Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

Eduardo Castellanos Nájera

Evaluating mobile edge-computing on base stations

Case study of a sign recognition application

Master's Thesis
Espoo, August 18, 2015

Supervisors:	Professor Antti Ylä-Jääski, Aalto University
	Professor Peter Sjödin, KTH Stockholm
Instructor:	Teemu Kämäräinen M.Sc. (Tech.)

Aalto University
School of Science

Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

ABSTRACT OF MASTER'S THESIS

Author:	Eduardo Castellanos Nájera		
Title: Evaluating mobile edge-computing on base stations Case study of a sign recognition application			
Date:	August 18, 2015	**Pages:**	vi + 53
Major:	Security and Mobile Computing	**Code:**	T-110
Supervisors:	Professor Antti Ylä-Jääski Professor Peter Sjödin		
Instructor:	Teemu Kämäräinen M.Sc. (Tech.)		

Mobile phones have evolved from feature phones to smart phones with processing power that can compete with personal computers ten years ago. Nevertheless, the computing power of personal computers has also multiplied in the past decade. Consequently, the gap between mobile platforms and personal computers and servers still exists. Mobile Cloud Computing (MCC) has emerged as a paradigm that leverages this difference in processing power. It achieve this goal by augmenting smart phones with resources from the cloud, including processing power and storage capacity. Recently, Mobile Edge Computing (MEC) has brought the benefits from MCC one hop away from the end user. Furthermore, it also provides additional advantages, e.g., access to network context information, reduced latency, and location awareness.

This thesis explores the advantages provided by MEC in practice by augmenting an existing application called Human-Centric Positioning System (HoPS). HoPS is a system that relies on context information and information extracted from a photograph of signposts to estimate a user's location. This thesis presents the challenges of enabling HoPS in practice, and implement strategies that make use of the advantages provided by MEC to tackle the challenges. Afterwards, it presents an evaluation of the resulting system, and discusses the implications of the results.

To summarise, we make three primary contributions in this thesis: (1) we find out that it is possible to augment HoPS and improve its response time by a factor of four by offloading the code processing; (2) we can improve the overall accuracy of HoPS by leveraging additional processing power at the MEC; (3) we observe that improved network conditions can lead to reduced response time, nevertheless, the difference becomes insignificant compared with the heavy processing required.

Keywords:	mobile cloud, mobile-edge computing, image recognition, edge-cloud
Language:	English

Aalto-universitetet
Högskolan för teknikvetenskaper

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV DIPLOMARBETET

Utfört av:	Eduardo Castellanos Nájera		
Arbetets namn: Evaluating mobile edge-computing on base stations Case study of a sign recognition application			
Datum:	3:e augusti, 2015	**Sidantal:**	vi + 53
Huvudämne:	Security and Mobile Computing	**Kod:**	T-110
Övervakare:	Professor Antti Ylä-Jääski Professor Peter Sjödin		
Handledare:	Teemu Kämäräinen M.Sc. (Tech.)		

Utvecklingen av mobiltelefoner har skett på en rusande takt. Dagens smartphones har mer processorkraft än vad stationära datorer hade för tio år sen. Samtidigt så har även datorernas processorer blivit mycket starkare. Därmed så finns det fortfarande klyftor mellan mobil plattform och datorer och servrar. Mobile Cloud Computing (MCC) används idag som en hävstång för de olika plattformernas processorkraft. Den uppnår detta genom att förbättra smartphonens processorkraft och datorminne med hjälp från datormolnet. På sistånde så har Mobile Edge Computing (MEC) gjort så att förmånerna med MCC är ett steg ifrån slutanvändaren. Dessutom så finns det andra fördelar med MEC, till exempel tillgång till nätverkssammanhangsinformation, reducerad latens, och platsmedvetenhet.

Denna tes utforskar de praktiska fördelarna med MEC genom att använda tillämpningsprogrammet Human-Centric Positioning System (HoPS). HoPS är ett system som försöker att hitta platsen där användaren befinner sig på genom att använda sammanhängande information samt information från bilder med vägvisare. Tesen presenterar även de hinder som kan uppstå när HoPS implementeras i verkligheten, och använder förmåner från MEC för att hitta lösningar till eventuella hinder. Sedan så utvärderar och diskuterar tesen det resulterande systemet.

För att sammanfatta så består tesen av tre huvuddelar: (1) vi tar reda på att det är möjligt att förbättra HoPS och minska svarstiden med en fjärdedel genom att avlasta kodsprocessen; (2) vi tar reda på att man kan generellt förbättra HoPS noggrannhet genom att använda den utökade processorkraften från MEC; (3) vi ser att förbättrade nätverksförutsättningar kan leda till minskad svarstid, dock så är skillnaden försumbar jämfört med hur mycket bearbetning av information som krävs.

Nyckelord: mobile cloud, mobile-edge computing, image recognition, edge-cloud

Språk: Engelska

Acknowledgements

I would like to thank Antti Ylä-Jääksi, Peter Sjödin, Teemu Kämäräinen and Zhonghong Ou for their support and guidance in the work on this thesis. I would also like to thank Maneesh Chauham, Matti Siekkinen, and Tuomas Ylipiha for their collaboration in the project.

Special thanks to my friends and family for their unwavering support throughout my studies.

Espoo, August 18, 2015

Eduardo Castellanos Nájera

Contents

Chapter 1

Introduction

In recent years, mobile phones have evolved from feature phones with limited processing power, to smart phones with processing power that rivals computers ten years ago. However, personal computers and servers have not stagnated, and their processing power has multiplied in the past decade. Accordingly, the gap between mobile platforms and personal computers and servers will continue to exist. In addition to the limited processing power, mobile devices are also limited by their battery life.

Cloud computing has been leveraged and analyzed as a solution for the lack of processing power and limited battery life of mobile devices. Researchers have accomplished significant improvements in battery life and processing speedup by offloading the most demanding parts of the application to the cloud. However, research has shown that cloud computing is limited for delay-sensitive applications because connecting to a cloud server has its own restrictions due to bandwidth and latency [1–3].

Mobile Edge-Computing (MEC) introduces a new option for offloading processing and distributing workloads. It brings the mobile cloud closer to the end user. If paired with Long-Term Evolution (LTE) technology improvements, it enables new applications and a set of advantages over mobile cloud computing. Besides the obvious latency benefits that brings the cloud closer to the user, MEC also augments applications by making network and location data available to the applications. Furthermore, MEC can benefit from their static location by only caching or keeping portions of databases relevant to its geolocation. For example Yelp [4], a crowd-sourced on-line review platform, can cache information and media only for nearby commerces.

1.1 Motivation

MEC is a recent initiative and has only been explored in a few research papers [5–7]. This creates a situation in which there is little practical knowledge on the applicability and real-world challenges of the specifics of MEC. In this thesis, we propose to explore this new territory by using the research application named Human-Centric Positioning System (HoPS).

HoPS is designed to mitigate the weaknesses of the current Global Positioning System (GPS): (1) in urban areas where tall structures block the line-of-sight to the satellites, the positioning error can be in the order of hundreds of meters [6]; (2) the initial acquisition time for enough satellites to calculate the position takes between 10 and 20 seconds [8]; (3) due to the long initialization delay, it becomes inconvenient to turn it on and off frequently, and most of the time it is kept on; (4) GPS is one of the most power hungry services of mobile devices, drawing between 200-300mW [9, 10].

In its current incarnation, HoPS intends to provide a location service without utilizing low level radio information or assistance from GPS. It is based on the real-time identification of roadside traffic signposts (which is referred to as traffic signs or directional traffic signs). The rationale is that sign posts, by definition, should convey enough information to enable navigation without the need of GPS. The response time of HoPS needs to be very short in order to provide a usable service.

MEC is advantageous for such a system. A MEC server can take care of the heavy image processing, save battery life for the mobile device, and speed up the whole process. This leads to a decreased response time and fulfills one of the application's objectives. Furthermore, the application leverages mobile network context information that is also available through the MEC interface. Additionally, MEC can contribute to the accuracy of the application by enabling further processing of the image data to identify the location with a lower error margin.

1.2 Problem statement

In order to assess the benefits conveyed by MEC on base stations, this thesis intends to implement a sign recognition application and deploy it on Nokia's RACS platform. Additionally, the application's performance will be measured when using computing power from a mobile cloud, as well as just a mobile handset. The thesis will contrast the benefits and drawbacks of each method.

In this thesis we focus on the MEC platform implementation by Nokia

Networks and the improvement opportunities it enables in terms of location accuracy and response time reduction. Specifically, we try to answer the following questions:

1. Previous work has found that the processing time improvements are frequently offset by the network transmission delay. Is processing offloading an effective strategy for improving the response time given the improved network conditions? What steps must be taken to minimize the impact of the network delay?

2. Given the additional processing resources available at the MEC server, is it possible to leverage them to improve the accuracy of the location identification algorithm? Are there other strategies that can also contribute to the accuracy?

3. MEC offers services such as location awareness and mobile network context information; how can these be leveraged to augment the application?

1.3 Structure of the thesis

This thesis is structured as follows:

Chapter 2 introduces the technologies surrounding MEC, and how they evolved and contributed to the development of the MEC concept.

Chapter 3 introduces the application's current development status and the challenges it faces. It provides a snapshot of the current status of the application.

Chapter 4 analyses and proposes solutions to the identified challenges the application faces by leveraging multiple strategies enabled by MEC.

Chapter 5 describes the detail of the implementation of the improved application and it's integration with Nokia RACS.

Chapter 6 evaluates the effectiveness of the solution and discusses the results of the experiments we performed to assess the effectiveness of the implementation.

Chapter 7 discusses the challenges in the implementation and the evaluation results.

Chapter 8 offers a conclusion and future work.

Chapter 2

Mobile-edge computing

Mobile-edge computing (MEC) is a natural evolution of cloud technologies combined with the rise of mobiles' popularity. Mobile-edge computing leverages mobile base stations to bring cloud computing as close to the mobile user as physically possible in a mobile infrastructure. Before delving into details about MEC, the following sections introduce the technologies that make it possible.

2.1 Mobile cloud computing

The National Institute of Standards and Technology of the U.S. Department of Commerce (NIST) defines cloud computing as a model for accessing a shared pool of configurable computing resources that can be configured and provisioned with minimal management effort and service provider involvement [11]. By pooling the resources, cloud computing enables higher efficiency and cost reductions. This model has become quite popular, and its flexibility has enabled a wide variety of applications.

Cloud computing is a model that easily adapts to become a promising solution for mobile computing. One of the main applications of cloud computing is to augment mobile devices capabilities. This particular application has been called mobile cloud computing (MCC). MCC can augment mobile devices in terms of data storage, processing resources, and mobility.

MCC has been defined concretely by Sanaei et al. as *"A rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms based on the pay-as-you-use principle"* [1]. Most of the research has

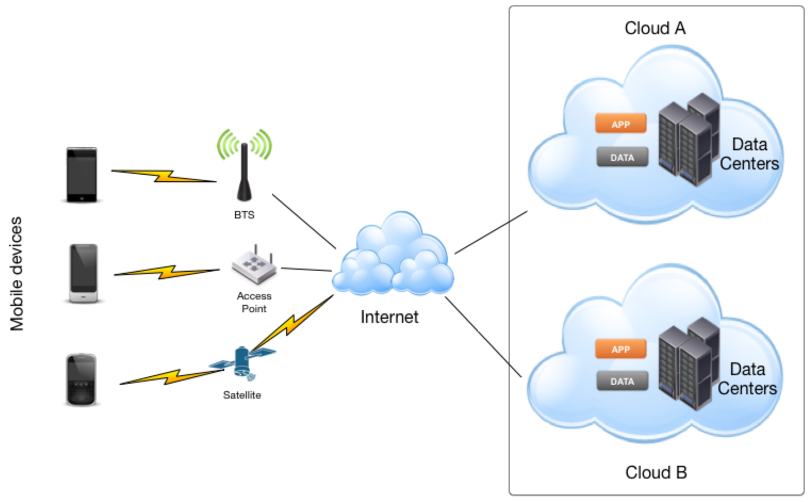


Figure 2.1: Mobile cloud computing architecture

been focused on ”cyber foraging” as Satyanarayanan initially described [12]. This approach introduces code offloading as a novel way to expand mobile devices’ limited resources. Many researchers have demonstrated the benefits and implemented code offloading frameworks that leverage MCC resources in novel ways. Commercial products such the Amazon’s Silk browser [13], the Apple Siri [14] voice activated assistant, and Google Goggles [15] are examples that use this approach.

