People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider, Biskra
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Order Number: RTIC20/M2/2018

Report

Presented to obtain the Academic Master Diploma in

Computer Science

Option: Networks and Technologies of Information and Communication

Crowdsensing for Driving Experience Improvement

By:

Noureddine Houssem Eddine

Defended the 25/06/2018, in front of the jury composed of:

Mrs Saouli Rachida	Professor	President
Mr Bachir Abdelmalik	Professor	Supervisor
Mr Bitam Salim	MCA	Examiner

Academic Year: 2017/2018

Acknowledgements

In the Name of Allah, the Most Beneficent, the Most Merciful.

All the praises and thanks be to Allah for giving me the strength to complete this project. I would like to express my gratitude to my supervisor Pr. Abdelmalik Bachir for his immense knowledge, motivation, understanding and support that contributed to the success of this project.

My deepest gratitude to my parents: Zohra and Bensayeh, for their love, kindness and support. Special thanks to my sisters and their children for their love and moral support during my studies.

Last but not least, I'd like to thank my professors and colleagues for the great learning experience we had through the past few years.

Abstract

Mobile crowd sensing (MCS) have proven to be a critical improvement for internet services and applications. It empowers the general population to participate in the sensing process using their smart phones that come equipped with different sensors and great processing capabilities. The data they generate can be used in various applications one of which is improving their daily routines such as driving. This thesis presents our solution to improve the driving experience of people as an android application based on MCS architecture, which addresses a phenomenon that might pose a danger to many drivers and that is speed bumps. We utilize the built-in accelerometer of smart phones as a sensor for bumps while driving. The resulting application managed to detect and alert the users of incoming bumps, while facing some limitations such as data quality or false alerts.

KEYWORDS: crowdsensing, driving experience improvement, internet, android, mobile application.

Contents

List of Figures

List of Source Codes

Abbreviations

MCS	**M**obile **C**rowd **S**ensing
MSN	**M**obile **S**ocial **N**etworking
LTE	**L**ong **T**erm **E**volution
GPS	**G**lobal **P**ositioning **S**ystem
PDO	**P**hp **D**ata **O**bject

Chapter 1

Introduction

Current advances in mobile phone and communication technologies have given people the ability to purchase smart phones with a wide array of sensors, and to be connected to the internet anytime and anywhere. Those advances have paved the way for a new technique for sensing and data collection: Mobile Crowd Sensing or MCS in short. MCS can be defined as a technique that give people with mobile devices capable of sensing to contribute data sensed or generated from their mobile devices. MCS provides a new way of perceiving the world, by involving anyone in the process of sensing and to greatly extend the service of IoT. Currently, many researchers and developers have explored different uses for MCS such as urban mobility, air pollution, weather prediction etc. However, some areas haven't been explored yet, and providing a solution that covers the whole world has proven to be a challenge. Applications that improves the driving experience of the general population has been either lacking, not being available for everyone or failing to meeting the requirement of different environments.

This thesis aims to present a solution for improving the driving experience based on MCS. This solution is represented in an android application that can detect and alert users about incoming bumps. This thesis is divided into 6 chapters: Chapter 2 discusses MCS, its features, architecture, challenges and also presents some related work. Chapter 3 discusses the android platform and its features. Chapter 4 presents the architecture and the design our the application. Chapter 5 outlines the implementation of the proposed solution and discusses the obtained results after testing it. Chapter 6 concludes this thesis with our future expectations of the proposed solution.

Chapter 2

Mobile Crowd Sensing

2.1 Introduction

In this chapter we will be introducing a new sensing technique called Mobile Crowd Sensing, present its history, features and the challenges that it faces. We'll be also presenting some related work based on MCS.

2.2 Definition of MCS

MCS is a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices, aggregates and fuses the data in the cloud for crowd intelligence extraction and people-centric service delivery[2].

2.3 MCS History

Crowd powered problem solving has been addressed in many research efforts. In 2005, Surowiecki revealed in a book titled The Wisdom of Crowds [3] a general phenomena of the aggregation of information in groups, resulting in decisions that are often better than could have been made by any single member of the group [2]. Another similar concept "collective intelligence" [4] has been presented as groups of individuals doing things collectively that seem intelligent. The similarity between MCS and those two concepts is in the crowd powered data collection and analyzing process [2].

Jeff Howeand and Mark Robinson, two senior editors from Wired Magazine coined the term "crowd sourcing" which is defined by the Merriam-Webster Dictionary[1] as the practice of obtaining needed services or content by soliciting contributions from a large group of people, and especially from an online community. Compared to MCS, crowd-sourcing focuses on the participation of online crowds[2].

2.4 Key features of MCS

2.4.1 Citizen Participation

MCS involves citizens in the sensing loop same as participatory sensing, but it moves it a step further by introducing new features:

- The data generation mode: MCS generates data using two methods instead of just one like in participatory sensing. One, Mobile sensing which functions in an individual manner leveraging the sensing capabilities from individual devices. Two, Mobile social networking (MSN) that links the online society with the physical world. MCS benefits from the large scale user-contributed data from the MSN that would help understand the dynamics of the city and society. The combination of those two data generation modes is a unique feature of MCS [5].
- The sensing style: having two different data generation modes changes the sensing style of MCS, and we can categorize it from a new dimension: user's awareness to the sensing task. For mobile sensing, data collection is the primary purpose of the MCS application. Therefore, the sensing task is explicit to the user as they are informed of it. And for the MSN, it is used for social interactions which generates data that is used implicitly without the user's knowledge for collective intelligence extraction [5].
- Volunteer organization: participants in the MCS applications can be complete strangers or can form an organized group of neighbors working on a shared problem. We can categorize the volunteer organizations into three categories or groups [5]:
  - Urban: Generally for applications with wide appeal, such as traffic, road condition and air pollution monitoring. The data contributed by the citizens (mostly

[1]https://www.merriam-webster.com/dictionary/crowdsourcing

strangers) participating in this urban-scale sensing is generally considered as low quality [5].

- Community: According to the Cambridge Advanced Learners Dictionary[2], community is defined as the people living in one particular area or people who are considered as a unit because of their common interests, social group or nationality. Community-based MCS is the crowdsourcing where task takers come from an existing community or could easily form a new community. The members trust each other and are more likely to interact with each other during the task execution process, resulting in high-quality data [5].
- Group: This refers to a loosely or opportunistically organized group of neighbors (i.e., spatially nearby phones) that collaboratively address a shared problem. In this mode, people usually form the so-called opportunistic or ad hoc groups, and the relation among group members can be weak or strong. Depending on the strength of the relation and the interactions between the members, the generated data can be average to high quality. The key techniques include group formation/identification and management. For instance, we need to partition the set of mobile devices based on the associated physical or social contexts [5].

2.4.2 Hybrid Human-Machine Systems

To motivate full-scaled user participation and enhance user experience, MCS systems should be developed in a human-centric manner. Particularly, it should address the following key problems:

- Motivation of human participation: Having a successful MCS system that is deployed in a large scale requires a sheer number of participants with the latter requiring an incentive or a motivation to participate. The type of motivation varies from human to another, like financial gain which is an important incentive in a wide range of situations. Interest, entertainment, social and ethical reasons can also be motivators in some situations even with no financial gain [5].
- Fusion of human-machine intelligence: The involvement of human participation in the sensing and computing process will lead to a mixture of human and machine

[2]http://dictionary.cambridge.org/dictionary/british

intelligence in MCS. By integrating the intelligence of both crowds and machines, MCS allows the creation of hybrid human-machine systems. However, such human intelligence and machine intelligence could either be complementary or conflicting, and they show strength and weakness to distinct computing problems. Therefore, novel approaches and techniques should be studied to have optimal fusion of human and machine intelligence in MCS systems [5].

– User security and privacy: The sharing of personal data in MCS applications can raise significant concerns about security and user privacy. Therefore, new privacy preserving approaches need to exist in the MCS system to make sure that the personal information of the participants cannot be obtained by an unwanted party [5].

2.4.3 Transient Networking

MCS applications rely mainly on leveraging the continuous advances in modern ubiquitous and heterogeneous communication technologies to provide transient network connection and effective collection of mobile sensing data. While different MCS applications or systems may have various connection architectures and communication requirements, most of them share the following characteristics:

– Heterogeneous network connection: Current mobile devices such as smartphones come equipped with multiple wireless communication interfaces like GSM, WiFi, and Bluetooth interfaces. While GSM and WiFi interfaces can provide network connectivity with preexisting communication infrastructure in relatively large regions, Bluetooth or WiFi can also provide short-range connection among mobile devices and form self-organized opportunistic networks for data sharing. Heterogeneous networks raise new research challenges and enriched opportunities for MCS applications by supporting transient networking services, such as connectivity, collaborative sensing, and data routing/transmission, for the participants crossing these multiple wireless networks [5].

– Time-evolving network topology and human mobility: The mobility of mobile devices and their carriers not only provides a nice coverage for sensing tasks in MCS but also brings challenges to communications. Network topology evolves over time, which makes it hard to find stable routes among mobile devices. Traditional routing protocols designed for static wireless networks or wired networks cannot handle highly dynamic topology to fulfill basic communication tasks in MCS. Fortunately, recent advances in mobile ad hoc networks and wireless sensor networks provide many low-cost ad hoc routing solutions [5].

– Disruption tolerance service: In some MCS applications, the sensing data from each individual mobile device does not need to be transmitted in real time or guaranteed to be complete and accurate. Therefore, such MCS systems can take advantage of disruption-tolerant networks, which only rely on intermittent network connectivity and have a much lower deployment cost [5].

– High scalability requirement: Since MCS relies on sensing data from a large volume of mobile users, the scalability is clearly a basic requirement and key challenge for underlying communication systems. To achieve sufficient scalability, the MCS communication protocols and network architectures are usually highly distributed and decentralized. Such solutions can also improve the robustness of the overall MCS system. In addition, energy-efficient design has to be considered for MCS due to the limited power resources of each individual device and the large number of devices in the system [5].

2.4.4 Crowd Data Processing and Intelligence Extraction

MCS applications aim to extract high level intelligence from large volume of data contributed by its participants. Extracted intelligence can be classified into three dimensions regarding its value and beneficiary [5]:

– User awareness refers to the extraction of personal contexts (e.g., location, activity) and behavioral patterns (e.g., mobility patterns).

– Ambient awareness is to learn the status (e.g., noise level of a bus stop, traffic dynamics of a street) or the semantics (the logical type) of a particular space. The space can be small (e.g., a restaurant) or large (e.g., a city-wide area).

– Social awareness is about the contexts of a group or a community, such as social activity type, interpersonal relations, and so on.

2.4.5 Heterogeneous, Cross-Space Data Mining

The data collected in MCS systems is generated from both offline and online communities. Each community has its own features in relation to geographical coverage, infrastructure support and many more, and also different human interaction patterns. This leads to the need of studying the association and fusion of cross-space data, merging their complimentary features and fully combining their merits [2].

2.5 Architecture of MCS

The architecture of MCS is composed of five layers: crowd sensing, data transmission, data collection, crowd data processing, and applications as shown in figure 2.1 [2].

