which in turn fed the model with the collected data from the sensors. The sensors used were accelerometer, gyroscope, and magnetometer. The efficiency of the implemented system is questionable since the author itself states that CPU usage is rather high at some points, which in turn can also impact battery consumption. This work does not present an evaluation of battery usage, so we assume that the EE (energy efficient) requirement was not fulfilled. Besides, no study regarding ease of use was done and data from only two subjects was gathered, which is quite insufficient.

The solution proposed by Widhalm et al. [30] uses both GPS and accelerometer data. It detects the following means of transportation: bicycle, motorcycle, car, bus, tram, train, subway, and walking. The solution extracts features from the data in sliding windows of variable size. Using 100 random classifiers followed by a Discrete Hidden Markov Model (DHHM) [7] they were able to achieve 95% accuracy in detecting a cycling activity. The data set for cycling was quite significant, with 180 trips summing up 56 hours worth of data. No information on the time it takes to detect an activity was shown. No energy consumption study was done either and this solution uses GPS, which is one of the most energy-consuming sensors in a smartphone. Thus, based on the results shown, it fulfills neither the RT (soft real-time) nor the EE (energy efficient) requirements. Also, since there was no mention of what platform the solution was implemented in, it does not fulfill AD.

Reddy et al. [23] proposed a solution that detects several transportation modes such as stationary, walking, running, cycling, or in a motorized vehicle. The classifier is based on a Decision Tree (DS) followed by a DHHM [7]. The sensors used to gather trip data were the GPS and the accelerometer. To evaluate the system, sixteen individuals tested the application that was developed; there was no study on UF, so we assume this requirement is not respected. Regarding accuracy, this solution obtains an average of 93.6 % considering all transportation modes supported by the classifier. However, when it comes to cycling, the accuracy drops to 45.6% - 87.9%, thus not fulfilling the HA requirement. In addition, due to relying heavily on GPS, this solution does not properly fit into EE requirement. Furthermore, the classifier was targeted at a Nokia n95 smartphone that runs Symbian S60 operating system, thus the AD requirement is clearly not fulfilled as well.

Woorti [31] is a smartphone app that was developed in the MoTiV project [20] (European Union's Horizon 2020 research and innovation programme under grant agreement N^{o} 770145). Woorti's objective is to analyze a user's transportation behavior and understand how her/his travel time is spent. Thus, Woorti employs a ML based solution that is able to classify a user's transportation mode in each segment of the trip, although not always accurately. It allows the user to review and correct the transportation mode utilized in each segment and the corresponding data is used to retrain the model. Woorti was able to reach 94.62% accuracy in overall transport modality detection, i.e, considering all transportation modes that Woorti supports (walking, bicycle, car, bus, train).

Table 1 shows a summary of the ML based current solutions that were addressed and whether they satisfy the requirements we want to achieve. As shown in this table, we can observe that none of the listed systems respects all the requirements stated in Section 1.

3 ARCHITECTURE

In this section we first present a Biklio's overview. Then, we describe its overall architecture and the cycling activity detection modules.

3.1 Biklio Overview

As mentioned in Section 1, Biklio is a mobile app that rewards users for cycling; thus, Biklio needs an accurate cycling detection module to determine if a user has been in fact riding a bicycle and a way to track the traveled distance.

Along with the mobile app, Biklio's ecosystem is composed of a web application with all of its associated back-end infrastructure, including servers and databases (see Fig. 1). The participating shops also have access to the platform through the website so they can register their rewards and benefits available to the users. Biklio's