The general architecture of MCC is shown in figure 2.1. Mobile devices can be connected to the Internet through different means. For example, mobile networks, Wi-Fi, or satellite connections can provide the means to reach the Internet though Internet Service Providers (ISP). ISPs provide the network infrastructure that will route the links through the appropriate paths on the Internet in order to connect the mobile device with the cloud controller. Cloud controllers then process the incoming requests from mobile clients and deliver them to the appropriate cloud services. These services have been developed with the concept of utility computing, virtualization, and a service oriented architecture [2].

In addition, an alternative interpretation of the term mobile cloud computing exists. It envisions a collection of neighboring mobile devices that pool together their resources in order to share them. This model is termed an “ad hoc mobile cloud”. In this model, a task from a mobile device is distributed and processed in a collaborative fashion on the devices that belong to the ad hoc mobile cloud as can be seen in Figure 2.2. This model has been

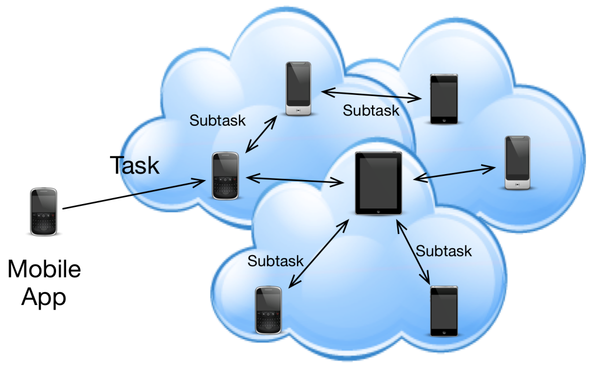


Figure 2.2: Mobile ad hoc cloud computing architecture

demonstrated in Virtual Cloud Provider [16] by distributing a Map Reduce application over a set of mobile devices.

However, there are still challenges in the MCC architecture. Connecting to servers in the cloud carries a price tag in the form of increased latency, sensitivity to the application availability, quality of service, and bandwidth limitations. These factors have limited the diversity of applications that MCC can enable. For example, augmented reality or assisted cognition rely on transmitting streams of sensor data and video to a resource rich server that can process them and deliver a result close to real time [17]. As a consequence, Satyanarayanan proposes a third vision of mobile cloud computing called a cloudlet.

A cloudlet is a trusted, resource-rich device that is self-managed, compact and can be deployed in business' premises. It is decentralized and owned by local business, leverages LAN latency and bandwidth, and is intended to serve few users at a time [17]. In this model a mobile device also leverages resources in the cloud. In contrast to the MCC architectures discussed earlier, the cloudlet model proposes bringing the cloud closer to the user by placing a device on the first hop of the network as shown in Figure 2.3. This is beneficial for several actors. First, for the end user the applications will seem more responsive and it will enable delay sensitive applications. In addition, network providers can exploit the location of the cloudlets to cache media and data and thus save bandwidth and resources on the core network. Lastly, application service providers benefit from the increased scalability of their applications since they can be distributed on the cloudlets.

Cloudlets achieve this by utilizing virtualization technologies. An application developer can create a virtual machine (VM) overlay by customizing

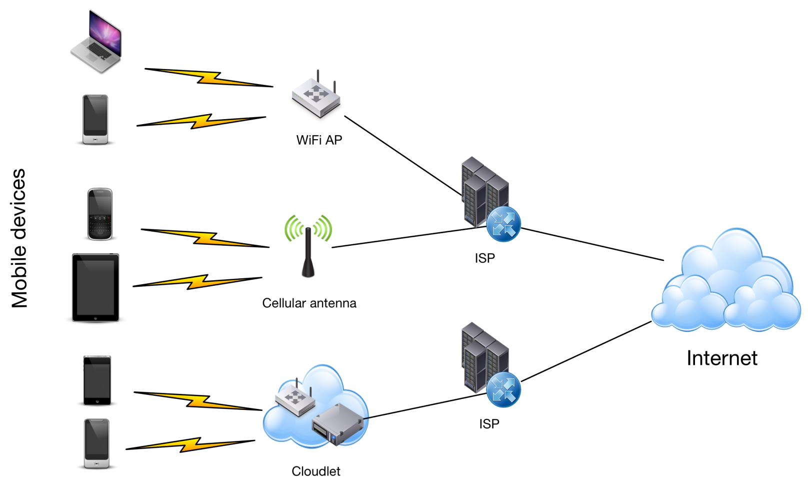


Figure 2.3: Cloudlet architecture

a base VM and then calculating the difference between the base VM and the customized one. Therefore, the size of the overlay is reduced and can be stored in the client and then transmitted to the cloudlet where it is combined with the base VM and executed. Afterwards, the mobile device can make use of it. Since cloudlets are defined as having a soft state, at the end of the execution the VM is discarded, along with any data on it.

Several applications in the academic world have exploited Cloudlets to implement applications that require low latency and high traffic rates. One such is the *Gabriel Architecture for Cognitive Assistance* [3]. This architecture proposes a framework to stream sensor data from sources like video, GPS, accelerometers, etc., to a control virtual machine (VM) in a Cloudlet which pre-processes and decodes the streams to make the data available to cognitive VMs specialized in tasks like face recognition, optical character recognition (OCR), object recognition, etc. The cognitive VMs process the streaming data and produce a symbolic representation that can be then used by complex higher level processing to produce an overlay or user feedback [3]. A concrete example is presented by Satyanarayanan, et al. [18] that utilizes this architecture to assist 2D Lego assembly.

Cloudlets continue to be researched in academia, but it was not until recently that commercial solutions that implement a similar paradigm have become available. The industry has termed this model as Mobile-edge computing, which the following will describe in detail.

2.2 Mobile-edge computing

Mobile-edge computing (MEC) and Cloudlets are very similar: they are both placed at the first hop of the network, offer storage and computing to nearby devices, and mobile users access them through wireless links. A MEC server can be deployed at an LTE macro base station (eNodeB) at the UMTS radio network controller (RNC), or at a multi-technology cell aggregation site. A multi-technology aggregation site controls a number of local, multi-technology access points that provide radio coverage on a premise [19]. However, MEC differs from Cloudlets in that it is managed by the mobile infrastructure provider, it integrates network operator related knowledge, and MEC servers are widely deployed and available to all mobile users. In addition, MEC servers have access to position and mobility information [20].

MEC is born as an European Telecommunications Standards Institute (ETSI) initiative in the form of an industry specification group (ISG). Huawei, IBM, Nokia Networks, Intel, NTT DOCOMO, and Vodaphone are the main sponsors. The purpose of the ISG is to standardize MEC in an open environment which will allow the integration of heterogeneous technologies from different parties to form multi-vendor MEC platforms. According to the ISG's Introductory Technical White Paper [21], MEC is characterized by being on-premises, proximity, lower latency, location awareness, and network context information.

The first characteristic of MEC is that it is on-premises. This means that the MEC server is local and can run independently and isolated from the rest of the network. In the event of connectivity loss to the core network, an application running on a MEC server would be unaffected and will be able to continue to operate normally.

MEC servers are also close to the source of information, the mobile devices. This proximity enables MEC servers to become agrregators of data and enables the capture of big data and analytics. Crowd-sensing applications and Internet of Things (IoT) devices benefit the most from this characteristic because all the data collected from them can be aggregated and pre-processed at a MEC server before uploading it to a central repository. Therefore, the information traffic is reduced and the mobile infrastructure provider as well as the application developer benefit from the reduction in bandwidth usage [21].

Another advantage of a MEC server's point of view is reduced latency to the end user devices. Reduced latency enables response time sensitive applications such as augmented reality or cloud gaming.

MEC servers also share information about their location as well as low-

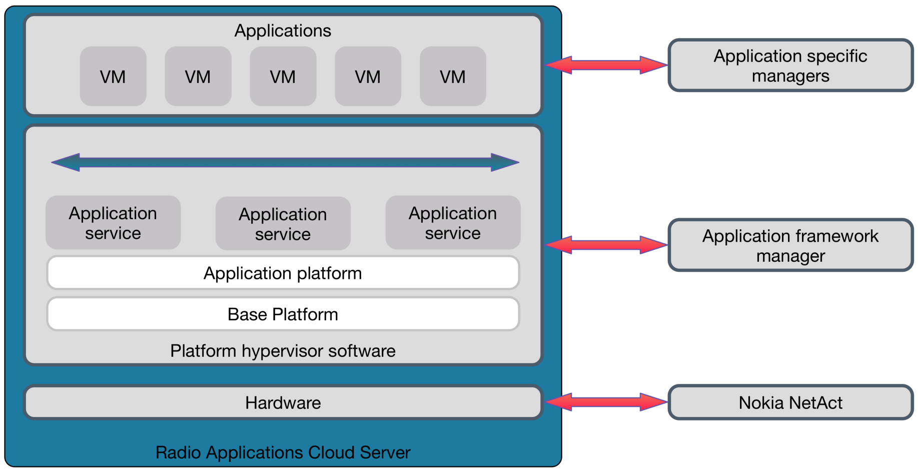


Figure 2.4: RACS Architecture

level signaling information with the applications. This enables location based services, analytics and differentiation of the content served in regard to the network conditions and location [21].

MEC had only been discussed as a concept until 2014 when Nokia Networks introduced the first example of a complete real-world MEC implementation [22]. This platform is called Radio Applications Cloud Server (RACS), and is discussed in further detail in the following section.

2.3 Radio Applications Cloud Server

The Nokia Networks Radio Applications Cloud Server (RACS) [22] is MEC solution that encompasses all the aspects required to create, deploy, and manage applications packed as virtual machines. A RACS contains a software platform that execute the virtual machines containing the applications. In a real-world deployment, the RACS can be placed in various locations within a mobile infrastructure provider's network. Then can be placed at the eNodeB, RNC, or at aggregation sites. In order to manage the multitude of RACS, the solution provides application framework managers and a proprietary tool called NetAct that interfaces with the RACS base system.

The software platform consists of three main components: a base platform, an application platform, and application platform services. The base platform provides the raw resources such as computing, storage, and connectivity. The application platform provides a hyper-visor upon which application virtual appliances run. Lastly, the application platform services encompass the RACS programming application user interface (API). These services include network traffic services that route IP traffic streams to and from the RACS, and information services that expose APIs that the applications utilize to retrieve the state of a UE as well as the radio network subsystem. These components work in conjunction to provide the platform on which the applications will be executed as shown in Figure 2.4.

The application framework managers are used to manage the virtual appliances deployed into RACS. It provides a user interface to manage the deployment of RACS applications. This includes application configuration management activities such as configuration policies, application deployment, execution, termination, and suspension. It communicates directly and manages collections of application platforms.

Nokia's NetAct network element management performs the monitoring of the base platform. It takes care of fault, configuration, accounting, performance, and security management.

As it is the only implementation of MEC available, this thesis makes use of Nokia Networks RACS platform to implement a distributed application and leverage the benefits the platform provides. The most important ones are the application platform services that provide network information, of which the CellID is particularly useful as will be described in later chapters; and low latency to the UE, which is crucial for augmented reality.

Chapter 3

Human-centered Positioning System

Global Navigation Satellite Systems (GNSS) such as Global Positioning System and the European Galileo have dominated the outdoor navigation systems for the past decades. The strengths of these systems are obvious, including weather resiliency and remarkable accuracy. However, these systems are not always the best choice for real-time applications because the initial acquisition time takes tens of seconds (10s-20s) and urban areas present many surfaces such as buildings that reflect and obstruct radio signals, thus the positioning error can be off by hundreds of meters [23].

