2014•2015
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef deel 1

Performance evaluation of cloud based image processing on RACS for Augmented Reality devices

Promotor :
dr. Kris AERTS

Copromotor :
De heer Koen GILISSEN

Promotor :
Ing. TEEMU KAMARAINEN

Copromotor :
dr. ANDREY LUKYANENKO

Thomas Machiels

Eerste deel van het scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2014•2015
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterproef deel 1
Performance evaluation of cloud based image processing on RACS for Augmented Reality devices

Promotor :
dr. Kris AERTS

Copromotor :
De heer Koen GILISSEN

Promotor :
Ing. TEEMU KAMARAINEN

Copromotor :
dr. ANDREY LUKYANENKO

Thomas Machiels
Eerste deel van het scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Preface

This year I had the opportunity to go abroad to do my Thesis at the Aalto University in Helsinki, Finland. It was a very nice experience to do my Master's Thesis abroad and I met a lot of new people.

I would like to thank Vincent Boerjan with helping to me to contact different people to find a topic for my Master's Thesis. I would also like to thank my supervisor Teemu Kämäräinen and my co-promotor Andrey Lukyanenko for the good governance and open-minded attitude towards the subject with my choice. I would also like to thank my internal supervisor Kris Aerts for his help.

Contents

Tables

Figures

Abbreviations

AG	Augmented Reality
BGR	Blue Green Red
CN	Core Network
HSV	Hue Saturation Value color
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
PC	Personal Computer
QoS	Quality of service
RACS	Radio Application Cloud Server
RAN	Radio Access Network
RGB	Red Green Blue color
RGBA	Red Green Blue Alpha color
TCP	Transmission Control Protocol
VM	Virtual Machine

Abstract

In recent years, augmented reality (AR) applications have become more popular. Current applications on the market typically use an AR device to process the data. However, advanced AR applications require extensive processing power on the smartphone or Google Glass with higher power consumption. This results in unexpected draining of the battery or malfunctioning of the application. Therefore, it is advantageous to use a cloud server to accomplish the real-time image processing. Because the continuing growth of mobile traffic is overloading current cloud servers and networks there will be a high demand for moving cloud functionality to the edge. One example is the Radio Application Cloud Server (RACS), which allows high-speed data storage and processing.

This Master's Thesis studies the performance of cloud based image processing on for AR devices. Different applications are developed for a Smartphone and a Google Glass that cooperate with the RACS to process the real-time images. To determine if the cooperation with the server results in a better performance, the total processing time of the image on the device is compared with the processing time on the server including the transfer over the network

Two examples have been implemented and tested. For the smaller example the connection costs outweighed the faster processing on the RACS, but for the larger example the offloading to the RACS reduced the time with a factor of approximately 80 %. Therefore the technique can be useful when the algorithm is sufficiently complex.

Abstract

Augmented reality (AR) applicaties zijn de laatste jaren volop aan het opkomen. De applicaties die momenteel op de markt zijn verwerken de data typisch op het AR apparaat zelf. Maar geavanceerde AR applicaties zorgen voor een hoger CPU- en batterij gebruik. Dit kan resulteren in een onverwachtse daling van de batterij of veel te traag werken van de applicatie. Daarom kan het handig zijn om de real-time beeldverwerking van deze applicaties te verplaatsen naar een performante Cloud server. Omdat de huidige Cloud servers overbelast worden door een grote toename van mobiele data gaat er een grote vraag zijn om de Cloud functionaliteit te verplaatsen naar de rand van het netwerk. Een voorbeeld hiervan is de Radio Application Cloud Server (RACS).

Deze master thesis is een studie van de performance van Cloud gebaseerde beeldverwerking op de RACS voor AR apparaten. Er worden enkele test applicaties gemaakt voor een smartphone en een Google Glass die gaan samen werken met de RACS om de images te verwerken. Hierbij wordt een vergelijking gemaakt van de totale verwerktijd van de afbeelding gemeten op het toestel en de totale tijd gemeten in combinatie met de server.

Er zijn twee voorbeelden geïmplementeerd en getest. Voor het kleinere voorbeeld zal de snelle verwerking van RACS teniet gedaan worden door hoge connectiekost maar voor het grotere voorbeeld zal de verplaatsing naar de RACS de tijd doen verminderen met een factor van 80 %. Daarom zal de techniek pas toepasbaar zijn als het algoritme voldoende complex wordt.

1 Introduction

This Master's Thesis took place in the Department of Computer Science of the Aalto University in Finland. The focus of the research at this department is on advanced computational methods for modeling, analyzing and solving complex tasks in technology and science. The research aims at the development of fundamental computer science methods for the analysis of large datasets and for the modeling and design of complex software.

Ongoing research in the department focuses on Mobile Edge Computing, a new concept that introduces an evolving role for the mobile base station. Together with voice and messaging , the base station also will be capable of processing data from Liquid Applications. At the root of Liquid Applications is the Nokia Radio Applications Cloud Server (RACS). Liquid applications are applications that are running in a RACS server in the base station. These applications improve the user experience by acceleration. RACS introduces the latest cloud technology and service creation capabilities into the base station [1].

An example of these Liquid Applications is an augmented reality GPS application that can detect road signs. The images that are captured with a smartphone, tablet or Google Glass are sent to the RACS where the pictures are processed. The RACS is mainly used for caching and processing images from the application. But is image processing on the cloud server always faster than handling the images on the phone or Google glass? The research on the difference in performance between an application that processes the data on the device or on the cloud server is the main purpose of this Master's Thesis.

1.1 Problem statement

Augmented reality is a direct view of a real-world environment where elements are added on top of the real-time video on the screen [2]. Currently augmented reality applications on the market typically use the device for processing data, but this kind of applications require more and more CPU and power from the smartphone or Google Glass often resulting in unexpected draining of the battery or a malfunctioning of the application. Therefore it may be advantageous to use a cloud server to accomplish the real-time image processing. Currently there are almost no applications on the market that do this.

Ongoing research transfers the data processing to a cloud server to reduce CPU usage and power consumption. Nevertheless there is no knowledge whether this method has a better performance than handling the data on the mobile devices. This Master's Thesis compares the performance of an application that processes real-time images on the device with an application that relocates the image processing to a cloud server to determine which method has the best performance. The performance will be specifically measured for an application on an Android phone and Google Glasses.

1.2 Objectives

The creation of several applications on different devices is required to measure the difference in performance. First a cloud server application needs to be constructed to process the real-time image stream from the application on the Android phone or Google Glasses. The server needs to ensure that the images from the incoming stream are saved locally. Secondly it needs to implement color blob detection on the saved images with OpenCV to detect objects with a specific color. The color will be determined by a touch on the screen of the Android application. And finally when the server is ready with the color blob detection, it has to send the detection back to the application. The processing time to do the color blob detection will be measured both on the server and the smartphone. Both measurements will be compared to determine which of the two processed the color blob the fastest.

For the development of the different augmented reality applications we use four set-ups: Only the Android phone, Android phone merged with the cloud server, only Google Glasses and Google Glasses combined with the cloud server. Each set-up needs another Android application to accomplish the object detection and measure the performance. There are four applications developed, one for each set-up. The application for only the phone and only Google Glass must do the same color blob detection as the server. The color of the detection on the phone will be determined with a touch on the screen. For Google Glass a predefined color will be used to detect objects. When the phone or Google Glass are merged with the server, the color blob Android application uses another approach: the application streams the camera images from the device to the server and renders the received images from the server on the screen.

The execution times and energy consumption for the four scenarios are measured and compared to find out whether the applications that are merged with the server are faster and consume less energy than the applications that are only using the phone or Google Glass.

In a second phase we added a more fine granularity for sending data from the device to the cloud server. Streaming the data at different points in the application gave us a better insight of which functions of the color blob detection are best processed in the server, allowing us to create an optimal ratio between functions processed on the server and the application. The ratio can be acquired by comparing the execution times of the functions for all variants. At the optimal ratio, the application will have the best balance between energy consumption and processing time.

1.3 Structure and focus

The rest of the Thesis is structured as follows. Chapter 2 covers briefly mobile cloud computing before introducing the concept of mobile edge computing in Chapter 3. Chapter 4 covers the RACS, the mobile edge concept of Nokia. It also explains Liquid Applications and briefly covers an augmented reality Liquid application example.

Chapter 5 covers the setup of the design of the application and the server, as well as the connection between the application and the server. It also describes the used platform on which the server is built. Furthermore it also introduces OpenCV and the different object detection methods that are implemented in the application and the server. The implementation of the server and the applications are covered in detail in Chapter 6. The applications are evaluated in Chapter 7 by measuring the execution times of the different applications and streaming modes. Also the power consumption of the phone using the different applications is measured. Finally the Thesis concludes with discussion and conclusions in Chapter 8.

The main focus is on assessing the benefits gained from moving image processing to the RACS. This is done by constructing different object detection applications that will move the processing of the images to a cloud server. These applications use different modes to stream the images to the server to find the best trade-off of offloading the image processing to the server. The implementation of the cloud server and the various applications are described and the execution times and power consumption of the applications are measured.

2 Mobile Cloud Computing

Chapters 2 until 4 introduce the background concepts needed in the implementation of the server and the applications created in the cloud system. This section gives an explanation about the concept of Mobile Cloud Computing (MCC) and defines the architecture. The emerging of MCC is given to introduce Mobile Edge Computing (MEC).

2.1 Introduction

The fast growing amount of mobile applications and the emerging of cloud computing led to the introduction of Mobile Cloud Computing that has become a new technology for mobile users. In the concept of MCC the mobile devices cooperate with the Cloud to gain more advantages out of applications and to improve their functionality. This technology is emerging as one of the most important functionalities of cloud computing and the expectation is that it will ensure an expansion of the mobile ecosystems [3].

2.2 What is Mobile Cloud Computing?

MCC is a combination of cloud computing, mobile computing and the use of the wireless network to deliver very powerful processing to the mobile subscriber. It is an infrastructure that is used as data storage as well as processing of data outside the mobile device. Mobile cloud applications relocate the processing power and data storage away from the device into the cloud allowing applications to have access to data that is not fixed on the device. This cooperation with the cloud makes it possible to run the same application from the smartphone on other mobile devices like tablets and laptops. The main goal of mobile cloud computing is to move heavy and slow data processing of applications to the cloud, so less powerful devices are also capable of running applications that they could not run before. It can also be an advantage for the powerful smartphones on the market, because even these devices can reach their limits in energy consumption and processing power. Even more extensive is that mobile cloud computing is a high performance cloud technology that implements the flexible resources from the cloud and network technology for great functionality. Moreover the data from the cloud will also be provided at any time and at any place across the internet [3].

2.3 Advantages

Data storage and processing power:

Data storage is limited for mobile users because of the limited internal data storage of the mobile devices. Mobile Cloud Computing is developed to process high capacities of data and saving them in the cloud over wireless networks. Cloud storage provides data storage and energy saving of the mobile device since saved data can be processed on the Cloud instead of the device.

Reliability:

A major advantage of using Mobile Cloud Computing is that the mobile subscriber can access his data everywhere in the world. The only requirement is a wireless connection between the device and the internet. The data is not only available on mobile devices but also on any other device that is connected to the internet and is capable of requesting and processing the data from the Cloud.

Scalability:
A big advantage for developers is the scalability of the server infrastructure. When usage increases server capacity will be automatically increased, often on a pay per use basis. And when less capacity is needed, servers scale down smoothly as well. The responsibility for scaling is moved from developer to infrastructure provider.

Real time data:
Mobile Cloud Computing provides access to real-time data where and whenever you want. Requesting and processing of data on the Cloud is real-time and multiple users can access the same data at the same time.

2.4 Disadvantages

Security:
One of the major disadvantages of Mobile Cloud Computing is the data security. Mobile subscribers are sending sensitive data over the internet to the cloud. Therefore it will be very important that the cloud has good data security for the users. When there is no data security or when there is a security leak, sensitive information of the users can be leaked to third parties that are not allowed to get this information.

Connectivity:
Another concern is the stability of the internet connection of Mobile Cloud Computing. A stable internet connection between the Cloud and the device is necessary for the usage of services from the Cloud. With a wired internet connection where there is a physical link between the client and the service, a stable connection and a fixed network bandwidth is easily established. This is not the case with a wireless connection, which has the additional complexity that the network capacity must be shared by many users. Besides, the stability of the connection will be very important to reduce the latency with the Cloud. When there is a poor internet connection between the cloud and the device, it will be very difficult to request data from the cloud or it will happen with high latency. So a good connection is crucial to acquire an optimal use of mobile cloud computing.

2.5 Architecture

The general architecture of Mobile Cloud Computing is shown in Figure 1. Mobile devices are connected to the mobile networks via base stations. Examples of base stations are Access Points and base transceiver stations. The base station controls and establishes the connections between the networks and the mobile devices. Than requests and information from the mobile user are transmitted to the central processors that are connected to servers providing mobile network services. Thereafter, the user's requests are delivered to the cloud through the Internet. Once in the cloud, the cloud will process the requests to provide mobile subscribers with the matching cloud services. These servers are developed with virtualization and service oriented architecture [4].

2.6 Rise of Mobile Cloud Computing

The improvements of mobile technologies ensure a higher amount of connected devices to the mobile networks. Most applications on the devices will be dependent on real-time data from the Cloud. Moreover, those devices and applications are requesting more and more data from the mobile network increasing the load of the network. And not only smartphones will cause the increasing load but other electronic devices like laptops and tablets will take advantage of the mobile network.

The servers and data storage locations that process and store data are typically situated in cloud-based data centers far away from the mobile network. These servers are constructed flexible so they can grow along with the amount of users of a cloud server. But those servers are not made to handle an increase of the load of the network. When the data load rises the server may become a bottleneck and the communication with the users device will be disturbed.