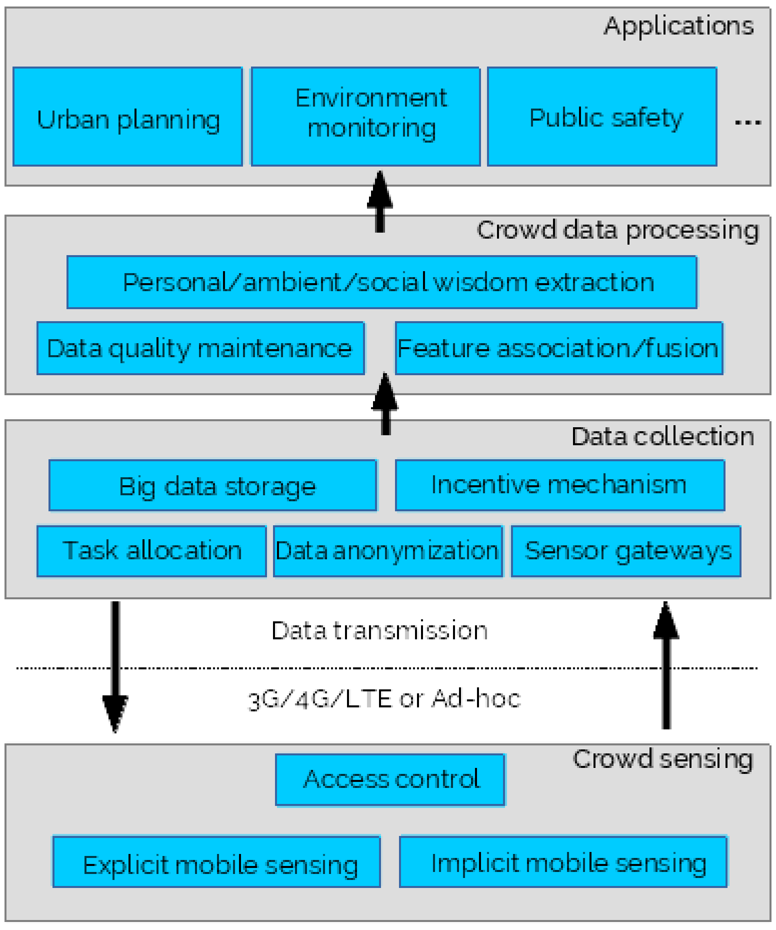


FIGURE 2.1: MCS architecture

2.5.1 Crowd sensing

This is the physical layer which is found in the mobile devices used by the participants. It generate two types of data: explicit and implicit mobile sensed data. Access control is a function that gives the users the power to decide with whom they share their sensed data [2].

2.5.2 Data transmission

The evolution in mobile networking and communication techniques with or without infrastructure (3G,4G/LTE, WiFi ...etc) can be used by MCS systems for uploading data while making that process transparent to the users and highly tolerant for transmission interruptions [2].

2.5.3 Data collection infrastructure

In this layer, MCS systems gather the data sent by the participants' devices while preserving their privacy [2].

2.5.4 Crowd data processing

This layer applies various machine learning and data mining techniques to transform the collected raw data into the expected intelligence [2].

2.5.5 Applications

This layer represents the different potential applications and services powered by the MCS systems. Those applications would typically include the functions of graphical user interfaces and data visualization for the intended purposes of that MCS system such as studying a natural phenomena or improving the different day to day tasks of their users [2].

2.6 MCS Challenges

2.6.1 Sensing with Human Participation

Compared with traditional static, centrally controlled sensor networks, the involvement of mobile human volunteers in gathering, analyzing, and sharing local knowledge in an interactive sensing infrastructure leads to a number of new challenges [5].

- Task allocation and data sampling: In MCS, a swarm of highly volatile mobile sensors can potentially provide coverage where no static sensing infrastructure is available. Nevertheless, because of a potentially large population of mobile nodes, a sensing task must identify which node(s) may accept a task. A set of criteria should be considered in filtering irrelevant nodes, such as the specification of a required region (e.g., a particular street) and time window, acceptance conditions (e.g., for a traffic-condition capture task, only the phones out of users pockets and with good illumination conditions can satisfy requirements), device capabilities, and termination conditions (e.g., sampling period) [5].
- Human grouping: Interactions among volunteers are necessary during the sensing process but absent in most existing crowd sensing systems. The interaction among users also enhances the data quality in MCS. Therefore, grouping users and facilitating the interaction among them should be a challenge of MCS. Key techniques to address this include community creation approaches, dynamic group formation metrics, social networking methods, and so on [5].
- Coverage, reliability, and scalability: MCS is akin to the event coverage in conventional sensor networks.To design and deploy successful MCS systems, the relation between the event coverage and the number of participants should be studied. Possible issues involve spatial-temporal coverage, the impact of user skills/preferences to task coverage, and so on [5].

2.6.2 Incentive Mechanisms

Incentive is another challenge to the human involvement in MCS. While sensing devices are usually possessed and administrated by a single authority in traditional sensor networks, they belong to different individuals with diverse interests in MCS. In order to sense, process, and collect the desired data, participants have to make either implicit efforts (e.g., energy and monetary costs) or explicit efforts (e.g., give some input or assessments). Without strong incentives, individuals may not be willing to participate in the sensing task with the cost because of their own limited resources. Therefore, an efficient incentive mechanism is essential for the success of MCS applications. Similar to other crowdsourcing systems, broadly two types of incentives can be used in MCS applications: intrinsic incentives or financial incentives [5].

Intrinsic incentives include interest (where the volunteers are willing to help when they think the task is interesting and important), enjoyment (making the MCS task an entertainment activity, such as a game), and social/ethical (where the participators are attracted by the chance of receiving public recognition). While these intrinsic incentives work well in certain types of MCS applications (e.g., environment monitoring, location services), financial incentives are probably the easiest way to motivate user participation in almost all types of MCS applications. In MCS systems with financial incentives, the financial rewards to their participates for performing sensing or communication tasks could be money, virtual cash, or credits (which are redeemable for later services or online goods). However, once money is involved, the participants are more likely to deceive the system to get more financial gains. Therefore, how to provide valuable incentive mechanisms that enable honest contributions in MCS becomes a critical challenge [5].

2.6.3 Data Delivery in Transient Networks

How to ship the sensed data from distributed participants to the backend server is another challenge due to a variety of MCS characteristics, such as the low bandwidth of wireless communication, frequent network partitioning caused by human mobility, and large number of energy-constrained devices. While this is a well-known research challenge in both wireless sensor networks and general mobile systems, MCS adds new requirements on scalability. In particular, the following four research issues need to be addressed for MCS [5]:

– Robust data delivery among highly mobile devices: Though many data forwarding and routing protocols have been developed for mobile ad hoc networks and mobile opportunistic networks in the past decade, it is still a hard problem to achieve robust and reliable data delivery with a large amount of mobile devices without infrastructure [5].

- Tradeoffs between communication and processing via localized analytics. In many MCS applications, certain localized analytics are performed on raw sensing data at the individual device level. By doing so, MCS systems can consume less net- work bandwidth than directly transmitting raw sensor readings. The challenge is how to make a balance between the energy cost on local computing and the data transmission cost [5].
- Distributed caching and replication schemes. In order to make the storing and retrieving of MCS data effective, efficient, and robust in a large-scale mobile system, it is essential to develop new distributed data caching and replication schemes. Such schemes should be integrated with the data processing and management issues [5].
- Hybrid networking protocols. The coexistence of heterogeneous network connections is a basic feature of MCS. Combining the complementary merits of heterogeneous networks provides opportunities to develop improved networking protocols for MCS [5].

2.6.4 Data Redundancy, Quality, and Inconsistency

In MCS, there can be multiple participants involved in the same sensing activity, for example, sensing the traffic information in an intersection. One of the issues caused by multi-participant sensing is data redundancy. In other words, to accomplish a particular sensing task, it is important to smartly select data from multiple available contributors. Note that for the same task, nearby mobile sensing devices may have various sensing qualities, which could be caused by the mobility of devices and the differences in energy levels or communication channels. Certain quality estimation and prediction methods are thus necessary to evaluate the quality of sensing data, and statistical processing can be used to identify outliers.

Another potential issue caused by ”redundant” sensing is data inconsistency. For example, due to the differences in sensing and computing capabilities, a set of collocated smartphones running the same algorithm (e.g., sound-based social context recognition) and sensing the same event can obtain different inference results (e.g., a party or a meeting), thereby causing the problem of inconsistency. Furthermore, a more complex issue is the inconsistence of semantics derived from multi-model sensory data (e.g., audio clips, images, videos, texts). For example, while both the audio and video data can be used to predict the social context of a user, the inference results can be different. Further studies should be done to address the inconsistency caused by the multi-model data contributed by crowds [5].

2.6.5 Cross-Space, Heterogeneous Crowdsourced Data Mining

The strength of MCS relies on the usage of crowdsourced data from both physical and virtual societies. The development of the Internet of Things and mobile Internet bridges the gap between the physical space and the cyber space. For the same sensing object (e.g., a social gathering in a street corner), it will interact with both spaces and leave fragmented data in each space, making the information obtained from different communities (online or offline) differ. For instance, we can learn social relationships from online social networks and infer group activities and interaction behaviors using mobile phone sensing in the real world. Obviously, the complementary nature of heterogeneous communities will bring new opportunities to develop new human-centric services. Therefore, we should integrate the information from heterogeneous data sources to attain a comprehensive picture about the sensing object. With the increase in the large-scale, interlinked data collected from heterogeneous communities, advanced techniques on complex network modeling, data mining, data association and aggregation, and semantic fusion will become more and more important.

2.6.6 Trust, Security, and Privacy

The involvement of human participation in crowd sensing also brings forth certain trust issues. MCS participants are likely to provide incorrect or even fake data to the system. For example, incorrect recordings might be collected when mobile devices are improperly placed by the participants; for example, one may put his or her phone in the pocket when assigned with a noise sensing task. Meanwhile, for their own benefit, malicious users may intentionally pollute the sensing data. The lack of control mechanisms to guarantee source validity and data accuracy can lead to information credibility issues. Therefore, a trust preservation and abnormal detection technologies must be developed to ensure the quality of the obtained data. To motivate user participation, an MCS system must be capable of providing effective privacy protection mechanisms so that participants can conveniently and safely share high-quality data using their devices. The following two ways are promising:

– Local versus remote data processing: One possible way to preserve privacy is to upload the processed data rather than the original raw data. For example, in case an MCS application needs to collect the environmental noise using microphones, by leveraging some phone-based noise identification methodologies,we might only need to upload the obtained results (i.e., noise level) rather than the raw audio files to the cloud. While such an approach avoids the disclosure of peoples conversation, phone-based algorithms would incur severe energy consumption. Meanwhile, it is more efficient to process audio files in the cloud than locally on the phone, but this requires implementing privacy preserving data mining techniques for remote processing.

– Privacy-aware sensing model and architecture. To effectively preserve the privacy of a huge amount of MCS participants, not only methodology efforts but also systematic studies are needed. In other words, a privacy-aware architecture should be provided to support the development of MCS applications.