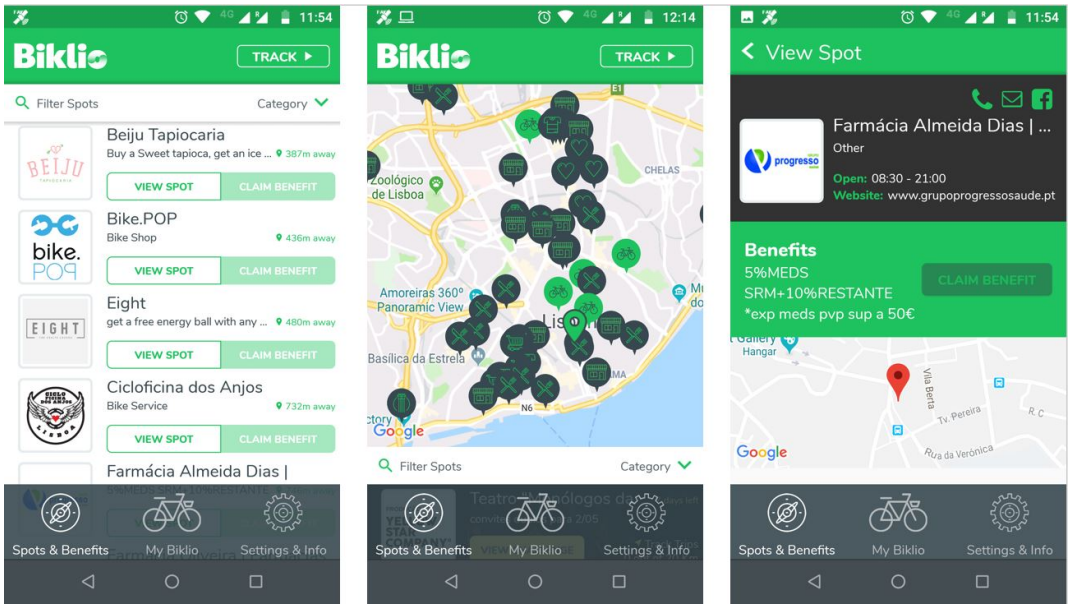


Figure 2: Biklio Android app: a) Spots listed by distance to the user; b) Spots on map; c) Spot details

web application (biklio.com) enables shop owners to sign in and register their shops (called spots in the app jargon). Basic information about the spot is provided during the registration, including the type of rewards they want to offer, as well as customizing their information to be displayed in the mobile app. The back-end servers and databases are used by both the website and Biklio app; the databases store users and spots data, as well as the tracking data from users' trips. In fact, for the sake of convenience and to analyze the data generated, the data of the recorded trips (distance, start time, and end time) are saved in the Biklio database. Note that this is not required for detecting cycling activity, which does not require network access at all. We opportunistically upload trip data to the database when an internet connection is available.

Fig. 2c shows (in a smartphone) a spot details screen of a registered shop. Regarding the kind of rewards offered by a spot, they can be either based on distance cycled, meaning that the user gets eligible for a reward after having cycled for a predefined distance or time, regardless of whether they were going to the shop or not.

After logging in, a user is presented with information on all the participating spots in her/his region. Figs. 2a and 2b show a list of spots sorted by distance to the current location of the user, and also the corresponding map, respectively. To see details of a given spot, the user only needs to click on the spot on the map or in the list.

In order to claim a benefit, the user must be eligible to do so, meaning that she/he has fulfilled the requirements to get the reward. This is determined by detecting if the trip was taken using a bicycle. If a user is eligible and close to a spot, she/he is able to claim the reward by simply clicking on the *Claim Benefit* button on the spot's page and showing the app to the shop's clerk.

3.2 Cycling Detection

As mentioned in Section 1, three solutions for cycling activity detection are proposed. Namely, using Google Activity Recognition API, BLE sensors, and a ML model. All solutions were developed using a long-running service in Android, which among other things, is responsible for detecting when a trip started or ended, and/or collecting data while the user is travelling.

With the exception of the solution based on BLE sensors (see Section 3.2.2 for mode details), the distance tracking feature uses periodic location information updates from Google's Fused Location Provider API [13]. This API provides an energy-efficient way to get location data by combining data from different sensors such as GPS, Wi-Fi, and the cellular network. The total travelled distance is calculated by summing up the straight line distance between each location update that is received by the app every 10 seconds.

Biklio automatically detects when a user starts a bicycle trip and when it ends, based on inputs from the sensors meaning that it requires no user interaction to detect the trips.

3.2.1 Google Activity Recognition API. There are essentially two Google's Activity Recognition APIs suitable for detecting transportation modes: **Activity Recognition Transition**, and **Activity Recognition Sampling**. The first one is adequate for cases when the developer needs to know when a specific activity using a given transport mode has started or ended, e.g., when the user started or ended a bicycle trip. This API works by sending event broadcasts representing the transitions the app has subscribed to; otherwise, it does not provide anything else. The Transition API is the preferred way of using Activity Recognition because, when compared to the other API, it improves accuracy, consumes less power (manages it automatically), and it is easier to use [11].