Relocating the datacenters to the edge of the network will provide a unique solution for the problem of servers that are located in normal datacenters. So instead of increasing the capacity of the servers in the centralized datacenters, there will be a transformation to the edge of the mobile network. This will give a better solution in terms of computational power and data storage. Furthermore the delay between the server and the users device will also be reduced. This relocation to the edge will meet to the needs of the communication industry for creating a robust network. Which means that the network will be resistant to errors that can occur in the network. Networks can even be capable of solving these problems [5].

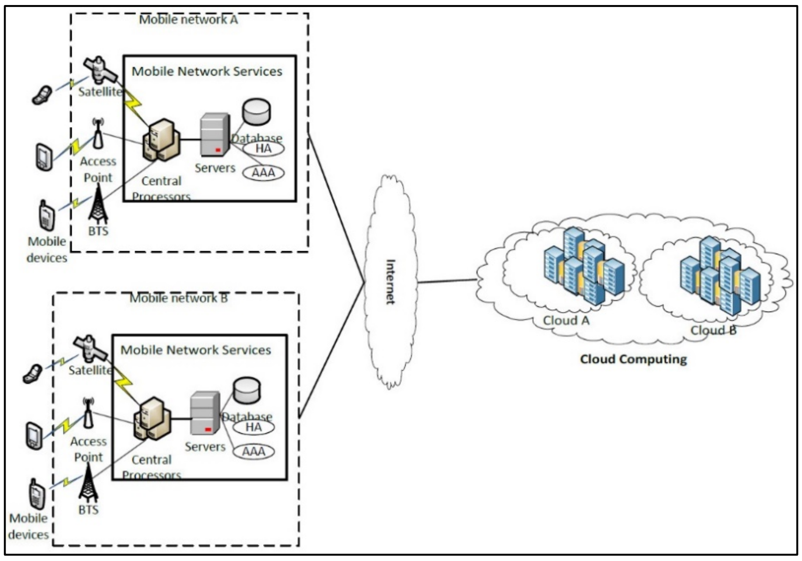


Figure 1. Mobile Cloud Computing architecture [3]

3 Mobile Edge Computing

3.1 Introduction

The fast increase of mobile video streaming, messaging and peer-to-peer applications cause a huge growth in mobile data traffic. This requires telecom companies to increase the capacity of the network to meet to the demand of mobile users. A solution to lower the mobile data traffic is to deliver the content faster to the subscribers. An example is Mobile Edge Computing (MEC), which moves the data storage and processing to the edge of the mobile network. Data and services can be provided to the mobile user to increase the responsiveness and speed up at the edge of the network and thus increase the user's experience [6].

Unlike Mobile Cloud Computing where the storage and processing of the data occurs in centralized servers at the core of the network, Mobile Edge Computing transfers processing and data storage to the Mobile Network Services at the edge of the mobile network. This allows fast delivery of services to the mobile user. The Characterization of MEC is listed below.

3.2 Characterization

Localization:

The edge is local, meaning that it can run isolated from the rest of the network while having access to local resources. This becomes important when networks needs to provide high data security. This will reducing data leakage by separating data from the rest of the network.

Proximity:

Being close to the source of information has several advantages. Mobile Edge Computing can be used to enable analyses, process and store big and complex data packets. The mobile providers can also encounter an advantage of implementing mobile edge computing in their network. This enables the development of specific applications that have direct access to the mobile edge. An example application can for example use the location awareness information to show specific information of the location to the mobile user.

Lower latency:

The short distance between the mobile edge and the mobile user reduces the latency. This can be used to improve the speed and user experience of applications. Furthermore, the reduction of the latency will also reduce the congestion with the rest of the network.

Location awareness:

The location of the base can be used to provide users with specific information. This positioning provides an innovative way for applications that can be developed to use the location of the base station. An example application can for example gather specific weather information from the base station and provide to the user of the application.

3.3 Evolution of the base station

In the past, the edge of the mobile network was only used by specialists that had knowledge about the network. The architecture of the edge was designed to deliver a good performance to some specific applications. This architecture and the connection with the core of the network where not developed to adjust. They used very specific protocols that date back to before the time that Internet Protocol (IP) became the standard for network communications. The quality of voice conversations was the main priority, not data transfer. With the arrival of IP a lot of changes were made to implement package based data transfer. This transformation to the implementation of IP enabled the development of new applications that can use the improvements of the telecom network.

The introduction of relocating the data processing to the edge of the network allows the execution of flexible services on strategic locations in the network, making them much more important and faster than applications or services that are running in the core of the network. These services can improve the QoS of the subscriber by reducing the latency of the network and providing location awareness applications to the user. An example of a service can be based on the location of the user where specific travel data in stations or airports can be provided to the user to guide him to the correct platform or gate.

The next three changes have improved the base station and the architecture of the mobile network [7]:

- Circuit switched connectivity services are replaced with packet switched services. As a result, in the future the legacy voice services will be replaced by voice-over-IP services;
- The mobile networks will evolve to a flat architecture, this allows more favorable economies in terms of capacity for the infinite growth of the data traffic;
- Cloud computing will also be introduced into the evolution of the network architecture, where it is expected that the control plane will be managed by sophisticated software that will run on very powerful servers what is known as virtualization. And the user plane will distribute the data by optimized routing platforms.

3.4 MEC server platform and architecture

Mobile edge computing provides a highly distributed environment that can be used to run applications and services. But it can also be used to process and store data in close proximity to the mobile user. These applications can access real-time network information and provide a personalized experience for the mobile user. Not only the QoS for the user will be improved but it enables also new monetizing opportunities which includes the development of new applications and services on the base station on the edge of the network.

An important element of mobile edge computing is the MEC IT application server that is part of the Radio Access Network (RAN). The mobile edge is located in the RAN because it is located between the mobile device and the core of the network [8]. The MEC server provides computing power, storage capacity, connectivity and access to network information. Moreover, the architecture contains components and functional elements that allow innovation and expansion of the servers. Furthermore, the servers need to expand to follow the fast growth in the data communication.

The MEC platform allows hosting of applications from mobile providers and consists of an application virtualization manager and application platform services. The virtualization manager supports a flexible, efficient run-time and hosting environment for the applications. This is enabled by the Infrastructure as a Service (IaaS) facilities. These facilities provides resources and security to the applications and the platform. Furthermore, virtual applications run on top of the IaaS layer and allows great flexibility to the implementation of applications. IaaS is a kind of cloud computing where the infrastructure of the cloud is provided virtually. The service providers contain the hardware like the server of the network infrastructure and the workstations. Moreover, the user of a server on the IaaS will only need to pay for part of the infrastructure that he is going to use. In the case of MEC virtual machines are provided as IaaS that can be used run an applications or save data [9].

The MEC hosting infrastructure that includes the connectivity to the RAN is beyond the scope of this Thesis and will not be discussed.

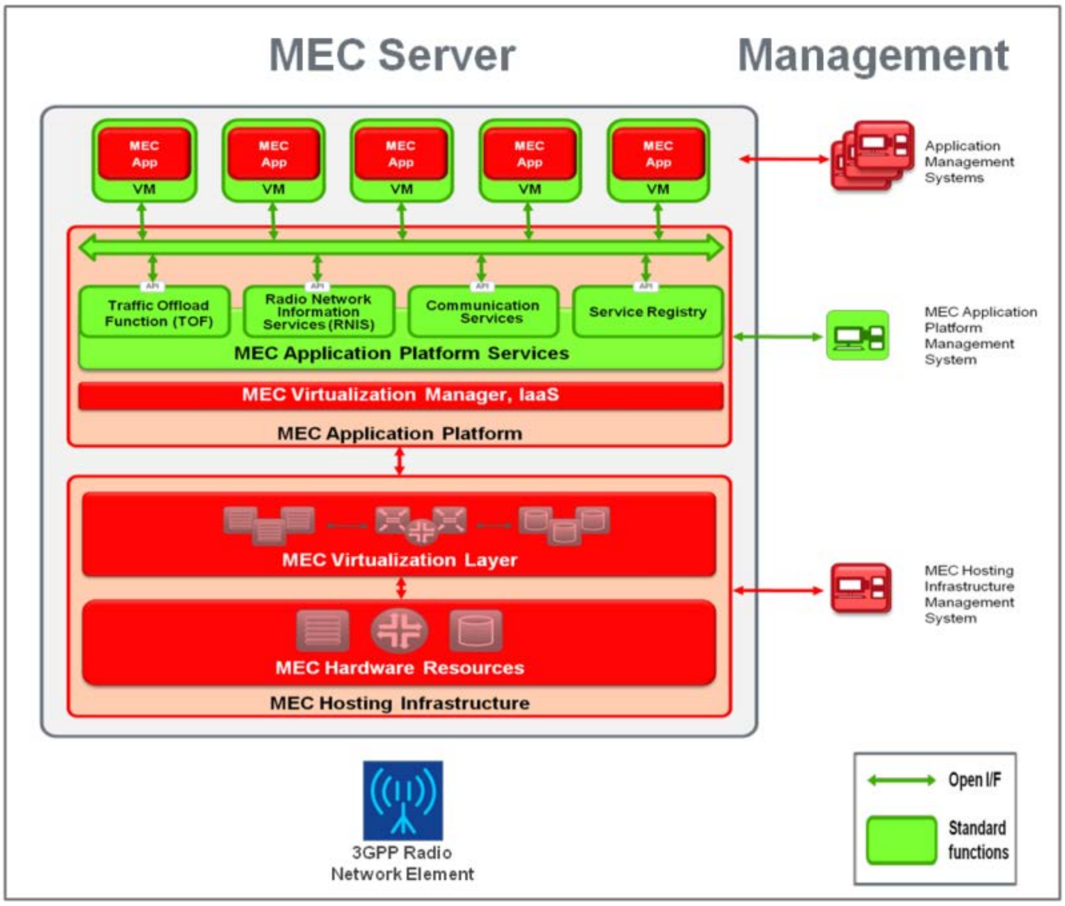

MEC Server
Management
MEC App VM
MEC App VM
MEC App VM
MEC App VM
MEC App VM
Traffic Offload Function (TOF)
Radio Network Information Services (RNIS)
Communication Services
Service Registry
MEC Application Platform Services
MEC Virtualization Manager, IaaS
MEC Application Platform
MEC Virtualization Layer
MEC Hardware Resources
MEC Hosting Infrastructure
Application Management Systems
MEC Application Platform Management System
MEC Hosting Infrastructure Management System
3GPP Radio Network Element
Open I/F
Standard functions

Figure 2. MEC server platform[6]

4 RACS

4.1 Introduction

The base station of mobile networks has not changed much in recent years. They were created in the Radio Access Network and were only used to connect mobile devices with the network. These base stations in the RAN were only used to transport data and voice conversations but now the base stations are evolving to intelligent service hubs which enables the creation and provision of services on the edge of the network, something that was never done before. For example, Nokia Solutions and Networks totally changed the role of the base station in the mobile network. Moreover, the applications running inside the base station are called Liquid Applications, running specifically in a Radio Application Cloud Server (RACS) that is based on high performance Intel processors [10].

The RACS creates a new environment that allows the development and implementation of new applications and services in a way that gives full control to the mobile operators. As a result, they can reposition themselves within a value chain since their infrastructure is becoming more important. This will allow mobile providers to get more value out of the network by not only using is as a bit-pipe where data is just passing through the network but using the network to provide personalized services to the mobile users.

4.2 Benefits of RACS

The explosive growth of video data is becoming one of the most important loads of the mobile network. Furthermore, more users are going to download the same data (e.g. viral video or live streams) and most of the time even in the same location, which results in extensive usage of the network.

A solution can be to cache the data locally on the edge of the network so it does not need to travel repeatedly over the internet, reducing the load of the network. Today this data is typically cached just outside or just inside the RAN of the mobile operator. With the introduction of RACS the popular data will not be stored in the RAN but inside the base station on the edge of the network. This enables the possibility to send the data directly to the mobile devices reducing the load of the network. Furthermore, the advantages of the mobile edge computing technique are listed below.

Better user experience:

Data will be loaded up to five times faster from the base station than when the data is requested from servers that are typically deeper in the network. As a result, mobile users will receive requested data much faster which will improve the user experience. For example, videos or live streams will be loaded faster when the user is connected with the RACS instead of receiving the data from elsewhere in the network.

Cost savings:

The intelligent base stations significantly increase the efficiency of the network. Data storage and processing will be carried out faster in the base station. This increase will reduce the time that services are used on the server. Meaning that a service provider needs to pay less for the use of an application or a service that is running on the base station.

Exclusive control:

Mobile operators control the applications and services that are running on their base station. They are also responsible for the delivery of data and services to mobile devices. So they can provide a guarantee to subscribers as well as other content providers about the quality of the services.

Localization features:
Unlike services in the core network, the services of the base station are running locally so they can provide and deliver location based information to the users. For example providing information to a tourist that is watching a monument or a building.

4.3 Base station architecture

One of the design principles of the innovated base station architecture is to separate the IT domain (which contains the users applications and the IP traffic between the server and the applications) from the telecom domain. For economic reasons, the mobile networks are typically built between a few core networks. Accordingly there can be a certain gap between the Core Network (CN) and the user device. This geographical distance will be translated in a propagation delay that can be extended with the delay of the different network hops between the user device and the CN. With the conversion of the architecture of the base station, the distance between the user device and the CN will be significantly reduced resulting in a lower the propagation delay.

Figure 3 shows the application architecture of the base station. The applications in the base station are split up into two groups, embedded and add-on applications. Each group has a different lifecycle. The provider of the base station develops the embedded applications and will provide a software release of the application on the base station. In contrast to the embedded applications, the add-on applications are released independent of the software release on the base station and can be developed by the provider of the base station or a third party. This concept provides a faster lifecycle of the applications in contrast to the traditional telecom networks.