2.7 MCS applications for driving experience improvement: Related Work

2.7.1 Pothole

Pothole[3] is an android application that detects potholes when the users drive over them and plots their location in a website[4]. The application detects potholes using data from the smart phone's accelerometer, by constantly checking the collected data for irregularities then sending the location of the potholes to the website.

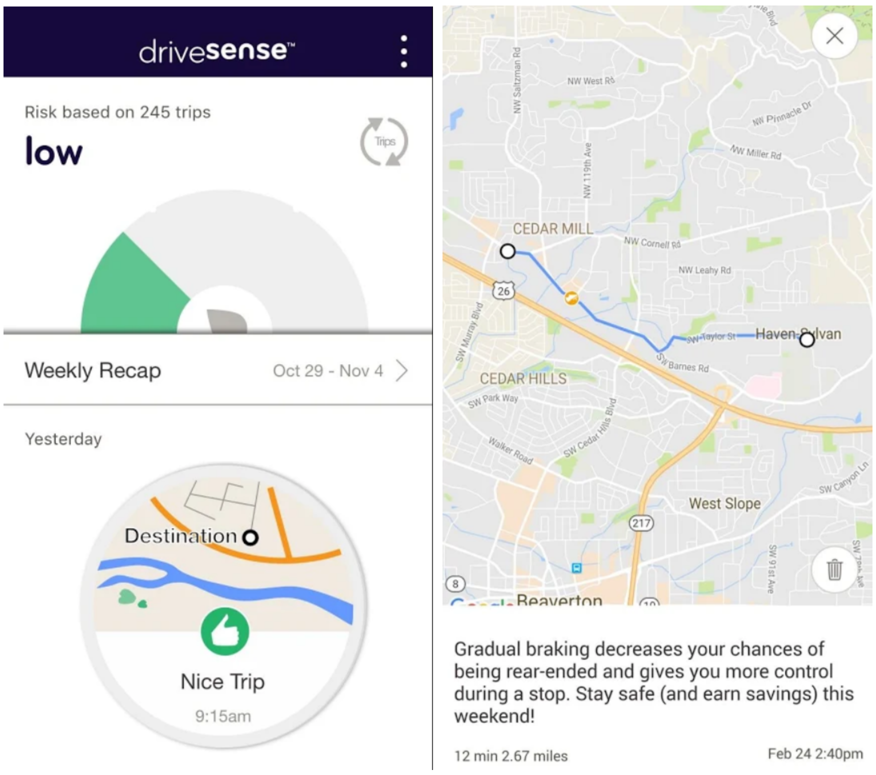
The application has been abandoned and never received updates since 2015. It was available on the play store until it was removed from it when I last checked it in March 2018. The website used to plot the potholes location suffered the same fate and it no longer available.

[3]www.play.google.com/store/apps/details?id=slashn.pothole&hl=en

[4]www.potholeweb.com

2.7.2 DriveSense mobile

DriveSense[5] is an android application published by an American insurance company named Esurance, it is used to monitor driving trips where the application uses the smart phone's sensors to determine if the user has driven safely during his trip or not. The collected data which consists of speed, traveled distance, accelerometer data and location. This data is used by the application to provide the users with information about the risks they encountered during their travels such as speeding and sudden braking. Trip results are used to give Esurance's clients a discount for driving safely.



Gradual braking decreases your chances of being rear-ended and gives you more control during a stop. Stay safe (and earn savings) this weekend!

FIGURE 2.2: DriveSense mobile.

The application is only available for Esurance clients in the United States of America, Canada and Mexico as a way of promoting the company's insurance services.

2.7.3 CitySensing

CitySensing is a framework developed to support the development of mobile crowd sensing applications for various smart city scenarios. The mobile crowd sensing solutions provided

[5]https://play.google.com/store/apps/details?id=com.esurance.drivesense&hl=en

by this framework could detect and report traffic events and conditions, the state of road infrastructure, driver behavior and activities, as well as accidental events [1].

The framework is not available to the public, it can be assumed that it is only used for research purposes of the laboratory that created it.

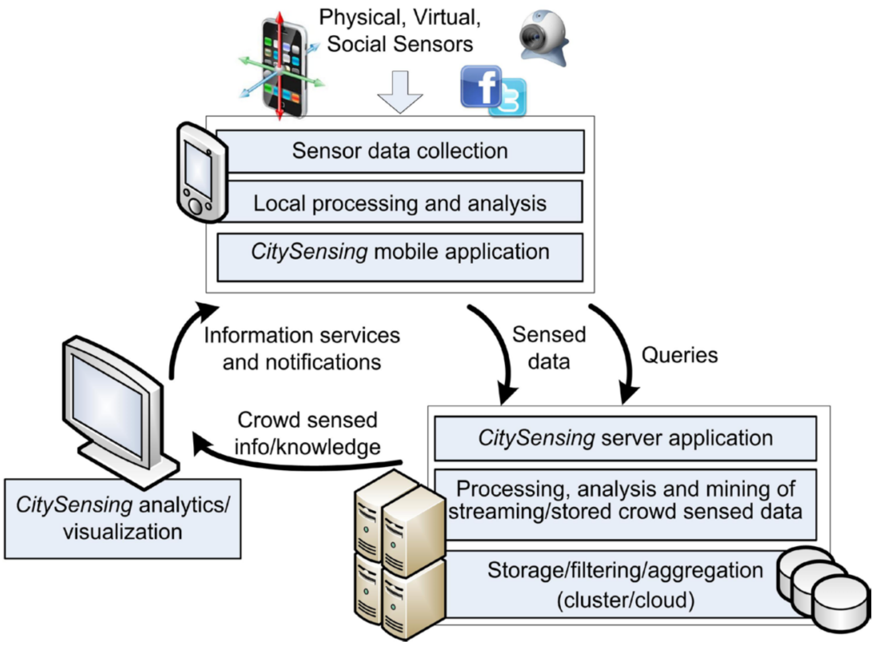

Physical, Virtual, Social Sensors
Sensor data collection
Local processing and analysis
CitySensing mobile application
Information services and notifications
Sensed data
Queries
Crowd sensed info/knowledge
CitySensing server application
Processing, analysis and mining of streaming/stored crowd sensed data
Storage/filtering/aggregation (cluster/cloud)
CitySensing analytics/ visualization

FIGURE 2.3: General architecture of CitySensing framework[1].

2.8 Conclusion

We can deduce from this chapter that MCS is a crucial element in elevating IoT services and providing many benefits such as improving the daily routines of people.

Chapter 3

Android Mobile Operating System

3.1 Introduction

We will present in this chapter one of the widely used mobile operating systems: Android. We'll discuss the features that made it reach a high degree of success and it's architecture.

3.2 What is Android

Android is a software environment built for mobile devices. Android includes a Linux kernel-based OS, a rich UI, end-user applications, code libraries, application frameworks, multimedia support, telephone functionality, and much more. Whereas components of the underlying OS are written in C or C++, user applications are built for Android in Java [6].

Android is developed and maintained by the organization called Open Headset Alliance (OHA). OHA was established in 2007 with Google being its foremost member. OHA includes a lot of prominent hardware and software companies [7].

Originally, Android was created by a company called Android Inc. Google acquired this company in 2005. After then, Google made it open source and Android gained a big momentum [7]. Android has the market share of around 85.9% as of Q1 2018 [1].

[1]https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

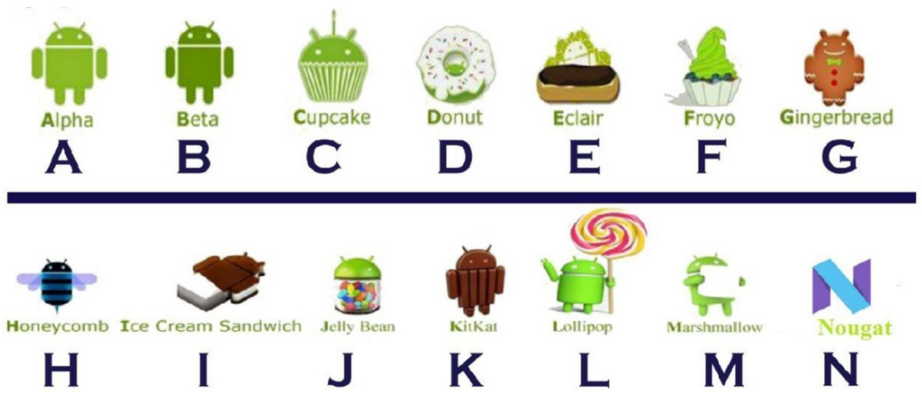


FIGURE 3.1: Android versions till now

3.3 Android features

3.3.1 Open source

After Google made Android open source and released its source code to the public, it became its strongest feature. Open source is a double-edged sword. On one hand, the power of many talented people and companies working around the globe and around the clock to deliver desirable features is a force to be reckoned with, particularly in comparison with a traditional, commercial approach to software development. On the other hand, how far will the competing manufacturers extend and potentially split Android [6].

3.3.2 User interface

Android provides a user interface based on touch inputs such as swiping, tapping, pinching and reverse pinching to manipulate what's on screen. Phone sensors can be used to provide more functionality such as changing the screen orientation [6].

3.3.3 Rich SDK

The android SDK provides a wide array of tools to help develop applications such as a debugger, emulator, sample codes, documentation and a rich software library. Android applications can be programmed in Java or Kotlin or the combination of both. Developers can even use C or C++ for better performance [6].

3.3.4 WebKit

The WebKit browser engine is an open source project that powers the browser found in Macs (Safari) and is the engine behind Mobile Safari, which is the browser on the iPhone. It is not a stretch to say that the browser experience is one of a few features that made the iPhone popular out of the gate, so its inclusion in Android is a strong plus for Android's architecture [6].

3.3.5 Licensing

Android is released under two different open source licenses. The Linux kernel is released under the GNU General Public License (GPL) as is required for anyone licensing the open source OS kernel. The Android platform, excluding the kernel, is licensed under the Apache Software License (ASL). Although both licensing models are open sourceoriented, the major difference is that the Apache license is considered friendlier toward commercial use [6].

3.4 Android platform architecture

The Android stack (figure 3.2) includes an impressive array of features for mobile applications. In fact, looking at the architecture alone, without the context of Android being a platform designed for mobile environments, it would be easy to confuse Android with a general computing environment. All the major components of a computing platform are there [6].

3.4.1 Linux kernel

This layer provides a foundational hardware abstraction layer, as well as core services such as process, memory, and filesystem management. The kernel is where hardware specific drivers are implemented capabilities such as WiFi and Bluetooth are here. The Android stack is designed to be flexible, with many optional components that largely rely on the availability of specific hardware on a given device. These components include features such as touch screens, cameras, GPS receivers, and accelerometers [6].