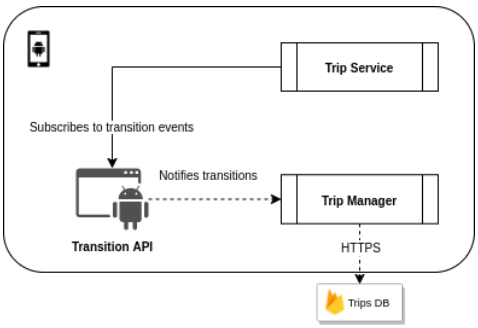


Figure 3: Architecture based on Google's Activity Recognition Transition

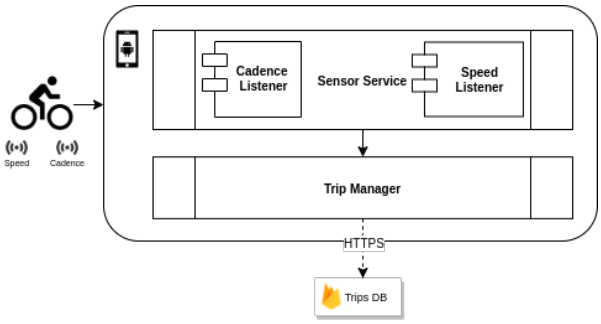


Figure 4: Architecture based on BLE sensors

The overview of the architecture of this solution is depicted in Fig. 3 and has two main components: the **Trip Service**, and the **Trip Manager**. The Trip Service has the following responsibilities:

(1) *Initialization of the Transition API* - upon startup of the app, this service does all the necessary steps to communicate with the Transition API, subscribe to the events we are interested in (when a bicycle trips starts or stops), and handles notifications to the user interface.
(2) *Registering the Trip Manager*, which is a broadcast receiver of the Transition API events.

The Trip Manager basically listens to specific global broadcasts (in the smartphone) indicating that a trip has started or ended, with the following responsibilities:

(1) *Determine when a trip starts or ends* - this component is constantly listening to events from the Transition API; whenever a transition is detected it takes actions to start or stop tracking a trip, depending on the information in the broadcast.
(2) *Tracking distance* - when a new trip is detected, this component is also responsible to start tracking the distance travelled; this is done by receiving periodic location updates, about every 10 seconds, and calculating the distance between each one. There is a trade-off in choosing the interval between location updates; it was chosen 10 seconds (after testing) because this interval yields a distance which is close to reality.

3.2.2 BLE Sensors. There are several BLE sensors in the market developed to track several kinds of activity. Among those, there are some designed for use in bicycles that allow us to obtain speed and cadence data that can be used to determine if one is cycling. When properly paired and connected to a smartphone, such a sensor can send data periodically. Such data includes speed and cadence values which can be then processed by the smartphone app.

The solution based on the usage of BLE sensors attached to a bicycle allow us to obtain speed and cadence (number of crank revolutions) information while cycling. We chose to use a set of BLE sensors of the Wahoo brand [28]. Those sensors were chosen given the mature and well-documented software development kit (SDK) they provide; this allowed an easy integration between the sensors and the app.

Each sensor (one for speed and one for cadence) is powered by a small single button battery, type CR2032 3V, that according to the manufacturer can last up to one year and is easily replaced when needed. In addition, the kind of sensors chosen are easily installed and removed without any extra resource or tool.

The speed sensor is installed on the rear wheel axis, and the cadence sensor can either be installed on the pedal crank or attached to one of the feet of the person riding the bicycle; in the evaluation tests (see Section 5), both ways were used. The sensors power up and start providing data as soon as movement is detected, and shut off after a predetermined period when they are idle.

When the app receives valid data coming from both sensors at the same time it assumes that the user has started a trip and shows a notification indicating that. A trip is considered to have ended when no data from the sensors is received for a period longer than 2 minutes (which can be adjusted as necessary). This period of 2 minutes was determined by experimentation and it is long enough to take into account waiting for traffic lights and possible stops one might take while cycling. Note that, even if the user stops for longer than 2 minutes during a trip, the current trip would finish and a new one would start as soon as the app started receiving sensor information again. From a Biklio point of view, this is not really an issue since the cycling activity is still going to be detected (and a reward can be claimed).