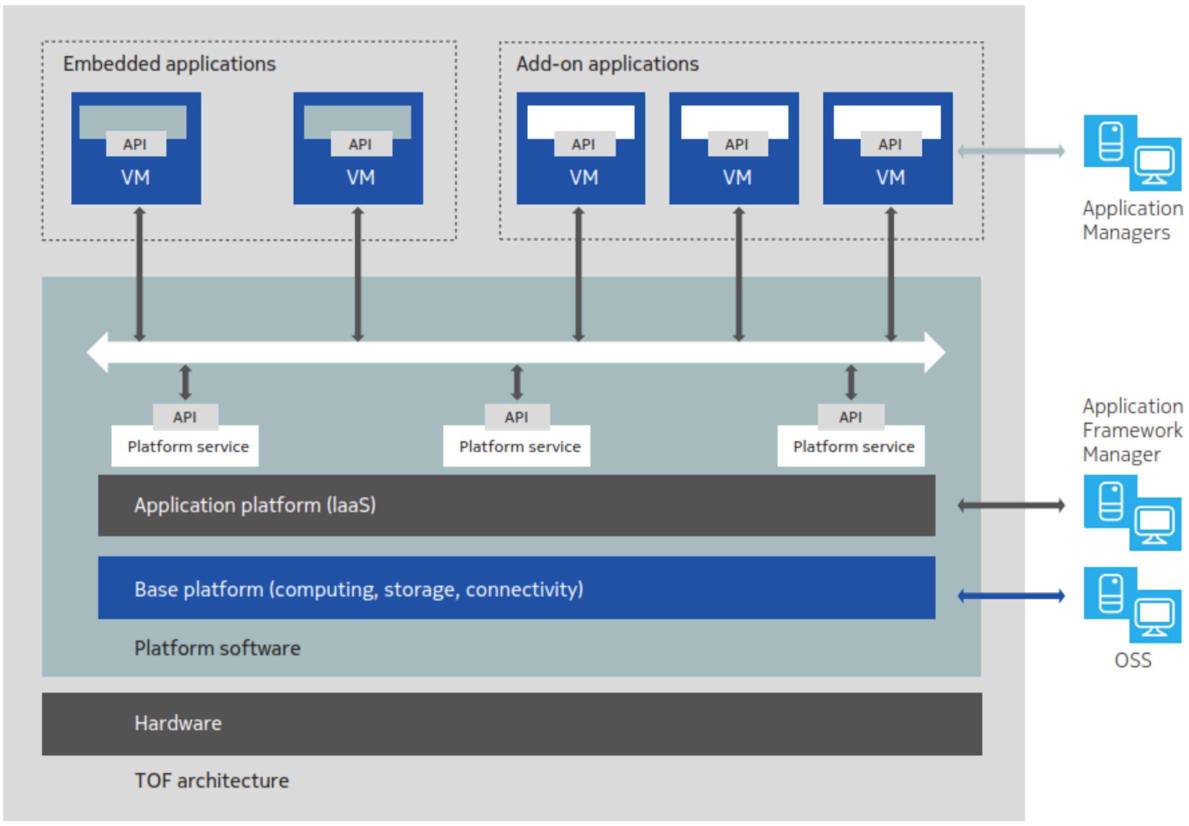


Figure 3. Base station application architecture [7]

The RACS is optimized with standardized 3GPP functions making the servers fully programmable. The servers are built around the lifecycles of the telecom, providing more flexibility from the datacenters in the server and containing data functionality in close proximity to the end user which allows fast data processing and storage [11].

4.4 Liquid Applications

Liquid applications are applications that are running in a RACS in the base station. These applications improve the user experience by acceleration because applications are running in close proximity to the end user and thus can be delivered very fast. This is translated in a significantly increased throughput and time-to-content improvements that are only available on the network edge. Liquid applications have the possibility to retrieve and process real-time data from the network, which allows network operators to detect problems very quickly in the network, implying that they can guarantee a good quality of the network. This real-time data of the Liquid Applications can not only be used to improve the network operations but also provide a good review of the behavior of the users. These improvements unleash a new eco-system for services in and around the base station. Furthermore, location based applications and services can be provided to the user by the direct communication between the user and the network [1].

These liquid applications contains properties that make them robust, secure and able to deliver a better user experience. The properties are listed below:

- Agility and flexibility allows the base station to transform into highly efficient data processing and storage on the edge of the network;
- The time-to-content and the response time will be significantly improved because the server is located in a close proximity to the user;
- Real-time exploitation of the network data, which dynamically responds to the changing environment of the server, can transform data, services and applications;
- The state of the art technologies for the highly distributed application environment translates into fast development and deployment of the applications.

4.5 Augmented reality application example

Augmented reality applications are possible examples of applications that can use the capabilities of the RACS to improve the speed of the application. This cooperation with the RACS can improve the individual experience of the user. Augmented reality is not a new concept but it can be drastically improved if the augmented content is located in close proximity of the mobile user inside the base station of the RACS.

An augmented reality application can for example be an application for tourists, where the tourist is pointing its camera towards a building or a monument and the information of the viewpoint will be displayed on the screen. This application needs content from the internet. As the content travels over the internet, congestion and latency will typically degrade the quality and the user experience of the video delivery. What could happen if the content of the same application is stored in close proximity to the mobile user [7]:

- The video or virtual content is always requested and streamed from one base station, making it possible to build services and databases that only store location specific content. So the data does not need to travel over the internet.
- The mobile user experience is enhanced, because the virtual content will be streamed faster to the user than in a centralized architecture. The streaming quality of the video can be much better because there will be no delay of the central network.

- When the virtual content is running on the device, the remainder of the mobile network will not be occupied by the transmission of video. Thus the network can serve more traffic and handle more data.

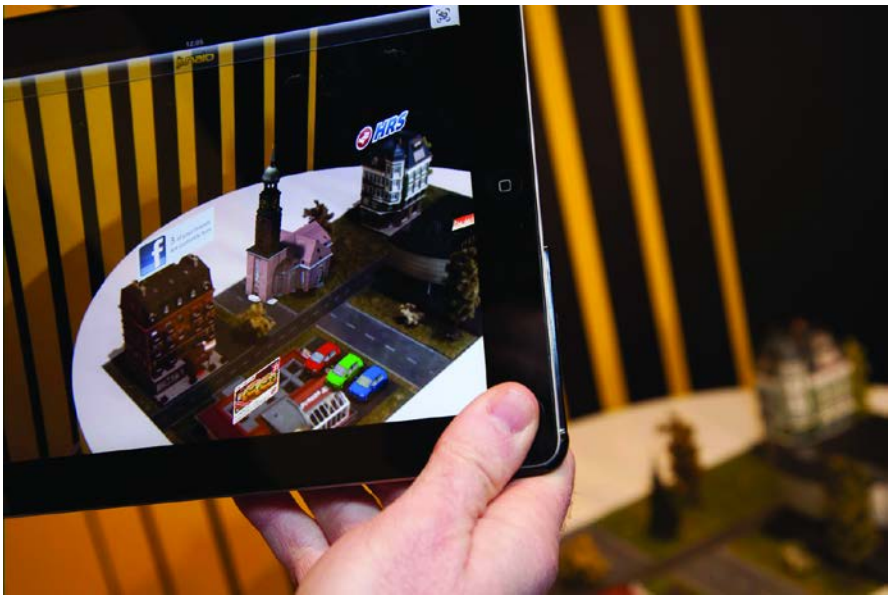


Figure 4. Augmented reality application example [7]

5 Application design

This Chapter will give the design of the application and the server that are created. Two streaming methods are compared that can be used for the data connection between the server and the application. OpenCV is also introduced just like the datatypes used for the images will be discussed. Furthermore an explanation of the color blob and square detection methods used in the applications are given.

5.1 Server design

To relocate the processing of the images to a more powerful location and to reduce the energy consumption of the devices a remote server application will be running on the RACS. This server will process the images that are received from the application and send them back to the application. When a connection between the server and the device is established data can be transferred to and from the server. Moreover, the connection between the server and the application needs to be a real-time data exchange because the image needs to be presented in real-time on the screen. So the delay between the server and the application needs to be as small as possible to deliver the best quality to the user. Two connection types between the server and the device are discussed. The first is a http long polling connection between the server and the application and the second is a full duplex communication with web sockets. In addition there will be a brief summary of the platform that is used to run the server.

5.2 HTTP long polling

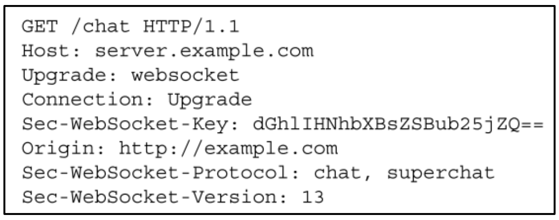
HTTP polling consists of a sequence of request – response messages. First the client sends a request to the server. When the server receives this request it will respond with a new message if there is a message ready to send back. The server will respond with an empty message if there are no response messages available. Finally when the client is not receiving any messages after a request is sent, the client will poll the server again to see if there are any new messages ready. A weakness of http long polling is the number of unnecessary requests sent to the server when it has no new messages for the client [12].

5.3 WebSocket protocol

With HTTP long polling, the client must repeat the HTTP headers in each request to the server and will receive these headers back in each response from the server. This extended adding of headers in the messages results into an increased communication overhead. Therefore the WebSocket protocol will eliminate the network overhead of the HTTP long polling and permits the connection to remain idle until the client or server initiates a request. This means that the client will be able to send image data to the server even when the server is not replying with processed images [13].

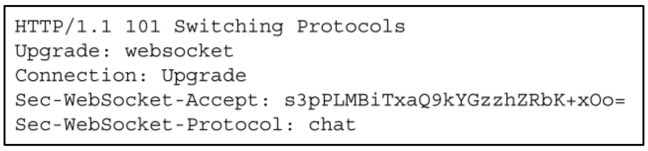
The WebSocket protocol provides a full-duplex, bidirectional communication channel that operates through a single socket over the web and is built to create real-time applications. The communication occurs over a single TCP connection and the protocol consists of two parts. The first part is the handshake that is constructed out of a message from the client and a handshake response from the server. This handshake is shown in Figure 5 and Figure 6 and is used to establish the connection between the client and the server. The second part is the data transfer [12]. In addition the communication occurs over TCP port number 80, which is of benefit for those environments which block non-web internet connections using firewall [14].