The research group in the Aalto University has developed an application called Human-Centered Positioning System (HoPS) that is able to recognize traffic signs in an image and then combine this with context information in order to effectively calculate a user's position without relying on GPS, WiFi fingerprinting, or any other radio fingerprinting. The main task of the thesis is to understand the functionality of the application and identify the parts that can be offloaded or otherwise augmented by leveraging the MEC architecture via making use of Nokia's RACS server. Hence, this chapter lays the groundwork by describing the implementation of the application.

3.1 Location detection process

The application relies on identifying context information from the mobile network and combining it with information extracted from a camera snapshot. In particular, the application identifies traffic signs within the camera snapshot and counts them. Then it uses the information gathered from the detected signs together with the available context information such as the altitude, quality of the connection to the base station, and connected base station Cell ID to determine the location of the user. The overview of the location algorithm can be seen in Figure 3.1.

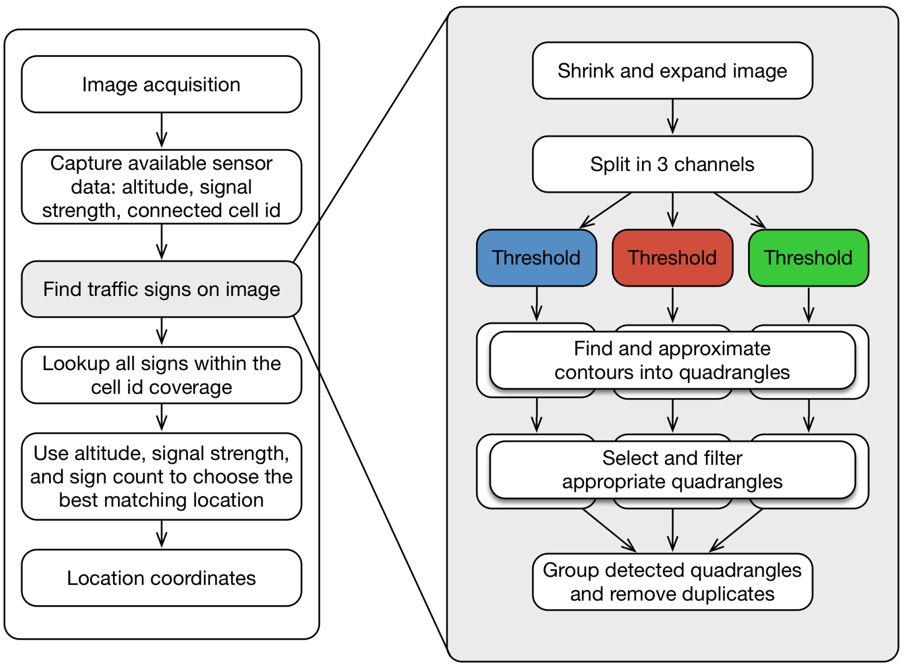


Image acquisition
Capture available sensor data: altitude, signal strength, connected cell id
Find traffic signs on image
Lookup all signs within the cell id coverage
Use altitude, signal strength, and sign count to choose the best matching location
Location coordinates
Shrink and expand image
Split in 3 channels
Threshold
Threshold
Threshold
Find and approximate contours into quadrangles
Select and filter appropriate quadrangles
Group detected quadrangles and remove duplicates

Figure 3.1: Traffic sign location application.

3.1.1 Sign detection

The process of the signs detection in the application starts by first blurring the image in order to minimize the noise and improve the accuracy of the contours algorithm. The method utilized to blur the image is by using a Gaussian pyramid to reduce the image to half of its size and then utilizing a Laplacian pyramid to reconstruct the image to the original size [24].

After removing the noise, the next step is to split the image into 3 images, one for each color channel. Next, each of the 3 channels is treated as a grayscale image.

The algorithm then takes the images and generates 10 images per input image by choosing a gray value as a threshold and turning every value below it into a white pixel, and anything above the threshold into a black pixel. In addition, the algorithm produces 3 more images by using a Canny edge detector [25] 2 times with different settings to highlight the edges on the images, and an adaptive threshold technique that highlights edges but is also effective when there is unevenness in lightning [26]. In total, this step produces 13 black and white images for each one generated on the previous step. In total 39 black and white images are produced.

Afterwards, each of these 39 images is processed with a function designed to find contours with an algorithm designed by Satoshi Suzuki [27]. The contours detected are then approximated using the Douglas-Pecker algorithm into straight lines [28]. Then, the groups of lines that form closed convex quadrilaterals are selected. Furthermore, the algorithm inspects the inner angles of the quadrilaterals and discards those that that are more than 5 degrees away from 90 degrees. Next, the area of the the quadrangle is calculated so that is it between 0.05% and 12.5% that of the original image. These percentages are needed in order to avoid selecting regions that are too large, or too small to be considered as traffic signs. This process will typically yield a multitude of quadrangles believed to be signs. An evaluation of the signs detection algorithm on a sample set of 95 images yielded an approximate 58% of average identification accuracy.

3.1.2 Sensor data

The Android API provides access to a myriad of sensors on mobile devices. The application makes use of the pressure sensor to determine the altitude. However, before being able to accurately measure the altitude, the pressure sensor must be calibrated with an on-line service. In the application, the device queries the Finnish Meteorological Institute [29] by using their public API to find out the reference pressure at the current location. In addition to the pressure sensor, the algorithm makes use of the signal strength to the connected base station together with the base station's CellId. This information is used to narrow down the search to a specific area before performing further filtering and querying.

3.1.3 Database search

The last step the application has to perform to approximate the user's location is to perform a database search using the gathered and processed data from the camera and sensors. The first step is to query for all the recorded locations that are within range of the base station identified by the CellID recorded. If this operation produces only one record, then this location is returned. Otherwise, the signal strength and the altitude are compared against the candidate locations and matched as closely as possible. If there is more than one location that matches, the last operation is to count the number of traffic signs in the image and use this to select a single location. If there is still more than one match after this last operation, then the process return several possible locations.

```
Data: Context information [$cell\text{-}ID, signalstrength(ss), altitude(alt)$]
      associated with the input image
Result: Estimated unique location, or multiple candidate locations
        for further processing
Retrieve the set of candidate locations ($Set_{loc}$) from the database
using $cell\text{-}ID$
if $Size(Set_{loc}) > 1$ then
    $alt_{th} \leftarrow 0.2$
    $ss_{th} \leftarrow 0.2$
    while $Size(Set_{candidates}) < 0$ do
        for $loc \leftarrow Set_{loc}$ do
            if $|loc[alt] - alt| < alt_{th}$ and $|loc[ss] - ss| < ss_{th}$ then
                Append $loc$ to $Set_{candidates}$
            end
        end
        $alt_{th} \leftarrow alt_{th} + 0.2$
        $ss_{th} \leftarrow ss_{th} + 0.2$
    end
end
else
    $Set_{candidates} \leftarrow Set_{loc}$
end
Return the candidate locations to the end user
```

Algorithm 1: Database matching algorithm.

3.2 Original implementation

The research group already built a first prototype of a HoPS application. In the original implementation, the mobile device performs the sign recognition, and afterwards the resulting data is sent to a remote server through an HTTP post request or looked up in a local database. The advantage of this approach is that the resulting data is small and takes very little to transmit when querying a remote service. Keeping a local database is challenging because the database would have to grow significantly as the application's coverage area increases. The following describes the implementation specifics of the system's first prototype.

3.2.1 Sign detection and sensor data collection

Most of the application is self contained and operates in the mobile device itself. The sign detection algorithm is implemented in C++, and leverages the widely used OpenCV library [30] for the computer vision related tasks such as thresholding, contour finding, and contour approximation necessary in the sign recognition process described earlier. This process is executed in the mobile device and uses the output information to query a local location database or a remote location service to retrieve candidate locations and narrow down the real location of the user.

3.2.2 Remote location service

Field	**Type**
location_id	integer
cell_id	integer
signal_strength	integer
altitude	real
sign_count	integer
location_area_code	integer
longitude	real
latitude	real

Table 3.1: Basic location database

The remote location service is implemented to run on the cloud. The service has been implemented using Python with the Flask framework [31] as the server. Flask is a lightweight and flexible Python framework. A MongoDB database stores all the data for the locations, signal strength, altitude, and sign information. The database consists of a single table, shown in Table 3.1. The Python application queries the database for all the records matching a provided CellID, and then iterates over the results comparing the signal strength and altitude for the best match.

Chapter 4

Analysis and Design

This thesis focuses on leveraging MEC via Nokia's RACS to improve the performance of the HoPS application. Leveraging a remote server also comes with a new set of challenges. The most evident is the network usage since it introduces a delay in the form of the data transmission latency and the response delay. The data transmission latency is more apparent when large quantities of data, like images, have to be transmitted and processed remotely. The response delay is affected by the round-trip time of a packet in a mobile network. Therefore, the first step in this process is to measure the performance of the application and identify possible improvement strategies. With these strategies in mind, this chapter will analyze how MEC's most prominent characteristics, e.g., on-premises, proximity, lower latency, location awareness, and network context information, can be leveraged by the HoPS application while minimizing the effect of the challenges described above.

4.1 Application performance measurements

The first step before analyzing the advantages of MEC is to define the metrics on which to base the comparison. The HoPS application aims to provide a quick response to a user's positioning query based on an image snapshot. Thus, the main performance metrics should be the algorithm execution speed and its accuracy. The execution speed is measured in milliseconds, and the accuracy is the percentage of correct traffic sign detections. Therefore, we utilize a data set of 95 images to build the database. The signs within each image are manually identified to provide a baseline of comparison for the accuracy of the application.

The sample data set consists of 95 images taken around Helsinki, Finland.

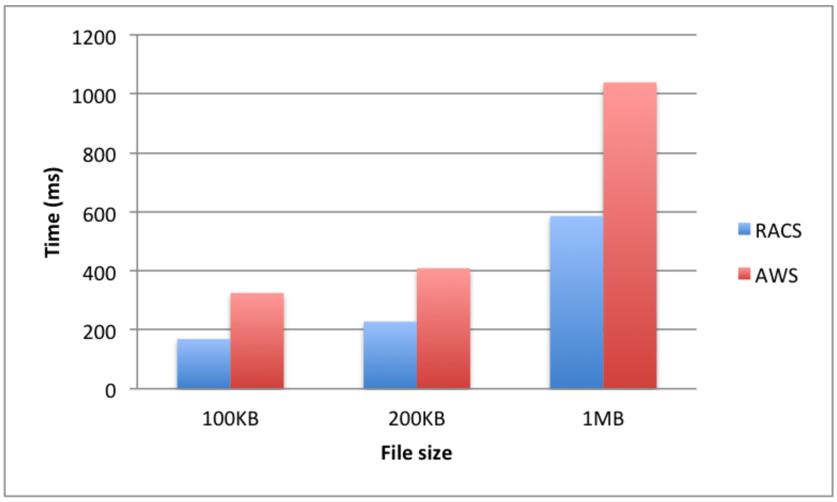


Figure 4.1: Image transmission time

The images are 2448 by 3264 pixels in size and compressed with a JPEG quality factor of 96. All of the images show signs of varying sizes, some have more than one, and 5 at most. For our purposes, adjacent signs are grouped together whenever they form a larger rectangle. In total, the image set contains 219 signs.

This section then uses the sample data set to perform experiments that show concretely how much each of the RACS features can affect the performance and accuracy of the application.

4.1.1 Network transmission latency

The first step to assess the performance of the application is to evaluate the impact that the network conditions will have on the response time. The first experiment focuses on measuring the round trip time (RTT) from the application on an Android device to a RACS, and also to a remote server in the cloud. For the remote cloud server we utilize an Amazon Web Services (AWS) server located in Frankfurt (eu-central-1) since it is the closest to the Aalto network. We use Internet Control Message Protocol (ICMP) echo request packets and then wait for the corresponding ICMP echo reply packet and measure the time difference between the packets to calculate the RTT by using the Linux ping tool. The average network latency in milliseconds for a packet from the mobile device to the RACS is 14ms, and the average latency between AWS and the mobile device from Aalto's network is 47ms.

The next step is to compare the bandwidth between the Android device and the remote servers. Interestingly, the bandwidth is not used fully when transmitting a 1MB file, much less when transmitting smaller ones. The latency and the network conditions impact the congestion window growth, and thus the transmission time of the image files between the device and the remote processing server as can be seen in Figure 4.1.

