Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

User activity type and transportation mode detection using embedded mobile device sensors

Degree: Telecommunications Engineering
Author: Rok Slamek
Director: Mónica Aguilar Igartua
Co-Director: Božidar Potočnik
Date: September 2017

Abstract

This study aims to improve an existing mobile application MobilitApp (http://mobilitat.upc.edu) for citizen mobility analytics. By eliminating the use of APIs with limited user activity and transportation mode detection, and energy wasting GPS, we developed our own system, using two approaches and only embedded mobile device sensors. We captured various user activity and transportation modes such as stationary, walking, running, riding a bicycle, motorcycle, driving a car, taking a bus, tram and train. We recorded all activities with video camera to be aware when activity actually happened. At first approach, we build feature vectors through experimentation and then use this data in machine learning. In the second approach, we experimented with neural networks if they are capable recognizing features by them self, if we provide them only raw data from embedded mobile device sensors. We also do studies if controlled capturing data with camera is required or can be done without supervision on larger scale.

Resumen

Este proyecto tiene como objetivo mejorar la aplicación para Android llamada MobilitApp (http://mobilitat.upc.edu) que consiste en el análisis de la movilidad de la ciudadanía de Barcelona. Hemos diseñado nuestro propio sistema al cual le hemos eliminado el uso abusivo del GPS para ahorrar energía del dispositivo y las APIs de detección del modo de transporte que tienen un uso limitado para el usuario. Hemos estudiado dos métodos de detección y utilizamos únicamente los sensores de los smartphones. Detectamos diversos modos de actividad y transporte del usuario tales como estático, a pie, corriendo, en bicicleta, en coche, en autobús, en tranvía y en tren. Nuestra mejora ha consistido en grabar con una cámara de vídeo todas las actividades para tener registrado todo lo ocurrido en la actividad. En el primer método, construimos vectores característicos a través de la captura de los datos de los sensores del dispositivo y utilizamos estos datos en el aprendizaje automático. En el segundo método, experimentamos con redes neuronales para ver si el sistema es capaz de reconocer las características por si mismo si solo le proporcionamos datos de los sensores. También hemos analizado la posibilidad de hacer estos estudios sin la necesidad de capturar vídeos de la cámara y observar solo los datos proporcionados por los sensores de los dispositivos.

Resum

Aquest projecte té com a objectiu millorar l'aplicació per a Android anomenada MobilitApp (http://mobilitat.upc.edu) que consisteix en l'anàlisi de la mobilitat de la ciutadania de Barcelona. Hem dissenyat el nostre propi sistema, hem eliminat l'ús abusiu del GPS per estalviar bateria del dispositiu i les APIs de detecció del mode de transport que tenen un ús limitat per l'usuari. Hem estudiat dos mètodes de detecció i utilitzem només els sensors dels telèfons intel·ligents. Detectem diversos modes d'activitat i transport de l'usuari tal com estàtic, a peu, corrents, amb bicicleta, amb cotxe, amb autobús, amb tramvia i amb tren. La nostra millora ha consistit en enregistrar amb una càmera de vídeo totes les activitats per tenir registrat tot el que ha ocorregut a l'activitat. Al primer mètode, vàrem construir vectors característics a través de la captura de les dades dels sensors dels dispositius i utilitzem aquestes dades en l'aprenentatge automàtic. Al segon mètode, vàrem experimentar amb xarxes neuronals per observar si el sistema és capaç de reconèixer les característiques per si mateix si només li proporcionem les dades dels sensors. També hem analitzat la possibilitat de fer aquests estudis sense la necessitat d'enregistrar amb la càmera i observar només les dades proporcionades pels sensors dels dispositius.

Dedicated to my family, friends and all supporting me.

Table of Contents

Table of Figures

List of Tables

Glossary of acronyms

ANN – Artificial neural network

API – Application programming interface

ATM - Autoritat del Transport Metropolità

CPU – Central processing unit

DFT – Discrete Fourier transform

DHMM – Discrete hidden Markov model

ELM – Extreme learning machine

FFT – Fast Fourier transform

FPS – Frames per second

GPS – Global positioning system

K-NN – K nearest neighbors

MEM – Microelectromechanical system

OSELEM – Online sequential extreme learning machine

PII - Personally Identifiable Information

RF – Random forest

RMS – Root mean square

SDK – Software development kit

SMO – Sequential minimal optimization

SVM – Support vector machine

TransELM – Transfer extreme learning machine

UPC - Universitat Politècnica de Catalunya

ZRC – Zero-crossing rate

1 INTRODUCTION

Mobile devices have become an indispensable device in our everyday life. With constant presence in our immediate vicinity, they have become convenient for monitoring our activities, location etc. Such information may be particularly interesting for some companies, which could improve their services and hence customer satisfaction, based on analysis of gathered data from mobile phones of their costumers. At Universitat Politècnica de Catalunya (UPC) in Barcelona, we joined the project Mobilitat, which is developing a mobile application MobilitApp, for purpose of implementing such a system in Barcelona city. The aim of the application is to provide the data for improvement of the services provided by public services companies and raising awareness of the users on the reduction of the carbon footprint with more rational use of means of transport. This is possible by tracking users location, mode of transport and other user activities. Goal is to offer an application or data to companies that provide services of public transportation, such as Autoritat del Transport Metropolità (ATM) of Barcelona, automated bike rental, etc. For these provider of public transportation services, information such as what is the distance from users home to nearest stop of public transportation, what is the distance from users of private motorized transport, to the nearest stop of public transport, the number of users going from the stop of public transport to a certain direction etc. Based on these data the company could by adjusting the routes and schedules of public transport improve their services, hence customer satisfaction and possibly even acquire new users. In the case of a company that provides rental bicycles at various points in the city and return of bicycle not necessarily to the same point, the information of the direction and distance of users after the use of bicycles, can contribute to the setting up new meaningful locations to rent or return a bike. However, since the capture of such a data interferes the privacy of the users, they rarely use such applications. The idea is to encourage the use of application by offering free public transport rides, free minutes to rent bicycles and discounts on other urban services. At the same time the application also offers other useful services and information such as traffic information, the index of user's carbon footprint, the number of steps walked, number of calories burned and system for detecting an accident.

Since the constant recording of Global Positioning System (GPS) location, relative to the recording of embedded sensors, is very energy-consuming, we would like to eliminate usage of GPS in our detection algorithm. We figured out that recording the location at endpoints and intermediate points of trip at a change in the type of means of transportation or user activity, is sufficient enough for our use. It would therefore be interesting to see whether the embedded mobile device sensors can determine the activity and transportation mode used by mobile device user, if we use the known techniques in the field of signal processing and pattern recognition. The subject of the final work represents engineering approach to develop model for detection of the type of user activity and transportation mode, using signals from embedded mobile device sensors.

1.1 Objectives and thesis

The purpose of the final work is to determine whether it is possible to detect the type of user activity and transportation mode in pseudo real-time, by eliminating use of GPS and using only signals from embedded mobile device sensors. The aim is to develop a system that will be able to identify various user activities, such as stationary, walking, running, riding a bicycle, motorcycle, driving a car, taking a bus, metro, tram and train, and will give results comparable with existing solutions. The system will be developed using two approaches. In first approach we will build model using vector of features extracted based on analysis of signals. In second approach we will build model using neural networks, where we would like to explore whether neural networks are capable to extract the features by them self, if the captured signal is directly feed into neural network. We would also like to know, if separating of stationary parts from actual activity effects on classification. With this we can tell if supervised data capturing Is required or capturing can be done without supervision. In this final work we would like to acknowledge the following two hypotheses:

- ***Hypothesis 1:*** *By using only embedded mobile device sensors, without usage of GPS, is possible to build vector of features and model, which is by efficiency of detection of type of user activity and transportation mode, comparable with "state of the art" solutions, if we use short time window up to 3 s.*
- ***Hypothesis 2:*** *From raw signal of embedded mobile device sensors captured using short time window up to 3 s, that we feed directly into the neural network, neural network is capable to extract features and build model, which is by efficiency of detection of type of user activity and transportation mode, comparable with "state of the art" solutions.*
- ***Hypothesis 3:*** *Treating stationary parts of different user activities and transportation modes as one common "stationary" class, gives better classification results than using stationary parts as part of it's activity.*

1.2 Structure of work

This work is structured in 6 main chapters. In chapter 1 we introduce us with problem, MobilitApp application and curse of our work. In chapter 2 we review existing solutions as final applications, Application Programming Interfaces (APIs) and research papers. In chapters 3 and 4 we present methods used and process how we developed our system. In chapter 5 we present data and metrics used and results of our test. And in final chapter 6 we open the discussion about results, we look in future work and conclude this work.

1.3 MobilitApp

MobilitApp (http://mobilitat.upc.edu) is a mobile application developed in project Mobilitat at UPC. Main goal of project is to provide mobility data to ATM of Barcelona, to improve the current transportation infrastructure in Barcelona. Through few years application was built and updated by various students working on this project. For now, application is only available on Android platform.

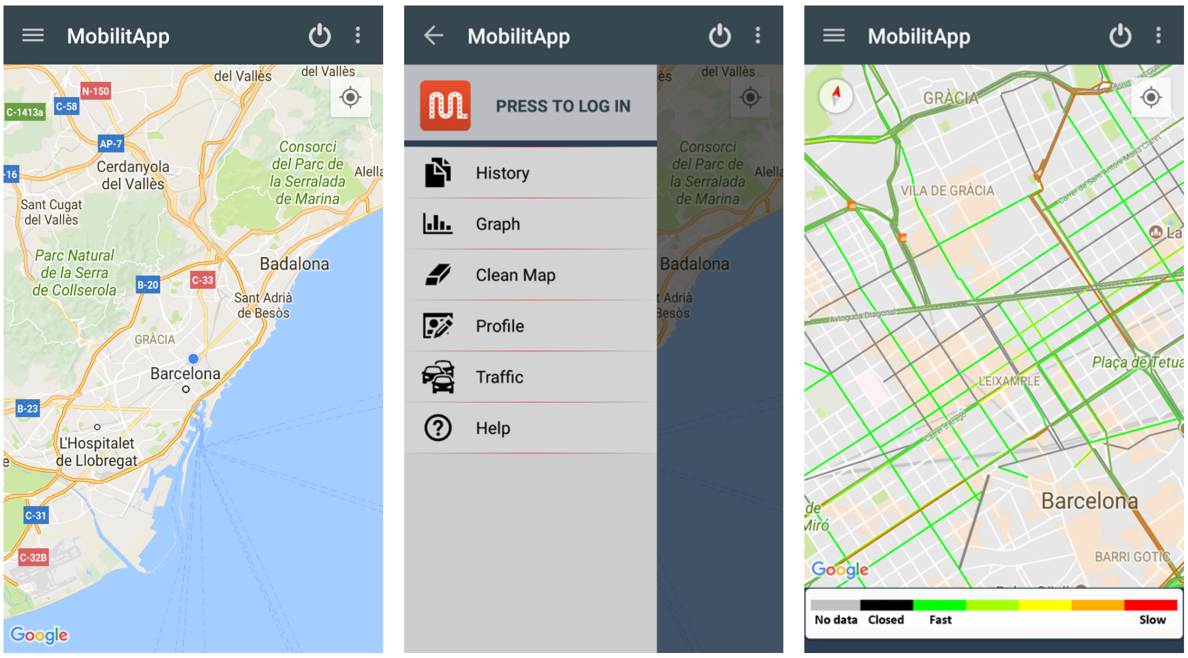
Main objective of the application is to obtain mobility data from the citizens of the metropolitan area of Barcelona. The aim is to provide this data or complete application to the public service companies for further analyzes for purpose of improving their services and raise user's awareness of importance of the reduction of the carbon footprint with more rational use of means of transport.

Some functionalities are still in development and some are in state of improvement. Over the years of different students working on project, different approaches for detecting user activity and transportation modes were used. Currently Google Activity Recognition API is in use for detecting following states: stationary, in vehicle, on bicycle and on foot. Because of the maximum active users limitation of Google Activity Recognition API, and goal to detect larger variety of specific transportation modes, main goal of the project is to develop own system for detection of user activity and specific transportation modes.

Application also offers other useful services and information such as traffic information (Figure 1 a), the index of user's carbon footprint, the number of steps walked, number of calories burned and system for detecting an accident. Information about the state of traffic and incidents on the road is provided in real-time, allowing the citizens to take decisions on their journey. On Figure 1 we can see screenshots of MobilitApp application. On screenshot a) we see initial screen with map and top bar with menu, shutdown and settings buttons. With shutdown button user stops all tracking activities in background and closes the application. In screenshot b) we see the main menu and live traffic information on screenshot c).

[1] As sensitive data, such as GPS location, is used in the application, before using the application privacy policy is presented to the users who are concerned with how their Personally Identifiable Information (PII) is being used online. Data collected in application is not associated with specific person and it is completely anonymous.

In future, plan is to provide user with detailed information of activities and use of means of transportation (range and time spent in transport or doing activity) and other relative information.



a) b) c)

Figure 1: Screenshots of MobilitApp application

2 STATE OF THE ART

[2] Many systems exist to classify human motion activities and transportation modes. Commercial devices such as FitBit, GoWear, Philips Trackmor, etc. are widely available and fairly convenient, but they provide limited activity information, and they are separate entity. With the advancement of sensors on mobile phones, researchers are looking to use this device as a platform for activity detection. Most of the solutions really on GPS position and speed information.

In this chapter we will focus only on those existing solutions, that are using embedded mobile device sensors.

2.1 Existing APIs and applications

There is few solutions that enable you as developers to implement their functionalities to your application. But down side of these solutions is that they have some limitations such as not detecting particular activities and limited number of active users.

2.1.1 Google Activity Recognition API

[3] Back in 2013, Google launched the ActivityRecognitionAPI to developers. The ActivityRecognitionAPI is an interface that allows android app developers to know

what "activity" the user is currently engaged in without the hassle of getting raw data from individual sensors and then having to run a complex analysis to come to a conclusion. The API returns the detected activity together with the confidence of its results.

[4] The activities are detected by periodically waking up the device and reading short bursts of sensor data. It only makes use of low power sensors in order to keep the power usage to a minimum. For example, it can detect if the user is currently on foot, in a car, on a bicycle or still. In Table 1 all supported activities in this API are displayed. The activity detection update interval can be controlled with the detectionIntervalMillis parameter. Larger values will result in fewer activity detections while improving battery life. Smaller values will result in more frequent activity detections but will consume more power since the device must be woken up more frequently.

Table 1: Supported activities in ActivityRecognitionAPI

Type	Description
int **IN_VEHICLE**	The device is in a vehicle, such as a car.
int **ON_BICYCLE**	The device is on a bicycle.
int **ON_FOOT**	The device is on a user who is walking or running.
int **RUNNING**	The device is on a user who is running.
int **STILL**	The device is still (not moving).
int **TILTING**	The device angle relative to gravity changed significantly.
int **UNKNOWN**	Unable to detect the current activity.
int **WALKING**	The device is on a user who is walking.

2.1.2 Apple CMMotionActivity

[3] [5] Also in 2013 Apple introduced their CMMotionActivity activity recognition into their iOS.

On devices that support motion, you can use a CMMotionActivityManager object to request updates when the current type of motion changes. When a change occurs, the update information is packaged into a CMMotionActivity object and sent to your app. The motion-related properties of this class are not mutually exclusive. In other words, it is possible for more than one of the motion-related properties to contain the value true. For example, if the user was driving in a car and the car stopped at a red light, the update event associated with that change in motion would have both the cycling and stationary properties set to true. It is also possible for all of the properties to be set to false when the device is in motion but the movement does not correlate to walking, running, cycling or automotive travel. In Table 2 all activities, supported by CMMotionActivity, are displayed.