In this project WebSockets will be used instead of HTTP long polling for the creation of a real-time data communication between the server and the client. These WebSockets are used because a real-time data connection needs to be established between the server and the client. When HTTP long polling would be used a big overhead of the continuous polling to the server will slow down the data transfer between the client and the server [12]. So the WebSocket protocol will be the best solution to establish the connection between the application and the server.



```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Figure 5. Handshake from client [14]



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Figure 6. Handshake from server [14]

5.4 OpenCV

OpenCV will be used in the applications and server to detect objects in the images. OpenCV is an open source computer vision library that includes a lot of computer vision algorithms. The library is written in C and C++ and runs on Linux, Windows and Mac OS X. OpenCV is used for image processing with high computational efficiency and with a strong focus on real time applications. It is created and optimized in C and it takes advantage of multiple processors [15].

The main goal of OpenCV is to provide a simple-to-use computer vision infrastructure that is created to help people build advanced vision applications. The OpenCV library contains over 500 functions that span many areas in computer vision, including security, user interface, camera calibration, user interface and robotics. Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All transformations are done for achieving some particular goal. The input data can be used for example to detect an object in a moving image or to track an object. The image from the camera of the device will be used in this Thesis to detect an object depending on its color or shape.

5.5 OpenCV datatypes

OpenCV images can be represented as three image datatypes. The three datatypes are the CvArray, the CvMatrix and the IplImage and they have an object-oriented design as shown in Figure 7. These images can be represented in different color fields: Grayscale, color (RGB) or four-channel (RGB+apha). Each channel of a color field contains several integer or floating-point numbers. The four-channel color type is more general than the three-channel 8-bit RBG Image. The OpenCV matrix will be used in OpenCV for Android to process the images. The matrix consists of 32-bit floats or un-signed integer 8-bit triplets. An element of a CvMatrix is not necessarily a single number, the capability to represent multiple values for a single entry in the matrix allows the representation of multiple color channels in a RGB image where each element of the matrix contains the RGB color of a pixel in the image [16].

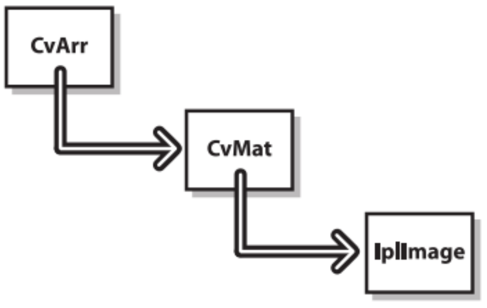


Figure 7. OpenCV datatypes [16]

5.6 Color blob detection

This Thesis will deal with two object detection methods to create the augmented reality applications. These AR applications can be used to detect road signs on the side of the road. The first method is a low processing color blob detection, the signs will be detected based on the color. The second method is a more complex square detection process, where the shape of the road signs will be used for the detection.

Color blob detection will detect an area with a specific color. It refers to mathematical methods that are aimed at detecting regions in an image that have different properties, such as a specific color, compared to areas surrounding those areas. A blob is a region of the image in which some properties are constant or vary within a prescribed range of values [16]. When a property is specified, in this case the color of the road sign, the area can be detected on an image using OpenCV libraries. The detection will occur when passing through several steps. All the steps and the different datatypes used in the steps are shown in Figure 8.

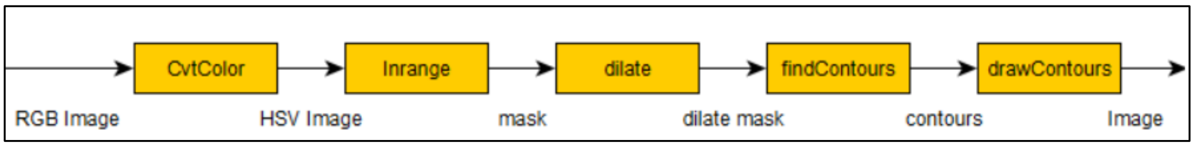


Figure 8. Color blob detection method

First a color conversion from RGB to HSV needs to be done because RGB images are exposed to lot of environmental effects like difference in lighting, shadows, motion blur and vibration that can occur when taking a picture. This can mean that some objects with the specified color are not detected and others are. A conversion to the HSV color space is needed to get rid of the environmental effect and detect the object that needs to be detected [17]. The HSV color space stands for Hue-Saturation-Value and is a different color system than RGB. This color system is more natural for the human perception. So the detection with HSV will be more accurate, objects will be detected in images with diverse ligting. Because when an object is in a shadow environment the color will not be the same as when the object is in light. A human will think that the object has a specific color but the RGB detection will not detect it. When translating the image to the HSV color space, the colors are split into different values and are not affected by shades, motion blur or vibrations [18].

The second step in the color blob detection is to create a black and white mask. This needs to be created to tag all objects in the image with the same color. In order to create a mask, the detected color needs to be converted in an upper and lower boundary using a specified radius. The color that must be detected will be in range of the upper and lower boundary. The inRange() method from OpenCV is used to create the mask. This function will check if the pixels in an image fall within a certain specified range. Each pixel of the image will be compared with the range, when the value of the pixel is between the upper and the lower boundary, then the corresponding value in the mask will be set to white (0xff). Otherwise the value in the mask will be set to black (0x00). The result is a mask with white spots representing the detected object. Next the mask is parsed with the dilate() function that will remove noise and melt small spots together. When the inRange() method is used a large detected region can be broken apart during the creation of the mask. The dilate function will melt the components back together.

In the next step the contours are detected in the dilated mask. A contour is a list of points that represents a curve in an image. In OpenCV contours are represented by sequences in which every point in the sequence encodes information of the location of the next point on the curve. The findContours() method is used to find the contours of the white spots in the dilated mask. There are a few possible methods to represent contours in an image. The CV_CHAIN_APPROX_SIMPLE will be used in the application and on the server to represent the points. This mode will compress the vertical, horizontal and diagonal segment of all the points, leaving only the ending points. This will reduce the amount of points that need to be drawn the contour on the image thus speeding up de the detection. CV_CHAIN_APPROX_NONE is the other possible method that can be used to store the points of the contours. This method stores all the contour points and is not used in this Thesis because the data size of the contour points will be bigger than the other method and thus slowing the application.

Finally the outlines of the contours are drawn on the original image using the drawContours() method from the OpenCV library.

5.7 Square detection

The second algorithm used in this research is a more complex square detection method. This can be used to detect square road signs near the edge of the road. Squares can be detected by looping through all the detected contours on the image and extracting only the contours that contain four points. This technique will detect all rectangles, the selection when a rectangle is exactly a square is not implemented. The process of finding rectangles out of an image is shown in Figure 9.

First a color conversion from RGB color space to Grayscale needs to be done to convert the four channel RGB (each pixel is represented by Red, Green, Blue and Alpha)image into a one channel Grayscale image. The value of each pixel will be a single value that will carry only the intensity information. This will create a black and white image that is composed out of shades of grey, ranging from black at the weakest intensity to white at the strongest[16]. This greyscale image will be processed to detect the contours of squares.

In the second part of the square detection, the Greyscale image needs to be converted in black and white images that can be used to find the contours. There are two possible modes to convert the image to detect squares. The first mode is the iterative method in which a threshold is used to create a mask and the second is the adaptive mode which requires an adaptive threshold to generate the mask. Both methods will generate almost the same detection, but in some cases the iterative mode will generate a more precise detection because it covers more threshold values.

In the iterative mode, the image will loop several times through the threshold method using a different threshold value. The threshold is used to reject all the pixels above or below some value while keeping the other pixels. A different threshold value will be used to create a mask each times it loops through the threshold() function to eliminate strong lighting or reflection that can occur when the picture is taken. This method will give slightly better results than the adaptive mode. A black and white mask that is created to find all the contours of white spots in the image using the findContours() function. Next each contour that contains four points is extracted from the contours list leaving only a list with squares that can be drawn on the original image using the drawContour() function.

The adaptive mode uses an adaptive threshold to create the mask. This is a modified threshold technique in which the threshold level is self-variable. This function is an automated version of the iterative mode where the threshold value is automatically changed. The adaptive technique is useful when there is strong lighting or reflection that will adapt the greyscale image and resulting in a bad mask when the threshold function is executed only one time. Once the mask is created using the adaptiveThreshold(), the contours of the white spots can be detected on the mask. Only the contours that contain exactly four points are extracted, leaving only squares that can be drawn on the original image.

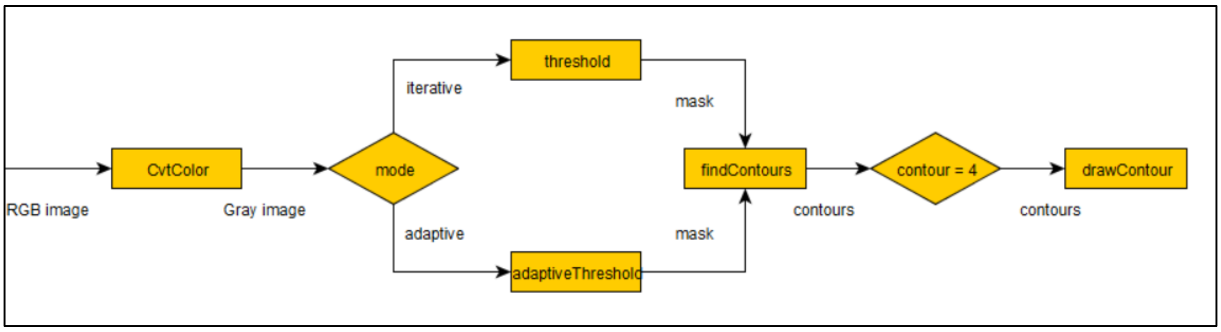


Figure 9. Square detection method

5.8 Server Platform

There is a lot of software available to create a server that is capable of processing the real-time image data. There were three possible platforms that could be used to create an OpenCV image processing server. The first is PHP, second is Node.js and the third is Python. First PHP was researched to create a data connection between the server and applications. But PHP was not the right platform to develop the server because there are almost no libraries for implementing OpenCV. Then the server was first developed in Node.js to test the connection between the applications and the server. Node.js is a very efficient and lightweight platform to create a communication server and there were some OpenCV libraries for face detection. This worked smoothly but the supported face detection libraries where not sufficient. The color blob and square detection functions that we needed were not available for Node.js. Therefore we decided to switch to Python, as OpenCV fully supports Python.

Another option might have been to create JavaScript modules for Node.js that convert C++ functions from the OpenCV C-library to JavaScript functions, but this method was too exhaustive and would have taken a long time to create this module.

6 Implementation

This Chapter explains the implementation of the cloud server and the different applications that will move their processing to this server. The creation of AR cloud applications is required to measure the difference in performance between an AR application processing on the device or processing on the server. A cloud server is created to process the images from the AR device. Furthermore a color blob and a square detection application are developed that will co-operate with the server to process the detection.