4.1.2 Sign detection accuracy

We propose two different strategies to reduce the file size of the images sent to the RACS in order to minimize the network transmission delay penalty. First, we evaluate the how the accuracy is affected when reducing the resolution of the image since reducing the resolution has a direct effect on the image size. The second strategy is to compress the image data. The image can be compressed by using either lossy or lossless algorithms. A lossless algorithm won't affect the accuracy of the algorithm since no data is lost, but won't reduce the file size significantly. On the other hand, a lossy algorithm will reduce the file size significantly, but it will affect the accuracy of the algorithm because it introduces artifacts, or distortions of regions in the original images [32]. Since lossless algorithms don't affect the file size much, we focus on testing how the accuracy is affected by the JPEG Discrete Cosine Transform (DCT) compression algorithm [33].

First we need to establish a baseline accuracy, and for this, we assessed the accuracy of the original data set first. For this experiment, the sign detection accuracy is measured as the percentage of the known signs that were detected, while the false positives are ignored since they can easily be filtered out. The accuracy of the sign detection algorithm on the original data set is 58.45%. Each of the images in the sample set is rescaled to varying resolutions with representative megapixel (MP) count. The original images have a pixel count of 8MP, and we generate copies of the sample set in 1 to 7 MP and 0.3MP. At the same time, each new set of rescaled images is compressed with even JPEG compression levels ranging from 2 to 96. This amounts to 432 sample sets. The detection algorithm is executed on each of the sets and the averages calculated for the detection accuracy.

The first test is used to determine exactly how much the information density varies with the compression and resolution. The results of these tests are shown in Figure 4.2. From the results, it becomes obvious that using JPEG compression doesn't affect the algorithm's accuracy significantly until the JPEG quality factor is set below 20. On the other hand, reducing the resolution of the images does have a greater effect on the detection accuracy. The accuracy is affected when the resolution is lower than 6MP.

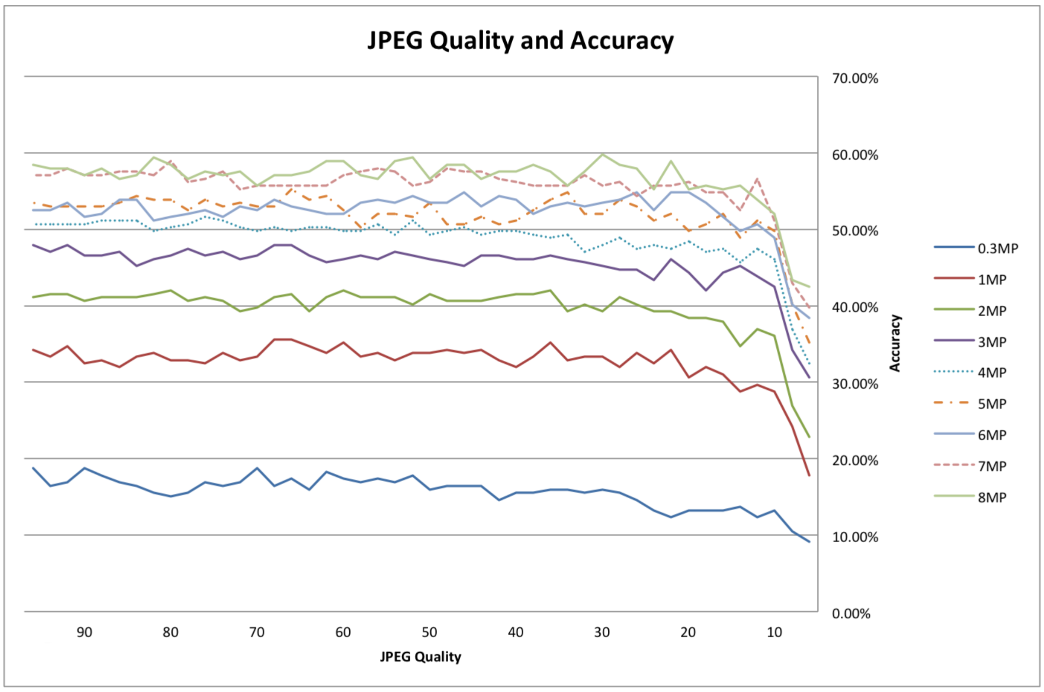


Figure 4.2: Algorithm accuracy related to compression and JPEG quality.

4.1.3 Sign detection processing time

Next, we assess the time it takes for the algorithm to execute in different devices. We choose to compare a wearable device, Google Glass; an average smartphone, Samsung Galaxy S4+; and a MEC server, Nokia RACS as shown in Table 4.1. The results of processing the original images are shown in Figure 4.3.

Device	Processor	RAM	Storage
RACS	Intel Xeon	16 GB	400 GB
Galaxy S4+	Qualcomm Snapdragon 800	2 GB	16 GB
Google Glass	TI OMAP 4430 SoC	2 GB	12 GB

Table 4.1: Devices used for testing.

From these results, it is evident that utilizing a RACS server provides an advantage over processing the images on the mobile or on Google Glass. The other interesting result is that the processing time is also diminished when the image resolution is reduced.

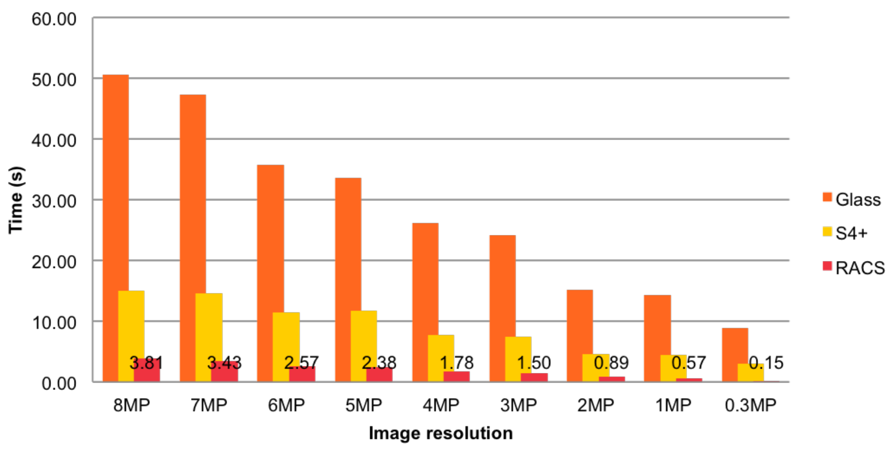


Figure 4.3: Sign detection execution times on different platforms

4.1.4 Database location matching

In the original implementation, the database location matching algorithm is dependent on the CellID of the station the phone is connected to, the signal strength, and the altitude. Afterwards, and as a last recourse the algorithm counts the number of signs found at the location in order to locate the user. After testing with different mobile devices, we found out that the signal strength measurement is a reliable factor even though it can vary slightly depending on the weather and other atmospheric conditions or the mobile chipset. In addition, the atmospheric pressure sensor is not available on most devices according to OpenSignal [34], and when it is available, it needs calibration. In order to calibrate a pressure sensor, one must compare the sensor's reading with the expected pressure calculated by taking into account the temperature and height of the location. This is usually achieved in smartphones by querying an online service with the current location. Finally, counting the number of signs is not enough for the algorithm to find a location. The way a photo is taken can influence on the detection, and a user could take a picture that doesn't include all of the signs in the area. The detection algorithm's accuracy is also too low at this point for the approach to be effective.

Database location matching is an area that can be improved. The following section takes into account the above described shortcomings of the current algorithm to propose a solution to them.

4.2 Design of a RACS-enabled solution

Considering the previous results, it becomes evident that the current algorithm can be improved in several ways by using the advantages provided by the RACS platform. The advantages that the platform provides and we can readily use are: code offloading, improved network conditions, distribution, and location awareness.

4.2.1 Code offloading

Code offloading allows not only the faster processing of the data, but it also allows us to execute more complex processes such as optical character recognition (OCR) or further image analysis. The previous section shows the time benefits of code offloading compared to running the application on the device. Together with the need to find a more reliable way of detecting the location, OCR is a good candidate for a new location algorithm.

Using the data from the analysis it becomes obvious that the best choice for the JPEG quality setting is 22. In Figure 4.2 we can see that the 22 is the peak after which the quality starts to degrade. Choosing the lowest quality setting enables us to reduce the image size considerably. A 2403 kB image can be compressed to 10% of its size to 235 kB. The resolution also affects the file size directly since less pixels have to be encoded. However, it also has a noticeable impact on the sign detection algorithm, and quickly lowers the detection accuracy. In addition, we intend to use the images for text detection. This requires higher resolution images for the text extraction to be successful. Consequently, we chose to sacrifice the resolution of the image for the benefit of the smaller file size. Consequently, we choose the 7MP resolution setting.

The traffic signs text provides meaningful information about the direction the user is heading, but often the signs are repeated along a road. However, even though the signs share the same text, they have slightly different shapes, icons or direction arrows change, or the text occurs in different positions. Utilizing this information is not straight forward; The lighting conditions are varied and the signs appear distorted because of perspective. These factors have to be taken into account when processing the identified signs.

4.2.2 Improved network conditions

The improved network conditions also mean that the TCP connection can be established more quickly, the congestion window can grow faster, and thus a burst of image data can be sent in less time. Establishing a TCP connection requires a handshake which will consume 1 round-trip time (RTT) before the sender can start sending data. The lower latency to the RACS directly affects the connection establishment delay because the RTT is lower. In addition, the congestion window growth in TCP is dependent on the RTT. The congestion window will grow faster as the RTT is lower because the congestion window limits the rate at which the sender can send data and grows with each packet confirmation. A lower RTT means that packets can be confirmed more quickly, and thus, the congestion window can grow faster and the data is transmitted at a greater rate.

4.2.3 Location awareness

A RACS's location awareness and network information services let the application query the current CellID. The amount of information stored in the local database can be reduced by taking into account the cell's coverage area information, and only storing data relevant to the cell's coverage area. In this way, the first query for the CellID can be skipped, and the database on which the location algorithm searches are performed will be smaller and produce faster responses to even more complex queries.

4.2.4 Distribution

Another factor that is relevant due to the location of the RACS inside the base station is that the application can become independent of Internet services and rely solely on the RACS. The application can be used without the need to contact the core network or a server in the cloud. In this way, network operators can benefit from the reduced bandwidth usage on their back-haul links between the eNodeB and the core network. Also, in emergency scenarios where the link between the eNodeB and the core network is lost, the application could still provide its service and would remain functional.

Our application benefits directly from the distribution because then each RACS server would only need to keep a small local portion of the database that is relevant only to its coverage. However, a central entity is necessary to manage all the distributed nodes and keep them updated. In our case, the traffic sign positions are assumed to be static over time and thus the updates can be performed sporadically. The database is meant to aggregated from measurements from the mobile clients in a crowd-sourced fashion. The MEC servers can collect the data, process it, and compare it with their local databases before contacting the global server to push an update.

Chapter 5

Implementation

Following the analysis and design of the solution, this chapter describes the system implementation and technical implementation decisions taken. This chapter deals with the algorithms developed and the technologies and methods we used to achieve the design goals.

5.1 Overview

After taking into account the system described in Chapter 3, and the analysis and design from the previous chapter, we propose an improved system architecture and a new method for extracting information from the image data.

The system is composed of three main components. The first one is the client application which runs on the mobile device. The purpose of the client application is to interact with the end-user and capture the data for later analysis by the second component. The second component is the server application, which runs on the remote server. In our approach the remote server can either be a MCC server located in a third-party provider such as Amazon or a MEC server installed in an eNodeB. The third component is a distribution server which would have to reside in the cloud, or somewhere in the internal network of the mobile infrastructure. This server's task would be to coordinate and update the data in the MEC servers. In this thesis, the implementation of such a server is left for future work.

The basic work flow of the system is that context information and a photograph is captured at the mobile device via the client application. The client application then minimizes the size of this data, and queries the remote application running on either the MEC or MCC server. We call this process *image optimization*. The server offers a web service that accepts these queries and first produces a set of candidate locations based on the context information. If the context information produces only one candidate location, then the server application sends the candidate location back to the application and it does not perform any further processing on the image data. However, this is rarely the case, and most of the times the server will need to perform further processing on the image to detect the traffic signs present in it, remove the false positives, and analyze the textual data contained in them. Finally, after the processing the server application will score each of the candidates and will pick the best suited one. This candidate is then sent back to the user. The following sections describe in detail the implementation of such a system.