An overview of the architecture for this solution is shown in Fig. 4. The **Sensor Service** is a foreground running service that is the main component of the app; it has the following responsibilities:

(1) *Sensor discovery and connection* - upon startup, this service checks if there are any saved sensor connection information on the smartphone storage from previous runs; if so, it connects to the sensors using the parameters in those files, otherwise, it starts the discovery process. This is done in order to avoid having to discover sensors that have been previously connected.
(2) *Attach listeners to sensors* - in order to receive the sensors data, this service adds both cadence and speed listeners when a connection to a sensor is acquired.[2]

Just like the previous solution (using the Google Transition API) this one also has a **Trip Manager** that listens to the broadcasts from the Cadence and Speed listeners with the following responsibilities:

[2]A listener is essentially a class that listens to a specific event and performs some action when it happens.

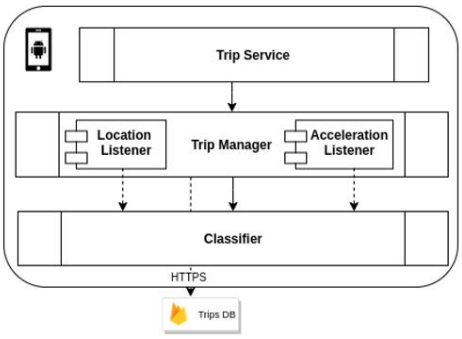


Figure 5: Architecture based on a ML classifier

(1) *Determine when to start and stop a trip* - process the broadcasts from both sensors to determine when a trip has started or stopped.
(2) *Keep track of the connected sensors* - since we can only start a trip when both sensors are connected, this component keeps track of the connected sensors by listening to broadcasts from the Sensor Service that inform if a sensor has been connected.
(3) *Tracking distance* - the sensors broadcast information that includes the number of crank revolutions per time unit; with that data, the Trip Manager can keep track of the distance travelled by using Equation 1 (see below).

Differently from the other two solutions, to track distance we take advantage of the fact that the speed sensor provides the number of wheel revolutions, Thus, we can easily use that value and calculate the travelled distance by using the following equation:

$$distance = wheelRevolutions * (2 * \pi * wheelRadius) \quad (1)$$

where *wheelRadius* is the *radius of the bicycle wheel in meters*. Obviously, the wheel radius value can be freely adjusted. Regarding the other detection aspects, we basically use the same approach as described in Ferreira et al. [9].

3.2.3 Machine Learning. Within ML, two different approaches regarding the training of the model can be taken: *supervised* and *unsupervised*. In supervised learning, the ML algorithm used receives labeled data as input in order to learn how to produce an output later when it is given new unlabeled data.

The Random Forest algorithm follows a supervised approach and, as already mentioned, it is used in the ML based solution. Random forest is a type of ensemble classification method, meaning that instead of working with the construction of a single model, it combines numerous learning models in an attempt to improve the quality and performance of the achieved classification.

This solution is based on the classifier used in Woorti [27, 31]. It should be noted that the collection and processing of the data required to train the algorithm, as well as the generation of the model were done previously by the developers of Woorti. The trip data used to train the model was gathered using sensor data from several real trips using different transportation modes through the app Woorti. After the model was trained, it was exported to a PMML format [14], which made it suitable for importing and using in an Android application.

An overview of the architecture for this solution is shown in Fig. 5. Its main components are described below. The **Trip Service** is a foreground service that has the following responsibilities:

(1) *Instantiate and register components* - upon startup, the service instantiates the Trip Manager, which is the main component of the app, and attaches a broadcast receiver for trip events (start and stop).
(2) *Attach listeners to sensors* - in order to receive the sensors data, the service adds both location and acceleration listeners when a connection to a sensor is acquired.

The **Trip Manager** is the main component that has the responsibilities listed below:

(1) *Start and stop trips* - it receives notifications from the Trip Service whenever a trip is started or stopped.
(2) *Manage listeners* - when a trip starts the app gathers location and acceleration data. In order to do that the Trip Manager registers the listeners that will receive data from the accelerometer and the GPS when a trip starts and de-registers them when a trip ends.

The **Acceleration** and **Location** listeners are simple components that have the following responsibilities:

(1) *Receive data from sensors* - after registered by the Trip Manager, the listeners will receive periodic bursts of sensor data containing acceleration and location.
(2) *Feed the classifier* - when the listeners receive data, they send it to the classifier to accumulate the data that will be used in the classification step after the trip is finished.