6.1 Server

A cloud server is developed to move the image processing from the mobile device to the RACS. This server will be used to measure the performance and determine whether offloading the processing will improve the application by processing the images faster and reducing the energy consumption. The cloud server needs to be capable of maintaining the connection to the device. It also needs to process the received image from the mobile device and furthermore it needs to measure the processing times for the comparison with the mobile device.

6.1.1 Used server libraries

The server is developed in Python and uses four libraries. Autobahn is the first library that is imported to create a web socket server. The second is the OpenCV library for Python to detect objects in the images received from the applications. Moreover, NumPy is the third library and is used to convert binary strings to a n-dimensional array (NumPy[]) that can be used in OpenCV. The last one is called time. This library is used to measure the execution times of the functions.

Autobahn is imported to create a web socket server that is able to communicate with any web socket client. The server has some protocols that can be used to communicate with the client. However a protocol class needs to be created to specify the behavior of the server. This behavior is determined by functions such as onOpen, onMessage and onClose. The onOpen method is called when a client opens a connection to the server. When the client disconnects from the server, the onClose method is called and the onMessage method is evoked when a message is received from the application. The type of the received and sent messages in Python is always a byte array (Byte[]).

The actual server is based on the behavior of the protocol class and is created by instantiating the protocol class. A TCP listening server is created that will use the protocol class and is listening on a predefined port [19].

OpenCV is an open-source library that includes computer vision algorithms and is imported in the server to do the color blob and the square detection [15] as explained in the previous section.

NumPy is an essential package for scientific computing with python. It is used to create a powerful n-dimensional array object that can be utilized to convert the byte array received from to client to a NumPy array that is used for the detection in OpenCV [20].

Time is a module that contains various time-related functions. It is imported to measure the execution time of the OpenCV color blob detection in the server [21].

6.1.2 Server implementation

The server is developed to detect objects in the received images from the Android applications. It can detect color blobs or squares depending on the application that streams the images. Furthermore the detection will be drawn on the received image and the processed image will be sent back to the android application. The server is forked in different parts to process the image and is shown in Figure 10.

The first part is receiving the image as a byte array from the application. Images will be sent from the application to the server as soon as the connection is opened. First the detection mode is sent to the server before the image is sent. When the mode is received it will be saved and used to detect objects in the stream of images from the application. When an image is received in the server it will be converted from a byte array to a NumPy array that is needed for the object detection.

The second part is the detection of objects in the incoming image stream. The color blob or square detection starts when the conversion to NumPy array is complete. There are some small differences in the implementation of the color blob detection between the server and the android application. One difference is that a color transformation from RGB to BGR is needed to receive the same detection results as the application. This is because the images on the application are processed as BGR images and the received image is in RGB format. The other difference is a threshold that is applied to the detection mask to remove some noise and to get a cleaner detection. Furthermore the contours of the detected object are added to the received image and the image is ready to be sent back to the application. But first the NumPy array needs to be converted to Byte array before the image can be returned to the application.

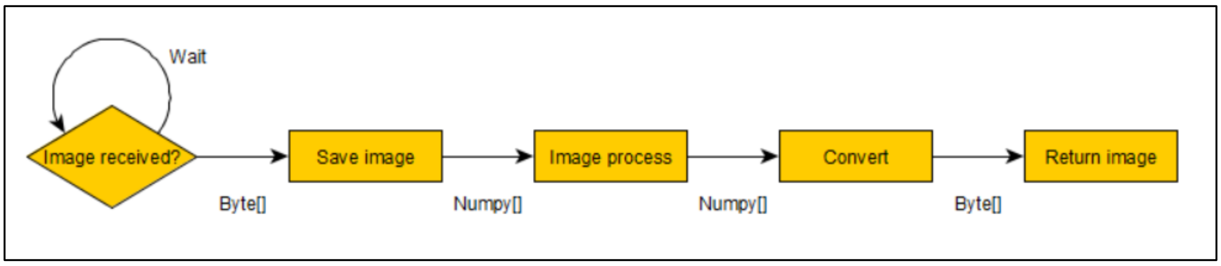


Figure 10. Server implementation

Different detection methods are implemented in the server to find the best trade-off of offloading the image processing to the server. For this purpose, the application will stream data from different points in the application to the server ranging from images to contours. Different functions needs to be executed on the server depending on the data that is streamed from the application. The server needs to receive the mode from the application to know which functions needs to be executed. This mode will be saved on the server and the following images that are received from the phone will be processed in the same way until the server receives a different streaming mode from the application. The server is capable of processing all the different streaming modes that are implemented in the color blob and square detection applications.

6.2 Augmented reality applications

A color blob and a square detection application have been created in this Master's Thesis to measure the trade-off between the AR device and the RACS. The applications are developed for the Android operating system and the performance is tested with a Samsung Galaxy S4 mobile phone and a Google Glass device. The smartphone is used to test the applications because it has a high CPU and a good battery. On the other side a Google Glass is used because it has less processing power and lower battery capacity. So when high processing needs to be done the Google Glass will be slower in processing than the phone and it can be advantageous to switch over to the RACS to do the processing of the images.

There are two AR applications developed for the smartphone and the Google Glass. The applications for the phone are almost the same as the applications for the Google Glass. The only difference between the two is the resolution of the images that are displayed on the screen. For the phone a resolution of 1920x1080 pixels is used and a resolution of 800x480 pixels for Google glass. Specifically there will be one color blob and one square detection application for the phone and the Google Glass. These applications will also have different streaming modes. Data can be streamed from various points in the application to the server to determine the best trade-off with the server. The applications and the different streaming modes will be briefly explained in the next section.

Figure 11 shows the general structure of both the color blob and the square detection application that cooperates with the cloud server. First the application will wait until an image from the camera is ready. Then a conversion of datatypes needs to be done because the image can only be streamed as a Byte array and the retrieved camera image is in the OpenCV matrix format. So the conversion from the OpenCV matrix to byte array is carried out. Next the image will be sent to the server and shortly after the processed image is received from the server. Then the received image is converted to the OpenCV matrix such that it can finally be displayed on the screen.

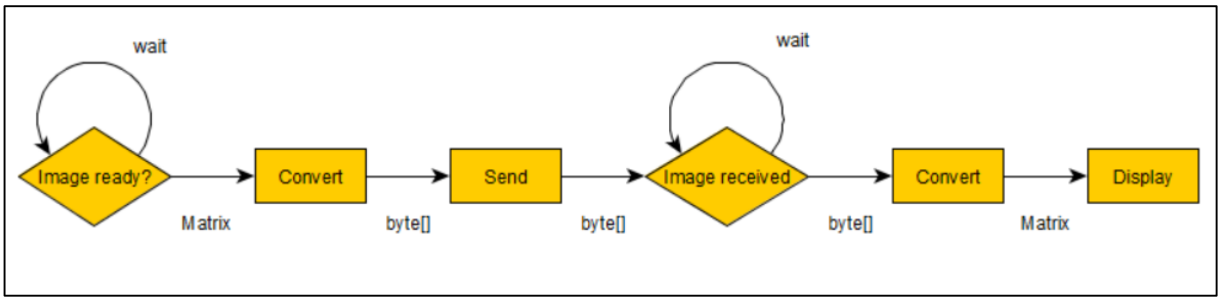


Figure 11. Augmented reality application structure

6.3 Color blob detection application

The color blob detection algorithm has been developed in two versions: the first one for the detection on the AR device and the other one for the detection in combination with de RACS. The application contains a button where the choice can be made between processing the image on the server or on the device. In the color blob detection application, objects of the same color will be outlined and shown on the screen. The color of the detectable object can be selected with a touch on the screen on the smartphone and a fixed color is used on the Google Glass. Figure 12 is a screenshot of the color blob detection application. In this scenario the blue color of the road signs were selected and the yellow contours on the image shows the color blob detection.

The image can be processed on the phone or in combination with the server. When it is processed on the device, the image from the camera will be processed in the device and the detection will be done in the device using the OpenCV library for Android. Furthermore, the OpenCV surface view is used to show the images that are processed on the screen. When an image from the camera is ready to be processed the onCameraFrame() function will be called. This function receives the image from the camera and needs to return an image that will be displayed on the screen. This camera image will be used to detect objects with the selected color. When the detection processing is ready it will be shown on the display of the mobile device.

When the option is selected that the images will be processed on the server, the applications will use the Autobahn library to introduce web sockets to create a connection to the server on the RACS. The connection to the server is established when the application starts. Whenever an image from the camera is ready it will be sent to the server through the web socket connection. But before the image can be sent to the server they need to be converted from the OpenCV matrix to a byte array that can be used to send data through WebSockets. When the data conversion is finished the images can be streamed to the server. As the image arrives in the server it will be saved, processed and returned to the device through the same web socket connection. When the image is retrieved back on the mobile device it will be shown on the screen.

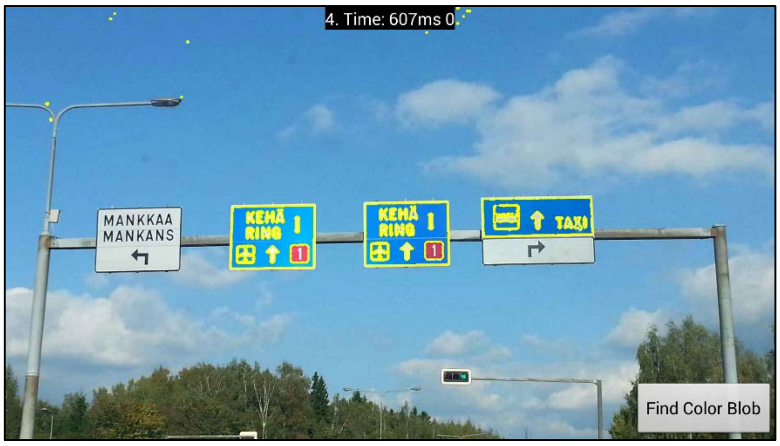


Figure 12. Screenshot color blob application

But sending images to the server and retrieving whole images back from the server is devious and a lot of data needs to be sent to the server. Because the images are quite big to send to the server and retrieve from the server it will introduce a delay when the server is used. To reduce this delay with the server different types of data are streamed from various points in the application to the server and received from the server to determine the best and fastest method to offload the image processing and to reduce the delay to a minimum. There are six methods tested that will each stream data from a different points in the application:

- Send image – retrieve image: The image from the mobile device is sent to the server, on the server the image will be saved, processed and the full image with the detection in returned to the device.
- Send image – retrieve contours: The image is streamed to the server and the server will only stream the contours back to the mobile device. Only returning the contours to the mobile device lowers the delay to the server because the sizes of the contours are lower than the size of the full image with the implemented detection. When the contours are received they need to be converted to the correct data type so they can be used to draw the contours on the original image.
- Send image – retrieve mask: The image from the camera is sent to the server and the processed mask from the server is returned to the mobile device. The delay will be lower because the size of the mask is a lot smaller than the size of the processed image.
- Send image – retrieve dilate mask: The image is streamed and the delate mask is returned. Just like retrieving the mask from the server, the delay will be lower because the size of the dilate mask is smaller.
- Send mask – retrieve contours: Sending the already processed mask on the device to the server and the server will detect the contours on the retrieved mask and return the