5.2 Client application

The purpose of the client application is to capture the context information and the image on the mobile device. This thesis focuses on Android mobile devices since it is the dominant platform in the market [35]. The prototype Android application must provide a simple interface to take a photograph of road signs, process the image, send the image and the context information to the server, and retrieve the resulting location. The following describes the process in detail.

5.2.1 Image and context information

In the previous chapter, we realize that it is necessary to re-size and compress the images. We can readily achieve this on the Android application because the Camera API provides a method of specifying how the camera will process the image before handing it over to the user. The processing compression and resizing of the image all happen in the camera chipset without a significant delay. Thus, performing the resizing and image compression on the Android headset produces no significant overhead.

The Android API also provides a way to query the signal strength via the PhoneStateListener together with the TelephonyManager. We can also obtain the CellID with this API, but it is not longer necessary since the RACS server also provides this context information.

5.2.2 Image transmission mechanism

After the application collects the relevant query information and the user has taken a photograph, the application contacts the RACS server by means of a Representational State Transfer (REST) request. This means that the application establishes a TCP connection to the server, and then performs an Hypertext Transfer Protocol (HTTP) POST request.

The REST API is simple, it consists of a single Uniform Resource Locator (URL) to which the information is to be sent by a POST request. The URL is an address of the form: “http://[ServerIP]/query”. Afterwards, the server produces an HTTP response which contains the location results in a lightweight data interchange format. The format chosen in JavaScript Object Notation (JSON) since it is easy for humans to read an write, which aids in the development and testing, and it is also easy for computers to generate and parse. The web service is described in further detail in the following section.

5.3 Server processing and matching

This section describes the server implementation, which receives the data from the client, performs the bulk of the processing of the image to detect relevant traffic signs, and then queries the database to determine the location.

5.3.1 Web service

The first step for the server is to publish a service that allows a mobile device to interact with it. The server must provide a reliable and robust API that the mobile client will utilize to query the service. The API must consider that the client needs to transmit image data and context information and obtain a location as a response. This API has been implemented utilizing REST.

The REST API exposes only one service: */query*. The query service expects a POST type request with the context information encoded in JSON. The format of the JSON data is shown in Listing 5.1. However, the image is not transmitted in the JSON structure because it contains binary data and JSON enconding would multiply the size of the data. Instead, the image file is transmitted using HTTP’s standard *multipart/form-data* encoding, which is suitable for binary files, and produces little overhead.

Listing 5.1: JSON query data

```
{
    "data": {
        "cellid": "13745",
        "ss": "-73"
    }
}
```

The server can respond to the requests in two ways. The first one is a successful query without errors. In this case the response consists of a JSON object that includes the location coordinates like shown in Listing 5.2.

Listing 5.2: JSON response data

```
{
    "location": {
        "latitude": "60.3273",
        "longitude": "24.3883"
    }
}
```

The server implementation makes the REST service available through HTTP. For this reason the server leverages Flask to provide the web service. Flask is a minimal Python framework for web applications. Being minimal, it is very flexible and can be easily adapted to the needs. Also, previous research work on the application is in Python. Choosing Flask enables us to re-utilize this code without much effort and adapt it to the MEC solution.

5.3.2 Traffic sign detection

The next action the server must perform is find the traffic signs in the image data. For this, we must take into consideration that Python is an interpreted language, which makes it slow compared to compiled programming languages such as C++. Consequently, we implement the traffic sign detection in C++ and leverage the OpenCV library of computer vision algorithms. The OpenCV library implements optimized versions of common and popular computer vision algorithms such as Canny edge filter, thresholding, contour detection, etc. The sign detection work flow remains unchanged from the original application described in Chapter 3, where these filters are first introduced. However, in the following we introduce additional filtering in order to minimize the unnecessary processing in later stages.

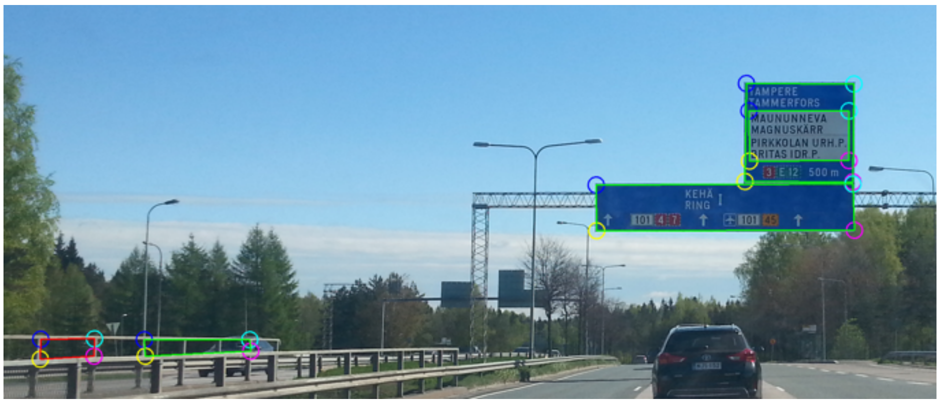


Figure 5.1: Detected signs before filtering.

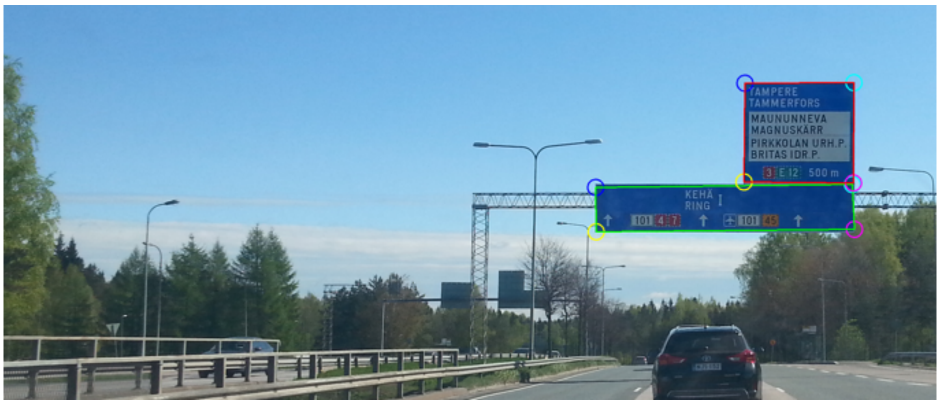


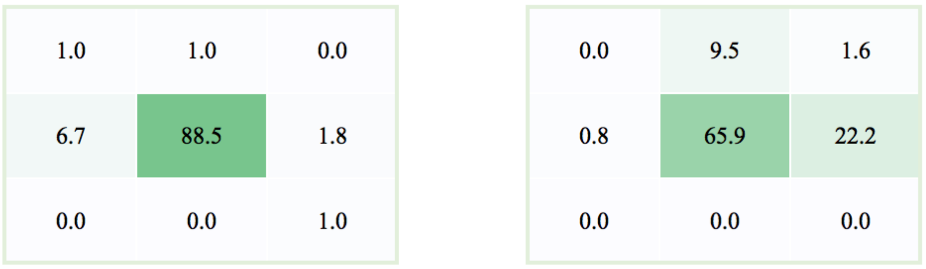
Figure 5.2: Correctly detected signs after filtering.

The traffic sign recognition algorithm produces many candidates of rectangular regions in the image as seen in Figure 5.1. Some of them are traffic signs, while others are just objects that also have a rectangular form, or form rectangular shapes. Before performing further analysis we need to perform filtering of the shapes based on their position in the image, their relative position to each other, inner angles, size, and closeness to the edge. Such filtering is necessary to avoid wasting processing time on rectangular regions with no information.

The first step of the filtering is to detect and merge duplicate rectangular regions. The traffic sign finding algorithm performs the detection several times on the same image, but on each iteration the image is processed with different thresholds. This produces better results, but also a lot of duplicates. The duplicates are detected by comparing the four vertexes of the rectangular areas, and if they are close enough (0.5% of the maximum image dimension) it merges the vertexes by calculating the average position of each vertex in the detected duplicates.

Afterwards, the rectangular regions are grouped in order to form larger rectangular units. The grouping is performed when two rectangular regions share the same edge, and the vertexes are within a 0.4% of the maximum image dimension. This step is necessary because often the traffic signs are themselves composed of several rectangular regions. Thus, it is necessary to stitch together all the rectangular regions into a macro region before further processing.

After grouping the rectangles into larger regions, we detect if there are any rectangles inscribed within larger rectangular regions, and finally we remove rectangles that are too close to the edges of the image. In our samples, we found that most of the signs in the pictures taken are closer to the middle or upper-middle areas of the photo as seen in Figure 5.3.



1.0	1.0	0.0
6.7	88.5	1.8
0.0	0.0	1.0

(a) Arterial road.

0.0	9.5	1.6
0.8	65.9	22.2
0.0	0.0	0.0

(b) Freeway.

Figure 5.3: Percentage (%) of locations of traffic signs on the image files (3×3 grid).

The end result is the correct detection of the rectangular images that contain traffic signs as seen in Figure 5.2. Now that the rectangles have been filtered and the false positives have been eliminated, we can perform further computations on the information rich rectangular regions.

5.3.3 Optical character recognition

Directional traffic signs have text indicating the destination of the road or highway. Since modern phone's cameras produce high quality images, it becomes possible to process the images to extract the text in them. Still, the biggest challenge we faced in detecting the text in the images is the varying lighting conditions, shadows, and camera image quality. These challenging conditions make traditional approaches to applying filters before performing optical character recognition (OCR) perform poorly. Most of the techniques have been developed to detect text in books or scanned media. Therefore, before performing OCR, we apply a custom process that binarizes the images and segments their irregular layouts so they can be processed by the OCR.

```
Data: Input image
Result: Noise reduced, binarized image
(1) Apply perspective-correction to the image (Figure 5.4).
(2) Rescale the image down. The optimal resolution for OCR is
300dpi.[36]:
(3) Split the input image into 3 separate channels, one for each colour
channel which is treated as a grayscale image.
    A. Apply Canny edge detection [25] to the 3 grayscale images.
    B. Recombine the images. (Figure 5.5)
(4) Perform Sobel edge detection [37] on the color image and equalize
the histogram (Figure 5.6).
(5) Remove Canny edges that are below certain threshold value in the
Sobel filtered image.
(6) Perform 8-connected component analysis on the remaining edges
to form connected contours.
(7) Filter and group contours based on their position, size, and
properties shared with neighbouring contours (Figure 5.7).
```

Algorithm 2: Text pre-processing algorithm. Note that the thresholds are empirical values from our experiments.

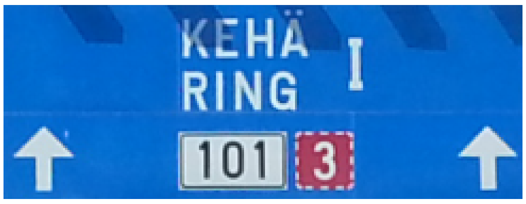


Figure 5.4: Perspective corrected image.

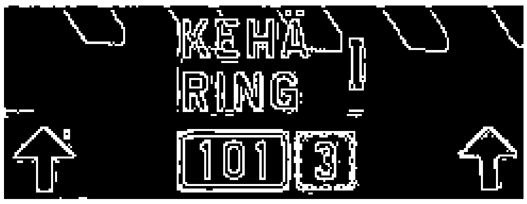


Figure 5.5: Canny edge detection.

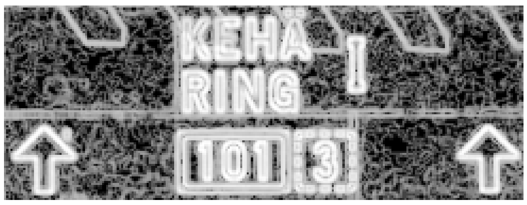


Figure 5.6: Sobel edge detection.