Table 2: Supported activities in CMMotionActivity

Type	Description
var **stationary**: Bool	The device is stationary.
var **walking**: Bool	The device is on a walking person.
var **running**: Bool	The device is on a running person.
var **automotive**: Bool	The device is in an automobile.
var **cycling**: Bool	The device is in an bicycle.
var **unknown**: Bool	The type of motion is unknown.

2.1.3 Samsung Digital Health - S Health Service

[6] S Health is an application that monitors the user's activities and helps the user has a healthier life. S Health's collected data can be categorized and expressed in various ways. It is important to present proper information to the user in the required time for advanced experiences. S Health 4.x supports Android devices with KitKat 4.4 including non-Samsung devices. With Samsung Digital Health software development kit (SDK) developers can implement functionalities of S Health 4.x into their application.

[7] Samsung has also developed their own application S Health. S Health helps users to better manage their health and track their fitness progress. It also offers various features and functions to make exercise fun, and fitness goals more attainable. With Detect Workouts function, the app can automatically detect and log user's running, cycling, walking and hiking sessions that occur for at least 10 minutes. Turning on the Auto Pause setting will also improve tracking accuracy, as it will recognize when the exercise session stopped.

2.2 Research papers

In this section we will present papers of similar work like ours. Results achieve in those papers, presented in Table 3, will be also use as benchmark for our solution. We could not find any solution that has all the activities, that we are interested, covered in their system. So In Table 3 beside activities of our interest, we also added classes Road and Rail, which are presenting two groups of some classes that we treat separate. From our standpoint we can also treat classes Motorcycle and Car as one class Road, and classes Train, Tram and Metro as class Rail.

Table 3: Benchmark results of similar work

Paper section		2.2.1	2.2.3	2.2.4	2.2.5	2.2.7
Paper reference		[2]	[8]	[9]	[10]	[11]
		Accuracy				
Class	Stationary					
	Walk	96.8%		95.8%	84.8%	96.2%
	Run	91.0%		98.5%	96.4%	98.6%
	Bicycle	92.8%		97.0%	77.6%	91.2%
	Motorcycle					
	Car			91.7%		
	Bus			92.4%		
	Metro					
	Train					
	Tram					
	Still	95.6%			97.0%	98.2%
	Road				90.4%	
	Rail				93.0%	
	Motorized	93.9%				94.3%
Sum accuracy		93.0%	94.6%	94.9%	89.8%	95.7%

2.2.1 Using Mobile Phones to Determine Transportation Modes [2]

[2] The focus of this work is on one dimension of context, the transportation mode of an individual when outside. They create a convenient (no specific position and orientation setting) classification system that uses a mobile phone with a built-in GPS receiver and an accelerometer. The transportation modes identified include whether an individual is stationary, walking, running, biking, or in motorized transport. To eliminate specific device orientation requirements, they express accelerometer as Root Mean Square (RMS) of 3 axes. They use 1 second sliding window and 5 features:

- GPS speed,
- accelerometer variance,
- magnitude of accelerometers Discrete Fourier Transform (DFT) at 1 Hz,

- magnitude of accelerometers DFT at 2 Hz,
- magnitude of accelerometers DFT at 3 Hz.

The overall classification system consists of a decision tree followed by a first-order discrete Hidden Markov Model (DHMM) and achieves an accuracy level of 93.6% when tested on a dataset obtained from sixteen individuals. They reported problems with classification of slow motorized transport for biking and slow running for walking.

2.2.2 Online Sequential ELM based Transfer Learning for Transportation Mode Recognition [12]

[12] To address the transfer learning problem for transportation mode recognition, this paper proposes an online sequential extreme learning machine (OSELM) based transfer learning method called Transfer Extreme Learning Machine (TransELM). This method can utilize valuable features and trustable samples to effectively transfer common knowledge across labeled source domain and unlabeled target domain. TransELM mainly includes three steps: Firstly, an initial Extreme Learning Machine (ELM) classifier is trained on the labeled training dataset from the source domain. Secondly, relevant mean and standard deviation values are separately computed as trustable intervals for each class of transportation modes. The unlabeled dataset of target domain is classified with the initial ELM model and trustable samples whose output values belong to corresponding trustable intervals are effectively extracted. Thirdly, for integrating these trustable samples, an incremental OSELM method is employed to incrementally update the original ELM classifier. They present experimental results from their user study, including five people with six typical transportation modes (staying still, walking, riding bicycle, taking bus, taking light-rail, and driving) in the daily life. To eliminate specific device orientation requirements, they express accelerometer as RMS of 3 axes. They use 8 second sliding window and 18 features:

- Maximum accelerometer value,
- minimum accelerometer value,
- mean of accelerometer,
- standard deviation of accelerometer,
- energy of accelerometer,
- zero-crossing rate of accelerometer,
- four amplitude statistics features,
- four shape statistics features of the power spectral density,

- mean GPS velocity,
- standard deviation of GPS velocity series,
- maximum GPS velocity,
- maximum GPS acceleration.

Experimental results show that TransELM obtains higher accuracy than the traditional ELM classifier in real world transportation mode recognition problems.

2.2.3 Detecting Changes of Transportation-Mode by Using Classification Data [8]

[8] In this paper they present a method for detecting changes of transportation mode on a multimodal journey, where the input data regard to the classification of human activities. They use a space transformation for extracting features that identify a transition between two transportation modes. The data are collected from the Google API for Human Activity Classification through a crowdsourcing-based application for smartphones. They are focusing on activities as walking, riding bicycle and driving. They use sliding window size of 5 samples and 6 features, witch they did not specify in paper. Results of 88 % correctly classified samples show improvements on precision and accuracy in comparison to initial classification data outcomes.

2.2.4 Applying Machine Learning Techniques to Transportation Mode Recognition Using Mobile Phone Sensor Data [9]

[9] This paper adopts different supervised learning methods from the field of machine learning to develop multiclass classifiers that identify the transportation mode, including driving a car, riding a bicycle, riding a bus, walking, and running. Methods that were considered include K-nearest neighbor, support vector machines (SVMs), and tree-based models that comprise a single decision tree, bagging, and random forest (RF) methods. For training and validating purposes, data were obtained from smartphone sensors, including accelerometer, gyroscope, and rotation vector sensors. K-fold cross-validation as well as out-of-bag error was used for model selection and validation purposes. Several features were created from which a subset was identified through the minimum redundancy maximum relevance method. In this paper beside accelerometer they also utilize gyroscope and orientation sensor, witch was never been done before. On sensors they apply RMS and extract 165 features using methods:

- Spectral entropy,

- energy,
- mean,
- maximum,
- minimum,
- variance,
- standard deviation,
- range,
- interquartile range,
- zero-crossing rate.

Data obtained from the smartphone sensors were found to provide important information to distinguish between transportation modes. The RF and SVM methods were found to produce the best performance.

2.2.5 Transportation mode recognition based on smartphone embedded sensors for carbon footprint estimation [10]

[10] This paper focuses on a particular type of context, the transportation mode used by a person for carbon footprint estimation and it summarizes a method for automatically classifying different transportation modes with a smartphone. The model was built using a random forest followed by a DHMM filtering and can classify 7 different classes (still, walk, run, bike, road, rail, plane and other). Beside GPS they are using 5 second sliding windows on accelerometer and magnetometer sensors extracting 13 based features using:

- Magnetic field norm standard deviation,
- standard deviation of linear acceleration components,
- proportion of energy in different frequency bands of the vertical acceleration,
- spectral centroid and spectral spread of the vertical acceleration,
- spectral centroid and spectral spread of the norm of the magnetic field,
- median of GPS speed.

In this article oppose to others they do not user RMS on accelerometer, instead they use accelerometer orientation estimation algorithm explained in [13]. They also use features calculated from different frequency bands such as [0.7Hz-3.5Hz], [3.5Hz-8.5Hz], [8.5Hz-18.5Hz] and [18.5Hz-45Hz]. The model was evaluated with real data and performed with accuracy around 94%, while the addition of the GPS feature improved the performance up to 96%.

2.2.6 Detecting the transportation mode for context-aware systems using smartphones [14]

[14] This paper presents a classification method for smartphone users mobility data in urban environments according to the used transportation mode. This classification is possible among several different transportation modes and using only the location data from user's mobility. Among the methods applied, includes data mining with machine learning techniques for the inference. This paper also presents the performance analysis for several machine-learning algorithms for the proposed task; the process used to collect mobility data for nine users along six months. They compared numerous classification methods such as Bayesian networks, Naive Bayes, SVM, Multilayer Perceptron, Decision Tree, Random Forest, Random Trees, K-Means, K-Nearest Neighbors (K-NN), Ada boost and Sequential Minimal Optimization (SMO), using 3 features, chosen by Weka Experimenter tool:

- Maximum speed,
- maximum acceleration,
- number of direction changes.

Beside classifying between walking, not walking, bicycle, motorcycle, bus and car, they also classified between walking and not walking, walking and bicycle, walking, bicycle and motorized, car, motorcycle and bus, motorized and not motorized modes. The classification algorithms, Decision Tree, J.48, Bayes Net, SMO, preformed best in every of classification groups mentioned before.

2.2.7 Determining Transportation Mode On Mobile Phones [11]

[11] We focus on one type of context, the transportation mode of an individual, with the goal of creating a convenient (no requirement to place sensors externally or have specific position/orientation settings) classification system that uses a mobile phone with a GPS receiver and an accelerometer sensor to determine if an individual is stationary, walking, running, biking, or in motorized transport. The target application for this transportation mode inference involves assessing the hazard exposure and environmental impact of an individual’s travel patterns. Their prototype classification system is consisting of a decision tree followed by a first-order Hidden Markov Model using 1 second sliding window and features:

- Variance,
- energy,

- sum of Fast Fourier Transformation (FFT) coefficients between 1-5 Hz from the accelerometer,
- speed from the GPS receiver.

They tested their system with device on different positions of users body, such as arm, bag, chest, hand and pocket, and discovered average decrease of 1.2% in accuracy between the generalized classifier and position specific classifier. When testing with their dataset consisting of twenty hours of data collected across six individuals, they achieved accuracy level greater than 90%.

2.2.8 Accelerometer based transportation mode recognition on mobile phones [15]

[15] In this paper, they introduce transportation mode recognition on mobile phones only using embedded accelerometer. In order to deal with uncertainty of position and orientation of mobile phone, acceleration synthesization based method (RMS) and acceleration decomposition based method (accelerometer orientation estimation method presented in [13]) are introduced. Using 8 seconds sliding window they classify between walking, running, bicycling, inline skating and driving car, using features:

- Mean,
- standard deviation,
- mean crossing rate,
- third quartile,
- sum and standard deviation of frequency components between 0-2 HZ,
- ratio of frequency components between 0-2 HZ to all frequency components,
- sum and standard deviation of frequency components between 2-4 HZ,
- ratio of frequency components between 2-4 HZ to all frequency components,
- spectrum peak position.

They compared results using DT J48, K-NN and SVM, and comparison indicates that acceleration synthesization based method outperforms acceleration decomposition based method.

3 METHODOLOGY

In this chapter we describe nontrivial methods that were used to develop our model for user activity type and transportation mode detection using embedded mobile device sensors.

3.1 Accelerometer orientation estimation using gravity

This approach [13] for obtaining orientation-independent acceleration information makes use of the fact that Microelectromechanical Systems (MEMS) accelerometers measure gravitational ("static") acceleration as well as ("dynamic") accelerations caused by the wearer's motion. The pull of gravity downward along some accelerometer axis manifests itself in the accelerometer output as an acceleration in the opposite direction along that same axis.

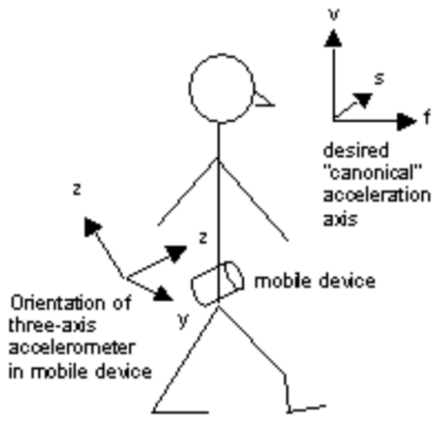

v
s
f
desired "canonical" acceleration axis
z
z
mobile device
Orientation of three-axis accelerometer in mobile device
y

Figure 2: Relevant coordinate systems

There are two relevant coordinate systems, as shown In Figure 2. The three axis accelerometer configuration is in some arbitrary orientation on the wearer's body. The three accelerometer axes are denoted in the figure as *x*, *y*, and *z*. Ideally, we would like to know acceleration information in terms of a coordinate system oriented to the user and his forward motion. In the figure, these axes are denoted *v* (for vertical), *f* (for the direction of horizontal forward motion), and *s* is a (usually of less interest) horizontal axis orthogonal to the direction of motion.

The algorithm works as follows: for a chosen sampling interval, typically a few seconds, obtain an estimate of the gravity component on each axis by averaging all the readings in the interval on that axis. That is, we are estimating the vertical acceleration vector *v* corresponding to gravity as

$$v = (v_x, v_y, v_z) \tag{1}$$

where *vx*, *vy* and *vz* are averages of all the measurements on those respective axes for the sampling interval. Let

$$a = (a_x, a_y, a_z) \tag{2}$$

be the vector made up of the three acceleration measurements taken at a given point in the sampling interval. We assume for the sake of simplicity that the three measurements are taken simultaneously. We set

$$d = (a_x - v_x, a_x - v_y, a_x - v_z) \tag{3}$$

to represent the dynamic component of *a*, that caused by the user's motion rather than gravity. Then, using vector dot products, we can compute the projection *p* of *d* upon the vertical axis *v* as

$$p = \left(\frac{d \cdot v}{v \cdot v}\right) v \qquad (4)$$

in other words, *p* is the vertical component of the dynamic acceleration vector d. Next, since a 3D vector is the sum of its vertical and horizontal components, we can compute the horizontal component of the dynamic acceleration by vector subtraction, as

$$h = d - p \qquad (5)$$

However, as opposed to the vertical case, we do not know the orientation of *h* relative to *f*, the horizontal axis we'd like to have it projected upon. Furthermore, it appears impossible to detect. There is no dominating static acceleration as there is in the vertical case. Accordingly, we simply compute the magnitude of the horizontal component of the dynamic accelerations, concluding that that is the best we can expect to do.

The result of the algorithm performed across a sampling interval is a pair of waveforms, estimates of the vertical components and the magnitude of the horizontal components of the dynamic accelerations, each of which is independent of the orientation of the mobile device containing the accelerometers.

This algorithm was proven to be useful in detecting and distinguishing several user motion activities, such as walking, running, climbing or descending stairs, or riding in a vehicle – in spite of the fact that the position and orientation of the device are not known. We conjecture that the vertical acceleration component is sufficient information for most such activity detection.

3.2 Spectral centroid

[16] The spectral centroid is a common measure used in digital signal processing to characterize a spectrum. It indicates where the "center of mass" of the spectrum is. Perceptually, it has a robust connection with the impression of "brightness" of a sound.

[17] It is computed considering the spectrum as a distribution which values are the frequencies and the probabilities to observe these are the normalized amplitude:

$$SC(m) = \frac{\sum_k f_k |X(m,k)|}{\sum_k |X(m,k)|} \qquad (6)$$

where *m* is the signal, $X(m,k)$ represents the value of signal *m* at index *k* and f_k is the amplitude of frequency *f*.

3.3 Spectral spread

[17] Following the definition of spectral centroid, spectral spread is defined as the spread of the spectrum around its mean value or in other words the variance of the above defined defined distribution:

$$SC(m) = \frac{\sum_k \left(f_k - SC(m)\right)^2 |X(m,k)|}{\sum_k |X(m,k)|} \tag{7}$$

where *m* is the signal, $X(m,k)$ represents the value of signal *m* at index *k*, f_k is the amplitude of frequency *f*, and *SC(m)* is spectral centroid of signal *m*.