Finally, the **Classifier** has the responsibilities described below:

(1) *Load model file* - load and parse the PMML file containing the ML model used for classification.
(2) *Calculate model features* - in order to be able to classify the trip, this component calculates all the required model features after the trip has ended, using the data it received from the listeners during the trip.
(3) *Classifying the trip* - determining if a user has ridden a bicycle by analyzing the result of the classification step using the model.

4 IMPLEMENTATION

The three solutions developed use Android services [3] and were programmed in Java. In Android, a service is an application component used to perform long-running operations that do not need user intervention. There are two main kinds of services: background and foreground. A background service is used to perform operations that are not noticeable to the user, e.g., compressing system files, performing maintenance tasks, etc. However, since Android 8 (API level 26) this kind of service has had some restrictions imposed on its usage, due to energy and efficiency concerns; e.g., not being able to receive event notifications from sensors, and having higher priority of being killed by the system when resources are needed [4]. On the contrary, foreground services have fewer restrictions and are less likely to be killed by the system; the only important aspect is that it has to show a notification to the user as long as it

is running. For these reasons, the services used in the solutions are foreground services.

5 EVALUATION

The evaluation focuses on the HA, RT, and EE requirements. In addition, the usability of Biklio, related to the UF requirement, was assessed through a test done in several cities. The remaining requirements (AD and OFF) were directly observed while using the app. In order to assess OFF, a trip was tracked using each detection system, with both Wi-Fi and mobile data disabled. The process of cycling detection happened on all of them.

The cycling detection tests consisted of 30 trips, each with a duration between 1 to 10 minutes, 10 trips being done with each solution. In most cases, the Biklio app was in the background; however, some tests were also made with the app on the foreground, with no noticeable differences regarding the cycling detection time.

The Google Transition API and ML solutions were tested while using transportation modes other than bicycles to verify whether they would erroneously detect a cycling trip since they both rely on ML based detection mechanisms.

5.1 Cycling Detection

We now describe the results obtained when detecting cycling activity with each solution developed.

5.1.1 Google Transition API. The tests using the solution based on Google's Transition API managed to detect correctly all bicycle trips during the tests. On average, it took less than 10 seconds to automatically detect the start of a bicycle trip in all cases. However, it took longer to detect that the trip was over, i.e., it took in average 3 minutes; in the worst case it took up almost 5 minutes. This considerable delay in detecting the transition from cycling to another activity, i.e., when a trip ends, most likely comes from the fact that it might be harder to detect this specific transition, as opposed to detecting a transition from a still position to cycling (at the beginning of a trip). Nevertheless, since the underlying details of the API are proprietary and the documentation does not contain further details about this, we can not be sure.

In addition, some tests using different transportation modes were performed to assess whether it would detect a cycling trip by mistake. As suspected, while using an escalator, a scooter, and a skate, the Biklio app erroneously detected all of them as bicycle trips: 6 trips were done, 2 with each transportation mode previously mentioned, and all were detected by the app as bicycle trips.

5.1.2 BLE Sensors. The tests using the solution based on the BLE sensors were performed with both sensors previously paired but not connected. During the tests, discovery and pairing took, on average, 15 seconds. Using this solution, all trips were correctly identified. After the initial pairing, every following trip was detected within 5 seconds, and instantaneously determined when it ended. Obviously, and contrary to the previous solution, there is no situation in which a bicycle trip is wrongly identified.

5.1.3 Machine Learning. This approach detected 9 out of the 10 bicycle trips done. The trip that was not detected was classified as being a car trip. Such miss-classification results from the fact that this trip was a rather short trip in comparison to the other ones with only 1 minute and 32 seconds; therefore, the classifier did not gather enough information to classify it correctly. The average time it took to classify a trip was 225ms, which is rather fast.

5.2 Battery Usage

This section evaluates each one of the three solutions regarding their energy consumption in order to understand its impact on the overall smartphone battery life. We let the smartphone charge to its maximum capacity and then reset the device's battery statistics before each test, by issuing the following command using ADB [12], so that the smartphone has a clean state: `adb shell dumpsys batterystats --reset`. After that, each solution was used (tracking a trip) until a drop of 1% in battery power was noticed. Since Android's internal battery usage information does not have decimal precision, an app called AccuBattery [1] was used instead to get a more accurate battery consumption information. The tests were done using a Motorola G7 Plus with a 3,000mAh battery cell.

Note that, even though the phone may be idle after a trip, the underlying service is kept running indefinitely in the background, therefore consuming memory and CPU resources.