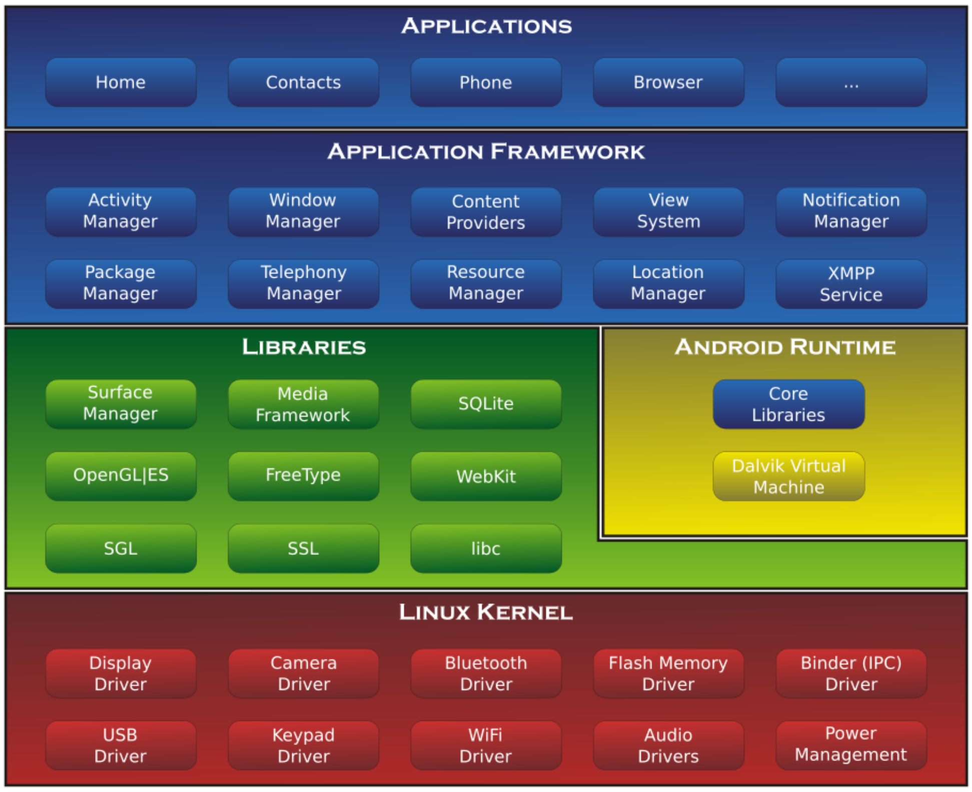


FIGURE 3.2: Android platform architecture

3.4.2 Code libraries

This layers provides a wide array of libraries to help develop android applications such as: Browser technology from WebKit, the same open source engine powering Mac's Safari and the iPhone's Mobile Safari browser. WebKit has become the de facto standard for most mobile platforms, database support via SQLite, advanced graphics support, Audio and video media support from PacketVideo's OpenCORE and Secure Sockets Layer (SSL) capabilities from the Apache project [6].

3.4.3 Android runtime

Provides Core Java packages for a nearly full-featured Java programming environment. And The Dalvik VM, which employs services of the Linux-based kernel to provide an environment to host Android applications [6].

3.4.4 Application framework

Represents an array of managers that provide services for: Activities and views, windows, location based services, telephony, resources, etc. [6].

3.4.5 Applications

The last layer where the android core and third party applications reside. They run in the Dalvik VM, atop the previous layers [6].

3.5 Conclusion

From what we have seen in this chapter, Android will be our best choice of a platform to build a MCS based application that improves the driving experience of people.

Chapter 4

Proposed solution

4.1 Introduction

This chapter presents our solution that is based on MCS architecture and built upon the android platform for improving the driving experience.

4.2 Objective

Our objective is to develop an android application that can sense bumps and send the sensed data to a server. The application should also be able to alert the users of any incoming known bumps while they are traveling.

4.3 General architecture

In order to utilize crowdsensing to improve the driving experience by detecting road bumps, we propose the following application architecture (figure 4.4):

– **Mobile application:** this is the first layer of our solution which represents the android application used by the participants. This layer offers two modes of travel: offline and online where the offline mode only gives a heads up to the user of incoming bumps from the local bumps database, and the online mode adds on top of that the ability to detect new bumps and sends them to the server for processing (figure 4.4).

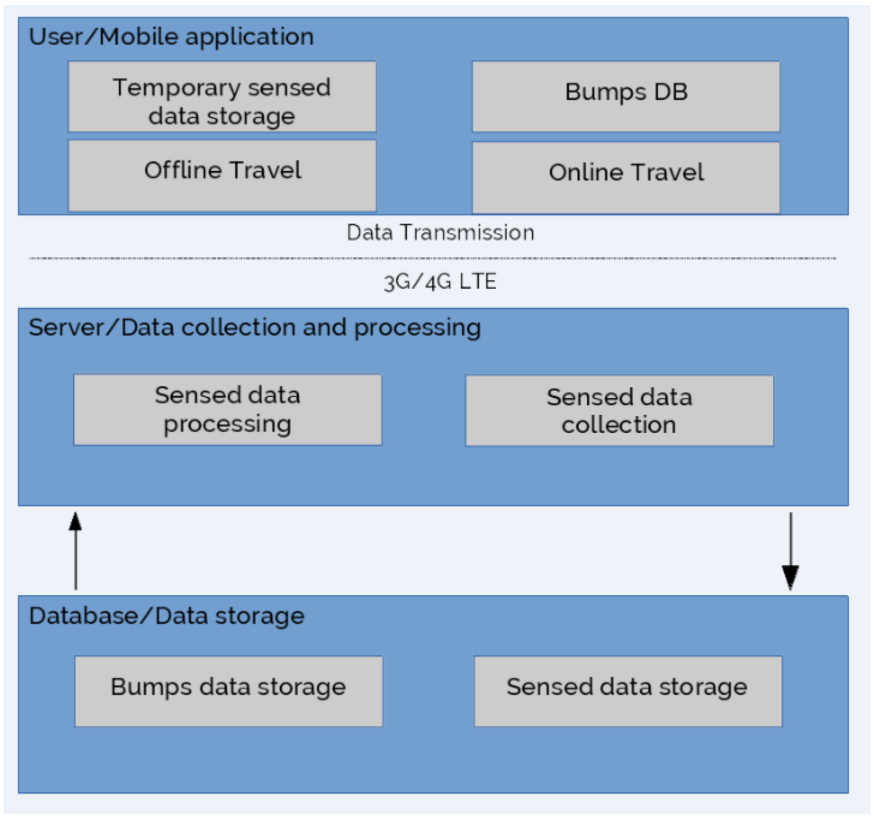


FIGURE 4.1: Proposed application architecture

- **Data collection and processing:** the second layer of our solution which resides on a remote server that collects all data sent from the participants and store it in a database. This data will be processed and filtered to determine the data presents real bumps and discard those that don't 4.4.

- **Data storage:** the last layer in our solution, this is where all data received from the participants would be stored and all known bumps with their locations after the data has been processed (figure 4.4).

4.4 Application design

4.4.1 Android application

As shown in figure 4.4, the android application will have two modules for travel: offline and online:

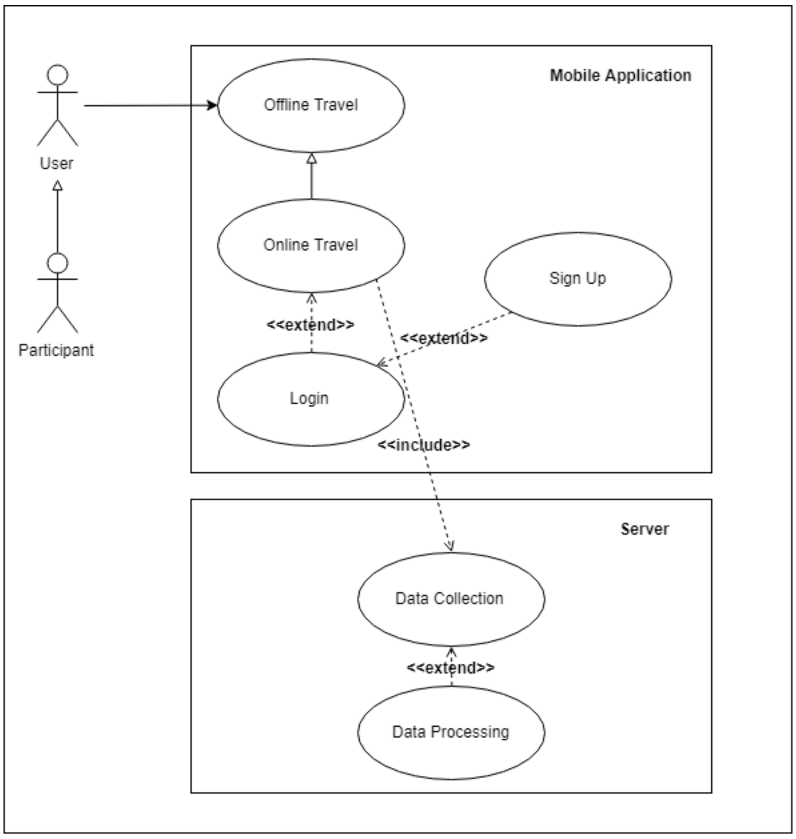


FIGURE 4.2: Solution use case diagram

4.4.1.1 Offline module

This module is for when users don't have mobile data or just don't want to participate in the sensing process but want to use application's feature of bumps alerts during travel. This module will have a thread working in the background, scanning the known bumps database for bumps which are very close to the user. Once a close bump is found, the module will launch a sound alert for the user.

4.4.1.2 Online module

The online module extends the offline module's features by giving the users the ability to participate in the sensing process. It uses the participants mobile phone's accelerometer to determine when the participants have driven over a bump (figure 4.6). The collected accelerometer data of the Y axis acceleration (figure 4.5 will be examined for any changes, if a significant change has been detected like in figure 4.6 the application will send the location of that change to the server for analysis, and only if there is no bump with same location on the local bumps database. If the accelerometer registers no significant changes

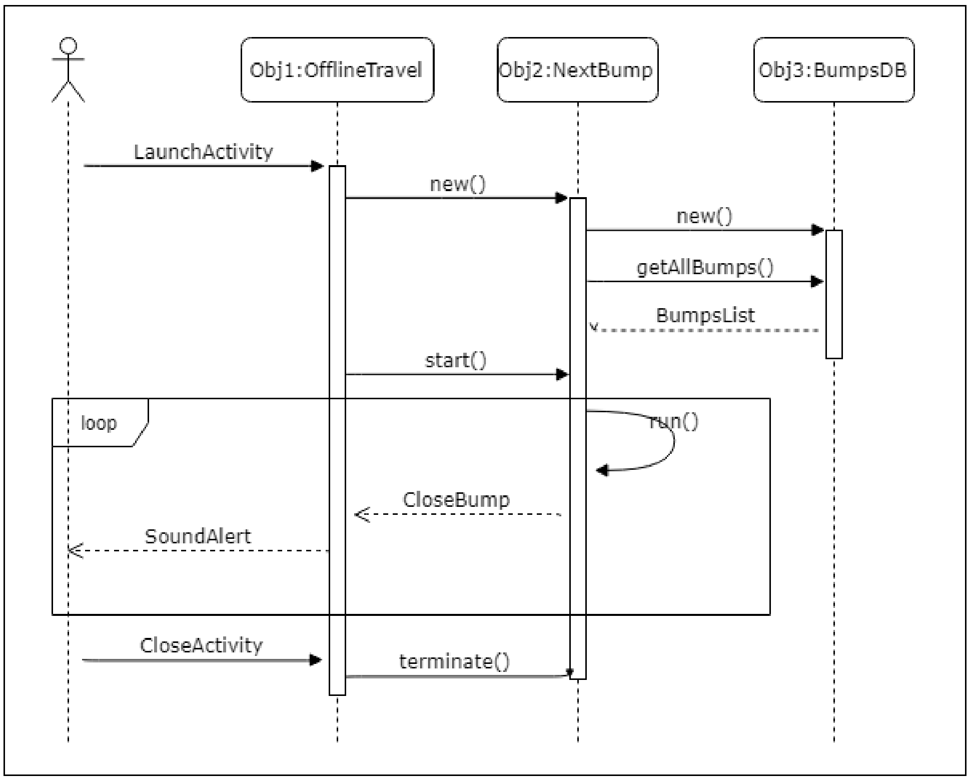


FIGURE 4.3: Offline module sequence diagram

when the user is at the location of a bump from the bumps database, then that bump doesn't exist and the application sends a bump removal request to the server.