3.4 Zero-crossing rate

[18] The zero-crossing rate is the rate of sign-changes along a signal, meaning the rate at which the signal changes from positive to negative or back. This feature has been used heavily in both speech recognition and music information retrieval, being a key feature to classify percussive sounds. Zero-Crossing rate (ZCR) is formally defined as:

$$ZRC = \frac{1}{T-1} \sum_{t=1}^{T-1} 1_{\mathbb{R}_{<0}} \left(s_t - s_{t-1}\right) \tag{8}$$

where s is a signal of length *T* and $1_{\mathbb{R}_{<0}}$ is an indicator function.

3.5 Vector norm

[19] In linear algebra, functional analysis, and related areas of mathematics, a norm is a function that assigns a strictly positive length or size to each vector in a vector space. [20] L^2 norm is one of the variations of vector norm and is defined as:

$$|x| = \sqrt{x_1^2 + x_2^2 + x_3^2} \tag{9}$$

where *x* is the vector consisted of three values.

4 PROPOSED ALGORITHM TO DETECT USER ACTIVITY AND TRANSPORTATION MODE

In this chapter we will present our work from capturing data to development of our algorithm using two approaches. In first approach based on research of existing work and introducing some of our ideas we build feature vectors, that we used to develop model using machine learning. In second approach we try to feed Neural networks with sliding window - signal sequence from embedded sensors. At the beginning of this work we defined three hypotheses, which we also try to prove in this chapter.

From examining existing solutions, we discovered that there are good solutions for detecting different user activities and transportation modes in general, but there is still not to be found good solution for detecting specific transportation type.

Our goal is to develop system (Figure 3), that will be able to detect user activities and transportation modes presented in Table 4, using only embedded mobile device sensors such as accelerometer and magnetometer, and completely eliminating usage of GPS. This way we intend to save the life of battery on mobile device, by setting GPS retrieval frequency or fetch GPS locations based on current state or transition of our system. We think retrieving GPS location on transitions between different states will be sufficient for our purpose. System will not be limited with specific device orientation, but we defined some restrictions for capturing data. From research of existing solutions, we figured out, that everybody is capturing larger data sets and uses specific activity’s data as whole from start to end of capture. Meaning that stationary parts of trip, such as stopping at bus, train, metro stops are also included. We believe, that stationary parts can be differentiated from moving parts so we suggest, that stationary parts of every activity should be treated separate from activity as one common “Stationary” class. To differentiate stationary sections from moving sections in sensors data, we decided to record capture sessions with video camera. System will be developed and tested on computer using Python programming language. While doing that we will keep in mind, that later this system can be also implemented on mobile platform.

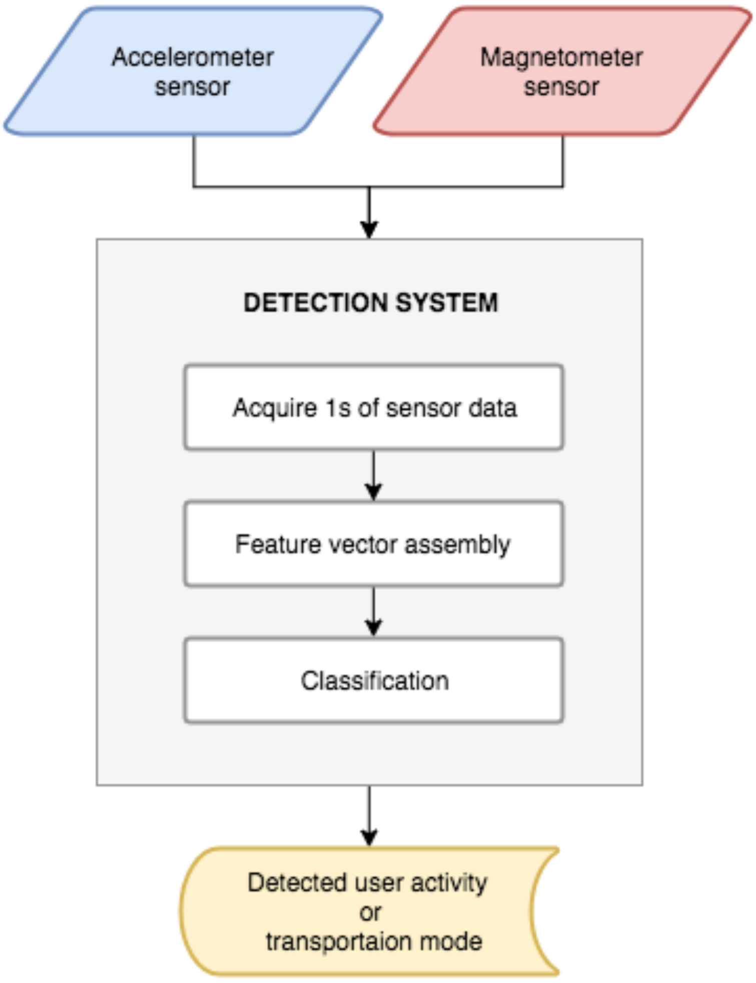


Figure 3: Flow chart of detection system

Table 4: Supported types of activity and transportation modes in our system

State	Description
Stationary	Device is stationary (including stationary in any of other state).
Walk	Device is on user who is walking.
Run	Device is on user who is running.
Bicycle	Device is on user who is riding bicycle.
Motorcycle	Device is on user who is riding motorcycle.
Car	Device is on user who is taking the ride with car.
Bus	Device is on user who is taking the ride with bus.
Metro	Device is on user who is taking the ride with metro.
Train	Device is on user who is taking the ride with train.
Tram	Device is on user who is taking the ride with tram.

4.1 Data collection

As we had project application available to work on it, we just updated it with sensor capture routines. We present changes in the application in section 4.1.1. These added functionalities are temporary, just for use of data gathering, and will be removed at the end.

We decided to record capture sessions with external video camera, so later when processing data, we will have better sense of environment. Capturing session will start with close up shoot of mobile device running applications with pop-up Start capture dialog containing early mentioned information for easier matching and synchronization of captured sensor data and video. When user presses Start button he puts his device in pocket or purse and we adjust framing of vide camera, so that user and surrounding environment is seen. At end user takes his device out od pocket or purse and stops capture by pressing Stop button. After this we also stop video recording.

At first we were recording video at 25 frames per second (FPS), but we discovered that this is not sufficient enough, to precisely synchronize sensor data and video and define sections of activity in sensor data. So we decided to increase FPS rate to 100, which is also the same number as out sampling rate. Initial plan to synchronize sensor data and video was by looking at a frame when start capture button in pressed. At moment when the button was pressed we assumed that sensors also started capturing data at this exact moment, so video and signals from sensors should be synchronized. But we discovered that this is not the case, because of the UI delay which is around 60m. So we went with different synchronization approach: playing 500ms long high pitch sound right at the beginning of capturing data from sensors. As shown in Figure 4 this high pitch sound can be in most cases easily differentiated from rest of the ambient sound.

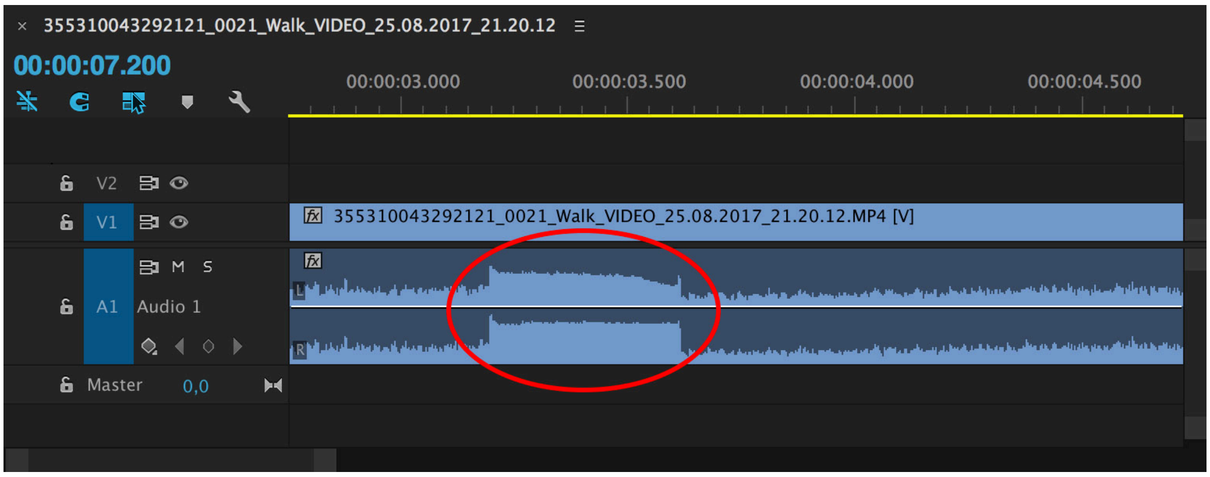

355310043292121_0021_Walk_VIDEO_25.08.2017_21.20.12
00:00:07.200
00:00:03.000
00:00:03.500
00:00:04.000
00:00:04.500
V2
V1
355310043292121_0021_Walk_VIDEO_25.08.2017_21.20.12.MP4 [V]
A1
Audio 1
M
S
Master
0,0

Figure 4: High pitch sound in video file

With increasing frame rate and improving synchronizing method, signals can be more precisely indexed, so that they correctly represent each activity. Still in some cases, it is difficult to exactly determine start of activity from video. For example, because of slight camera shake it is difficult to select correct frame when transitioning from stationary to moving or vice versa. Timeline of capture session is presented on Figure 5.

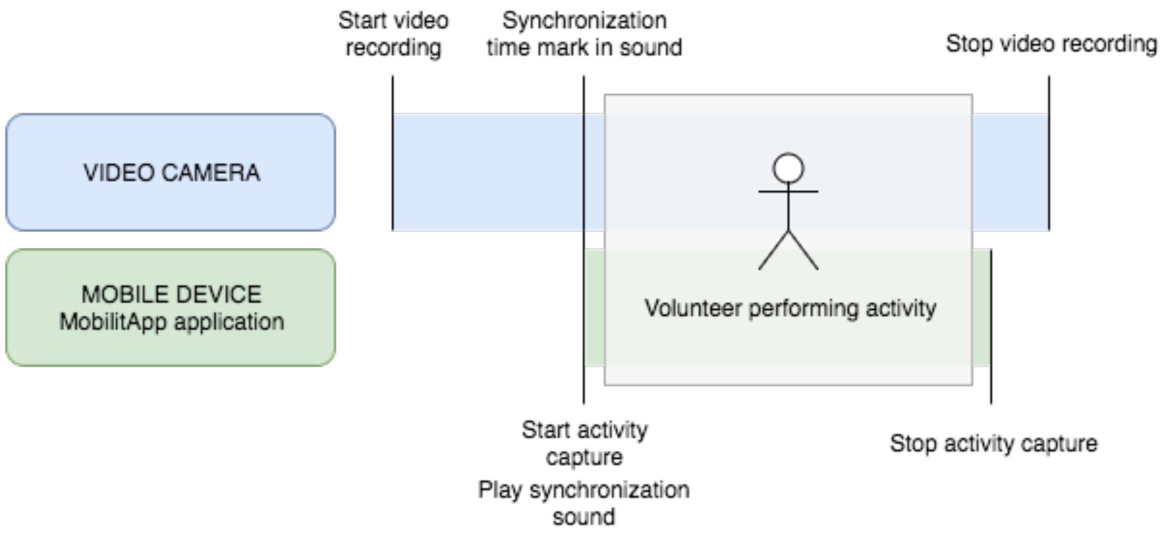

Start video recording
Synchronization time mark in sound
Stop video recording
VIDEO CAMERA
MOBILE DEVICE MobilitApp application
Volunteer performing activity
Start activity capture
Play synchronization sound
Stop activity capture

Figure 5: Timeline of capture session

Our plan was to capture sensors data at the highest possible sampling rate, so we defined sensor capture delay in out application to Android predefined sensor data delay value SensorManager.SENSOR_DELAY_FASTEST. This value defines that sensor value is saved with delay of 0ms. But after initial captures we figured out that after few minutes or running app we ended up with really large files. We also noticed that we did not get same file sizes for captures of same length on different and also on same phones. Fluctuations in file sizes occurred because of the way sensors work on Android. [21] On Android the data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the onSensorChanged() callback method. When you specify sensor data delay on Android, this is only suggested delay and Android system or other applications can effect this delay. So phones processing power also has effect on this. If phone is suddenly under a load by other application this can effect on sensor data delay. Based on all of this information we can conclude, that on Android it is impossible to capture sensors data at fixed sampling rate. In [2] they discovered that frequency of accelerometer did not have effect on classification. Because of that and issues presented before, we decided to increase sensor data delay value to 10000ms. That would be sampling rate of 100 Hz. For this sampling rate we decided based on [10], where they say that vibrations caused by motorized transportation are in frequency band [18.5 Hz – 45 Hz]. Using a larger delay also imposes a lower load on the processor and therefore uses less power.

From initial measurement we also discovered that in motorized activities like using car, train, tram, metro, or any other activity where there is no excessive movement of device, in some moments device goes to sleep and capture is interrupted – gaps in data appear. To solve this problem, while capturing activity, we acquired Android *PowerManager.PARTIAL_WAKE_LOCK*, which keeps phone's Central Processing Unit (CPU) awake to capture data without interruptions.

For each captured session we have four files: accelerometer, magnetometer, metadata and video file. Filename structure of these files is shown in Figure 6.

```
DeviceID_CaptureID_ActivityName_ACCELEROMETER_date_hour.csv
DeviceID_CaptureID_ActivityName_MAGNETICFIELD_date_hour.csv
DeviceID_CaptureID_ActivityName_METADATA_date_hour.txt
DeviceID_CaptureID_ActivityName_VIDEO_date_hour.mp4
```

Figure 6: Filename structure of data files

Accelerometer and magnetometer file structure is shown in Figure 7. File starts with header line and it is continued with actual timestamp, x, y, and z sensors values with each sample in his own line.

```
Timestamp, x, y, z
Timestamp₁, x₁, y₁, z₁
...
TimestampN, xN, yN, zN
```

Figure 7: Accelerometer and magnetometer file structure

To describe what is happening in the session we introduced metadata file where, with help of captured video, we define sections of activity when it actually happened. Structure of metadata file is presented in Figure 8. File starts with integer number *StartCaptureTime* in first line, representing time in milliseconds used for synchronization of metadata and sensor files. In following lines, two integer numbers, *SectionStartTime* and *SectionEndTime,* and string *ActivityClassName* are representing start and end time in milliseconds and class name for each observed section in its own line.

In some cases, capture was interrupted or something specific happened, that we would like to exclude from data. For this purpose, we introduced dummy class name “ignore”, so these sections can be then excluded in further process. Typical metadata file includes sections of actual activity, stationary and ignored parts, if they happend.

```
StartCaptureTime
SectionStartTime₁ SectionEndTime₁ ActivityClassName₁
...
SectionStartTimeN SectionEndTimeN ActivityClassNameN
```

Figure 8: Metadata file structure

4.1.1 Updates in MobilitApp application for gathering data

To gather data from sensors we had to make some changes in the existing project application. We used existing select transport pop-up dialog from previous student [22] working on this project, but we written our own class *SensorLoger*, to capture data from sensors. Instead of pop-up dialog for transportation type selection showing when the application is opened, we introduced *start/stop* button in top bar (Figure 9 screenshots a and c), which starts and stops capture of accelerometer and magnetometer. When user presses button *start*, he is prompted with pop-up dialog as shown on Figure 9 screenshot b, where he choses which kind of activity or transportation mode will he use. In this pop-up dialog we also included device ID and serial number of capture, so that we can match sensor data with captured reference video. After pressing stop button, user is prompted with pop-up dialog, with option to discard or save current captured data, as shown on Figure 9 screenshot d.