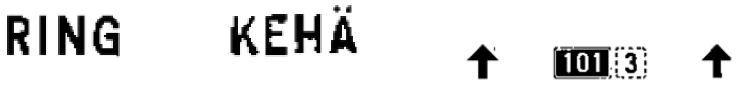


Figure 5.7: Resulting segmented text.

After applying the algorithm the image is fed to an OCR library. We chose Tesseract OCR [38] by Google because it is open source, freely available, and it is used widely in academia. Tesseract OCR is one of the top performers at the Fourth Annual OCR Competition held in 1995 [39]. After the text detection by Tesseract, the detected text and the context information is matched against the information in the database as described in the following.

5.3.4 Database implementation

The first step to implement the database matching algorithm is to design a database that contains the context information and also contains the possible locations in a way that is easy to search for the right one. The database consists of 4 tables: *signs*, *locations*, *connections*, and *cells* as shown in Figure 5.8

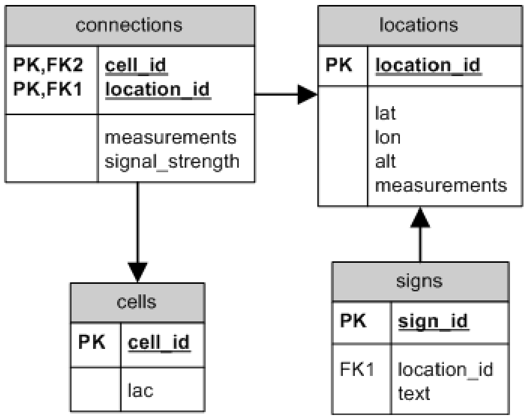


Figure 5.8: Database design.

The *locations* table has the coordinates and altitude of the places where directional traffic signs are located. The *cells* table has the information about the base station identifiers and their location area code (LAC), which is a regional identifier. The *signs* table contains the information about the text found in the traffic signs. Finally, the *connections* table has the information for the signal strength between a mobile device and a eNodeB. Therefore, the *connection* table creates a many-to-many relationship between a *location* and a cellID or eNodeB.

The database is populated by measurements of the context information at each of the locations, and the information of the text in the traffic signs. The database is meant to be built up by crowd-sourcing the measurements at each location. This is why in the *connections* and *locations* tables have a *measurements* counter. We use this counter to keep a running average of the signal strength and location coordinates of the traffic signs when we update the database with new data.

When using the database, the first step is to query the *connections* table with the cellID of the mobile device connected to it. This way, we get the candidate locations covered by a specific cellID. After getting the connections and the associated locations, the next step is to examine the signal strength from the context information and compare it to the ones in each connection. We assign a score to each candidate *location* based on how close the signal strength in the database is to the one from the query. The score is the difference between the signal strength values, divided by the sum of all the differences between the candidate's signal strength and the query signal strength. The scores of all the candidates add up to 1. The process is summarized in Algorithm 3.

```
Data: Context information (cellID, signal strength)
Result: Set of candidate locations
(1) Query the database for the connections involving the cellID.
(2) For each connection, extract the associated location and add it to
the candidate set.
(3) Score each location by calculating the difference between the
queried signal strength, and the connection signal strength.
(4) Normalize the scores of all the locations so they all up to 1.
```

Algorithm 3: Signal strength-based query algorithm.

The second step is to perform the matching with the detected text. For this, we first obtain the expected text from the candidate locations by following the relationship between the *locations* and *signs* tables. We then match each sign on an individual level in the database. The text matching algorithm is described in Algorithm 4.

```
Data: Detected text.
Result: Scores for the candidate locations based on the text
        matching.
(1) Retrieve the text from each sign in associated with each location
candidate.
(2) Use the retrieved text to compare it against the detected text in
each of the detected signs and use the Levenshtein distance to
calculate a score.
(3) Choose the highest scoring sign detected sign-database sign
combinations.
(4) Assign scores to the locations based on the scores of each of the
highest scoring sign combinations.
```

Algorithm 4: Text matching algorithm.

The database is designed in such a way that it can be used either on the mobile device or on the RACS server. The version of the database in the RACS server would only have one entry in the cells table, and few ones for the connections and locations tables. In our implementation, we chose PostgreSQL as the back-end database, and SQLAlchemy as the object-relational mapping (ORM) [40] we use to query the database from the Python Flask server. Choosing SQLAlchemy as the ORM gives us the flexibility to use a SQLite database on the mobile devices and for testing, and a PostgreSQL database in the production environment without any changes to the code since SQLAlchemy abstracts the interaction with the database.

The final score of each candidate is the sum of the scores of the text matching, and the context information matching, the signal strength. Using this score, the system is able to pick the appropriate location and then craft the HTTP REST response to send it to the mobile device.

Chapter 6

Evaluation

After re-implementing the system while taking into account the benefits the MEC server provides, we set out to test measure quantitatively how much of an impact it had. The objective of this Thesis is to improve the performance of the application by leveraging a MEC server. The application's goal is to provide the user with a quick response to a query based on context information such as signal strength, and an image of directional traffic signs. The response must be a unique location of where the user captured the photo. Based on this objective, we measure the accuracy of the system and the algorithms in identifying the correct location, and the total time the system takes to respond to a query in each of the 3 implementations: MEC, MCC, and on the mobile phone.

In this chapter, the first section introduces the new data sets collected. Next the second section introduces the experiments and results performed to compare the accuracy of the new algorithms and the system created for their use in the MEC environment against the original ones developed for the mobile only application. Finally, the last section deals with the system response time experiments and measurements.

6.1 Testing data set

The two data sets consist of images and context information recorded while driving from Otaniemi to Myllypuro on a 28.4 km section of the Kehä I freeway. Figure 6.1 shows the location of the images. The both sets include 74 matching images of traffic sign locations and the corresponding context information of each taken on different days. The fist test set was collected on the 12th of May, 2015 and the second set on the 15th of May, 2015. The weather conditions on each of the days were opposite. The first day was overcast and there were moments of light rain. The weather conditions improved on the second day. It was a sunny day with a clear sky. Each of the days present unique challenges for the detection of the traffic signs.

Name	**Date**	**Locations**	**Weather**
Test set 1	May 12, 2015	82	Overcast, light rain
Test set 2	May 15, 2015	76	Sunny

Table 6.1: Testing data sets.

In addition to the image data sets, we have recorded the context information at each of the locations 21 times on different days and with 4 different mobile devices: a Google Nexus 5, a Samsung S3, and two Samsung S4. It became necessary to collect 21 measurements of the context information at each of the points because in each measurement we found that the mobile phones connected to new base stations in each measurement, and it wasn't until we had 20 measurements that we stopped finding new base stations.

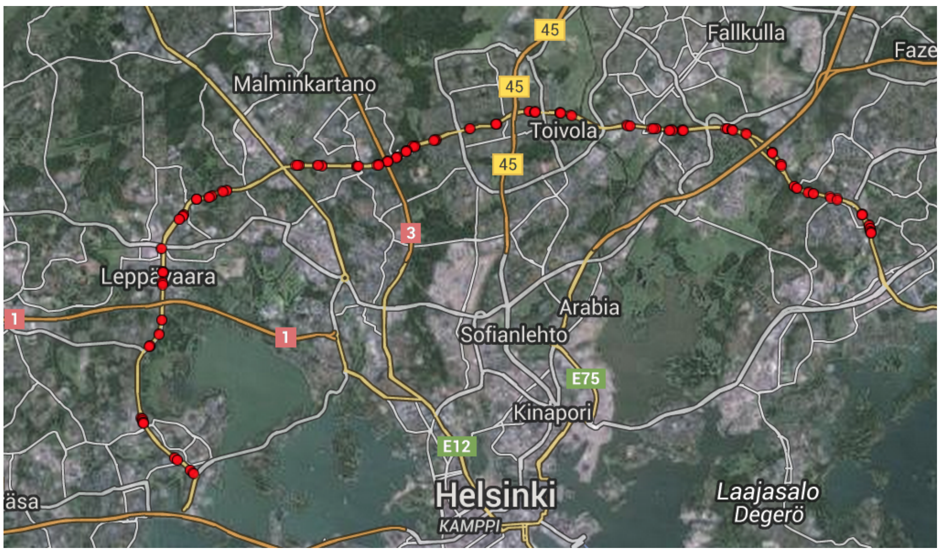


Figure 6.1: Map showing the location of the traffic sign sets.

6.2 Detection accuracy

In this section we analyze the impact of optimizing the image sizes on the sign detection accuracy, and how this in turn affects the OCR detection accuracy.

We also compare the improved candidate selection algorithm which only relies on the signal strength and cellID information.

6.2.1 Sign detection accuracy

In this experiment we compare how the sign detection accuracy is affected by the compression and re-sizing required, which we call image optimization, before transmitting the image over the network. We have manually identified the traffic signs on both test sets of images. Using the manual identification data, we can measure the detection accuracy of the sign detection implementation when processing the optimized images by comparing the positions of the vertexes of the rectangular regions that the sign detection produces. For this purpose we process the images in both sets in their original resolution of 8MP and 96 JPEG compression quality, and then compare the results to the results of optimization of the images with 22 JPEG compression quality and downscaled to 7MP resolution. The results of the percentage of the signs detected on both sets are summarized in Figure 6.2.

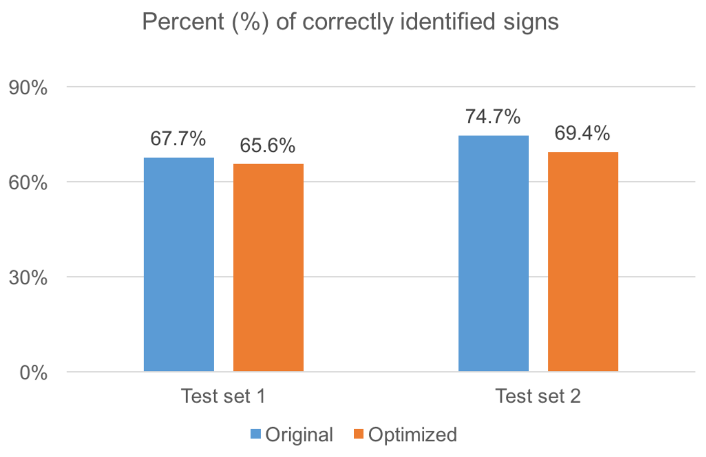


Figure 6.2: Sign detection accuracy of the image sets.

From Figure 6.2 we can notice the impact of optimization on the images on the sign detection algorithm. Interestingly, the algorithm performs better than expected on the new image data sets. When analyzing the detection data closely, we can notice that, as the figure shows, the difference is not significant. However, there are slight differences in the sign detection in 13 images from test set 1, and in 14 images from test set 2.

Image file	**Optimized**	**Original**
20150512_123856.jpg	1/1	0/1
20150512_124851.jpg	3/3	1/3
20150512_125342.jpg	2/4	1/4
20150512_131628.jpg	1/3	0/3
20150512_124128.jpg	3/4	4/4
20150512_124508.jpg	0/1	1/1
20150512_124822.jpg	1/3	2/3
20150512_124912.jpg	2/4	3/4
20150512_125215.jpg	0/2	1/2
20150512_125328.jpg	2/3	3/3
20150512_125749.jpg	2/3	3/3
20150512_130259.jpg	0/1	1/1
20150512_131452.jpg	2/3	3/3

Table 6.2: Sign detection differences in test set 1.

First we are going to look at the results from test set 1. Table 6.2 shows the differences in the detection. In 9 of the optimized images the sign detection algorithm misses one traffic sign that was previously detected in the original set. This produces the sign detection algorithm to fail completely in 3 images where it has previously only detected one sign. Interestingly, in the remaining 4 images the sign detection algorithm performs better. As a result, the sign detection is able to detect signs in 2 images where it had not been able to in the original set.

The results of test set 2 are consistent with the ones in test set 1, but the optimization has a stronger negative effect. Table 6.3 shows the differences in the detection. In the test set 2, the optimization affects negatively 11 images. It makes the sign detection fail completely in 4 because only one traffic sign had been detected before. However, the detection is boosted in 3 optimized images. Interestingly, in 2 of them no signs had been detected before. As a result, the net difference of complete detection failures is only 2 images.