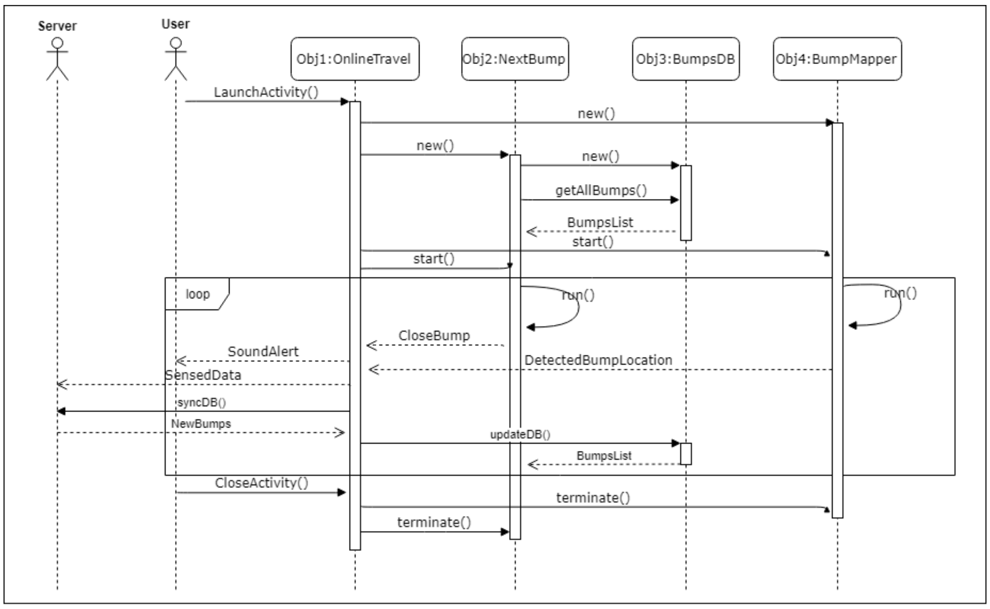


FIGURE 4.4: Online module sequence diagram

The online module requires the user to create an account and login. This helps with the data processing and updating the bumps database as mentioned in section 4.4.3.

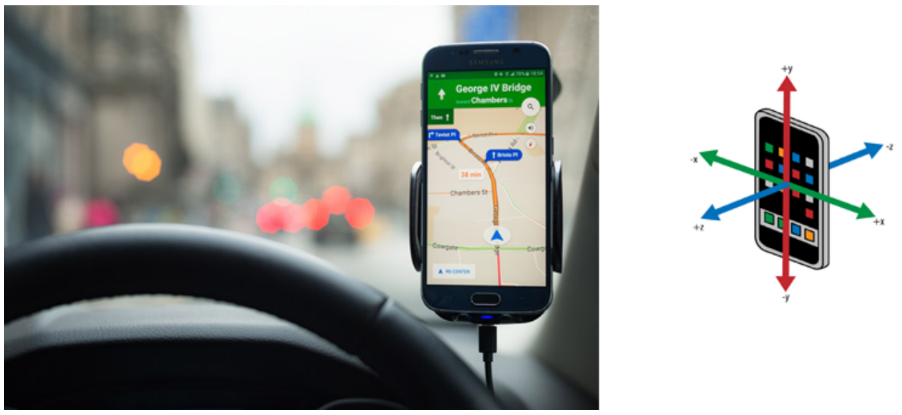


FIGURE 4.5: Smart phone accelerometer axis and the required position when using the application

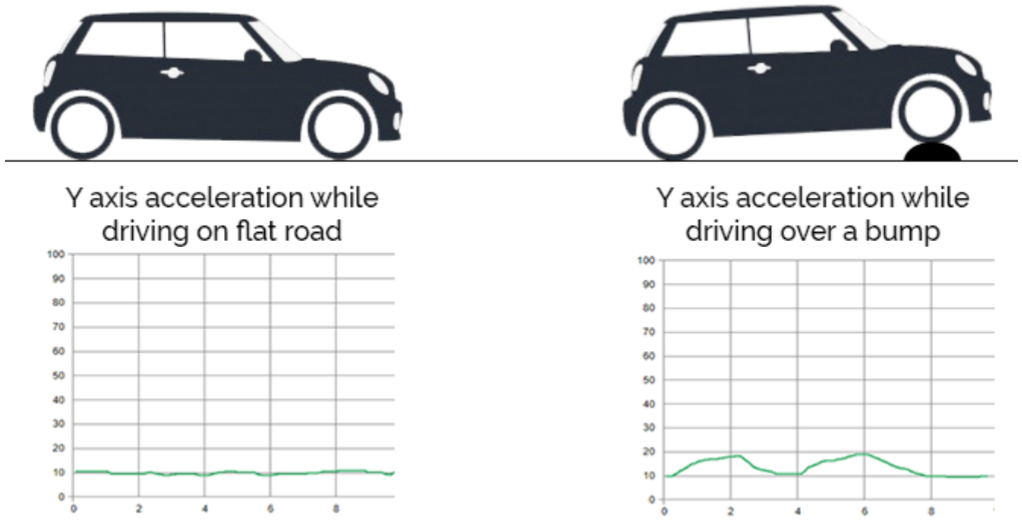


FIGURE 4.6: Bump detection using accelerometer

4.4.2 Data collection and storage

The server has the capability of receiving data from the participants. This data is stored in a database for later processing. The data can be either new bumps locations or bump removal requests. The result from processing that data will be used to update the bumps database by either adding new bumps or removing existing ones that are no longer on the roads (figure 4.7).

4.4.3 Data processing

To determine which data represents an actual bump, we would use a voting system on the server. If a certain number of users have detected bumps at the same location, then the

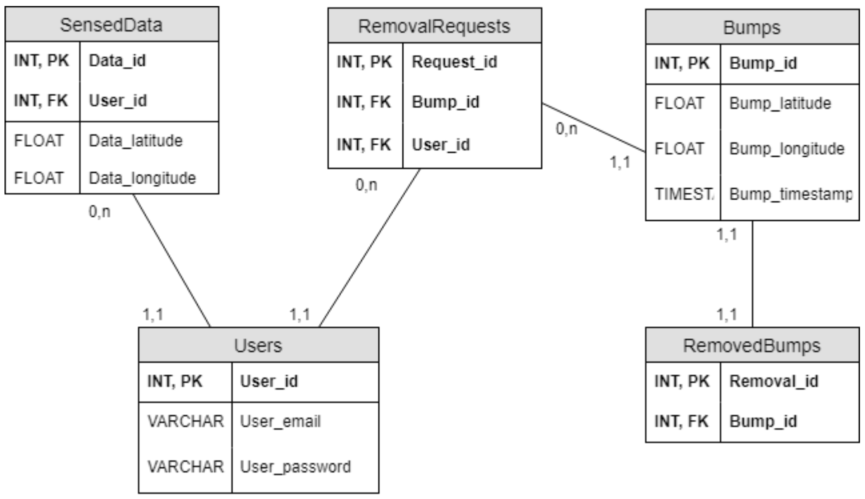


FIGURE 4.7: Database design

server will mark that location as an actual bump. The same applies to bumps removal, if a certain number of participants detect no accelerometer change while passing by a bump location from the database then that bump would be removed.

The new additions or removal would be sent back to the participants on a regular basis to ensure that they are having the latest bumps database and to avoid any data redundancy or waste of their mobile internet.

4.5 Conclusion

We have presented in this chapter our idea of improving the driving experience using MCS architecture, which will be implemented and tested in the next chapter.

Chapter 5

Implementation and results

5.1 Introduction

In this chapter we will presenting the implementation of our solution into a mobile application and the tools we used in that process. We will also provide the result of a brief test of the application and a the discussion of it.

5.2 Used tools

5.2.1 Java

THE Java programming language is a general-purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough that many programmers can achieve fluency in the language. The Java programming language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included. It is intended to be a production language, not a research language [8].

5.2.2 PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was[1].

5.2.3 Android Studio

Android Studio is the official IDE for android. It provides the fastest tools for building apps on every type of Android device. World-class code editing, debugging, performance tooling, a flexible build system, and an instant build/deploy system all allows to focus on building unique and high quality apps[2].

5.2.4 Android

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touchscreen mobile devices such as smartphones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

5.2.5 SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file [3].

[1]http://php.net/manual/en/intro-whatis.php
[2]https://developer.android.com/studio/index.html
[3]https://www.sqlite.org/about.html

5.2.6 MySQL

MySQL is the most popular Open Source SQL relational database management system that is developed, distributed, and supported by Oracle Corporation. The MySQL Database Server is very fast, reliable, scalable, and easy to use that works in client/server or embedded systems[4].

5.2.7 Heliohost hosting services

Heliohost is a nonprofit organization powered by a community of hobbyists that offer full-featured web hosting services for free. HelioHost has been offering web hosting services to the online community for more than a decade. The companys headquarters is in a professional data center in Silicon Valley. As a result, its clients can access features such as unlimited domains and almost unlimited bandwidth [5].

5.2.8 Hardware (Smart phone)

– **Name:** Huawei Y6II.

– **SIM:** Dual SIM.

– **Chipset:** Huawei HiSilicon Kirin 620.

– **CPU:** Octacore ARM Cortex-A53 1.2 GHz

– **RAM:** 2 Gb.

– **Internal memory:** 16 Gb.

– **Communication:** GSM, HSPA, LTE, WiFi.

– **OS:** Android 6.0 Marshmallow.

– **Sensors:** GPS, accelerometer, light, proximity.