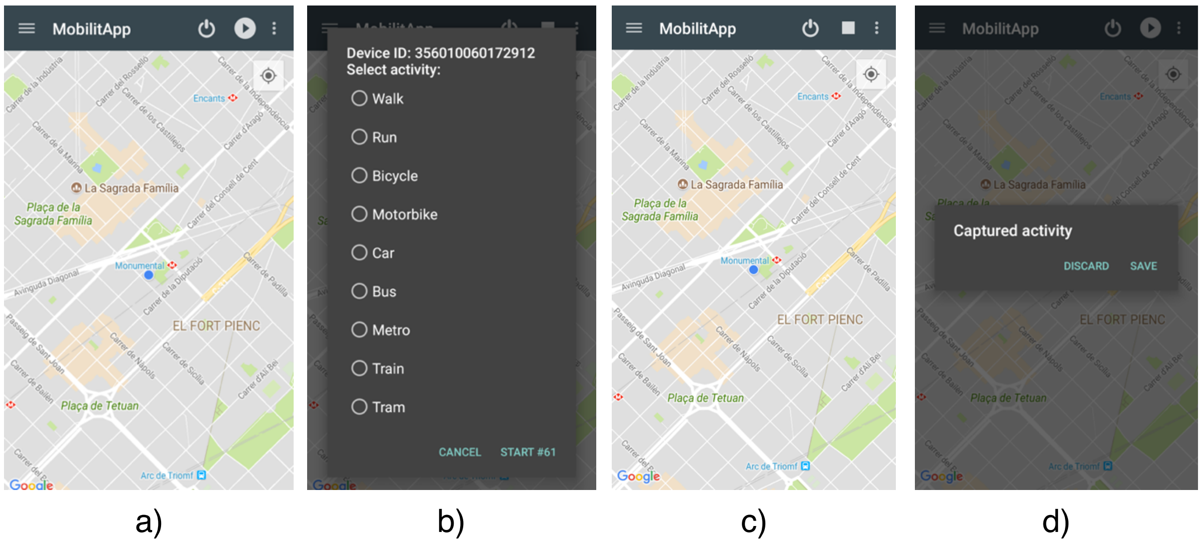


Figure 9: Screenshots of updated MobilitApp application

4.1.1.1 Main activity

In main activity in *onOptionsItemSelect* function, we added code (Figure 10) to respond to start/stop record activity button. When pressing *start* button, if capture is not running, we open *transportation type selection* dialog, else we stop capture and open *discard or save capture* dialog.

```java
else if (itemId == R.id.addActivity) {

   this.setFinishOnTouchOutside(true);

   if (sensorLoger.isCaptureRunning()) {

      sensorLoger.stopCapture();
      item.setIcon(android.R.drawable.ic_media_play);

      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setTitle("Captured activity");

      builder.setPositiveButton("SAVE", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int which) {
            sensorLoger.saveCapture();
         }
      });
      builder.setNegativeButton("DISCARD", new DialogInterface.OnClickListener()
      {
         @Override
         public void onClick(DialogInterface dialog, int which) {
            sensorLoger.discardCapture();
            dialog.cancel();
         }
      });

      builder.show();

   } else {
      SelectTransportFragment selectTransportFragment =
         new SelectTransportFragment();
      selectTransportFragment.show(getSupportFragmentManager(),
         "select_transport");
      item.setIcon(R.drawable.ic_stop_capture);
   }
   return true;
}
```

Figure 10: Added code in MainActivity.java in onOptionsItemSelected function

To avoid continuous capturing in background in case of when capture is running and application is exited, we added single line of code (Figure 11) to *onDestroy* function and shutdown button handler.

```java
sensorLoger.discardCapture();
```

Figure 11: Line of code to discard capture

4.1.1.2 SensorLoger class

As we mentioned before, we developed our own class for capturing sensor data, which is derived from *Services* and *sensorEventListener* classes. We also written *CSVFile* class for easier writing data to file, but we will not present it here as it is not relevant to our work.

On Figure 12 we see constructor of *SensorLoger* class. In constructor we define *WakeLock with PowerManager.PARTIAL_WAKE_LOCK* paramter, which is required for uninterrupted capturing of data from sensors. Instances of accelerometer and magnetometer sensors are also defined in constructor.

```java
public SensorLoger(Context mContext) {

    this.mContext = mContext;

    powerManager = (PowerManager) mContext.getSystemService(POWER_SERVICE);
    wakeLock = powerManager.newWakeLock(
       PowerManager.PARTIAL_WAKE_LOCK,
       "WakeLockSensorCapture");

    mSensorManager = (SensorManager)
                     mContext.getSystemService(Context.SENSOR_SERVICE);
    sAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    sMagneticField = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
}
```

Figure 12: Constructor of SensorLoger class

In onSensorChanged rutine, as shown on Figure 13, we get value from sensor and save it to the list of corresponding sensor.

```java
@Override
public void onSensorChanged(SensorEvent event) {

    float x = event.values[0];
    float y = event.values[1];
    float z = event.values[2];

    timestamp = new Date().getTime();

    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

        accelerometerData.add(new SensorSample(timestamp, x, y, z));
    }
    else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {

        magneticFieldData.add(new SensorSample(timestamp, x, y, z));
    }
}
```

Figure 13: onSensorChanged rutine in SensorLoger class

We also written few supporting functions as shown in Figure 14.

```
public boolean isCaptureRunning() {

    return isCaptureRunning;
}

public boolean isReadyToSave() {

    return !isCaptureRunning && !isCaptureSaved;
}

public boolean isNewCaptureAvailable() {

    return !isCaptureRunning && isCaptureSaved;
}
```

Figure 14: Support functions in SensorLoger class

In *startNewCapture* rutine on Figure 15, we acquire *WakeLock* and start the capture with registering sensor listeners. Wright before capture is started we also call function *playSynchronizationSound*, which generates and plays high pitch sound for synchronization of video and sensor data.

```
public void startNewCapture(int captureID, String activityType) {

    if (isCaptureRunning()) {
        Toast.makeText(this, "Capture start failed! Please try again.",
            Toast.LENGTH_SHORT).show();
        return;
    }

    // Lock phones CPU from going to sleap
    wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
        "WakeLock");
    wakeLock.acquire();

    Log.d("SENSOR", "Start capture");

    accelerometerData = new ArrayList<SensorSample>();
    magneticFieldData  = new ArrayList<SensorSample>();

    this.captureID = captureID;
    this.activityType = activityType;

    // Play sound for synchronition of video and sensor files
    playSynchronizationSound();

    // Register sensor listeners - start capture
    mSensorManager.registerListener(this, sAccelerometer, SENSOR_SAMPLING_PERIOD,
        new Handler());
    mSensorManager.registerListener(this, sMagneticField, SENSOR_SAMPLING_PERIOD,
        new Handler());

    this.isCaptureRunning = true;
    this.isCaptureSaved = false;
}
```

Figure 15: startNewCapture rutine in SensorLoger class

On Figure 16 we see *stopCapture* rutine, which unregisters sensors and releases *WakeLock*.

```
public void stopCapture() {

    if (!isCaptureRunning()) return;

    Log.d("SENSOR", "Capture finished");

    // Unregister listeners - stop capture
    mSensorManager.unregisterListener(this);

    wakeLock.release();

    isCaptureRunning = false;
    isCaptureSaved = false;
}
```

Figure 16: stopCapture rutine in SensorLoger class

Rutine *discardCapture* on Figure 17 is used in case if user discards the capture or when application is exited while capture is still running.

```
public void discardCapture() {

    if (isCaptureRunning()) {
        stopCapture();
    }

    Log.d("SENSOR", "Discard capture");

    this.isCaptureSaved = true;
}
```

Figure 17: discardCapture rutine in SensorLoger class

Figure 18 shows *saveCapture* rutine, where data from both sensors is saved in separate files. This is done in separate thread, to insure uninterrupted workflow of application.

```java
public void saveCapture() {

    if (!isReadyToSave()) {
        return;
    }

    new Thread(new Runnable() {
        public void run() {

            Log.d("SENSOR", "Saving sensor data to files");

            final TelephonyManager tm = (TelephonyManager)
                mContext.getSystemService(Context.TELEPHONY_SERVICE);

            String FILENAME_FORMAT =
                tm.getDeviceId()
                + "_" + String.format("%04d", captureID)
                + "_" + activityType
                + "_%s" // sensor type
                + "_" + new SimpleDateFormat("dd.MM.yyyy_HH.mm.ss")
                    .format(Calendar.getInstance().getTime())
                + ".csv";

            // Save accelerometer data
            String filename = String.format(FILENAME_FORMAT, "ACCELEROMETER");
            CSVFile csv = new CSVFile(FILE_STORE_DIR, filename);

            csv.open();
            csv.writeLine("timestamp,x,y,z");
            csv.writeData(accelerometerData);
            csv.close();

            // MAGNETICFIELD
            filename = String.format(FILENAME_FORMAT, "MAGNETICFIELD");
            csv = new CSVFile(FILE_STORE_DIR, filename);

            csv.open();
            csv.writeLine("timestamp,x,y,z");
            csv.writeData(magneticFieldData)
            csv.close();

            isCaptureSaved = true;

            Log.d("SENSOR", "Data successfully writed to files");
        }
    }).start();
}
```

Figure 18: saveCapture rutine in SensorLoger class

4.2 Model development using feature vectors built based on signal analysis and experimentation

The raw input data is often too large or complex, noisy and redundant for machine learning. So standard practice in pattern recognition is transformation of signal into a new (smaller) space of variables (features) that simplify analysis. Feature is as measurable property of the observed phenomenon, usually containing information relevant for pattern recognition.

After we gathered all the data it was time to review and preprocess data. As we gathered data from two sensors which each has tree axis, we are now dealing with 6 signals. Analyzing each axis separate would not make sense, as different orientation of device has different impact on each separate axis.

In following section, we present our first approach of developing model, using feature vectors.

4.2.1 Interpolation

Inconsistent capture frequency of embedded mobile device sensors leads us to use of interpolation on accelerometer and magnetometer sensor. We decided to interpolate norm of accelerometer to exact 100Hz. After examining FFT before and after interpolation or norm of accelerometer, we discovered that difference, as shown on Figure 19, is negligible, and so we decided not to use interpolation. This way, in future when we implement algorithm on mobile device, we save some processing power and with that also battery life. We should also mention, that frequencies of accelerometer and magnetometer are not always the same and they vary time to time. So number of samples in sliding windows might also vary.

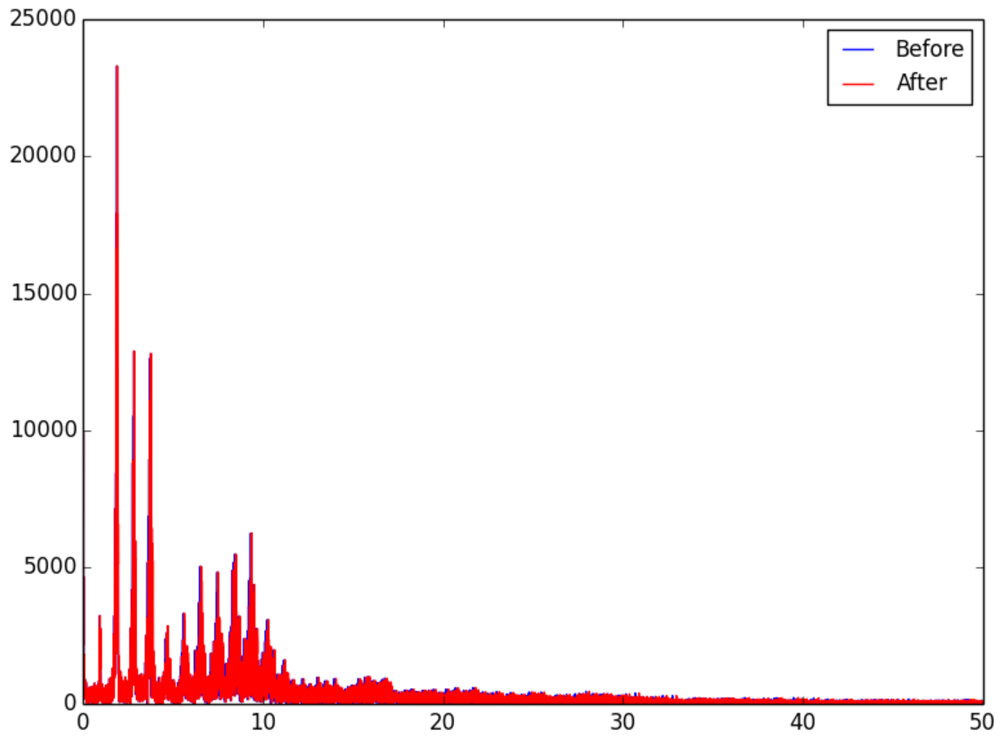


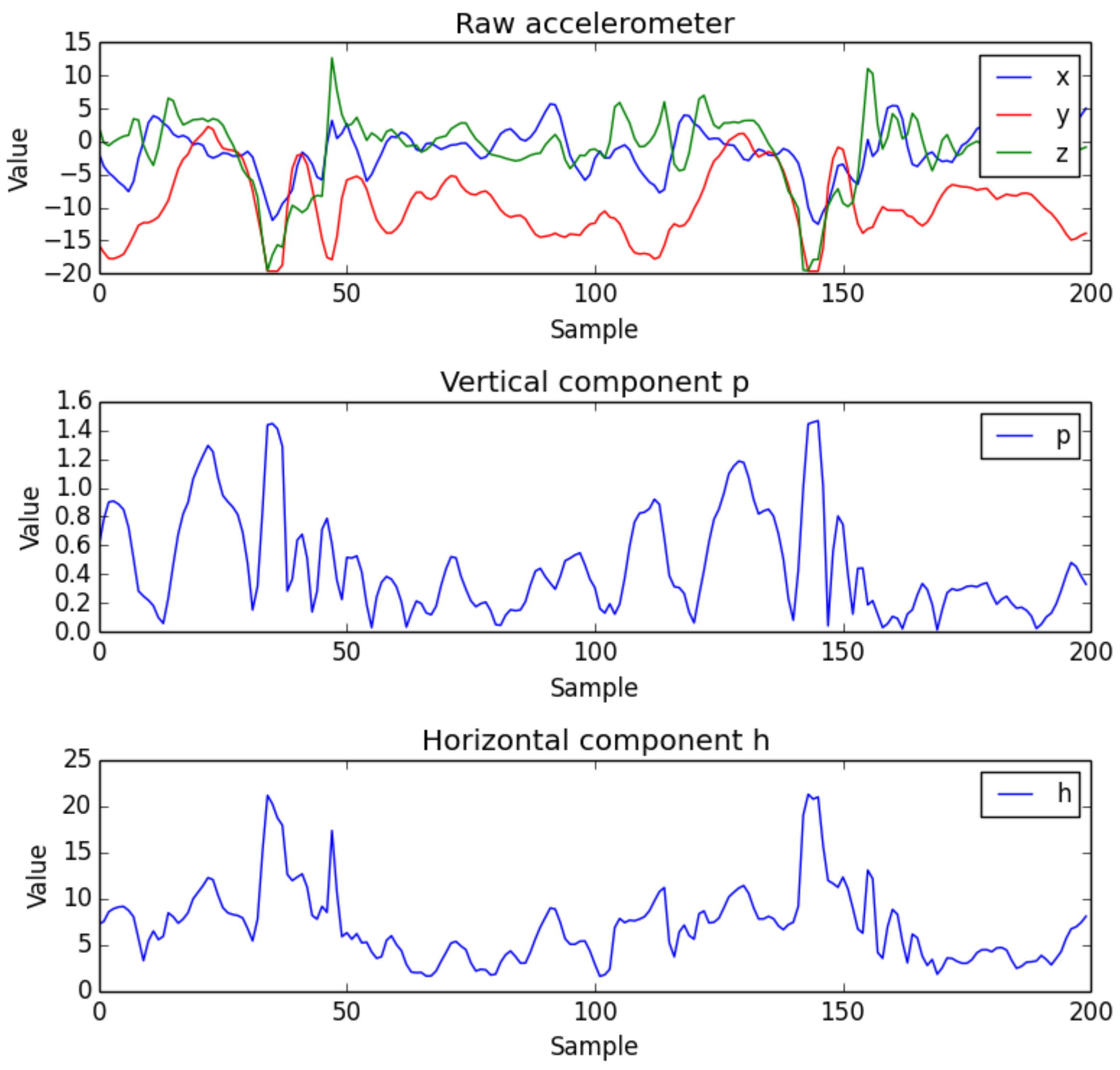
Figure 19: FFT of norm of accelerometer before and after interpolation

4.2.2 Eliminating mobile device orientation restrictions

To eliminate mobile device orientation restrictions, we had to implement some method to neutralize the orientation. We decided to use two approaches.