5.2.1 Google Transition API. This solution showed a very low energy consumption. It took approximately 1.5 hours to drop the battery level 1%, thus, using approximately 0.85% of device battery per hour. This is due to the fact that this API optimizes resources by using several different techniques to preserve battery power, as stated in the associated documentation [11]. If used in daily short urban trips, this detection method might not affect a user's battery life in any noticeable way due to its low power usage.

5.2.2 BLE Sensors. The BLE based solution took around 22 minutes to consume 1% of the device's battery, thus yielding an hourly consumption of 2.7%. This is due to the overhead of having to discover and pairing with two BLE sensors as well as exchanging data with them. This consumption can impact negatively a device's battery life depending on the frequency it is used.

5.2.3 Machine Learning Model. The ML based solution took approximately 20 minutes to drop the battery level by 1%, therefore having an estimated hourly energy consumption of around 3%. This energy usage is due to the frequency of reading from the accelerometer (every second) and GPS (every 10 seconds); these intervals would have to be adjusted to make the Biklio app use less energy. Similarly to the previous solution (BLE), this also shows a battery consumption that can degrade battery life on long trips.

5.3 Usability

Biklio was used in several different cities (Southend, Breda, Luxembourg, Bologna, Plovdiv) from different countries (see Table 2). Interestingly, 82% of all the Biklio app downloads are related to its use in Bologna and in Luxembourg. Consequently, the three other cities have generated 18% of the app downloads.

There is no correlation between cities population and the application metrics such as the number of active users for instance. Luxembourg has generated a relatively high number of downloads even if Luxembourg-city has few inhabitants. Breda with an important bicycle users community has not generated an important number of downloads. In addition, given the small number of paths

	Southend	Breda	Luxembourg	Bologna	Plovdiv	Total
Population	177900	181000	116328	388367	376000	–
Total # of Downloads	74	36	381	299	44	834
Total # of Users	45	20	297	254	40	656
Total # of Active Users	5	5	66	92	14	182
Total # of Sessions	109	74	1160	1772	240	3355
Total # of Claims	2	11	46	110	8	177
Total # of Recorded Tracks	1	9	414	529	49	1002

Table 2: Biklio cross site analytics

recorded, the information collected in Southend, Breda and Plovdiv can hardly be usable by public authorities.

The differences between Total # of Downloads and Total # of Users is not straightforward (some people who downloaded the app did not use it a single time). The hypothesis for this behaviour is a lack of information before downloading the app. Among the five cities, the dropping rate ranged between 10% (Plovdiv) and 44% (Breda). In addition, some people uses the app a limited number of times (less than 5), thus not being considered as active users.

We can see that the dropping rate is most relevant between Total # of Users and Total # of Active Users. Indeed, the dropping ranges between 64% in Bologna and 89% in Southend. We believe that one of the reasons that explains this behaviour is the lack of interest for the app and its idea in general, as well as a lack of interest for the proposed incentives.

One can see the aggregated statistics of the Biklio app in Table 2, column named "Total". The app has been downloaded more than 800 times. In total, 78,6% of the participants who downloaded the app have used it at least once. This is clearly a positive outcome. The disparities observed between the five different cities do not seem to be explainable by city characteristics such as size or existing modal split. Thus, the differences in terms of application download and usage are most probably the results of strategical decisions (commutation & marketing choices, press coverage, number of local partners involved, etc.).

6 CONCLUSION

This paper presented the Biklio system with a focus on the app running in a smartphone and the three possible solutions to detect cycling activity. These solutions are based on the Activity Transition API from Google, a set of BLE sensors, and in a ML classifier.

The most interesting solution regarding the requirements that Biklio has (high accuracy, soft-real-time, offline, low energy consumption, and easiness of use), is the one based on ML. The Google Transition API showed to be a very energy-efficient way to detect cycling activity, at the cost of low accuracy when using different means of transportation and a high latency to detect the end of a trip. The BLE solution demonstrated a flawless cycling detection, with the only downside of requiring an extra, albeit small effort of mounting and pairing the sensors with the app. Finally, the ML based solution proved to be a responsive and fast way to detect cycling activity, although not perfectly. However, being a ML model we expect it can be better tuned for cycling and get more precise with more training.