contours. The received contours needs to be converted on the device to the correct datatype so they can be drawn on top of the original image.

- Send dilate mask – retrieve contours: The dilate mask is processed on the mobile device and streamed to the server. On the server the contours of the selected color will be detected and streamed back to the device and drawn on the original image.

6.4 Square detection application

The android square detection applications is more complicated and takes more time to process an image than the color blob detection application. So for this application it can be more useful moving the image processing to the cloud server. This application is also split up into two versions, one part for processing the images on the mobile device and the other part for cooperating with the cloud server. There is also an option to choose if either the squares are detected using the fast adaptive threshold detection or the slower iterative way using the normal threshold for the detection of the squares. Figure 13 is a screenshot of the square detection application. The green squares on the figure show the detected road signs using the iterative square detection method. Also the buttons are shown where the selection can be made between processing the image on the server or on the mobile device.

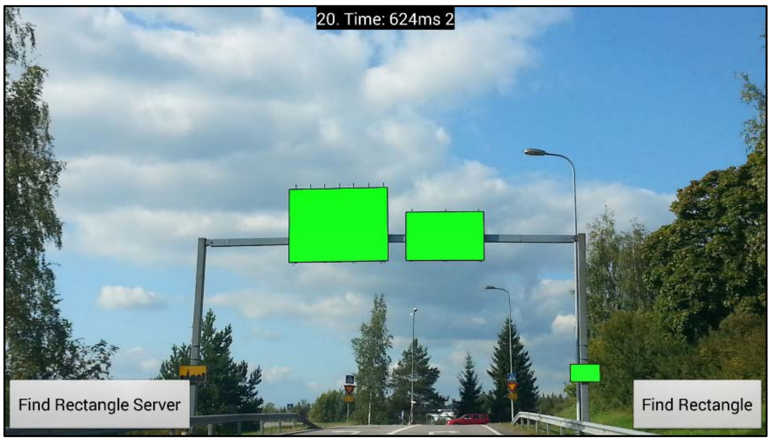


Figure 13. Screenshot iterative square detection

The image from the camera of the mobile device will be used to detect the squares. When the onCameraFrame() method is called in the application an image from the camera is ready to be processed. When the image is received in the function, depending on which detection method is selected, the squares will be detected and drawn on the image. When the detection is drawn on the image it will be displayed on the screen. The processing time will depend on the used detection method. When the adaptive threshold is used the images will be processed much faster than the iterative way because the iterative mode needs to loop a lot of times through the threshold which makes it slower.

To reduce the image processing times of the square detection methods, a button in the application can be used to send the images to the server. When the button is pressed the connection with the server will be established. First the images needs to be converted from the OpenCV matrix to byte array before the image can be streamed to the server. When the image is received on the server it will be saved and processed to detect squares in the image. As soon as the image processing is finished the image with the added detection will be sent back to the mobile device. Once the image is received on the mobile device it needs to convert the data back into an OpenCV matrix so it can be displayed on the screen. The image processing method on the server will be the same as the method that is selected on the device. Just like the color blob detection there are different methods implemented in the server and the device to send and receive different data in the application to reduce the delay with the cloud server.

Below are the two different send and receive methods implemented for each square detection method:

- Send image – receive image: The image from the camera is streamed to the server. On the server the image will be processed using the adaptive or iterative mode depending on which method is selected in the application on the device. The image with the implemented detection is returned to the device as soon as the square detection is finished.
- Send image – receive contour: The image from the device is sent to the server. When the image is received it will be processed and only the contours will be returned to the android device. On the android device the received contours are conversed into the correct datatype so it can be drawn on the original image and shown on the display.

7 Evaluation

Augmented reality applications require powerful CPUs and consume a lot of power at the mobile devices. The latter can result in unexpected malfunction of the application. Therefore, it can be advantageous to use a cloud server to accomplish the real-time image processing. In this Thesis the performance of the applications are measured in collaboration with a cloud server that runs on a local PC and the distributed RACS.

7.1 Execution times measurements setup

The implementations of the different applications are evaluated in different modes in order to draw conclusions on which mode will result in the fastest and most energy efficient cooperation with the server. There are four different applications tested, the first and second are the color blob detection for either the phone or Google Glass. Furthermore, the third and fourth are the square detection applications that are developed for the phone and the Google Glass. The applications use different modes to send and receive data from the server in order to find the best trade-off between the server and the application. These different modes where the data is sent to the server and received from the server for the color blob detection application are given in Table 1. Moreover, Table 2 provides the various methods used in the square detection application.

Table 1. Color blob detection modes phone and Google Glass

Mode	**Send**	**Receive**
Image	Image	Image
Contour	Image	Contours
Mask	Image	Mask
Dilate mask	Image	Dilate mask
Stream mask	Mask	Contours
Stream dilate	Dilate Mask	Contours

The phone and the Google Glass will be connected to the server using the Wi-Fi connection of the mobile device. The server is running on a PC that has a wireless network card to create a local network where the devices can connect. The delay with the server is measured on the phone as well as the necessary time to convert the received data from the server and the data conversion to send the image to the server. The server will measure the total processing time of the image processing on the server and will send this data to the phone. All the gathered data is collected on the phone and logged into a text file.

There were two scenarios carried out when measuring the different delays and execution times. All possible modes in the applications are tested with the same images in order to come to a conclusion. In the first scenario one image was applied for the color blob and square detection application. Because the execution times vary each time a functions is executed, an averages is taken out of looping twenty times through the image. The second scenario uses a loop of twenty different pictures to create an average delay and execution time. This reconstructs the scenario where the application will be used in a real world scenario because when it will be used to detect the square road sign in the real world the images will vary continuously. Both scenarios are tested and measured but this Thesis will deal only with one of the two scenarios. The slowest scenario of the image processing on the server will be chosen to determine the best trade-off between the server and the application.

This scenario is the worst possible and the other scenario will always be faster. Further a fixed color will be used for the color blob detection application to detect blue road sign in the images from the different scenarios. The server is tested with multiple applications that uses the server on the same time but all execution times are measured with the server that is not loaded and only dealing with one user.

Table 2. Square detection modes phone and Glass

Mode	**Send**	**Receive**
Adaptive	Image	Image
Adaptive contour	Image	Contours
Iterative	Image	Image
Iterative contour	Image	Contours

7.1.1 Image streaming quality

The delay with the server is the major drawback of the application and can be reduced by lowering the data size of the image to speed up the data transfer between the server and the application. The data size of the image will be reduced when the quality of the image is lowered. Sending data with different qualities is shown in Figure 14. The delay with the server will be very high when streaming the image in full quality. When the quality of the image is lowered the delay will be reduced. The images that are streamed in the application will be streamed with 40 % of the quality which results that the delay will be three times smaller than streaming in full quality. There will be almost no difference visible when the streamed image quality is reduced from 100 % to 40 % .

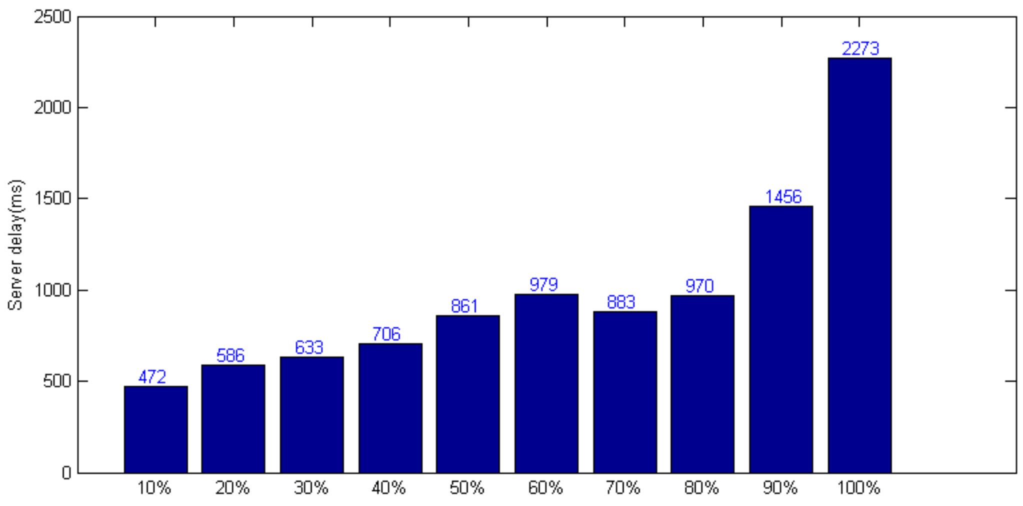


Figure 14. Image streaming quality

7.2 Execution times measurements

The execution times of the image processing for different scenarios are measured on a server that runs on a local PC and a server on the RACS. Therefore, the benchmark performance of the operating systems are measured and compared. The performance of the systems will be used to discuss the image processing times on mobile phone, PC and RACS. Furthermore, the server delay between the RACS and PC are compared just like the two measurement scenarios to select measurement results for the server delay measurements. The streamed data sizes are also measured in order to understand the different time measurements.

7.2.1 Benchmark comparison

The processor benchmark scores are measured for the phone and the different systems that use the server. Benchmarking is a way to characterize the performance of the device hardware by measuring different operations of the hardware, for example, the floating point operation performance [22]. Geekbench 3 from Primate Labs was used to measure the different benchmark scores [23]. This is a cross-platform benchmark tool to measure the computational power of processors. This cross-platform tool is used to compare the computational speed of the Android device with the servers that run on Linux. The processor benchmark scores are measured for the phone, the PC and the RACS. The phone contains a 2.27GHz Qualcomm Snapdragon 800 processor with four cores, the PC includes a 2.83Ghz Intel Core 2 Quad Q9550 processor with four cores and the RACS system has a 2.00GHz Intel Xeon E5-2640v2 processor with eight cores. The single core and multi core benchmark scores for the different processors are given in Table 3. The results show that the RACS has the best computational power for single and multicore processing. Furthermore the phone has the least computational power so it will be advantageous to move the processing from the mobile device to a cloud server either on the PC or on the RACS to accelerate the processing.

Table 3. Processor benchmark score

	Single core (points)	**Multi core (points)**
Phone	962	3123
PC	1650	5686
RACS	2099	16841

The application that processes the images on the phone can benefit of multiple cores, OpenCV automatically divides the image processing among the different cores of the device [15]. This will result in faster processing than when only one core is used to process the images. The image processing on the server on PC will also benefit from the multiple core advantage of OpenCV. The RACS consists of multiple Virtual Machines (VM), each VM for different applications. The RACS will allocate one core for each VM and the server that cooperates with the detection applications on the device is running in one VM. There is only one core allocated for the VM that runs the server which causes that OpenCV cannot benefit of dividing the images processing over different cores. This is the case for the PC and mobile phone that contain multiple processing cores.

7.2.2 Image processing times comparison

Figure 15 shows the comparison of the processing time of the color blob detection between the mobile phone, server on PC and the server on the RACS. Figure 16 and Figure 17 shows the comparison of processing times of the square detection methods with the different servers. The results shows that processing the images on the server is twice as fast for the color blob detection on the PC and a little bit faster on the RACS. The processing on the RACS will be slower than the PC because the PC has more computational power and has the advantage of using more cores. The processing of the image is carried out seven times faster on the server than on the phone for both the adaptive and iterative square detection methods. These measured processing times on the server are without the delay with the server and without the send and receive conversion times of the data. Hence, the total processing time on the server will be higher and it is possible that the total processing time is larger than the processing time on the phone due to the extra delay that needs to be added.