In literature [13] we discovered method of estimating accelerometer orientation using gravity. We used this method as first approach to eliminate orientation restrictions as described in section 3.1. Result of this method implemented on accelerometer data is decomposition to vertical *p* and horizontal *h* component as shown on Figure 20.

Figure 20: Accelerometer orientation estimation using gravity



We also addressed the problem with second solution of our own. With calculating norm of a samples of x, y and z values of accelerometer we get signal that should not be effected by device orientation. We discovered that norm values are shifted up by some constant. From stationary state of device, we figured out, that this constant is constant of gravity. By subtracting gravity constant from norm values, we get signal that is floating around x axis. NormG is our own method to eliminate device orientation restrictions and is calculated as:

$$NormG_i = \sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2} - G \qquad (10)$$

where x, y and z stands for values of accelerometer and *G = 9.80665* stands for gravity constant. Example of method applied to accelerometer data is shown on Figure 21.

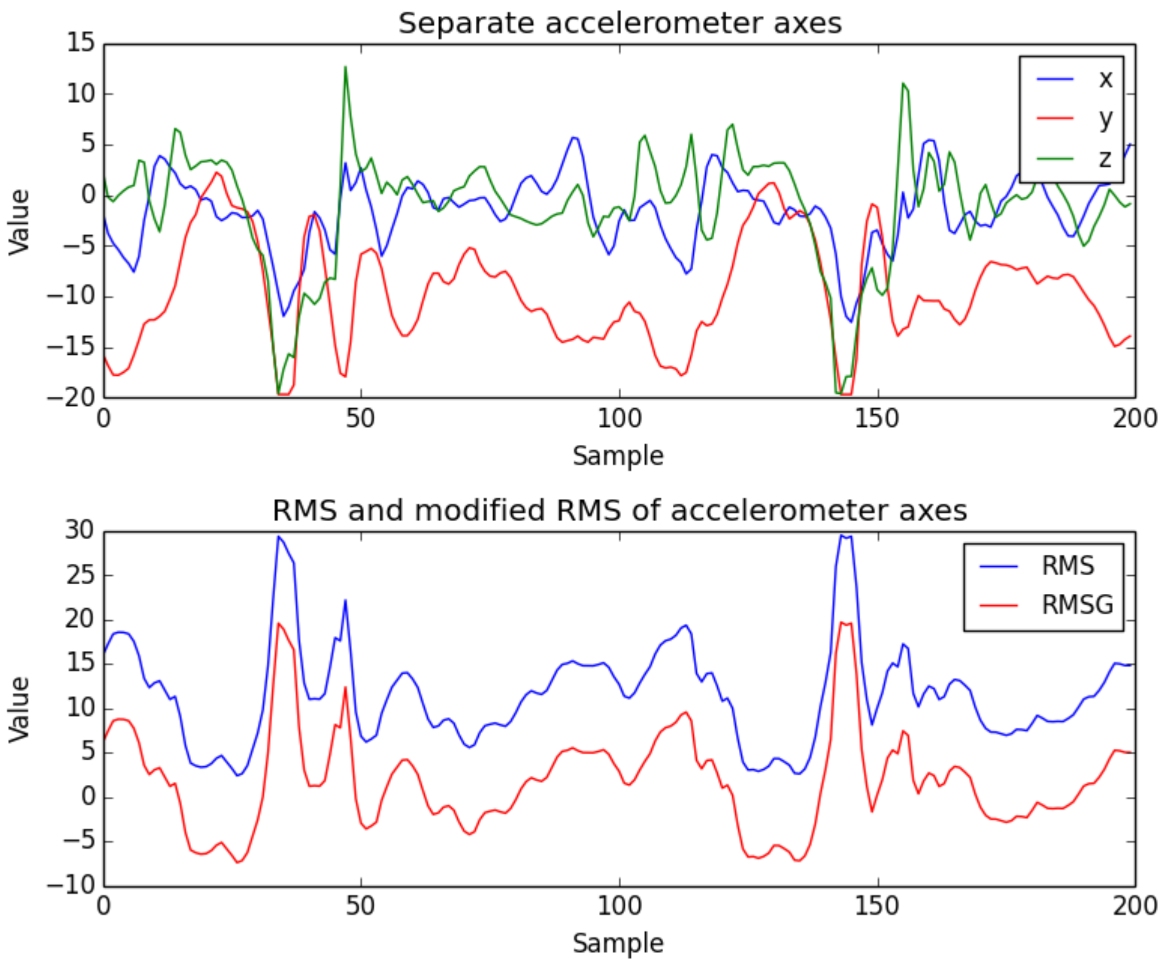


Figure 21: Development of norm without gravity

4.2.3 Overview and analysis of captured samples of separate activity

Good way to start analysis of signals is to visualize them. For each activity we drawn graphs of NormG of accelerometer, norm of magnetometer, vertical component *p* and horizontal component *h* of accelerometer and FFT of NormG and vertical component *p* of accelerometer. On graphs we marked the stationary parts with gray background color, and white background presents the time when activity was actually happening.

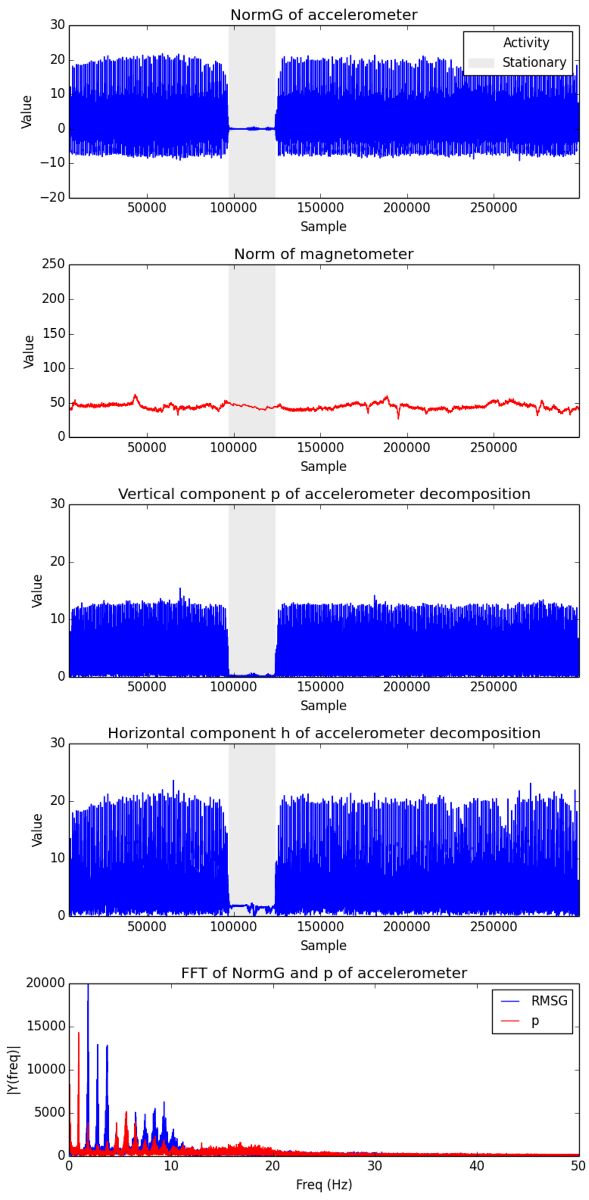


Figure 22: Walk

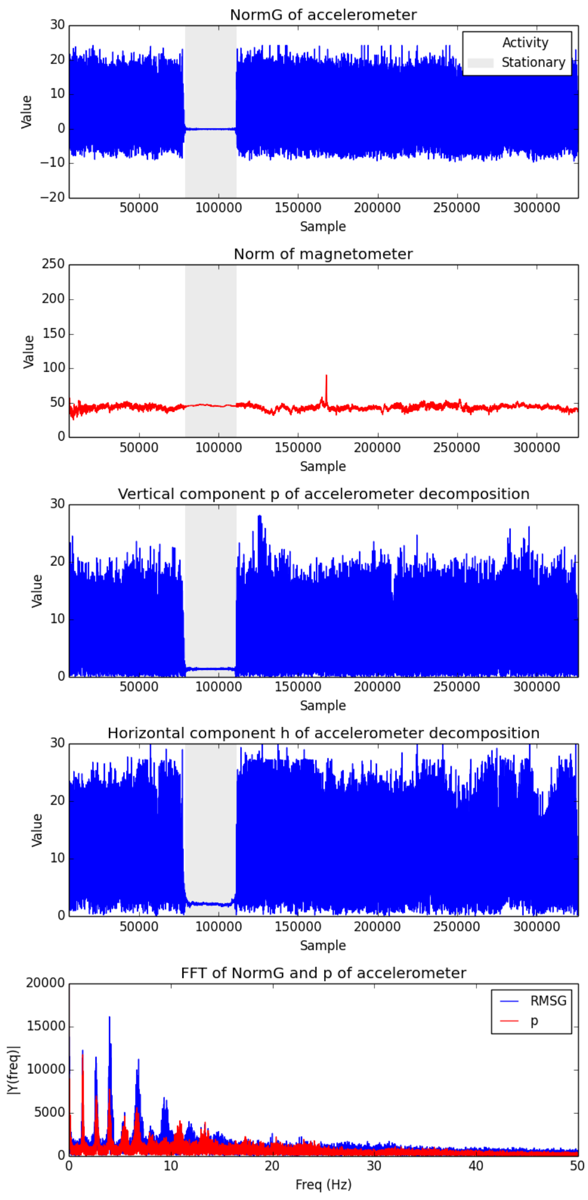
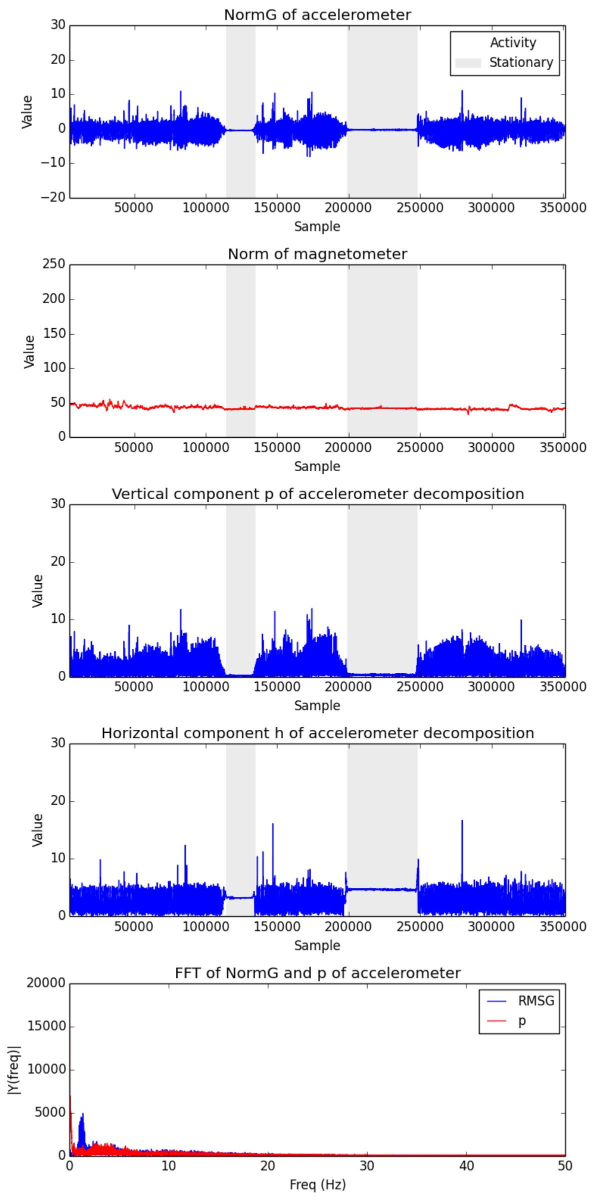


Figure 23: Run

On Figure 22 and Figure 23 we can see graphs for activities walk and run. We can differentiate those two activities from each other and the rest of activities by just looking at the vertical and horizontal components of accelerometer and FFTs.



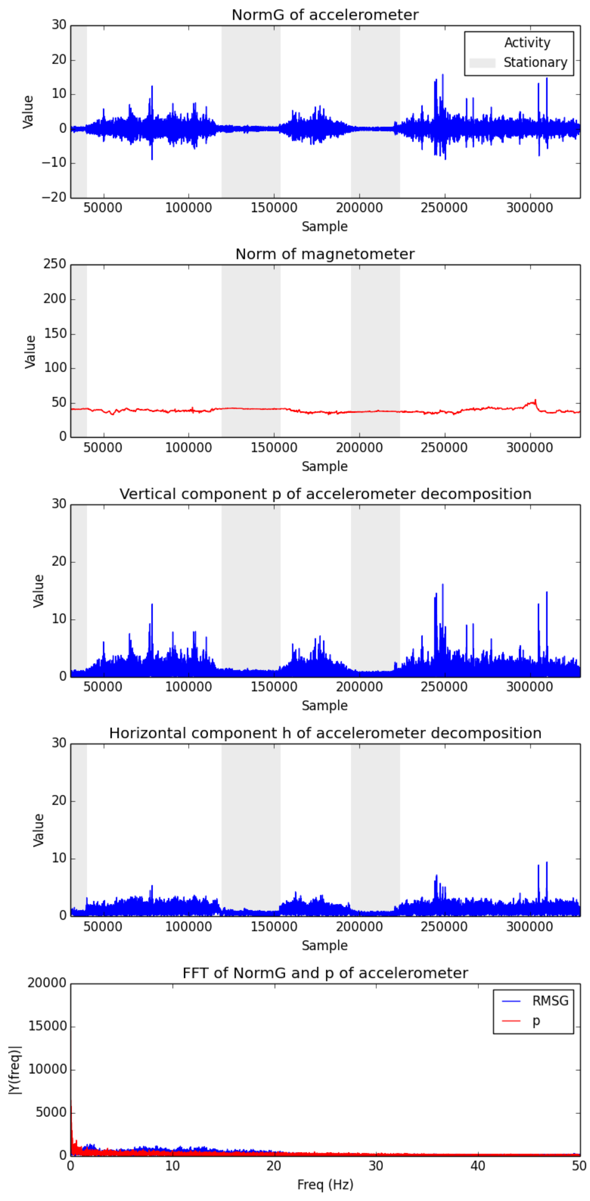


Figure 24: Bicycle

Figure 25: Motorcycle

From Figure 24 of bicycle activity we can also see larger values in vertical and horizontal component of accelerometer. We have more or less consistent value of horizontal component. In vertical component we can start to see some spikes, which we assume are from the vibrations on the road. This is also seen on Figure 25, which present motorcycle activity.