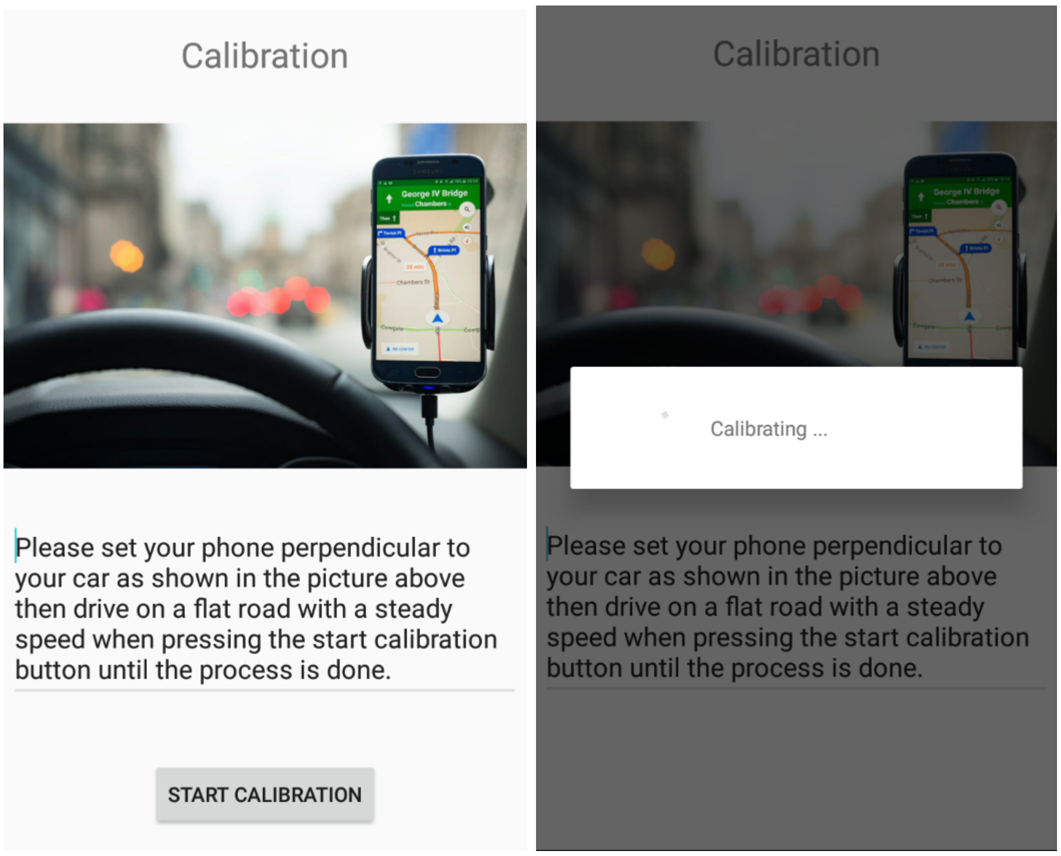
[4]https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html
[5]https://www.heliohost.org/

5.3 Solution Implementation

5.3.1 Mobile Application

5.3.1.1 First use

When the participants install the application for the first time, after the welcome activity they will be sent to the calibration phase which will show them the required position of the phone. Once the participant placed their phone on the car correctly, they can start the calibration of the app by driving their car on a flat road for 5 seconds (5000 milliseconds as shown in source code **??**). This process will calculate the moving average of the Y axis data of the accelerometer to record the threshold which would be used later in the sensing activity. Once the threshold has been saved, the app starts downloading the latest bumps database from the server.



Please set your phone perpendicular to your car as shown in the picture above then drive on a flat road with a steady speed when pressing the start calibration button until the process is done.

Please set your phone perpendicular to your car as shown in the picture above then drive on a flat road with a steady speed when pressing the start calibration button until the process is done.

FIGURE 5.1: Calibration activity

```java
long startTime = System.currentTimeMillis();
while (System.currentTimeMillis() < startTime + 5000) {
        tsList.add(acc_y);
}

SharedPreferences sharedPref =
 ↪  PreferenceManager.getDefaultSharedPreferences(contextActivity);
SharedPreferences.Editor editor = sharedPref.edit();
for(Float acc:tsList) result+= acc;
        result = result/(tsList.size());
editor.putFloat("threshold", result);
editor.apply();
```

Source Code 5.1: Calculating threshold

5.3.1.2 Login and sign up

If the participants want to help in the sensing and data collection of bumps while traveling, they can create an account for the first time by signing up as show in figure. Once they made an account, they can login (figure) and proceed to sensing and collecting bumps data if their local bumps database is up to date to avoid redundancy. The login and signup processes are done via HTTP POST requests to the server.

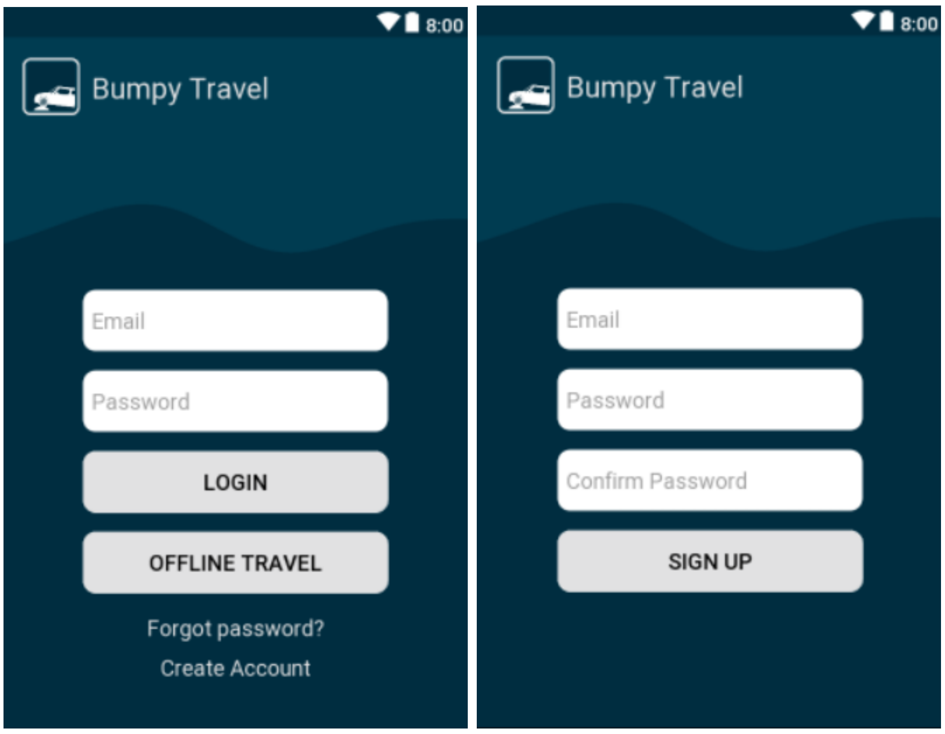


Figure 5.2: Login and signup activities

5.3.1.3 Online travel

The online travel activity is where the participants can collect new bumps data when they have connectivity and get alerts about existing bumps once they are close to them. The activity launches two threads in the background to avoid blocking the user interface. One thread is responsible for the data collection by comparing the accelerometer data that is collected every 200 milliseconds to the threshold value calculated in the calibration process (section 5.3.1.1). Once the threshold has been surpassed, the application sends an HTTP POST request to the server with the location of the detected bump and the participant's ID.

```
String url = "http://crowdsensing.heliohost.org/collection3.php";
StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
                                        //Success
            Toast.makeText(OnlineTravel.this, response, Toast.LENGTH_SHORT).show();
        }
    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
                                        //Error
            Toast.makeText(OnlineTravel.this, "Error while sending data!",
             ↪  Toast.LENGTH_SHORT).show();
        }
    }) {
    @Override
    protected Map<String, String> getParams()
    {
        Map<String, String>  params = new HashMap<>();
        params.put("lat", ""+lastLocation.getLatitude());
        params.put("long", ""+lastLocation.getLongitude());
        params.put("user", ""+id);
        return params;
    }
};
VolleySingleton.getInstance(OnlineTravel.this).addToRequestQueue(stringRequest,
 ↪  "sensedData");
```

Source Code 5.2: Sending collected data using android's volley

The second thread is for incoming bumps alerts, it scans the application's bumps database which is implemented using android's SQLite database management system, and when it detects that the participant at 50 meter or less from any bump, it launches a sound alert to inform the participant about it. If the participant is close to a bump location and the application doesn't register a significant change in the Y axis accelerometer data, it sends a bump removal request to the server.

```
if(!nextBumps.isEmpty()){
    for(Location loc:nextBumps){
        if(lastLocation.distanceTo(loc) <=3){
            long startTime = System.currentTimeMillis();
            accData.clear();
            while (System.currentTimeMillis() < startTime + 3000) {
                accData.add(acc_y);
            }
            float data = 0;
            for(Float acc:accData) data+= acc;
                data = data/accData.size();
            if(data<threshold+0.05){
                String url = "http://crowdsensing.heliohost.org/remove.php";
                StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                    new Response.Listener<String>() {
                        @Override
                        public void onResponse(String response) {
                                        //Success
                        }
                    },
                    new Response.ErrorListener() {
                        @Override
                        public void onErrorResponse(VolleyError error) {
                                        //Error
                        }
                    }) {
                    @Override
                    protected Map<String, String> getParams()
                    {
                        Map<String, String>  params = new HashMap<>();
                        String csv = "";
                        for(Float acc:accData){
                            if(csv.equals("")) csv = Float.toString(acc);
                            else csv = csv+","+acc;
                        }
                        params.put("lat", ""+lastLocation.getLatitude());
                        params.put("long", ""+lastLocation.getLongitude());
                        params.put("user", ""+userID);
                        return params;
                    }
                };

                ↪  VolleySingleton.getInstance(contextActivity).addToRequestQueue(stringRequest,
                ↪  "removeBump");
            }
        }
    }
}
```

Source Code 5.3: Detecting bumps that are no longer present and sending removal requests

Local bumps database is always updated, as the application synchronizes the database every 3 minutes to get new bumps and remove those that are no longer existing, by requesting updates from the server using HTTP POST requests and receiving the updates as a JSON Object.

```
String url = "http://crowdsensing.heliohost.org/syncdb.php";

StringRequest jsonObjectRequest = new StringRequest
(Request.Method.POST, url, new Response.Listener<String>() {

    @Override
    public void onResponse(String response) {
        try {
            JSONObject jObj = new JSONObject(response);
            boolean error = jObj.getBoolean("error");
            if (!error) {
                String stamp = ""+0;
                for (int i = 0; i < jObj.length()-2; i++) {
                    int id = jObj.getJSONObject("" + i).getInt("id");
                    double lat = jObj.getJSONObject("" + i).getDouble("latitude");
                    double lon = jObj.getJSONObject("" + i).getDouble("longtitude");
                    stamp = jObj.getJSONObject("" + i).getString("created");
                    db.addBump(new Bump(id, lat, lon, stamp));
                }
            }

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
                        // TODO: Handle error
    }
}){
    @Override
    protected Map<String, String> getParams() {
                // Posting params to register url
        Map<String, String> params = new HashMap<String, String>();

        String last = db.getBumpStamp();
        params.put("lastBump", ""+last);
        return params;
    }
};

VolleySingleton.getInstance(contextActivity).addToRequestQueue(jsonObjectRequest,
 ↪   "updateDB");
```

Source Code 5.4: Synchronizing local database using android's volley

5.3.1.4 Offline travel

The offline travel activity is for participants that don't have internet connection or just don't want to participate in the sensing process. It launches only one thread responsible for alerting the user about incoming bumps. It is the same second thread from section 5.3.1.3.