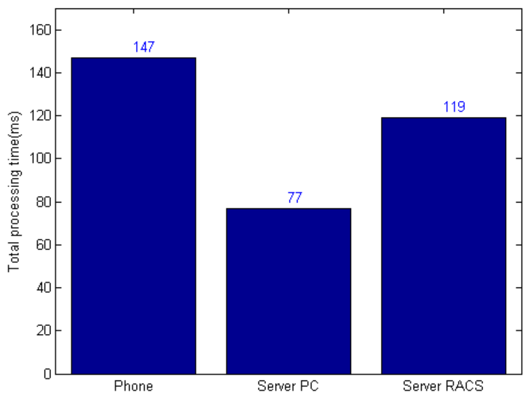


Figure 15. Color blob detection processing times

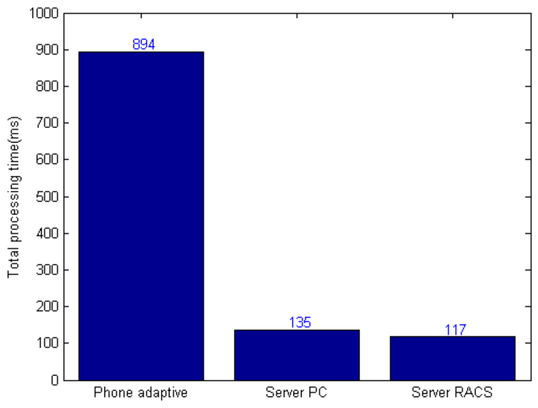


Figure 16. Adaptive square detection processing times

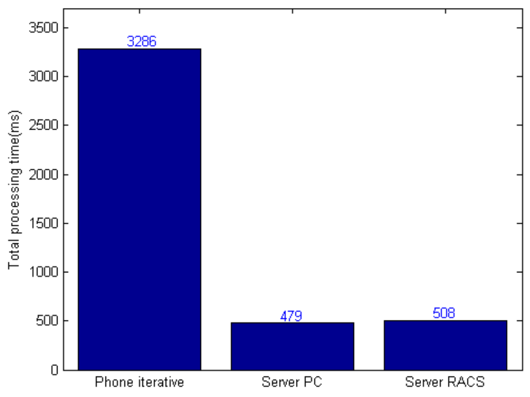


Figure 17. iterative square detection processing times

7.2.3 Server delay comparison PC and RACS

The execution times of the different modes and applications are measured for both the server on the PC and on the RACS. Figure 18 shows the comparison of server delay between the PC and the RACS for the color blob detection processing. Furthermore Figure 19 compares the server delay for the square detection application. This server delay consists of uploading data from the application to the server, processing the data on the server and finally downloading the processed data from the server.

The results for the color blob application shows that the delay with the server on PC will be two times larger than the delay with the RACS. This is because the RACS is situated in the base station at the edge of the mobile network, which allows a very fast communication with the mobile device. The server on PC is running in a local network that cannot reach the download and upload speed from the RACS. These results also show that the RACS is more powerful than the server on PC.

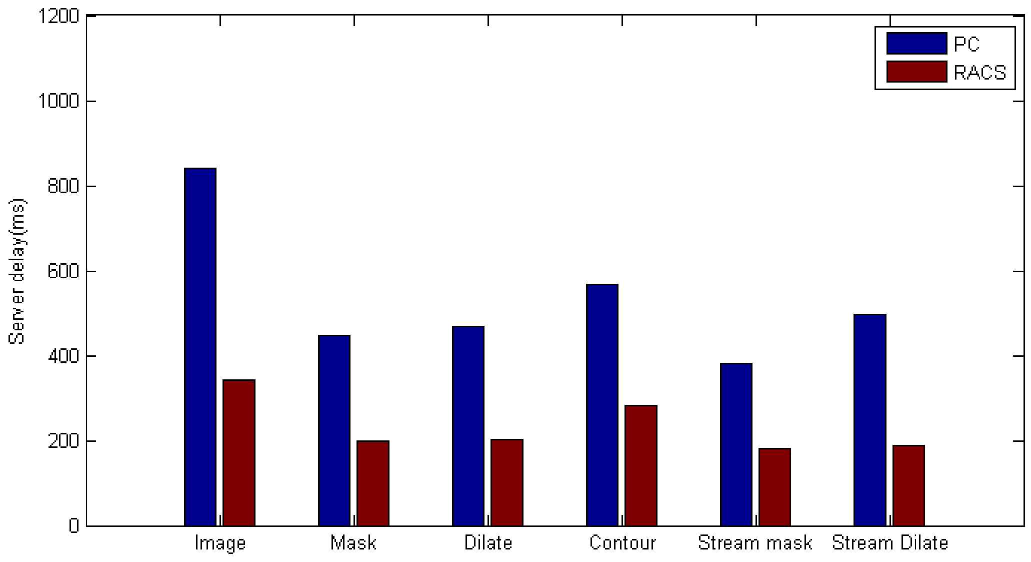


Figure 18. Color blob server delay comparison between PC and RACS

The server delay between the application and the server will also be smaller for the square detection when the server is deployed on the RACS instead of the PC. In general, the delay of the iterative square detection will be very high compared with the adaptive detection method. This is because in image processing on the server will take more time for the iterative than the adaptive square detection method.

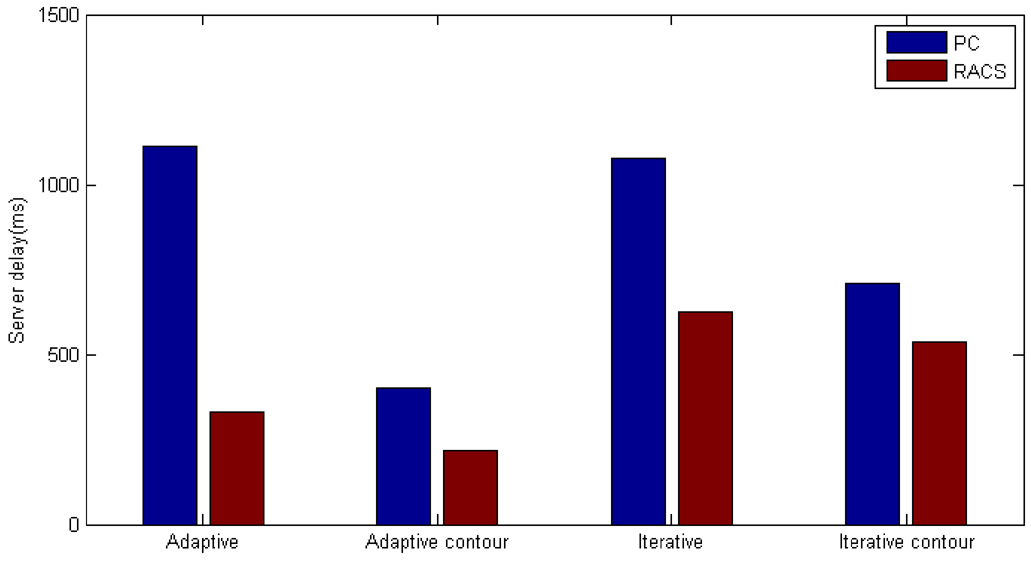


Figure 19. Square detection server delay comparison between PC and RACS

7.2.4 Measurement scenarios comparison

The applications are deployed with two different testing scenarios. In the first scenario the application will be tested with one static image that will be used for the detection. For the second one a loop of different images will be used to measure the execution times of the different functions and detection techniques. Both scenarios will loop twenty times through the images to receive an average execution time of each function. The applications will be tested with a Samsung Galaxy S 4 and a Google Glass that will be cooperating with a server that is running on either the PC or the RACS. First a comparison between the two different measurement methods is carried out to choose the method with the slowest execution times on the server. And that method will be used to compare all the execution times with the processing times on the device because when the slowest scenario results in faster processing of the images, the other scenario will be even more advantageous.

The comparison of the two measurement methods for the color blob detection on the phone and Google Glass is shown in Figure 20. When the server cooperates with the device, an average execution time of the different trade-off approaches is given for the comparison. The execution time of the color blob detection on the phone itself is faster than in cooperation with the server.

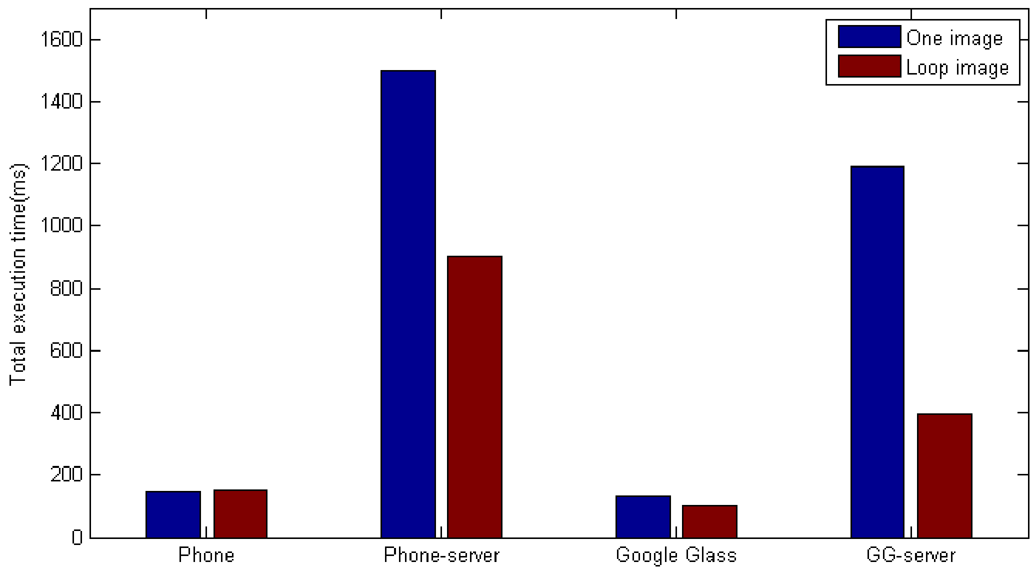


Figure 20. Comparison of the different scenarios for color blob detection

The average execution time for each method that uses one complex test image on the phone and Google glass is slower than when a loop of different images with varying complexity is used. The big difference between the two testing techniques is carried out by the amount of contours that will be detected on one image. In general, there will be more contours detected in the scenario that uses one complex image for the detection compared with the other scenario that loops through the different images because looping through images with different complexity will result in different contours for each image or even no contours can be detected resulting that the data size of the contours will be smaller. This is not always the case, It can be possible that looping through images with different complexity can result in more contours per image but with this test scenario the amount of contours of looping through one complex image is larger than the other scenario. The execution times of the method that handles one complex image for the color blob detection will be used in the Thesis for the comparison of the trade-off between the devices and the server.

In Figure 21 and Figure 22, the comparison of the execution times of the two measurement scenarios are given for the adaptive and iterative square detection application on the different devices. These execution times for the adaptive square detection are almost equal when cooperating with the server, but the measurement method that requires one image for measuring the execution time is slower than looping through different images. We will use the slowest method for comparing the trade-off between the server and the device. In this case the approach with one image is used further in the Thesis. This scenario will be less realistic than a series of images is used.

Looping through a series of different images on the phone results in a slower execution time than looping with one image. This is not the case for the Google Glass, when one image is used the execution time will be faster than looping through the series of images. This difference between the phone and the Google Glass is due to the difference of image size. The phone uses 1920x1080p and the google glass 800x460p, this different sizes caused that the iterative square detection can detect more squares when looping through different images and when the size of the image is bigger on the phone. Using the Google Glass with the smaller images causes more squares will be detected when looping with one image instead of with a series of images.

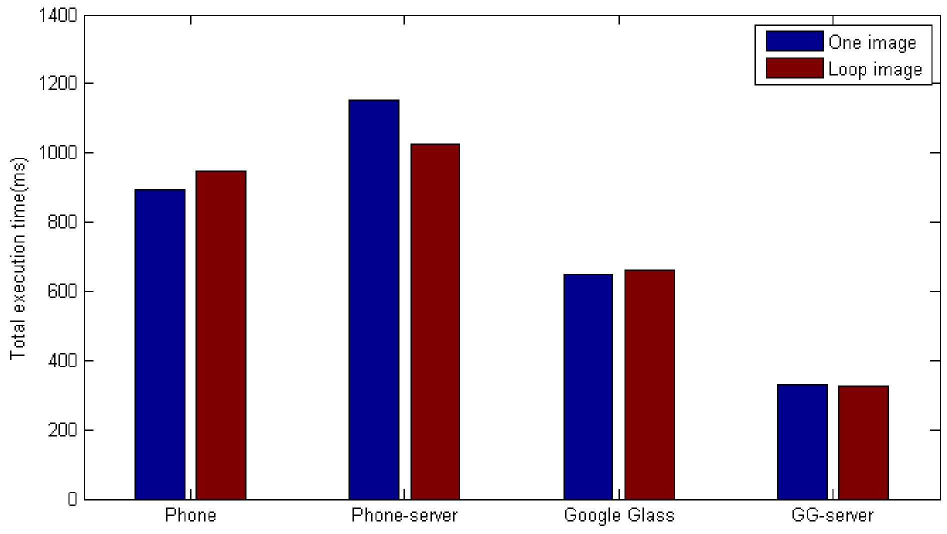


Figure 21. Comparison of the different scenarios for square detection adaptive

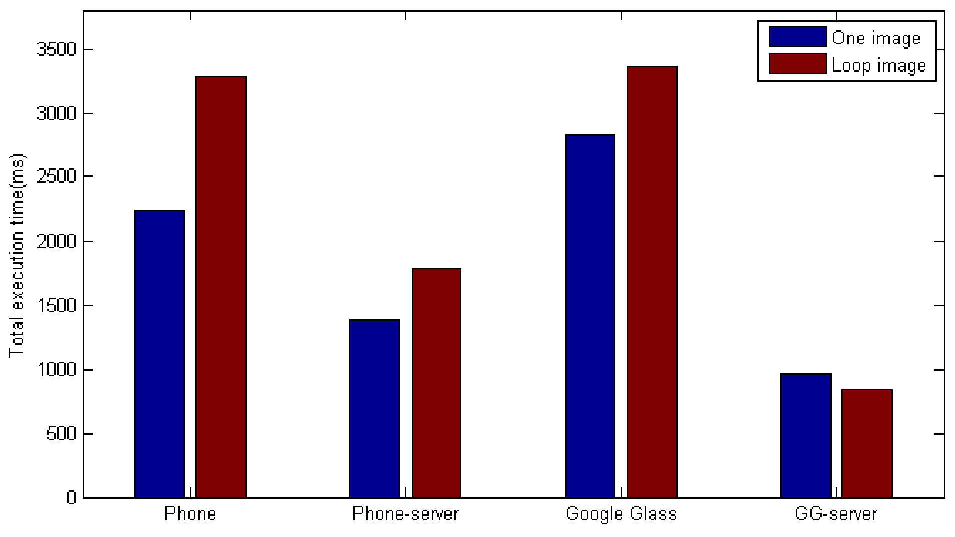


Figure 22. Comparison of the different scenarios for square detection iterative

7.2.5 Streamed data sizes

The data sizes that are streamed to and received back from the server have been explained before the trade-off between the mobile device and the server was discussed. The size of the data is involved in the object detection as well as the delay with the server. The bigger the size of the image the longer the object detection will take. Furthermore the size of the data is also important when data is sent to the server or received from the server. There will be a larger delay to the server when the data size is bigger than when fewer data is streamed. This delay with server needs to be as small as possible to improve the application that cooperates with the server.

Table 4. Send-receive sizes color blob detection phone

Mode	**Send size(kB)**	**Receive size (kB)**
Phone	359	359
Image	165	536
Mask	165	166
Dilate	165	178
Contour	165	94
Stream mask	90	92
Stream dilate	96	92

Table 4 shows the send and receive data sizes in the color blob application for the smartphone. These data sizes will vary depending on the used streaming mode. When the color blob detection is carried out on the phone there will be no connection with the server, but the given data size is the size of the image that needs to be processed on the phone. Furthermore, the received data from the server will be three times larger than the sent size. This is because the detection of the object will be added to the image causing it to grow in size. The received data from the contour mode will be smaller than the sent data because only the detection of the objects is returned to the application on the mobile device. The difference between the data size used to process the images on the device and the images that are sent to the server can be explained by the quality of the images that are streamed to the server. To reduce the delay with the server the images are streamed with 40 % of the quality, this reduction of the quality is not visible to humans but the data size will be reduced with 55 % of the original image.

Table 5 shows the send-receive sizes for the color blob detection application for the Google Glass. These image sizes for the Google Glass are an example when the data size of the received contour is larger than the sent data. This is carried out when the mask and the dilate mask are streamed to the server from the Google Glass. This is because the data sizes of the mask are very small in comparison with the detection that is returned. The contours returned from the server have almost always the same size independent from which mode is used to stream to the server. The received image size from the mask and dilate mask mode is smaller than receiving the whole image with detection from the server. When these modes are selected only the mask will be returned to the application on the mobile device in which this mask just contains black and white pixels indicating the detection. The size of the mask will be also smaller because the detection has not yet been added to the original image. Furthermore the image sizes from the phone are bigger than the images used in the Google Glass, this is due to the different image sizes used on the phone (1920x1080p) and on the Google Glass (800x460p).

Table 5. Send-receive sizes color blob detection Google Glass

Mode	**Send size(kB)**	**Receive size (kB)**
Google Glass	158	158
Image	50	155
Mask	50	55
Dilate	50	57
Contour	50	36
Stream mask	28	40
Stream dilate	28	40

Table 6 and Table 7 shows the data sizes from the square detection application. The received adaptive contours from the server will be up to twenty times smaller for the phone and fifty times for the Google Glass than the sent image. This big difference is because there are only a few squares detected in the images. This will reduce the image size significantly.

Table 6. Send-receive sizes square detection phone

Mode	**Send size(kB)**	**Receive size (kB)**
Phone adaptive	359	359
Adaptive	165	393
Adaptive Contour	165	9
Phone iterative	293	293
Iterative	140	337
Iterative Contour	140	57

Table 7. Send-receive sizes square detection Google Glass

Mode	Send size(kB)	Receive size (kB)
Google Glass adaptive	158	158
Adaptive	50	103
Adaptive Contour	50	1
Google Glass iterative	158	158
Iterative	50	103
Iterative Contour	50	23

7.2.6 Server delay measurements

The delay with the server is divided in three parts to compare the execution time of the processing on the device with the application that cooperates with the server. The first part is the time needed to convert the image to the right format so it can be streamed to the server. Then the second part is the delay with the server what contains the upload time to the server, the processing time on the server and the download time of the image from the server. And final part is the conversion of the received image so it can be displayed on the screen. Moreover, some processing needs to be carried out on the device depending on the selected streaming mode.