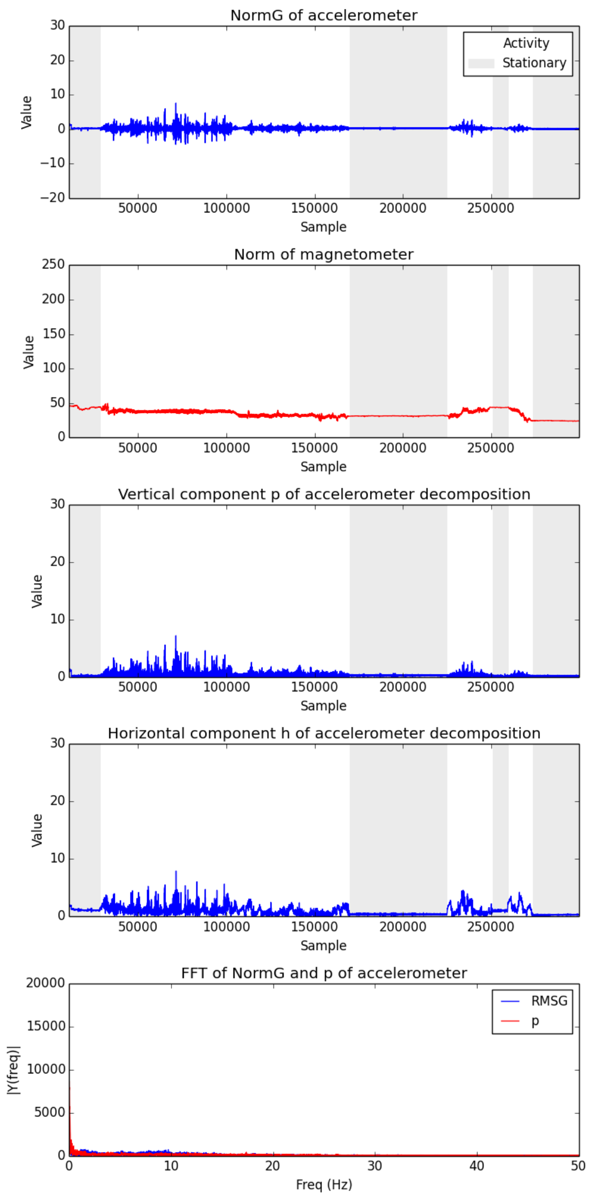


Figure 26: Car

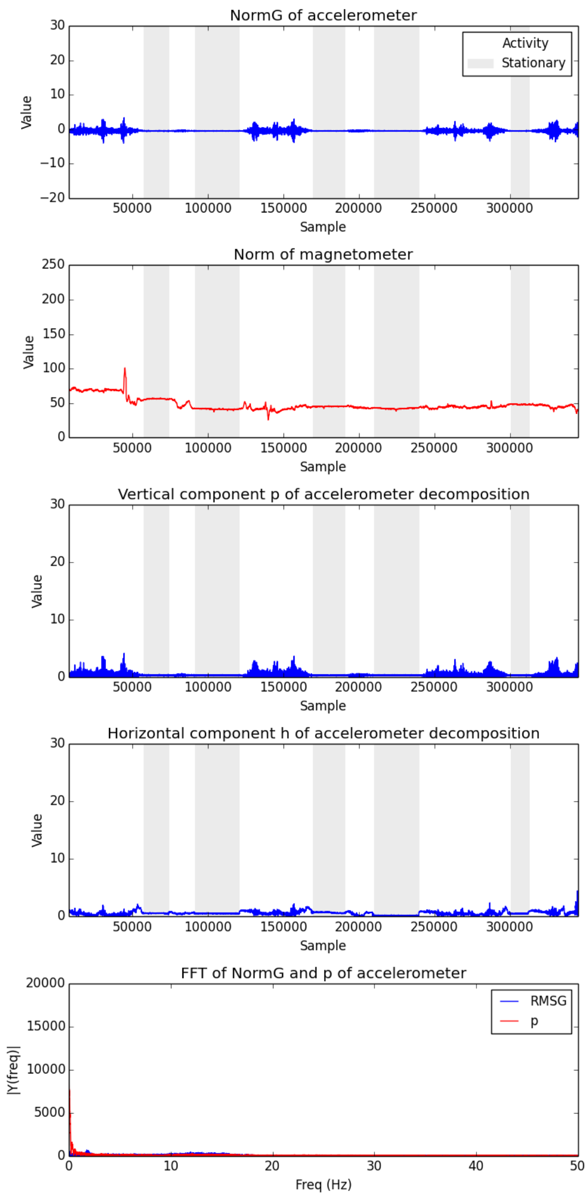


Figure 27: Bus

We can see similarities in vertical and horizontal component on Figure 26 when driving with the car. Compared to bus activity on Figure 27, at car horizontal component has higher presence, as car is much shorter and so more G force is applied to device when driving fast through corner or changing lane. Ride on bus is usually smooth with occasionally strong vertical accelerations coursed by holes or bumps on the road. This is clearly seen on graph.

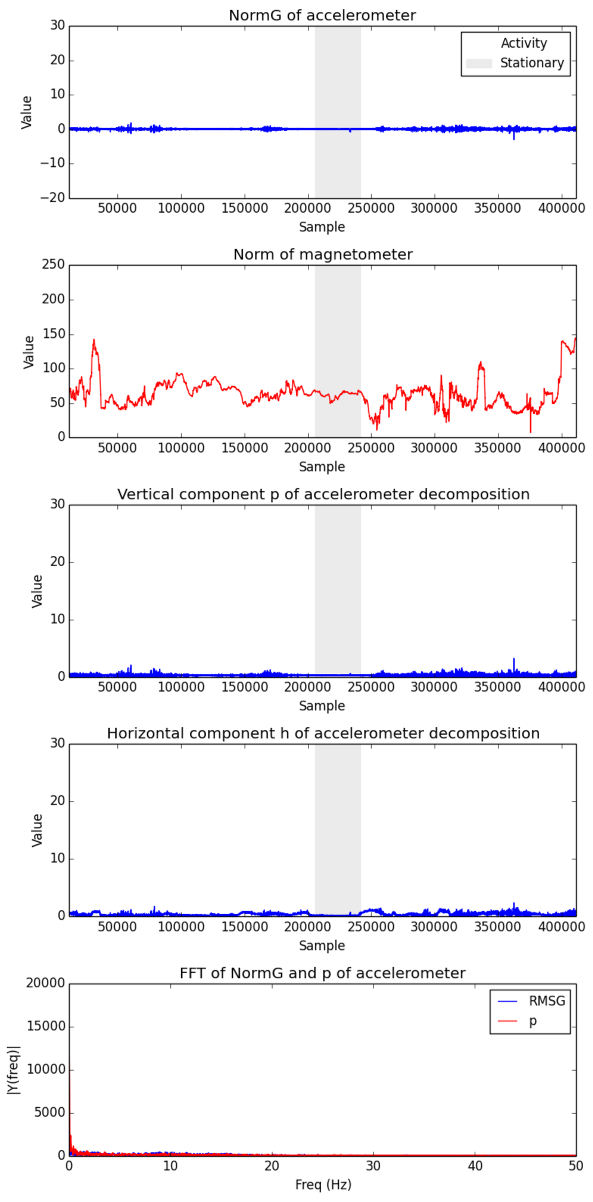


Figure 28: Train

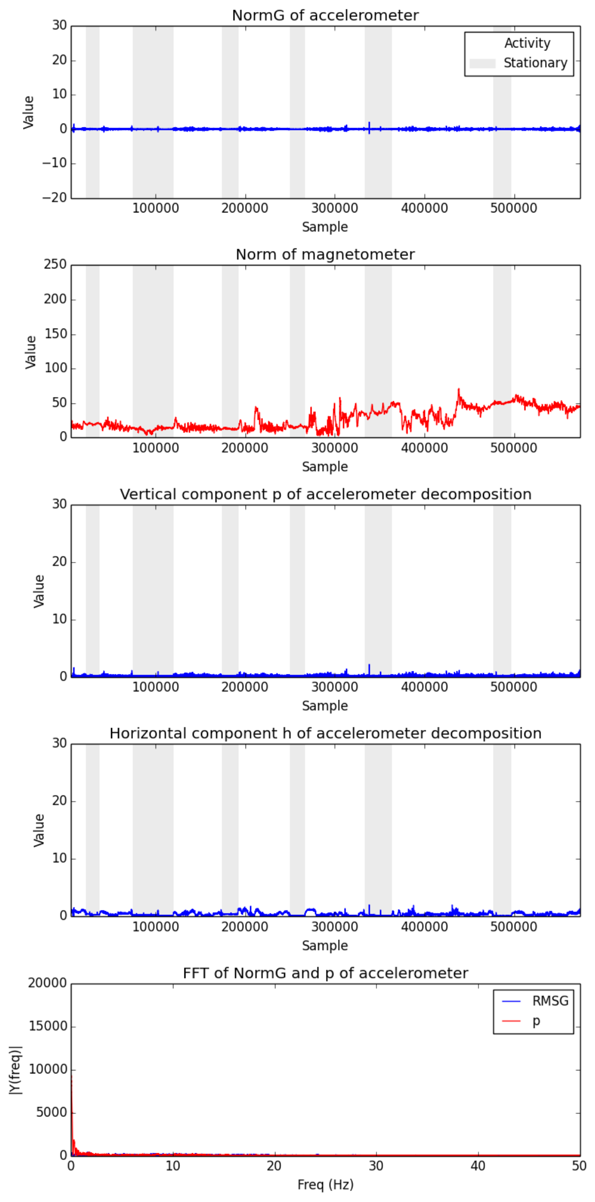


Figure 29: Tram

Rides with train and tram are usually very smooth with slow accelerations and decelerations. On Figure 28 and Figure 29, which represent train and tram activities, we can see low values of vertical and horizontal acceleration. We can also notice the change in accelerometer behavior, as train and tram run on electric motors.

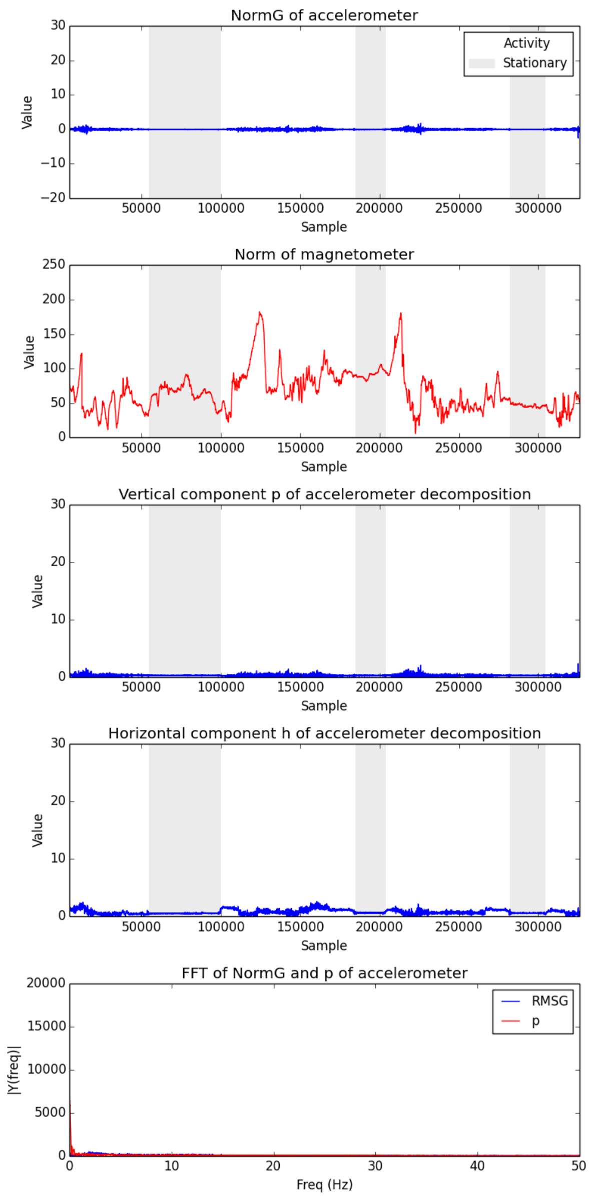


Figure 30: Metro

On final Figure 30 of metro activity, we can see the biggest effects on magnetometer as metro is also running on electric motors and in most of the time it runs in closed space underground, what probably contains in and magnifies magnetic field. Horizontal accelerations on metro are also higher than on train and metro.

4.2.4 Feature vector assembly

From suggestions in literature of existing solutions and overview and analyze of our data, we used following methods to extract features:

- Standard deviation,

- Energy,
- Spectral centroid,
- Spectral spread,
- Zero-crossing rate,
- Minimum,
- Maximum.

Before applying these methods, we transformed our raw signals using accelerometer decomposition and NormG, methods explained in section 4.2.2. In [10] they also calculate FFT of vertical component *p* of accelerometer, and from that energy in four separate frequency bands. From analyzing our data, we discovered that FFT of NormG of accelerometer has more energy and different characteristic for different activities, than FFT of vertical component *p*. Energy is calculated in *[1Hz-4Hz], [5Hz-9Hz], [10Hz-19Hz]* and *[20Hz-45Hz]* bands, similar as in [10]. Based on experimentation we assembled the following feature vector, that produces best results with machine learning:

$$\begin{bmatrix} acc\_std \\ mag\_std \\ acc\_p\_std \\ acc\_h\_std \\ acc\_normg\_e\_b1 \\ acc\_normg\_e\_b2 \\ acc\_normg\_e\_b3 \\ acc\_normg\_e\_b4 \\ mag\_norm\_e\_b1 \\ mag\_norm\_e\_b2 \\ mag\_norm\_e\_b3 \\ mag\_norm\_e\_b4 \\ acc\_p\_sc \\ acc\_v\_ss \\ mag\_rms\_sc \\ mag\_rms\_ss \\ acc\_rmsg\_zcr \\ \max\_p \end{bmatrix} \qquad (11)$$

where:

- ***acc_std*** is standard deviation of accelerometer,
- ***mag_std*** is standard deviation of magnetometer,
- ***acc_p_std*** is standard deviation of accelerometers vertical component *p*,
- ***acc_h_std*** is standard deviation of accelerometers horizontal component *h*,

- ***acc_normg_e_b1*** and ***mag_norm_e_b1*** are energies of FFT of NormG of accelerometer and norm of magnetometer in frequency band [1Hz-4Hz],
- ***acc_normg_e_b2*** and ***mag_norm_e_b2*** are energies of FFT of NormG of accelerometer and norm of magnetometer in frequency band [5Hz-9Hz],
- ***acc_normg_e_b3*** and ***mag_norm_e_b3*** are energies of FFT of NormG of accelerometer and norm of magnetometer in frequency band [10Hz-19Hz],
- ***acc_normg_e_b4*** and ***mag_norm_e_b4*** are energies of FFT of NormG of accelerometer and norm of magnetometer in frequency band [20Hz-45Hz],
- ***acc_p_sc*** is spectral centroid of vertical component *p* of accelerometer,
- ***acc_p_ss*** is spectral spread of vertical component *p* of accelerometer,
- ***mag_norm_sc*** is spectral centroid of vertical component *p* of magnetometer,
- ***mag_norm_ss*** is spectral spread of vertical component *p* of magnetometer,
- ***acc_normg_zrc*** is zero-crossing rate of NormG of accelerometer
- ***min_p*** is minimum value of vertical component *p* of accelerometer.
- ***max_p*** is maximum value of vertical component *p* of accelerometer.

Our goal is for the detection to happen in pseudo real-time, so we chosen 1s sliding window to generate data set for training and testing our system. Choosing larger sliding window would also increase the probability of other activity happening in the same window. Using 1s sliding window (75% overlapping) and information in metadata files, we iterated through accelerometer and magnetometer data files. For every 1s window, we calculate feature vector and save it to data set. With this approach we created following data sets *DS1*, *DS2* and *DS3* (Figure 31) where in:

1. ***DS1***: stationary parts of all activities are grouped into one global “stationary” class,
2. ***DS2***: stationary parts of each activity, are part of it’s activity class, except stationary parts of walk, run and bicycle, which are grouped into “stationary” class,
3. ***DS3***: classes car, motorcycle, bus are grouped into “road” class and classes metro, train, tram are grouped into “rail” class, where stationary parts of each activity, are part of it’s activity class, except stationary parts of walk, run and bicycle, which are grouped into “stationary” class.