To get accurate user position data, the application requires the "ACCESS FINE LOCATION" permission and a location update interval of 5 seconds. This would enable the application to get frequent location updates with an accuracy of one meter or less.

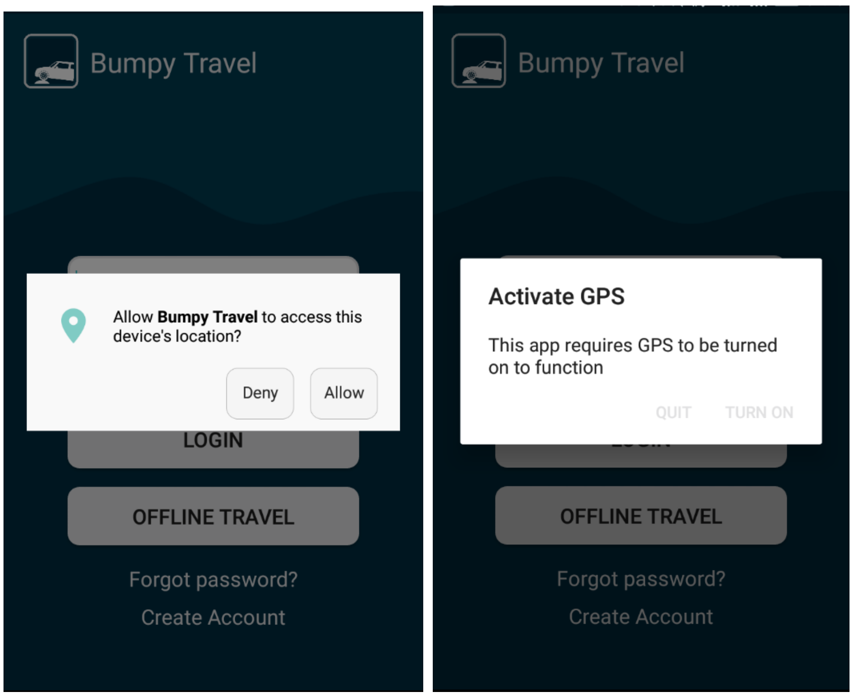


Figure 5.3: Requesting location permission and to activate GPS

5.3.2 Server

The server layer of our solution handles participants login and signup, data collection and data processing. It is hosted on HelioHost hosting services as mentioned in section 5.2.7. We connect to the MySQL database using Php Data Object as show in the code below.

```php
<?php
$response = array("error" => FALSE);
try {
        $host = 'localhost';
        $user ='houssemn_admin';
        $pass = 'password';
        $db = 'houssemn_crowdsensing';
        $connect = new PDO('mysql:host=' . $host . ';dbname=' . $db, $user, $pass);
        $connect->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
        $response["error"] = TRUE;
        $response["error_msg"] = "Could not connect to server!";
        echo json_encode($response);
}
?>
```

Source Code 5.5: Connecting to our database

5.3.2.1 Participants login and signup

The server receives HTTP POST requests from connected participants with their emails and passwords using a PHP script. During signup, the script checks if that email is already registered, if not if proceeds to finalizing the signup process by adding the user information to our users table with the password secured using PHP's latest hashing algorithm bcrypt from the function "password_hash()". For login, the script checks if the user exists then compares the provided password with the hashed password on our database using PHP's function "password_verify()", if the provided user credentials are correct, it returns the user's information to the application using as a JSON Array.

```php
<?php

if(isset($_POST['email']) && isset($_POST['password'])){
        $email = $_POST['email'];
        $password = $_POST['password'];
        $response = array("error" => FALSE);


            // Login process
        if(checkExistingUser($connect,$email)){
                        // User exists, authenticate
                $query = $connect->prepare('SELECT * FROM User where User_email = :email');
                $query->bindParam(":email", $email);
                $query->execute();
                $row = $query->fetch();
                if(password_verify($password, $row['User_password'])){
                                // User credentials are all correct
                        $arr = explode("@", $row['User_email'], 2);
                        $name = $arr[0];
                        $response["error"] = FALSE;
                        $response["user_id"] = $row['User_id'];
                        $response["user_name"] = $name;
                        $response["user_email"] = $row['User_email'];
                        echo json_encode($response);
                }
                else{
                                // Wrong password
                        $response["error"] = TRUE;
                        $response["error_msg"] = "Login credentials are wrong. Please try again!";
                        echo json_encode($response);
                }
        }
        else{
                        // User doesn't exist
                $response["error"] = TRUE;
                $response["error_msg"] = "Login credentials are wrong. Please try again!";
                echo json_encode($response);
        }

}

?>
```

Source Code 5.6: Handling user login

5.3.2.2 Data collection

The data collection consists of a PHP script that receives the sensed data or bump removal requests from the participants (user ID, latitude and longitude) using HTTP POST requests, then saves it to the collected data table or the removal requests in our database as shown in section 4.4.2 using the MySQL database management system.

```php
<?php

try{
	$query = $connect->prepare("INSERT INTO CollectedData(latitude,longitude,user)
	↪  VALUES(:lat,:long,:user)");
	$query->bindParam(":lat", $lat);
	$query->bindParam(":long", $lon);
	$query->bindParam(":user", $user);
	if($query->execute()){
		$response["error"] = FALSE;
		$response["error_msg"] = "Data saved, thank you for your contribution.";
		echo json_encode($response);
	}
	else{
		$response["error"] = TRUE;
		$response["error_msg"] = "Error while saving data, please contact the
		↪  developers.";
		echo json_encode($response);
	}
}
catch (PDOException $e) {
	$response["error"] = TRUE;
	$response["error_msg"] = "Error while connecting to server, please contact the
	↪  developers.";
	echo json_encode($response);
}
?>
```

Source Code 5.7: Processing collected data to add new bumps

5.3.2.3 Data processing

The processing is based on a vote system, if 3 participants or more send data about bumps in the same location then that location would be selected as an actual bump and update the Bumps list with the time stamp of when each bump has been added in our database using a PHP script.

```php
<?php
try{
	$query = $connect->prepare("SELECT * FROM CollectedData");
	$query->execute();
	$result = $query->fetchAll();
	$number = 0;
	foreach ($result as $row) {
		if($row['user'] != $user){
```

```
                        $distance = vincentyGreatCircleDistance($lat, $lon,
                         ↪   $row['latitude'], $row['longitude']);
                        $distance = round($distance);
                        if($distance <= 3){
                                $vote[$number] = $row['id'];
                                $number++;
                        }
                }
        }
        if($number >= 2){
                $query = $connect->prepare("INSERT INTO
                 ↪   Bumps(Bump_latitude,Bump_longtitude) VALUES(:lat,:lon)");
                $query->bindParam(":lat", $lat);
                $query->bindParam(":lon", $lon);
                if($query->execute()){
                        for($i = 0; $i < $number; $i++){
                                removeData2($connect,$vote[$i]);
                        }
                        $response["error"] = FALSE;
                        $response["error_msg"] = "Bump created, thank you for your input.";
                        echo json_encode($response);
                }
                else{
                        $response["error"] = TRUE;
                        $response["error_msg"] = "Error while processing data, please
                         ↪   contact the developers.";
                        echo json_encode($response);
                }
        }
}
catch(PDOException $e){
        $response["error"] = TRUE;
        $response["error_msg"] = "Error while connecting to server, please contact the
         ↪   developers.";
        echo json_encode($response);
}
?>
```

Source Code 5.8: Processing collected data to add new bumps

The same is done with the removal requests, when a bump has been confirmed to be removed after receiving 3 or more different removal requests for it a PHP script would remove the bumps from the Bumps list in our database and updates the Removed Bumps table which would be used to synchronize the participants local databases when they go online.

```
<?php

try{
        $query = $connect->prepare("SELECT * FROM DeleteRequests WHERE bump_id = :id AND
         ↪  user_id <> :user");
        $query->bindParam(":id", $bump_id);
        $query->bindParam(":user", $user);
        $query->execute();
        $number_of_rows = count($query->fetchAll());
        if($number_of_rows >= 2){
                try{
                        $query = $connect->prepare("INSERT INTO DeletedBumps(bump_id)
                         ↪  VALUES(:id)");
                        $query->bindParam(":id", $bump_id);
                        if($query->execute()){
                                $query = $connect->prepare("DELETE FROM Bumps WHERE bump_id
                                 ↪  = :id");
                                $query->bindParam(":id", $bump_id);
                                if($query->execute()){
                                        $query = $connect->prepare("DELETE FROM
                                         ↪  DeleteRequests WHERE bump_id = :id");
                                        $query->bindParam(":id", $bump_id);
                                        $query->execute();
                                        echo 'Bump deleted';
                                }
                                $response["error"] = FALSE;
                                $response["error_msg"] = "Bump removed, thank you for your
                                 ↪  input.";
                                echo json_encode($response);
                        }
                        else{
                                $response["error"] = TRUE;
                                $response["error_msg"] = "Error while processing your
                                 ↪  request, please contact the developers.";
                                echo json_encode($response);
                        }
                }
                catch(PDOException $e){
                        $response["error"] = TRUE;
                        $response["error_msg"] = "Error while connecting to server, please
                         ↪  contact the developers.";
                        echo json_encode($response);
                }
        }
        else{
                try{
                        $query = $connect->prepare("INSERT INTO
                         ↪  DeleteRequests(bump_id,user_id) VALUES(:id,:user)");
                        $query->bindParam(":id", $bump_id);
                        $query->bindParam(":user", $user);
                        if($query->execute()){
                                $response["error"] = FALSE;
                                $response["error_msg"] = "Request saved, thank you for your
                                 ↪  input.";
                                echo json_encode($response);
                        }
                        else{
                                $response["error"] = TRUE;
                                $response["error_msg"] = "Error while processing your
                                 ↪  request, please contact the developers.";
                                echo json_encode($response);
                        }
                }
                catch(PDOException $e){
                        $response["error"] = TRUE;
                        $response["error_msg"] = "Error while connecting to server, please
                         ↪  contact the developers.";
                        echo json_encode($response);
                }

        }
}
catch(PDOException $e){
        $response["error"] = TRUE;
        $response["error_msg"] = "Error while connecting to server, please contact the
         ↪  developers.";
        echo json_encode($response);
}

?>
```

Source Code 5.9: Processing bump removal requests

5.4 Results and discussion

5.4.1 Preparations

To test the functioning of the application we used a ”Huawei Y6 II” smart phone and a ”Ford Fiesta 2009” car. The first phase of our test will be by driving around the area presented in figure 5.4 and test if our application can detect most of the bumps in those roads marked with stars on the map.

The second phase of the test will be by driving around the area presented in figure 5.4 and test if our application can alert us about the marked bumps which are already registered with their locations in our bumps database.