Our approach at comparing the detected signs by matching the vertexes overlooks the fact that sometimes only portions of the signs can be detected. For example, if in an image there is only one road sign and only the bottom section is detected, the comparison will be negative. Still, the partial rectangular region may contain enough information that can be extracted with OCR.

Image file	Optimized	Original
20150519_105917.jpg	2/4	1/4
20150519_111155.jpg	1/1	0/1
20150519_112749.jpg	1/2	0/2
20150519_105156.jpg	3/4	4/4
20150519_105652.jpg	0/1	1/1
20150519_105704.jpg	1/3	3/3
20150519_105729.jpg	1/2	2/2
20150519_105748.jpg	1/2	2/2
20150519_105804.jpg	0/2	1/2
20150519_105829.jpg	2/3	3/3
20150519_105856.jpg	2/3	3/3
20150519_110301.jpg	0/2	1/2
20150519_110326.jpg	0/3	1/3
20150519_110744.jpg	2/3	3/3

Table 6.3: Sign detection differences in test set 2.

6.2.2 Candidate selection

The purpose of the following experiments is to measure how accurately the system, and each of the contributing components can determine the right location from the subset of locations at a specific cell. The accuracy is measured by the percent of locations within 50 meters of the expected corresponding location. To measure the distance between two coordinate sets, we use Vincenty's formula [41] which takes into account Earth's curvature.

We perform three experiments. First we test the system with only the context information and compare the accuracy of the two algorithms developed for this purpose. Next, we test the accuracy of the OCR text matching applied to to the candidate selection. Finally we combine both approaches and test the overall accuracy.

6.2.2.1 Context information based candidate selection

In this experiment, we utilize only the context information available to us to choose the correct location. We compare the original algorithm (Algorithm 1) and and the one proposed in this Thesis (Algorithm 3).

Figure 6.3 summarizes the results by presenting a comparison of the percentage of the locations identified correctly by both algorithms in both data sets.

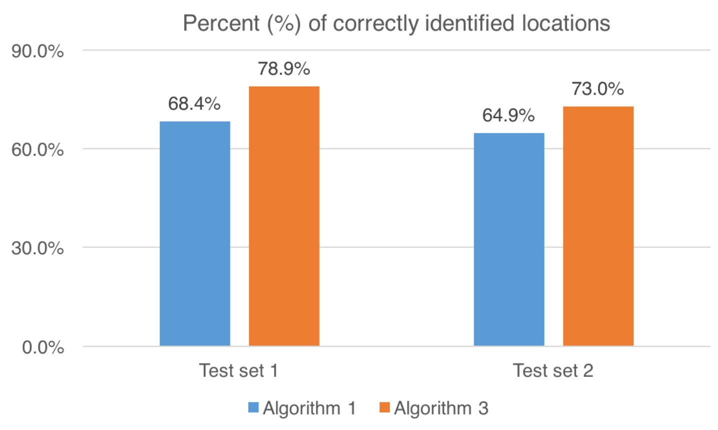


Figure 6.3: Location accuracy of both algorithms.

Based on the results, we can readily observe that Algorithm 3 performs better for this purpose. The main cause for a difference in the accuracy of the algorithms is due to the altitude measurements. When we look closely at the data of the altitude measurements, we notice that it varies within a range of 50 to 100 meters. Having taken the measurements in a portion of Finland, it is not surprising that most locations are within 10 meters elevation of each other. This makes the altitude data useless for our purpose, and further, it has a negative impact on the accuracy of the algorithm. Possibly the altitude information can be useful in mountainous places, but definitely useless in most of Finland.

The variability of the signal measurements by different devices is also an issue that affects the accuracy. Most of the time the signal strength difference between the expected value at a location and the measured one would be within 10 units. However, sometimes we would notice that the devices would stop updating the signal strength value for several minutes. This would lead to the signal strength difference between the expected value at a location and the measured one to be more than 30 units off. Fortunately, this is not very common in our data, it only occurred in 2 out of 63 measurements, but it is still a factor that contributed to the failure rates of the accurate detection of the locations.

6.2.2.2 OCR based candidate selection

In this experiment we use only the OCR information together with the cellID to select the best location candidate according to Algorithm 4. In addition, we test the algorithm on the original images as well as the compressed and rescaled ones.

Figure 6.4 summarizes the results by presenting a comparison of the percentage of the locations identified correctly by both algorithms in both data sets.

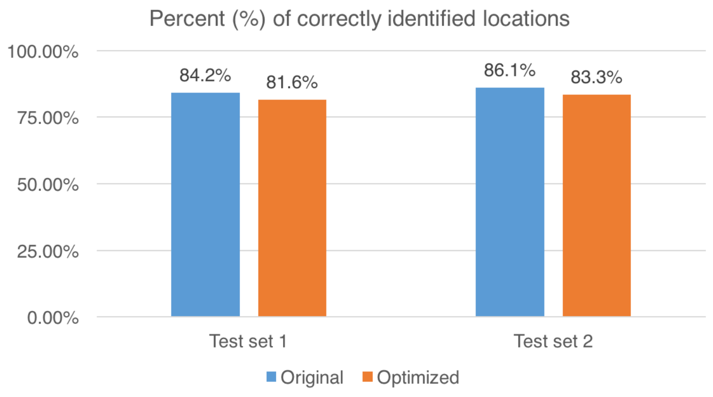


Figure 6.4: Location accuracy of text matching in both data sets.

6.2.2.3 OCR and context information candidate selection

Now we combine the two previous methods of choosing the location candidates, by text matching from the text detected with OCR together with the signal strength information. In this experiment we find out that combining both approaches makes them perform better and equally despite the image optimization.

Figure 6.5 summarizes the results by presenting a comparison of the percentage of the locations identified correctly by both algorithms in both data sets.

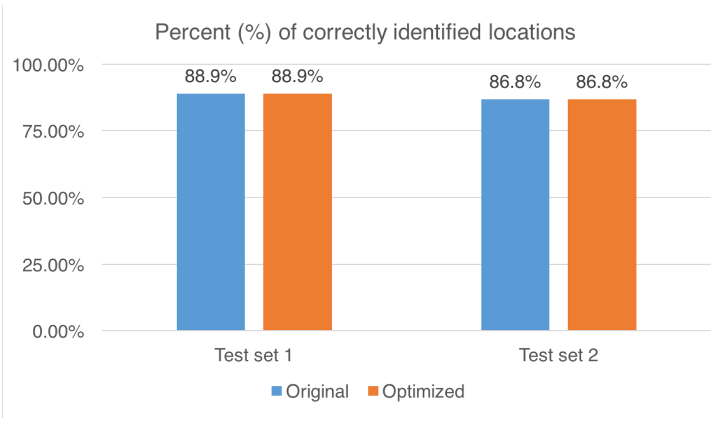


Figure 6.5: Location accuracy of text and context information matching in both data sets.

The results of this experiment are very interesting due to the fact that apparently the image optimization did not play a big role in the accuracy when the context information is used. Sign detection failed in most of the incorrectly chosen candidate locations. However, in some of them the sign detection worked perfectly and so did the OCR. The OCR was able to narrow down about 6 candidates to only 2. These 2 candidates did include the correct one, but the context information score made the algorithm select the wrong one.

The most common failure case was when neither the OCR or the context information scores were the same and made the algorithm select the first candidate, which often was the incorrect one. In one case, the OCR scored an incorrect candidate slightly higher, but then the context information score was much higher and ended up in the selection of the correct candidate. In most of the cases each score complemented the other and that is what is reflected on the results.

Two strategies remain to further improve the accuracy. One is to improve the sign detection algorithm, and the other one is to find other features in the image that can be used to identify the correct location. These features could be relative sign position and size for example.

6.3 Transmission and processing time

Implementing the MEC-based solution means that we can leverage code-offloading to process the images faster, but we have to balance it with the fact that the images still have to be transmitted over the network. In the analysis and design, we focused on reducing the image file size by using compression and rescaling. In this section we test how long the optimized images take to transmit over the network.

6.3.1 Network transmission delay

We use the LTE Netleap network in Aalto together with the RACS servers provided by Nokia Networks. We deploy the server application on the RACS server and use a mobile phone to send multiple images and measure the average time to send them. We perform the same measurements over a Elisa LTE network, but this time we deploy the application in Amazon Elastic Computing Cloud (EC2). In this way we are able to compare the network performance in the MEC and MCC scenarios.

In this experiment, we transmit the rescaled and compressed images from both data sets, and calculate the average time it takes to transmit them to the MEC server and to the MCC server. We measured the latency conditions prior to the test, and found that the latency to the RACS was 14 ms and the latency to the EC2 server was 79 ms on average.

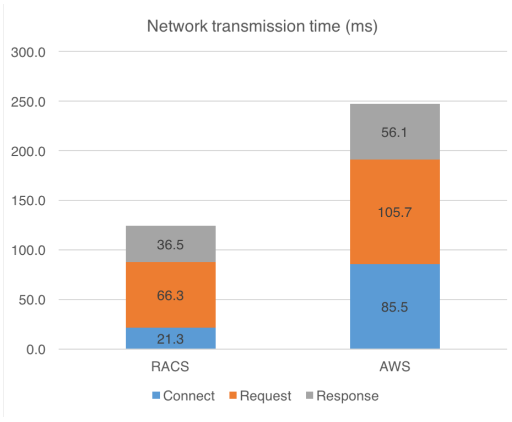


Figure 6.6: Comparison transmission time.

In figure 6.6 we can observe the advantage of sending the information to a server one hop away from the mobile device. Note that in the early measurements we performed during the analysis, the latency to the EC2 server was lower. We carried out the test in the same physical location while connected to the NetLeap LTE network. The application was able to transmit the files in less than 250ms total to the AWS and in less than 150ms to the RACS server. In augmented reality applications a user can notice 100ms difference. The upper latency bound for augmented reality applications is 500ms. The additional 100ms provided by the RACS server can mean an extra feature that cannot be available on the AWS server.

6.3.2 Processing delay

We also execute the application similarly to how we first did it for the analysis. We deploy the application in the RACS and test the time it takes to process the images. Similarly we test how long it takes to process the images on a mobile phone. In our testing we use a Galaxy S4+ mobile phone running the Android OS. We focus the tests in the sign detection and OCR processing since they are the most resource consuming and make up most of the processing time. Figure 6.7 summarizes the results.

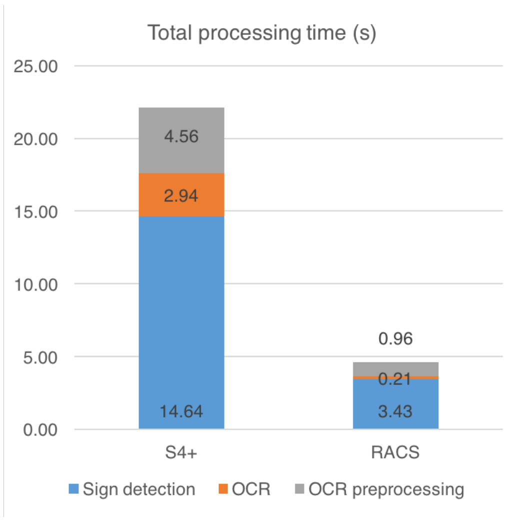


Figure 6.7: Comparison of processing times.

In these results we didn't expect the OCR pre-processing to consume so much of the processing time. The pre-processing is necessary to have a good accuracy when detecting the text, and without it the text detection performs poorly. In total, the execution of the whole process takes 22.14 seconds on the mobile phone and 4.82 seconds on the RACS server. Even though the RACS server provides a considerably faster response time, the application is still too slow to respond to be considered real-time. The most time consuming operation is the sign detection. The sign detection has to be performed most of the time, except for when the CellID filtering produces only one candidate which occurs about 4% of the time.

6.3.3 Overall delay

In this section we combine both processing and network delays to give an overall image of the system performance. Table 6.4 summarizes previous experiments and presents the total execution time for a query on average.