In *DS2* and *DS3* we treat stationary parts of each activity, as part of activity, except for activities walk, run and bicycle, of which stationary parts have similar characteristics and are not effected by surrounding vehicle.

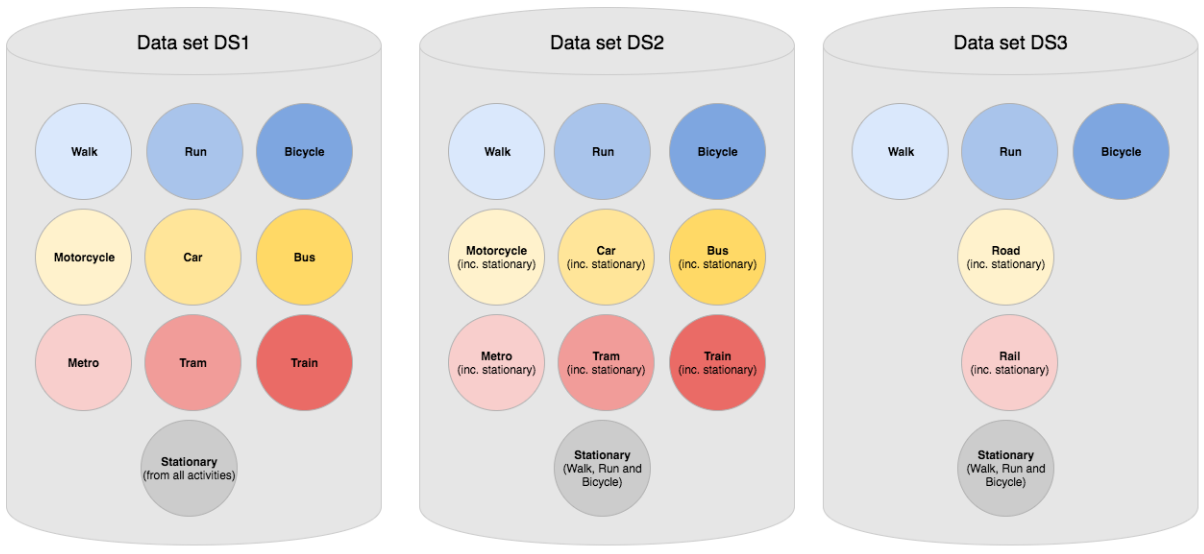


Figure 31: Data sets

4.2.5 Machine learning using developed feature vectors

For machine learning we used standard Decision Tree [23], Random Forest [24] and Gradient Boosting [25] method with Decision Tree classifiers.

[26] In statistics and machine learning we usually split our data into to subsets: training data and testing data (and sometimes to three: train, validate and test), and fit our model on the train data, in order to make predictions on the test data. When we do that, one of two thing might happen: we overfit our model or we underfit our model. We do not want any of these things to happen, because they affect the predictability of our model — we might be using a model that has lower accuracy and/or is ungeneralized. In order to avoid this, we can perform something called cross validation. It's very similar to train/test split, but it's applied to more subsets of data. There are few cross validation methods and one of them is K-fold cross validation, that we used. To asses the model we split our data into *k* subsets, and train on *k-1* one of those subset. What we do is to hold the last subset for test. We do this for each subset and present average of all *k* iterations. To evaluate our method with K-fold cross validation, we used parameter *k=4* or 4 folds. Before applying split to *k* parts we shuffled the data always using the same seed. On Figure 32 we can see code for K-fold cross validation for three different types of model. For each model classification metrics are printed to screen, so we can choose the best one. Later, the best performing model is trained with whole data set and tested on separate test samples as shown on Figure 33. Best type of model is them used for rest of the tests, that we performed.

```
# Initialize models
dt = DecisionTreeClassifier(max_depth=17, class_weight='balanced')
rf = RandomForestClassifier(n_estimators=80)
gb = GradientBoostingClassifier(n_estimators=10,
     learning_rate=0.1, max_depth=3, random_state=0)

# Initialize K-Fold split
kf = KFold(n_splits=4, random_state=12345, shuffle=True)

# Execute K-fold cross validation for each model
print "Decision Tree"
predicted = cross_val_predict(dt, data, target, cv=kf)
cm = metrics.confusion_matrix(target, predicted)
mbt.classificationMetrics(cm)

print "Random Forest"
predicted = cross_val_predict(rf, data, target, cv=kf)
cm = metrics.confusion_matrix(target, predicted)
mbt.classificationMetrics(cm)

print "Gradient Boosting"
predicted = cross_val_predict(gb, data, target, cv=kf)
cm = metrics.confusion_matrix(target, predicted)
mbt.classificationMetrics(cm)
```

Figure 32: K-fold cross validation

```
# Train model
model = rf.fit(trainData, target)

# Test model
predicted = model.predict(testData)
cm = metrics.confusion_matrix(target, predicted)
mbt.classificationMetrics(cm)
```

Figure 33: Train and test Random forest model

4.3 Model development using Neural Networks

[27] Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute real biological brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron and typically, neurons are organized in layers.

The original goal of our second approach using neural network was to solve problem in the same way that a human brain would. [27] Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as back-propagation, or passing information in the reverse direction and adjusting the network to reflect that information. The idea of this approach was to figure out if neural network can detect type of user activity or transportation mode, if we feed raw data from sensor to input of neural network. This might be difficult for human to solve, but machines are faster and better with numbers, and they also learn faster.

There are many variations of neural network systems and training methods. In our approach we used multilayer feed-forward neural network with supervised back-propagation learning.

Our goal was to use raw data from sensors, but we have to consider that we want to, for our system to work in every position and orientation of mobile device. We also have to merge data of accelerometer and magnetometer into one input vector. Concatenating x, y and z axes of two sensors would yield large input for neural network. Using 1s sliding window and capturing frequency 100Hz we should get 100 samples per each axes, but as we discovered, that is not always the case. As in this approach we are using raw data and the size of input data must be exact as number of input neurons, we were forced to use interpolation. To address problem with orientation restrictions we decided to do the decomposition of accelerometer, as we did in first approach in section 4.2.2, and calculate norm of magnetometers x, y and z axes. From accelerometer decomposition we used vertical component *p* and norm of magnetometer to form new concatenated input vector as:

$$[a_1, a_2, \dots, a_N, m_1, m_2, \dots, m_N] \qquad (12)$$

where *a* represent samples vertical component *p* of accelerometer, *m* represent samples of a norm of magnetometer and *N* is the length of signal of both sensors.

In our network we have input, output and one hidden layer. Number of neurons in input layer is defined by size of input vector, and number of neurons in output layer is defined by number of classes. From [28] we considered following rule to select the number of neurons in hidden layer:

$$H = O + (0.74I) \quad and \quad h < 2I \qquad (13)$$

where *H* is the number of hidden neurons, *I* is the number of input neurons and *O* is the number of output neurons. As we have 10 classes to differentiate (*O=10)* and input vector of 200 samples (*I=200)*, we get *H=158* hidden neurons.

To build and train neural network we used Python library Pybrain and code written in Figure 34 where *n* is the instance of neural network, and *t* is the instance of back-propagation trainer. Variable *numOut* represents number of output neurons, *numHid* number of hidden neurons and *numOut* number of output neurons. Instead of default linear activation function of output neurons, with *outclass* parameter, we defined sigmoid function instead. [29] With setting *bias* to *True*, we allow activation function to be shifted left or right, which may be critical for successful learning. *BackpropTrainer* constructor accepts instance of neural network, training dataset, *momentum* of learning and *weightdecay* parameters, which were selected by experimenting. Finally, function *trainUntilConvergence* trains network until convergence. To test the network, we activate it with test data set as shown on Figure 35. Result of *activateOnDataset* function are vectors of output values of output layer. Every output neuron corresponds with one activity, and the neuron with highest value is selected as detected activity.

```
n = buildNetwork(numOut, numHid, numOut, outclass=SigmoidLayer,
    bias=True)
t = BackpropTrainer(n, dataset=trainDS, momentum=0.5, weightdecay=0.01)
t.trainUntilConvergence()
```

Figure 34: Code used to train neural network

```
r = t.activateOnDataset(testDS)
```

Figure 35: Code to activate neural network

5 DATA COLLECTION PROCESS AND ANALYSIS OF RESULTS

In this chapter we present and compare results of both approaches used to develop our system. We also describe environment where and how data was collected.

5.1 Experimental environment and matrixes

While capturing data we did not specifically define position or orientation of mobile device. We let user to put away his phone to desired location. In most cases happened that users had their device in front pocket, in some cases in back pocket and few cases in lady's purse. Standing or sitting position restriction were also not defined, but we were strict not to combine activities like walking or running on bus, metro, train etc. We also asked user not do do any sudden movements while recording. Since riding bicycle and not pedaling has completely different characteristics we asked user to pedal as much as possible. Sections riding bicycle when user is not pedaling, were discarded.

We captured data with help of tree volunteers using their own device. We also introduced the fourth device. Every volunteer did all activities using his and our fourth device recording at same time. Then we captured all activities again using 3 devices each in separate session and in different environment (position of user in vehicle and location of capturing activity). We should mention that while gathering data we use only one motorcycle and two different cars. Each session lasted at least 5 minutes and in total we captured over 7 hours of data. In Table 5 whole amount of captured data is presented. At the end we also captured 2 minute sessions for final tests.

Table 5: Amount of captured data of activities and transportation modes

Activity	Activity time	Stationary time	Total time
Walk	00:38:50	00:04:16	00:43:07
Run	00:37:40	00:05:36	00:43:16
Bicycle	00:43:03	00:04:45	00:47:48
Motorcycle	00:35:15	00:12:37	00:47:52
Car	00:30:58	00:13:44	00:44:42
Bus	00:39:23	00:15:00	00:54:24
Metro	00:38:43	00:13:21	00:52:05
Train	00:39:05	00:13:47	00:52:52
Tram	00:36:42	00:12:11	00:48:53
Sum	05:39:39	01:35:17	07:14:59

To assess the error in our approach, we use the metrics such as precision, recall, accuracy, and F1 score which are defined as follows:

$$Precision = \frac{tp}{tp + fp} \tag{14}$$

$$Recall = \frac{tp}{tp + fn} \tag{15}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{16}$$

$$F1\,score = 2\frac{Precision * Recall}{Precision + Recall} \tag{17}$$

where tp, fp, tn and fn stand for true positive, false positive, true negative and false negative.

Precision (14) is the proportion between number of correctly detected samples of activity and number of samples when activity actually happened. *Recall* (15) is the proportion between correctly detected samples and number of all samples that should be classified as activity. In other words, recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives. *Accuracy* (16) is proportion between correctly predicted observation to the total observations. [30] *F1 score* (17) is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than Accuracy, especially if you have an uneven class distribution.

5.2 Qualitative and quantitative results

In this section we present the results of our system. Our second approach with Neural networks did not converge so we were unable to run tests. Neural networks proven not to be appropriate for this problem. All results in this section are from our first approach where we tested the model using K-fold cross validation with *k=4* parameter, and later we also tested the model on separate small test data set.

From results we can see that using only Accuracy metrics is not quite reliable if Precision and Recall have low value. Instead we evaluate our model by F1 score.

5.2.1 K-fold cross validation

In Table 6 we can see results of K-fold cross validation using three different types of model (Decision tree, Random forest and Gradient boosting using decision trees). In this data set (*DS1*), stationary parts of each activity are grouped into one global “stationary” class. Precision, Recall, Accuracy and F1 score of results are presented for each method. We will mainly focus on F1 score as, Accuracy can be deceiving. Random forest method proven to deliver best results of all three methods in F1 score for all activities. With 82%, it scored lowest for bicycle activity, with 98% highest for transportation mode tram and in average 88% in F1 score. For Decision tree and Gradient boosting methods we can see that they performed good in stationary, motorcycle, car, bus and tram but not in other activities.

Table 6: Results of K-fold cross validation with global stationary class (DS1)

	Decision tree				Random forest				Gradient boosting			
Class	Precision	Recall	Accuracy	F1 score	Precision	Recall	Accuracy	F1 score	Precision	Recall	Accuracy	F1 score
Stationary	0,92	0,92	0,98	0,92	0,93	0,96	0,99	0,94	0,87	0,92	0,97	0,89
Walk	0,71	0,73	0,94	0,72	0,85	0,84	0,97	0,85	0,53	0,57	0,90	0.55
Run	0,66	0,70	0,95	0,68	0,85	0,81	0,97	0,83	0,60	0,31	0,92	0,41
Bicycle	0,71	0,72	0,94	0,72	0,82	0.82	0,96	0,82	0,57	0,49	0,90	0,53
Motorcycle	0,80	0,86	0,97	0,83	0,84	0,91	0,98	0,87	0,64	0,87	0,93	0,74
Car	0,95	0,95	0,99	0,95	0,99	0,95	0,99	0,97	0,97	0,93	0,99	0,95
Bus	0,87	0,77	0,91	0,82	0,85	0,88	0,93	0,86	0,80	0,81	0,89	0,80
Metro	0,77	0,76	0,95	0,76	0,89	0,83	0,97	0,86	0,67	0,52	0,92	0,58
Train	0,68	0,78	0,94	0,73	0,83	0,83	0,97	0,83	0,51	0,65	0,90	0,57
Tram	0,96	0,96	0,99	0,96	0,97	0,98	1.00	0,98	0,94	0,95	0,99	0,94
Average	0,80	0,82	0,96	0,81	0,88	0,88	0,97	0,88	0,71	0,70	0,93	0,70

For all proceeding tests we used Random forest classifier, which produced the best results of all three methods. To prove our 3rd hypothesis we performed K-fold cross validation on data set (*DS2*), where stationary parts of each activity (except from classes walk, run and bicycle) are not treated as global “stationary” class, but as part of it’s own activity. Results of this K-fold cross validation are in Table 7. Compared to results of Random forest classifier in Table 6 we can say that performance improved significantly. With average F1 score of 92%, 87% for bus activity was the lowest score and highest for tram activity with 98%.

Table 7: Results of K-fold cross validation using Random Forest when stationary parts are part of activity (DS2)

Class	Precision	Recall	Accuracy	F1 score
Stationary	0,94	0,95	0,99	0,94
Walk	0,89	0,93	0,98	0,91
Run	0,91	0,89	0,98	0,90
Bicycle	0,88	0,90	0,97	0,89
Motorcycle	0,92	0,96	0,99	0,94
Car	0,99	0,96	1.00	0,97
Bus	0,92	0,83	0,99	0,87
Metro	0,93	0,90	0,98	0,92
Train	0,91	0,91	0,98	0,91
Tram	0,97	0,98	1.00	0,98
Average	0,93	0,92	0,98	0,92

In Table 8 we can see the results of K-fold cross validation of Random forest classifier on generalized data set (*DS3*), where classes motorcycle, car, and bus were grouped into "road" class, and classes metro, train, tram were grouped into "rail" class. With this data set we achieved the best results, compared to all other K-fold cross validations and two other data sets. With average of 94% of F1 score, the road class scored the lowest score with 83% and bicycle activity highest with 97%. From generalized group rail, we can say that metro, tram and train transportation modes share similar characteristic. But we can not say this for transportation modes motorcycle, car and bus that were grouped into road class, as they perform better in both previous K-fold cross validations with Random forest, when they are treated as separate class.

Table 8: Results of K-fold cross validation using Random Forest with simplified class range (DS3)

Class	Precision	Recall	Accuracy	F1 score
Stationary	0,94	0,95	0,99	0,94
Walk	0,95	0,96	0,97	0,96
Run	0,93	0,95	0,96	0,94
Bicycle	0,99	0,95	1,00	0,97
Road	0,95	0,73	0,99	0,83
Rail	0,98	0,98	1,00	0,98
Average	0,98	0,92	1,00	0,94

5.2.2 Tests with separate test data sets

After training the model using Random forest classifier and whole data set that was used for K-fold cross validation, we also performed tests with separate small test data set.