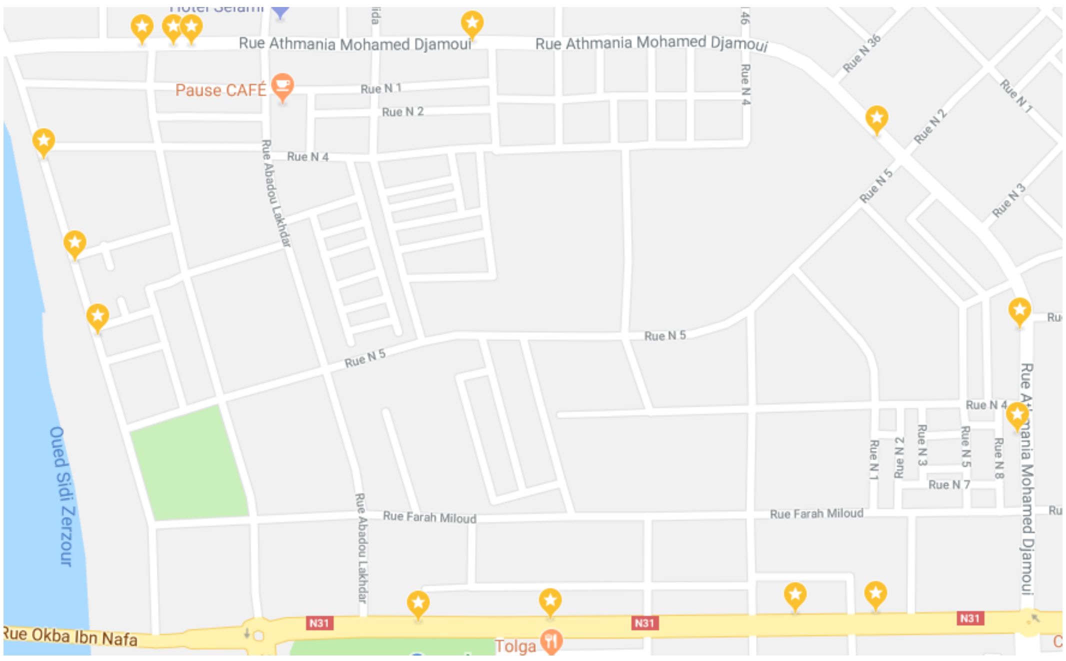


FIGURE 5.4: Map of the test area.

5.4.2 Results

5.4.2.1 Phase 1 results

The results of the first phase are shown in figure 5.5 where the green stars represent bumps that have been detected by the application, blue and grey stars represent bumps that the application could not detect.

The two blue marked bumps that were not detected due to the first being too small for the application to register a significant change in the Y axis accelerometer data.The second blue marked bump was an incomplete one that didn't span along the width of the road, so the car went over it by the left side tires and since the phone was in the center of the car, the application didn't register a significant change in the Y axis accelerometer data.

The four grey marked bumps were small metal bumps. Due to their small size the application didn't register a significant change in the Y axis accelerometer data, thus didn't mark them as bumps.

The application also detected a false bump after driving fast over a large pothole, which made it register a significant change in the Y axis accelerometer data.

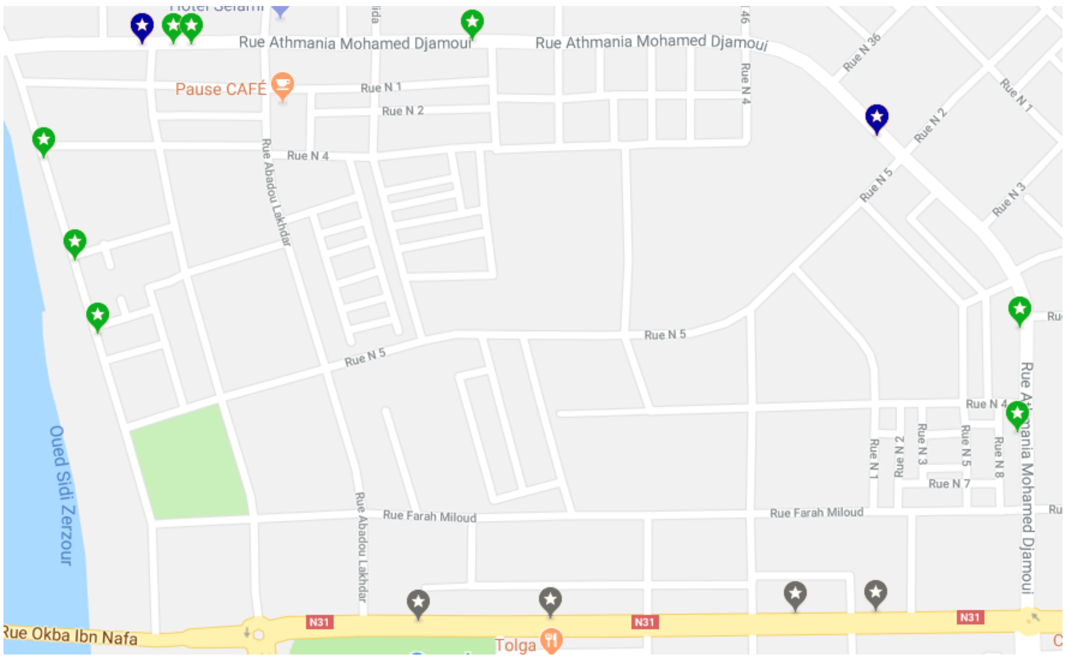


FIGURE 5.5: Phase one test results.

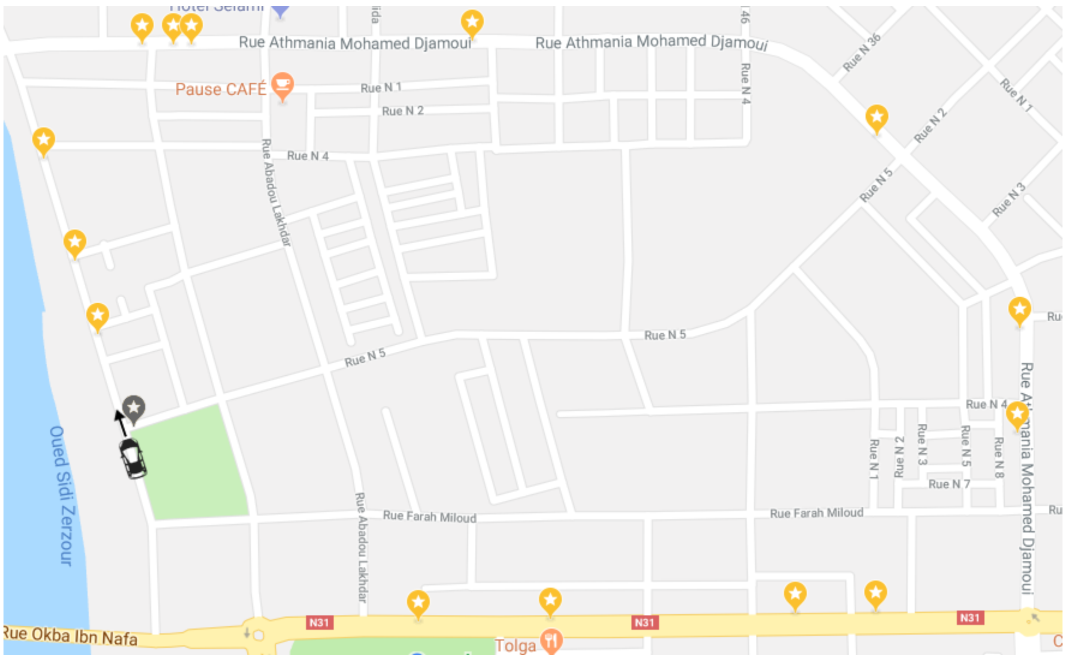


FIGURE 5.6: Phase two test results.

5.4.2.2 Phase 2 results

The application managed to send us sound alerts when being 50 meters away from all the marked bumps on the map. Since the application can't determine the user's travel direction, it can report a false positive in the case of driving past a bump that is too close to an intersection as shown in figure 5.6.

5.5 Conclusion

In this chapter we've seen the implementation of our solution and it's testing. The MCS based application can improve the driving experience but with few limitations that can be addressed in further studies.

Chapter 6

Conclusion

Mobile Crowd Sensing has proven to be a very important technique to help improve the daily life of people. One of the frequent routines of a person daily life is driving. Designing and developing a functioning application based on MCS is a challenge in itself. However we managed to provide an application that can improve the driving experience of people, as it can be used as the backbone of future MCS application. The application's limitations such as data quality can be addressed as follows: improving the sensing process by using time series pattern recognition algorithms, and improving the data processing phase by introducing machine learning to classify the collected time series data.

Bibliography

[1] Dragan Stojanovic, Bratislav Predic, and Natalija Stojanovic. Mobile crowd sensing for smart urban mobility. In *European Handbook of Crowdsourced Geographic Information*, pages 371–382. Ubiquity Press, aug 2016. doi: 10.5334/bax.aa. URL http://www.ubiquitypress.com/site/chapters/10.5334/bax.aa/.

[2] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to Mobile Crowd Sensing. *2014 IEEE International Conference on Pervasive Computing and Communication Workshops, PERCOM WORKSHOPS 2014*, pages 593–598, 2014. doi: 10.1109/PerComW.2014.6815273.

[3] James Surowiecki. *The wisdon of crowds*. Anchor, reprint edition, 2005. ISBN 978-0-385-72170-7.

[4] Thomas Malone, Robert Laubacher, and Chrysanthos N. Dellarocas. Harnessing crowds: Mapping the genome of collective intelligence. 1, 02 2009.

[5] Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y. Yen, Runhe Huang, and Xingshe Zhou. Mobile Crowd Sensing and Computing. *ACM Computing Surveys*, 48(1):1–31, 2015. ISSN 03600300. doi: 10.1145/2794400. URL http://dl.acm.org/citation.cfm?doid=2808687.2794400.

[6] Frank Ableson, Robi Sen, Chris King, and C. Enrique Ortiz. *Android in Action*. Manning Publications Co., Greenwich, CT, USA, 2011. ISBN 1617290505, 9781617290503.

[7] Serhan Yamacli. *Beginner's Guide to Android App Development A Practical Approach for Beginners*. CreateSpace Independent Publishing Platform, first edition, 2017. ISBN 9781548088163.

[8] Alex Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley. The Java® Language Specification - jls8.pdf. *Addison-Wesley*, page 688, 2014. URL https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf.

[9] Jinwei Liu, Haiying Shen, and Xiang Zhang. A survey of mobile crowdsensing techniques: A critical component for the internet of things. In *2016 25th International Conference on Computer Communications and Networks, ICCCN 2016*, pages 1–6. IEEE, aug 2016. ISBN 9781509022793. doi: 10.1109/ICCCN.2016.7568484. URL http://ieeexplore.ieee.org/document/7568484/.

[10] Bernd Resch. Progress in Location-Based Services. 2013. doi: 10.1007/978-3-642-34203-5. URL http://link.springer.com/10.1007/978-3-642-34203-5.