REFERENCES

[1] AccuBattery. 2020. AccuBattery. play.google.com/store/apps/details?id=com.digibites.accubattery&hl=en Accessed 25-08-2020.
[2] Ethem Alpaydin. 2010. *Introduction to Machine Learning* (2nd ed.). MIT Press.
[3] Android. 2020. Android Documentation, Services. developer.android.com/guide/components/services Accessed 01-04-2020.
[4] Android. 2020. Sensors Overview. developer.android.com/guide/topics/sensors/sensors_overview#sensors-practices Accessed 12-06-2020.
[5] J. Barreto, M. Costa, R. Lourenço, P. Ferreira, J. Bernardino, B. Evelien, C. Jan, D. Nick, F. Elke, and V. Ilse. [n. d.]. Increasing Bicycle Usage in Smart Cities. In *Proceedings of the 2nd International Workshop on Smart (SmartCities '16)*. ACM.
[6] Biklio. 2019. Biklio website. www.xpto.com Accessed 19-03-2019.
[7] F. R. Chen, L. D. Wilcox, and D. S. Bloomberg. 1995. A comparison of discrete and continuous hidden Markov models for phrase spotting in text images. In *Proceedings of 3rd International Conference on Document Analysis and Recognition.*
[8] Endomondo. 2019. Endomondo. www.endomondo.com Accessed 22-05-2019.
[9] P. Ferreira, A. Zabolotny, and J. Barreto. 2019. Bicycle Mode Activity Detection with Bluetooth Low Energy Beacons. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. 1–4.
[10] Google. 2019. Google Fit. www.google.com/fit Accessed 22-05-2019.
[11] Google. 2019. Google's Activity Recognition API. developers.google.com/location-context/activity-recognition/ Accessed 19-03-2019.
[12] Google. 2020. Android Debug Bridge. developer.android.com/studio/command-line/adb Accessed 08-08-2020.
[13] Google. 2020. Fused Location Provider API. developers.google.com/location-context/fused-location-provider Accessed 11-06-2020.
[14] Data Mining Group. 2020. Predictive Model Markup Language. dmg.org/pmml/v4-3/GeneralStructure.html Accessed 08-07-2020.
[15] Onel Harrison. 2020. Machine Learning Algorithms. towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 Accessed 11-05-2020.
[16] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. 2010. A survey of mobile phone sensing. *IEEE Communications Magazine* 48, 9 (2010).
[17] Nhac Ld, Nhan Nguyen, Thi-Hau Nguyen, and Ha-Nam Nguyen. [n. d.]. Vehicle Mode and Driving Activity Detection Based on Analyzing Sensor Data of Smartphones. *Sensors* 18 ([n. d.]).
[18] Jorge Luis Reyes Ortiz. 2015. *Smartphone-Based Human Activity Recognition.*
[19] MathWorks. 2019. Matlab. mathworks.com/products/matlab.html Accessed 22-05-2019.
[20] MoTiV. 2019. MoTiV. motivproject.eu Accessed 24-05-19.
[21] Marija Nikolic and Michel Bierlaire. 2017. Review of transportation mode detection approaches based on smartphone data. *17th Swiss Transport Research Conference* (2017). infoscience.epfl.ch/record/229181
[22] Pokemón. 2019. Pokemón GO. www.pokemongo.com/en-us Accessed 22-05-2019.
[23] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. 2010. Using Mobile Phones to Determine Transportation Modes. *ACM Trans. Sen. Netw.* 6, 2, Article 13 (March 2010), 27 pages.
[24] Tim Sonderen. 2016. Detection of Transportation Mode Solely Using Smartphones. *Twente Student Conference on IT* (2016).
[25] Strava. 2019. Strava. www.strava.com Accessed 22-05-2019.
[26] TRACE. 2019. TRACE Project. h2020-trace.eu Accessed 22-03-2019.
[27] L. Veiga and J. Bernardino. 2019. *D3.4-MoTiV-App.* Technical Report. INESC ID.
[28] Wahoo. 2020. Wahoo Bike Sensors. eu.wahoofitness.com/devices/bike-sensors Accessed 08-07-2020.
[29] Waze. 2019. Waze. www.waze.com/en-GB Accessed 22-05-2019.
[30] P. Widhalm, P. Nitsche, and N. Brändie. 2012. Transport mode detection with realistic Smartphone sensor data. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 573–576.
[31] Woorti. 2019. Woorti. www.woorti.com Accessed 15-05-2019.