In Table 9 we can see results of test of Random forest classifier trained on data set DS1 and tested with separate test data set. Tram, bus, car and stationary classes scored 89% in average F1 score. And the rest of classification results we would not consider good. Score of 15% for bicycle activity, was the lowest score in this test.

Table 9: Results of Random forest on test group with global stationary class (data set DS1)

Class	Precision	Recall	Accuracy	F1 score
Stationary	0,68	0,94	0,94	0,79
Walk	0,50	0,70	0,91	0,59
Run	0,57	0,23	0,89	0.33
Bicycle	0,31	0,10	0,89	0,15
Motorcycle	0,73	0,71	0,91	0,72
Car	0,98	0,95	0,99	0,97
Bus	0,81	0,90	0,91	0,85
Metro	0,57	0,56	0,89	0,56
Train	0,31	0,42	0,85	0,36
Tram	0,90	0,96	0,99	0,96
Average	0,64	0,65	0,98	0,62

In Table 10, where Random forest classifier was trained on data set DS2 and tested on separate test data set, compared to Table 9 we achieved worse results. Stationary, run, car and train classes scored the same F1 score as in Table 9. For rest of the classes F1 score dropped for between 5-7%. With average 59% this test scored the lowest in F1 score.

Table 10: Results of Random Forest on test group where stationary parts are part of activity (DS2)

Class	Precision	Recall	Accuracy	F1 score
Stationary	0,68	0,94	0,93	0,79
Walk	0,47	0,64	0,82	0,54
Run	0,47	0,25	0,78	0,33
Bicycle	0,23	0,06	0,84	0,10
Motorcycle	0,66	0,68	0,87	0,67
Car	0,99	0,94	0,99	0,97
Bus	0,71	0,75	0,96	0,73
Metro	0,45	0,57	0,84	0,50
Train	0,32	0,41	0,79	0,36
Tram	0,90	0,97	0,98	0,93
Average	0,59	0,62	0,88	0,59

From results in Table 9 and Table 10, we can say that results for some classes are good but not good for rest of them. From good K-fold cross validation results and this two tests we can assume that classes with bad results have low variance in training data set, which could be improved with acquiring larger data set.

Before we already mentioned that we achieved best results in K-fold cross validation with *DS3* data set, where we simplified class range. Benefit of generalization of classes is also shown on results of test with separate test data, which are presented in Table 11. With DS3 data set we achieved best results with K-fold cross validation as well as with this test among other two tests with separate test data. Overall these results are considered pretty good except for samples of class road, just like in K-fold cross validation in Table 8, which confirms our thought that metro, train and tram share similar features, but motorcycle, car and bus not as much. Comparing results in Table 10 and Table 11, we can se difference of only 11% in average F1 score.

Table 11: Results of Random Forest on test group with simplified class range (DS3)

Class	Precision	Recall	Accuracy	F1 score
Stationary	0,70	0,93	0,95	0,80
Walk	0,82	0,83	0,88	0,82
Run	0,82	0,77	0,84	0,80
Bicycle	0,98	0,94	0,99	0,96
Road	0.74	0.57	0,97	0.65
Rail	0,89	0,97	0,98	0,93
Average	0,83	0,84	0,94	0,83

5.2.3 Summary of the results

Overall we achieved good results with K-fold cross validation, but based on results of tests with separate test data, we think additional work is required, to improve variance in data with acquiring larger data set.

In Table 8 with results of K-fold cross validation using Random Forest with simplified class range (DS3) we achieved best results of all tests, with average Accuracy of 100% and F1 score 94%. But our main objective was to classify between specific transportation modes, so results in Table 7 of K-fold cross validation using Random Forest when stationary parts are part of activity (DS2), are more relevant to us. Here we achieved Accuracy of 98% and F1 score of 92%.

Our results are comparable or in some cases outperforming existing solutions. We should mention that all tests have been done offline and goal for future work is to include the algorithm within the MobilitApp application to do further tests, as it is pointed out in next chapter 6.

6 CONCLUSION AND FUTURE WORK

In final chapter we open the discussion about results, problems and challenges that we dealt with through the work. We also look in possible future work and conclude this work with final word.

In this work we reviewed existing solutions and worked up on them with our ideas, to develop system, that is able to detect specific user activities and transportation modes such as stationary, walking, running, riding a bicycle, motorcycle, driving a car, taking a bus, metro, tram and train. Since the energy consumption of GPS, relative to the usage of embedded sensors, is very high, we completely eliminated usage of GPS in our detection algorithm.

As we wanted for detection to happen in pseudo real-time we decided to use 1s sliding window. Using larger sliding window would also increase probability of two different activities happening in same window.

Our goal was to develop system using two approaches. In approach using neural networks we discovered, that neural networks are not suited for the problem that we presented to them. Our idea was to feed the raw data from embedded mobile device sensors to the input of neural network and the network will be able to differentiate between the activities. With assembled training data, our network never converged. In other approach we assembled feature vector based on analysis of signals of separate activities and experimentation. From initial tests with standard decision tree, Random forest and Gradient boosting methods, we chose Random forest as the best performing classifier to run the rest of the tests. To test the performance of model we used popular and effective K-fold cross validation method. Results just of K-fold cross validation are comparable and in some cases outperforms some of best existing solution. But later we also trained the model on whole data set used for K-fold cross validation and tested it on separate small test data set. From those tests, which were not as good, we discovered, that we have low variance in data, which can be improved with acquiring larger data set. We performed tests on three different data sets. In first data set we treat stationary parts of all activities as separate common “stationary” class. In second data set we treated stationary parts as part of activity, except for stationary parts of walk, run and bicycle, which we grouped into common “stationary” class. In the third data set where in addition to applying the same principal as in second data set, we also grouped corresponding classes to “road” and “rail” classes. In this master thesis we defined three hypotheses and attained following main goals:

- ***Hypothesis 1:*** *By using only embedded mobile device sensors, without usage of GPS, is possible to build vector of features and model, which is by efficiency of detection of type of user activity and transportation mode, comparable with “state of the art” solutions, if we use short time window up to 3 s.* Based on K-fold cross validation results, we can confirm this hypothesis. Using 1s sliding window, developed feature vector and machine learning, we can detect user activity and transportation mode without use of GPS.
- ***Hypothesis 2:*** *From raw signal of embedded mobile device sensors captured using short time window up to 3 s, that we feed directly into the neural network, neural network is capable to extract features and build model, which is by efficiency of detection of type of user activity and transportation mode, comparable with “state of the art” solutions.* Based on our neural network not converging, this hypothesis overruled. We discovered that neural networks are not able to differentiate between user activities and transportation modes just using raw data from sensor.
- ***Hypothesis 3:*** *Treating stationary parts of different user activities and transportation modes as one common “stationary” class, gives better classification results than using stationary parts as part of it’s activity.* As we got better results with second data set, where stationary parts are part of it’s activity, this hypothesis is overruled. Meaning that vibrations and magnetic field in vehicle is still present in stationary mode and identification of moving and stationary parts is not necessary. Walk, run and bicycle activity’s stationary sections can not be treated as part of activity as they posses different characteristic as moving sections.

As none of reviewed existing solutions performed classification with such range of specific transportation modes, we also run K-fold cross validation on our third generalized data set, to compare results with existing solutions. With this data set we achieved the best results in K-fold cross validation and with separate test data set as well.

Overall we achieved good results, that are comparable or in some cases outperforming existing solutions, but based on results of tests with separate test data, additional work is required. Pending objectives to develop in future work:

- Main goal is to eventually implement the system on the mobile platform.
- We think it is also essential to capture much larger data set. Capturing data is not as time consuming as reviewing the footage. With overruling our third hypothesis, we see that detailed dissection (identifying stationary and moving parts) of activity is not necessary and so capturing data could be executed with less effort in much larger scale (different positions, more devices/volunteers). But we would still suggest some level of control while capturing. Mobile devices have such a high presence in our life, that we are usually event not aware of them until we need them again. From experience at the beginning of our captures, we know that after you start the capture and you put your device in the pocket you eventually forget about it. You have to be focused on capturing all the time or other activities or interruptions could appear, which could corrupt the data. So we think that training or supervision of volunteers, to capture large data set, is necessary.
- We also think that there is still space to improve and optimize developed feature vector. One way to do this is to minimize number of features based on detailed analysis of feature importance.
- If we look at the transitions between the different activities we can tell, that transitions between some activities are less likely to happen. For example, after riding tram, train or metro you do not use the car, bus, bicycle right away, but instead walk, run or stationary activities probably happened between. From this observation we can suggest some kind of post processing of the classification. Hidden Markov model is one of the methods that could be used for that.

We were keen to join this interesting project, and we would like to thank all the members working on this project for support. We ran into some difficulties with gathering data, and after technical problems were solved, we successfully gathered proper data in third try. As we joined this project as part of ERASMUS student exchange program, we only had limited time to work on this project. We hope to continue this work and improve our system, so in future it can be used in the final application.

Bibliography

[1] Mobilitat project, Universitat Politècnica de Catalunya, "Privacy policy," [Online]. Available: http://mobilitat.upc.edu/policy.html. [Accessed 7 9 2017].

[2] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen and M. Srivastava, "Using Mobile Phones to Determine Transportation Modes," *ACM Transactions on Sensor Networks (TOSN),* vol. 6, no. 2, p. 13, 2010.

[3] K. Muthukumar, "Google's Activity Recognition API is awesome but where are the apps?," 10 10 2015. [Online]. Available: Back in 2013, Google launched the ActivityRecognitionAPI to developers. [Accessed 1 7 2017].

[4] Google Inc., " ActivityRecognitionApi," [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi. [Accessed 1 7 2017].

[5] Apple Inc., "CMMotionActivity," [Online]. Available: https://developer.apple.com/documentation/coremotion/cmmotionactivity. [Accessed 1 7 2017].

[6] Samsung Electronics, Co., Ltd., "Samsung Developers," 2016. [Online]. Available: http://developer.samsung.com/technical-doc/view.do?v=T000000211#none. [Accessed 1 7 2017].

[7] Samsung Electronics, Co., Ltd. , "New S Health Features Add Fun to Fitness," 5 7 2016. [Online]. Available: https://news.samsung.com/global/new-s-health-features-add-fun-to-fitness. [Accessed 1 7 2017].

[8] A. J. Lopez, D. Ochoa and S. Gautama, "Detecting Changes of Transportation-Mode by Using Classification Data," *18th International Conference on Information Fusion,* pp. 2078 - 2083, 2015.

[9] A. Jahangiri and H. A. Rakha, "Applying Machine Learning Techniques to Transportation Mode Recognition Using Mobile Phone Sensor Data," *IEEE Transactions on Intelligent Transportation Systems,* vol. 16, no. 5, pp. 2446 - 2456, 2015.

[10] O. Lorintiu and A. Vassilev, "Transportation mode recognition based on smartphone embedded sensors for carbon footprint estimation," *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC),* pp. 1976 - 1981, 2016.

[11] S. Reddy, B. Burke, D. Estrin, M. Hansen and M. Srivastava, "Determining Transportation Mode On Mobile Phones," pp. 25 - 28, 2008.

[12] C. Zhenyu, W. Shuangquan, S. Zhiqi, C. Yiqiang and Z. Zhongtang, "Online Sequential ELM based Transfer Learning for Transportation Mode Recognition," *2013 IEEE Conference on Cybernetics and Intelligent Systems (CIS),* pp. 78-83, 2013.

[13] D. Mizell, "Using Gravity to Estimate Accelerometer Orientation," *Seventh IEEE International Symposium on Wearable Computers,* vol. 1, no. 1, pp. 252 - 253, 3 11 2003.

[14] C. A. d. M. S. Quintella, L. C. V. Andrade and C. A. V. Campos, "Detecting the transportation mode for context-aware systems using smartphones," *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC),* pp. 2261-2266, 2016.

[15] W. Shuangquan, C. Canfeng and M. Jian, "Accelerometer based transportation mode recognition on mobile phones," *2010 Asia-Pacific Conference on Wearable Computing Systems,* pp. 44-46, 2010.

[16] Wikipedia, "Spectral centroid," [Online]. Available: https://en.wikipedia.org/wiki/Spectral_centroid. [Accessed 25 5 2017].

[17] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the CUIDADO project," 2004.

[18] Wikipedia, "Zero-crossing rate," [Online]. Available: https://en.wikipedia.org/wiki/Zero-crossing_rate. [Accessed 25 5 2017].

[19] Wikipedia, "Norm (Mathematics)," [Online]. Available: https://en.wikipedia.org/wiki/Norm_(mathematics). [Accessed 25 5 2017].

[20] Wolfram Research, Inc. , "L2 Norm," [Online]. Available: http://mathworld.wolfram.com/L2-Norm.html. [Accessed 25 5 2017].

[21] Google Inc., "Sensors Overview," [Online]. Available: http://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed 9 4 2017].

[22] G. M. Torregrosa, "Improvement of algorithms to identify transportation modes for MobilitApp, an Android Application to anonymously track citizens in Barcelona," Universitat Politècnica de Catalunya, Barcelona, 2016.

[23] Wikipedia, "Decision tree," [Online]. Available: https://en.wikipedia.org/wiki/Decision_tree. [Accessed 7 8 2017].

[24] Wikipedia, "Random Forest," [Online]. Available: https://en.wikipedia.org/wiki/Random_forest. [Accessed 7 8 2017].

[25] Wikipedia, "Gradient Boosting," [Online]. Available: https://en.wikipedia.org/wiki/Gradient_boosting. [Accessed 7 8 2017].

[26] A. Bronshtein, "Train/Test Split and Cross Validation in Python," 17 5 2017. [Online]. Available: https://medium.com/towards-data-science/train-test-split-and-cross-validation-in-python-80b61beca4b6. [Accessed 26 7 2017].

[27] Wikipedia, "Artificial neural network," [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed 22 8 2017].

[28] S. R. Shahamiri and S. S. B. Salim, "Real-time frequency-based noise-robust Automatic SpeechRecognition using Multi-Nets Artificial Neural Networks:A multi-views multi-learners approach," *Neurocomputing,* vol. 129, pp. 199-207, 10 4 2014.

[29] N. Kohl, "Role of Bias in Neural Networks," 10 3 2010. [Online]. Available: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks. [Accessed 27 8 2017].

[30] R. Joshi, "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures," 9 9 2016. [Online]. Available: http://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. [Accessed 23 8 2017].

[31] Google Inc., "Sensors," [Online]. Available: http://developer.android.com/reference/android/hardware/Sensor.html. [Accessed 9 4 2017].

[32] Vogella GmbH, "Android Sensor - Tutorial," 2016. [Online]. Available: http://www.vogella.com/tutorials/AndroidSensor/article.html. [Accessed 9 4 2017].

[33] A. Bloch, R. Erdin, S. Mayer, T. Keller and A. de Spindler, "Battery-Efficient Transportation Mode Detection on Mobile Devices," *16th IEEE International Conference on Mobile Data Management,* vol. 1, pp. 185-190, 2015.

[34] S. Hemminki, P. Nurmi and S. Tarkoma, "Accelerometer-Based Transportation Mode Detection on Smartphones," *11th ACM Conference on Embedded Networked Sensor Systems,* p. 13, 2013.

[35] K. Kamran and H. Howard, "Generation and Interpretation of Temporal Decision Rules," *International Journal of Computer Information Systems and Industrial Management Applications,* vol. 3, pp. 314-323, 4 2011.

[36] B. KamińskiEmail, M. Jakubczyk and P. Szufel, "A framework for sensitivity analysis of decision trees," 24 5 2017.

[37] R. Quinlan, "Simplifying decision trees," vol. 51, no. 2, pp. 497-510, 8 1999.