	RACS	**AWS**	**Mobile**
Sign recognition	3.4	3.4	14.6
OCR	0.2	0.2	2.9
OCR pre-processing	1.2	1.2	4.6
Network	0.1	0.2	0.0
Total	4.9	5.1	22.1

Table 6.4: Basic location database

The lowest response time is with the RACS server since it has the advantage thanks to the network transmission time. The processing time is the same on either the RACS or AWS server. In practice, the performance of the AWS server could even be better thanks to the elasticity of the cloud. However, an AWS server would have to serve a multitude of users at the same time, and this extra capacity would become irrelevant. On the other hand, the RACS server is intended to serve very few users at a time. Only those that are connected to the current cellID will make use of it services so the processing capacity won't need to scale as with the AWS server.

The mobile phone by itself performs the worst because the algorithms are too demanding for its hardware. The traffic sign processing and the OCR become too heavy for the phone's limited processing power, which ends up taking more than 4 times as much time on average to process an image.

Chapter 7

Discussion

This Thesis starts out with the objective to evaluate the benefits of MEC in base stations. In particular it focuses on the HoPS application that is under development by a research group at Aalto. The application takes context information captured by the user, together with a photograph of a sign post and using this data it identifies the location of the user by querying a database. The first step is to identify what services and benefits MEC can provide to applications, and then, with this knowledge, find ways to efficiently leverage MEC to improve the application.

The biggest challenges in improving the results further lie in the sign detection algorithm. The sign detection algorithm is a computer vision problem, and due to its complexity it can be a thesis work in itself. Improving the sign detection algorithm was out of the scope of this thesis. Any changes to decrease the processing time and increase the accuracy of it would directly impact and improve on the text matching and OCR of the traffic signs.

7.1 Code offloading

The Thesis identifies several strategies to improve the performance. The first and most obvious strategy is processing offloading. The Thesis presents a benchmark of the current state of the application. Based on this benchmark it compares the advantages of offloading the image processing and also takes into account the drawbacks of such an approach.

Among the drawbacks, the most prominent is the network processing delay. It soon becomes apparent that transmitting the full images to the server is not feasible. The images are commonly more than 2.5 MB in size, which translates to 1-2 seconds of delay even with the improved network conditions. Consequently, this work proposes to compress and reduce the size of the images. In order to find out the optimal parameters for compression and image size, we perform extensive tests on the impact of such transformations on the accuracy of the sign finding algorithm and the image size reduction. The analysis of the impact concludes that to be able to transmit the images to the server and minimize the delay, a small portion of the accuracy must be sacrificed. This leads to the selection of two methods to reduce the image size: JPEG compression with a quality factor of 22 and re-scaling the images to 7MP.

JPEG compression visually distorts the image, but it did not seem to affect the accuracy of the sign finding algorithm. This permitted us to set the JPEG compression to 22. On the other hand, reducing the size of the image had an immediate and direct effect on the accuracy. Therefore, the analysis concluded that the image size can only be marginally reduced from the original 8MP to 7MP, while keeping the accuracy penalty low.

Additionally, the Thesis proposes to improve the accuracy of the location scoring algorithm by leveraging the increased processing power to introduce OCR as a tool to extract more information from the text in the images. The reasoning is that while OCR is a processing intensive task, its use will be limited to small regions of the image that contain the traffic signs, and the traffic signs themselves have little text in them. Even though OCR did improve the accuracy and performed better than previous methods, the processing cost was higher than expected. Successfully extracting text via OCR required costly and complex image pre-processing that turned out to be more demanding than the OCR itself.

7.2 Database distribution

A second strategy to improve the performance is to partition and distribute the database. This Thesis proposes keeping a local small database on each of the MEC servers. This allows for an easier upkeep of the database, and independence from the core network. Thus, we leverage MEC's characteristic of being on premises.

If the connection between an eNodeB and the core network were to be disrupted, then the application could still work. While, on the other hand, keeping such a database in a mobile phone would be prohibitive as the system expands and more data is added to the database. Arguably, only a geographically relevant portion of the database could reside on the mobile device. We could argue that then when the user roams in another country she would need to update his application and download a new database. However, if MEC would become commonplace, it is still not clear how applications could be deployed and managed cross-infrastructure provider, much less cross-country borders.

7.3 Improved network conditions

The Thesis also attempted to leverage the improved network conditions, and it successfully made use of them to achieve a faster transmission of the images through the queries, and in this way reduced the response time. Nonetheless, this improvement became insignificant when compared to the processing time. The network transmission time is only a small fraction of the total query time, the processing time eclipsed the 75% reduction in the transmission delay that the improved network conditions provided. In order to leverage the improved network conditions, the research group must improve the application's computer vision algorithms such as traffic sign detection, and OCR pre-processing. In their current form, these algorithms are 2000% more time consuming that they should be. The 5 seconds it takes to process an image should be reduced to no more than 300ms.

Network operators see a larger benefit from the improved network conditions since the images from the application won't have to be transmitted over their back-haul links. All the data can be communicated at the edge, thus freeing the core network.

7.4 Context information availability

Finally, the last strategy this Thesis used was to leverage the context information provided by the RACS. The context information includes the cellID as well as network conditions. However, we soon noticed that the relevant network information is more easily retrieved from the mobile phone API than the RACS server's API since it is not completely implemented yet, and was only available to Nokia's applications.

7.5 Future work

Future work on the HoPS application should focus on improving the sign detection on the images. The current approach works, but it is highly redundant and, more importantly, time consuming. We recommend focusing on using a feature detection algorithm instead. These tend to be more reliable and resistance to noise, occlusion, and varying light conditions. In addition, feature detection algorithms tend to require less processing power compared to the sign detection algorithm. In this thesis we have analyzed the application in the context of a 4G network. However, in the future 5G will bring changes to the size of cells, overall improvements to the network, and most likely MEC will become a standard component of 5G deployments.

Chapter 8

Conclusions

MEC provides a platform to augment mobile applications in several ways. It provides services comparable to MCC like code offloading, and extended storage. In addition to these services, MEC also provides network context information, lower latency, distribution, and geolocation information. MEC is a recent development that is born from the Cloudlet idea. Since MEC is so recent, there are very few real-world implementations. One of them is Nokia Networks RACS. In this thesis we utilize Nokia Networks RACS to augment an existing HoPS application with the services provided by MEC with several strategies.

The HoPS application is an ideal candidate to test MEC because it requires heavy processing that is too time consuming on mobiles, and it also uses context information in order to estimate a the geographical location of the user. We evaluated the practical advantages of using MEC to augment this application.

We found that by leveraging MEC we improved on the performance of both the total query time, and the accuracy of the location selection. The new implementation reduced the query time by a factor of four, and raised the accuracy of the algorithm from 65% to 85%. The major contributor to the improvements was the capacity to offload the processing.

We successfully dealt with the network transmission delay by optimizing the image sizes while trading off a small amount of accuracy in the sign finding algorithm. However, the network latency improvements became insignificant to the response time when compared to the processing time. An application that is less processor intensive will benefit more from the latency improvement.

Finally, we also found that the current RACS implementation is immature since the API to query the context information is described in the manual, but it is not yet available for non-Nokia applications.

Bibliography

[1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *Communications Surveys Tutorials, IEEE*, vol. 16, pp. 369–392, First 2014.

[2] X. Fan, J. Cao, and H. Mao, "A survey of mobile cloud computing," *ZTE Communications*, vol. 9, no. 1, pp. 4–8, 2011.

[3] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81, ACM, 2014.

[4] "Yelp.com." `https://www.yelp.com`, 2015. [Online; accessed 1-July-2015].

[5] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, and A. Schneider, "Enabling real-time context-aware collaboration through 5g and mobile edge computing," in *Information Technology - New Generations (ITNG), 2015 12th International Conference on*, pp. 601–605, April 2015.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, (New York, NY, USA), pp. 68–81, ACM, 2014.

[7] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: Designing elastic android applications for computation offloading,"

[8] H. Hile, R. Vedantham, G. Cuellar, A. Liu, N. Gelfand, R. Grzeszczuk, and G. Borriello, "Landmark-based pedestrian navigation from collections of geotagged photos," in *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, MUM '08, (New York, NY, USA), pp. 145–152, ACM, 2008.

[9] Z. Ou, S. Dong, J. Dong, J. K. Nurminen, A. Ylä-Jääski, and R. Wang, "Characterize energy impact of concurrent network-intensive applications on mobile platforms," in *Proceedings of the Eighth ACM International Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '13, (New York, NY, USA), pp. 23–28, ACM, 2013.

[10] Z. Ou, J. Dong, S. Dong, J. Wu, A. Yla-Jaaski, P. Hui, R. Wang, and A. Min, "Utilize signal traces from others? a crowdsourcing perspective of energy saving in cellular data communication," *Mobile Computing, IEEE Transactions on*, vol. 14, pp. 194–207, Jan 2015.

[11] P. M. Mell and T. Grance, "SP 800-145. The NIST definition of cloud computing," tech. rep., Gaithersburg, MD, United States, 2011.

[12] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, pp. 10–17, Aug 2001.

[13] "Amazon silk browser." `http://docs.aws.amazon.com/silk/latest/developerguide/introduction.html`, 2015. [Online; accessed 4-July-2015].

[14] "Apple siri virtual personal assistant." `http://www.apple.com/ios/siri/`, 2015. [Online; accessed 23-May-2015].

[15] "Google goggles." `http://www.google.com/mobile/goggles`, 2015. [Online; accessed 15-May-2015].

[16] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, p. 6, ACM, 2010.

[17] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[18] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*, pp. 1–9, Nov 2014.

[19] "Executive Briefing - Mobile Edge Computing (MEC) Initiative," tech. rep., ETSI - European Telecommunications Standards Institute, 2015. `https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/MEC%20Executive%20Brief%20v1%2028-09-14.pdf`.

[20] M. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*, 2014.

[21] "Mobile-Edge Computing - Introductory Technical White Paper," tech. rep., ETSI - European Telecommunications Standards Institute, 2015. `http://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf`.

[22] "Nokia Networks Intelligent base stations white paper," tech. rep., Nokia Solutions and Networks Oy, 2012. `http://networks.nokia.com/sites/default/files/document/nokia_intelligent_bts_white_paper.pdf`.

[23] Z. Ou and A. Ylä-Jääski, "Towards Human-Centered Wearable Navigation System." 2015.

[24] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[25] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, pp. 679–698, Nov 1986.

[26] T. R. Singh, S. Roy, O. I. Singh, T. Sinam, and K. M. Singh, "A new local adaptive thresholding technique in binarization," *CoRR*, vol. abs/1201.5227, 2012.

[27] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985.

[28] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[29] "FMI Open data WMS services." `http://en.ilmatieteenlaitos.fi/open-data-manual-fmi-wms-services`, 2015. [Online; accessed 6-August-2015].

[30] "Open Source Computer Vision Library." `http://www.opencv.org/`, 2015. [Online; accessed 6-August-2015].

[31] "Flask Framework." `http://flask.pocoo.org/`, 2015. [Online; accessed 6-August-2015].

[32] K. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Elsevier Science, 2014.

[33] A. B. Watson, "Image compression using the discrete cosine transform," *Mathematica journal*, vol. 4, no. 1, p. 81, 1994.

[34] OpenSignal, "Barometer." `http://opensignal.com/sensors/barometer`, 2015. [Online; accessed 19-July-2015].

[35] IDC, "Smartphone OS Market Share, Q1 2015," June 2015.

[36] Z. Podobny, "Improving the quality of the output." `https://code.google.com/p/tesseract-ocr/wiki/ImproveQuality`, March 2015. [Online; accessed 19-July-2015].

[37] W. Gao, X. Zhang, L. Yang, and H. Liu, "An improved sobel edge detection," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 5, pp. 67–71, July 2010.

[38] R. Smith, "An overview of the tesseract OCR engine," in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, (Washington, DC, USA), pp. 629–633, IEEE Computer Society, 2007.

[39] S. V. Rice, F. R. Jenkins, and T. A. Nartker, "The fourth annual test of OCR accuracy," 1995.

[40] SQLAlchemy, *The Python SQL Toolkit and Object Relational Mapper*. 2015. [Online; accessed 6-August-2015].

[41] C. Karney and R. Deakin, "F.W. Bessel (1825): The calculation of longitude and latitude from geodesic measurements," *Astronomische Nachrichten*, vol. 331, no. 8, pp. 852–861, 2010.