Figure 23 shows the total execution time of the color blob detection on the phone for the different streaming modes. As a result of the high delay with the server the execution times of the different modes are ten times slower than when the mobile phone carries out the detection. This is due to the high processing power of the phone, the color blob detection is not a very powerful process so it is fast on the phone and the delay with the server will slow the application down. The processing of the images on the server is faster than the processing on the device but the delay to the server and the conversion times causes that the cooperation is slower for the color blob detection. Not only the delay with the server but also the conversion of the data will cause a delay that ensures that the application that moves the processing to the server will be slower than when the processing is accomplished on the phone. These conversion times of the images that are streamed to the server is almost equal to the processing time on the phone.

The Contour, Stream mask and the stream dilate will have a very high conversion time of the received data from the server. This is because these modes receive a lot of contours from the server and it will take some time to convert the received contours into to the correct data format so they can be drawn on the original image. The mask and dilate mask mode where there is a good trade-off between the server and the application are the fastest but still slower than when the processing is carried out on the mobile device. With these modes the phone streams the full image to the server and the server will create a mask of the detected color blob and the application will use this mask to detect contours and draw the color blob on the original image. These modes will be faster because the size of the mask that is return from the server will be smaller than the other modes. So the delay with the server is smaller and a lot faster than returning the contours from the server.

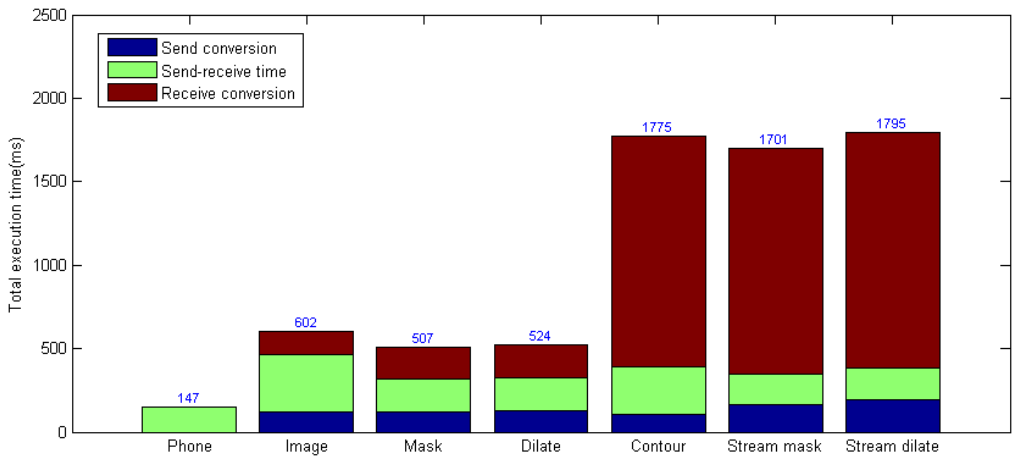


Figure 23. Server delay measurements color blob detection phone

The execution times of the color blob application for Google Glass are shown in Figure 24. The total processing time of the different modes are slower than when the detection is processed on the phone. This is because the data conversions and the delay with the server are too high to process the images faster on the server than the Google Glass. The color blob detection is not powerful enough to relocate the processing to the server. The results of some modes are better when the Google Glass is used instead of the phone but still slower than when the full processing is carried out on the Google Glass. In general, the data send conversion times and the server delay are smaller than when using the phone. This is due to the smaller size of the images. When the size is smaller, the conversion, uploading and downloading will be carried out much faster. All the streaming modes are slower but the Image, Mask and Dilate mask modes are the fastest of the modes where the application processes the images on the server. This is because the send and receive conversion times are lower due to the smaller image sizes. This smaller size ensure smaller delays. The modes that return the contours from the server are as slow as the same modes on the phone even if the send conversion and the delay with the server are lower. But it is the conversion of the received data that takes the most times. This is because the Google Glass has lower processing power than the phone. It will take a longer time to covert the received contours into the right format so they can be drawn on the image.

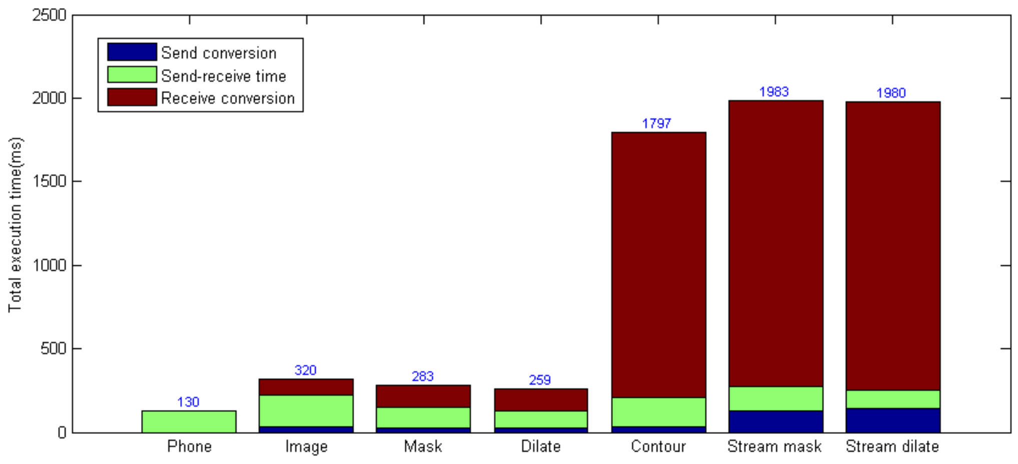


Figure 24. Server delay measurements color blob detection Google Glass

Because the color blob detection was processed faster on the devices than in combination with the server, a heavier processing square detection application was developed to detect road signs. The processing time of the square detection method is up to seven times slower than the color blob detection so a relocation of the processing to the server needs to speed up the application. There are two possible square detection methods measured for both the phone and the Google Glass, an adaptive mode and a slower iterative mode.

Figure 25 and Figure 26 shows the execution delay of square detection on the phone compared with the delays of processing of the square detection to the server. These results are more promising than the color blob detection. The Adaptive mode that returns the full image from the server will be faster when the square are detection on the server for both the adaptive and iterative contour mode. At this moment the connection cost of the server will be very low which causes that the application will be as fast as when it processes the image on the phone. In this case the use of a powerful server will be advantageous. Moreover, the adaptive contour mode will be slower than the adaptive mode. This is because the data conversion of the returned contours will take a longer time than when the full image needs to be converted. The iterative mode that processes the images on the phone is a very slow mode because it needs to execute the detection twenty times with different parameters to receive better detection results. When the same processing is carried out on the server, it will be twice as fast. This is because the server will process the iterative square detection seven times faster than the phone. In general, the data conversion of the received contours will be much faster than the color blob detection because only the contours that contain four points are returned to the phone so the conversion of the received contours to the correct format will be much faster. For the adaptive and the iterative square detection it will be very useful to relocate the processing to a powerful server to enhance the application.

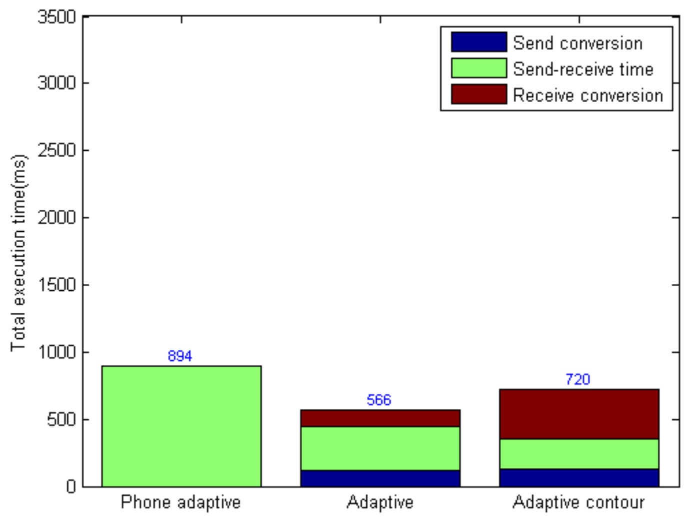


Figure 25. Server delay measurements adaptive square detection phone

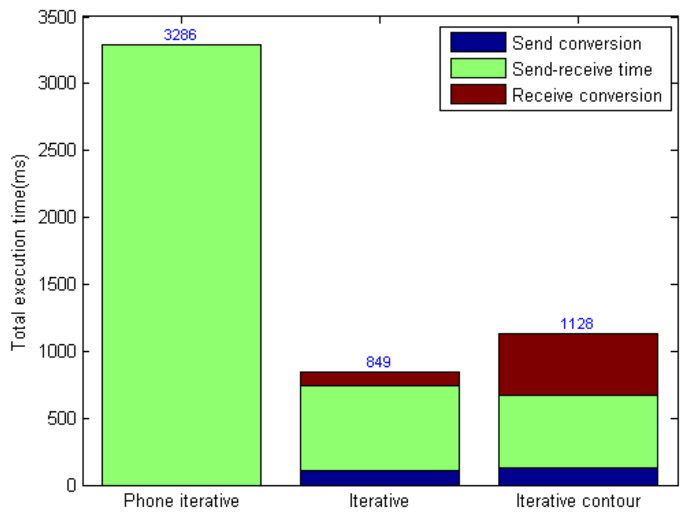


Figure 26. Server delay measurements iterative square detection phone

The same square detection methods are also measured for the less powerful Google Glass. The full execution times of the adaptive and iterative mode are shown in Figure 27 and Figure 28. The processing times for both methods are processed on the Google Glass as fast as the phone. This is because the images used in the detection on the Google Glass are smaller which allows the processing times to be smaller than when a bigger size of images are used. The results for the Google Glass are even better than the phone. Due to the smaller images the server will process the square detection even faster and the upload and download delay will also be smaller. The size of the image has more influence than the processing power when processing images on the RACS. So when the resolution of the camera image increases faster than the processing power of the mobile device, the image processing on the RACS becomes even more interesting. For Google Glass both adaptive modes that cooperate with the server are faster than the full processing on the device. The Adaptive contour mode is even faster than the Adaptive mode where the full image is streamed, this is because only a few squares will be detected and sent back what causes a very low delay with the server. It is particularly advantageous to move the processing of the iterative square detection to the server. This cooperation with the server results in seven times faster processing of the image. The iterative contour mode will be slower than the normal iterative mode but still faster than the processing on the server. This is due to the higher data conversion time of the received data. There will be more squares detected in the iterative method what causes that more data needs to be streamed back to the Google Glass and the conversion time will be longer. But in general for both square detection modes it will be profitable the relocate the processing of the image to the server.

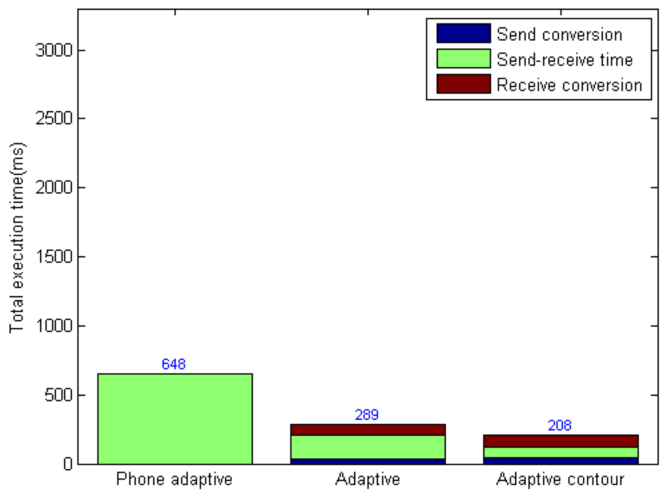


Figure 27. Server delay measurements adaptive square detection Google Glass

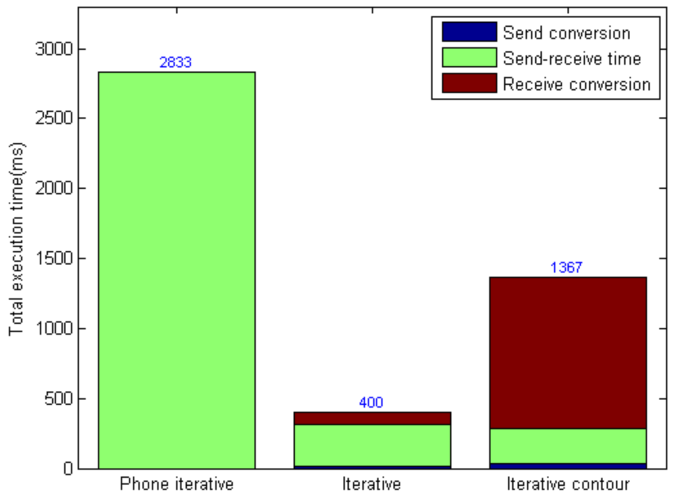


Figure 28. Server delay measurements iterative square detection Google Glass

7.3 Power measurements

Battery life is a crucial factor in today's mobile devices. The images streaming and heavy processing on the mobile device could drain the battery very quickly. Therefore, the power consumption of the applications that use cloud processing are measured and is a part of this Thesis. The average power of the phone is measured for the color blob and the square detection application. For each measurement the streaming mode of the application was changed to get a conclusion of the best trade-off between the server and the application. The average power of an object detection session is measured. These measurement session are sessions varying from three to four minutes.

7.3.1 Power measurement setup

For the power measurements the mobile device was connected to a Monsoon Power Monitor that powers up the device and measures its power consumption and current at the same time [24]. The power monitor logs the power consumption of the mobile phone several times a second. It will visualize the power consumption on the screen and the average power consumption of a measurement session will also be calculated.

The mobile device a Samsung Galaxy S4 was connected to the Monsoon Power Monitor by bypassing the battery of the device. The positive voltage clip of the battery was covered with insulated tape and a copper foil tape was used to connect the Monsoon Power Monitor to the phone. The bypassing of the battery of the phone is shown in Figure 29 and allows the phone to continue the communication with the battery as only the voltage and the ground clips are bypassed. PowerTool is the software used to record the data measured by the Monsoon Power Monitor. An overall picture of the setup is given in Figure 30.

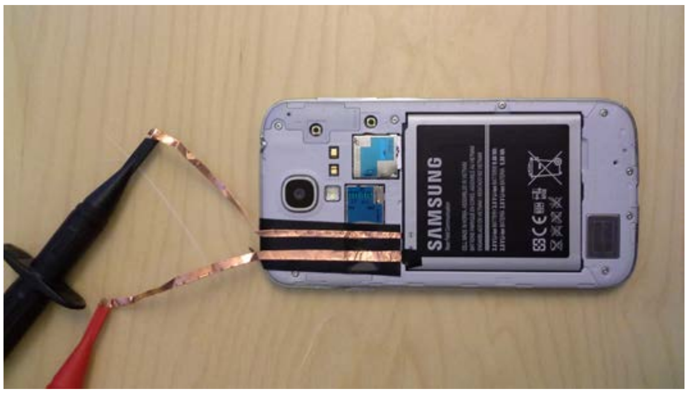


Figure 29. Bypassing the battery of the phone [25]

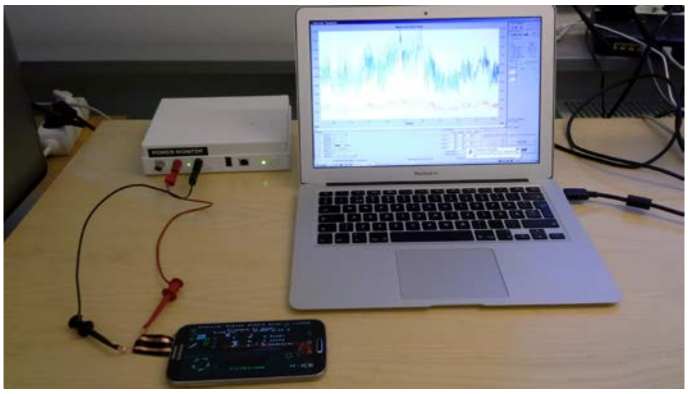


Figure 30. Power measurement setup [25]

7.3.2 Measurements

Figure 31 shows the power consumption difference between the different color blob detection streaming modes and the processing on the phone. The results show that the use of cloud processing does not necessarily save energy for all the modes. Even though the phone only needs to send and receive the images the power consumption will not be much lower than the full processing on the phone. A lot depends on the amount of processing the phone needs to do before the data is sent to the server and when the data is received. This can be seen in the different modes. The Image, Mask and Dilate mask mode have the lowest consumption, this is because the application does not need to process a lot on the phone. In comparison, the modes that receive the contour from the server will have a higher power consumption compared with the other streaming modes. This is due to the higher processing time of the conversion of the received contours. The power consumption also depends on the amount of transmitted and received data, but in this case it depends more on the processing that still needs to be carried out on the mobile phone. This can be seen in the results, the Image mode need the transmit and receive the highest amount of data but the power consumption is the lowest. Compared with the Contour mode that transmits the smallest amount of data but results in a higher power consumption than the Image mode.

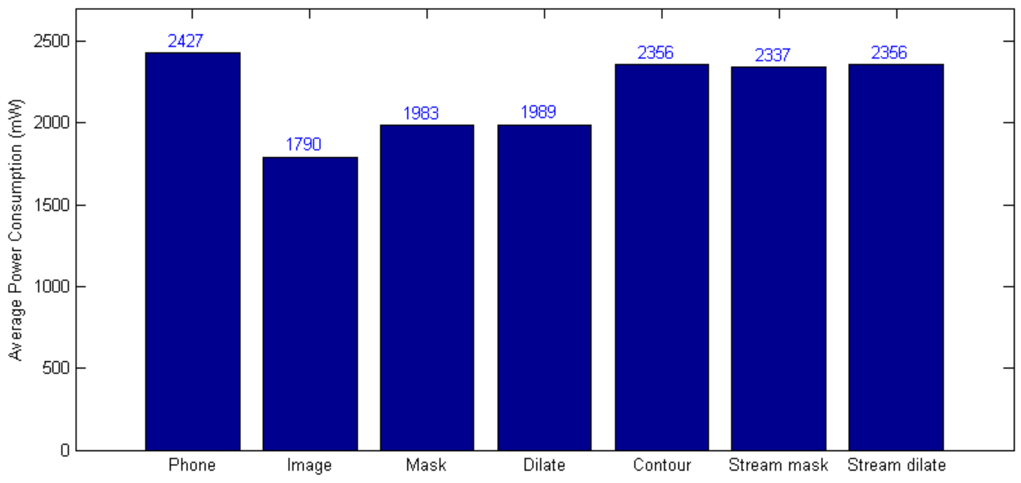


Figure 31. Color blob detection average power measurements

Figure 32 and Figure 33 shows the power consumption of the adaptive and iterative square detection methods. The results show that the higher processing times on the device are translated in high power consumption for both the square detection modes. But power consumption is lowered when the processing is carried out in the cloud. The Adaptive and Iterative Image mode consume the least power, this is because the device only needs to upload and download the image from the server so the connection will be the biggest power consumer but uses less power than the full processing on the device. The results of the modes that receive the contours from the server will consume a little bit more power compared with streaming the full image, this is because the device still needs to do some processing when the contours are received from the server.

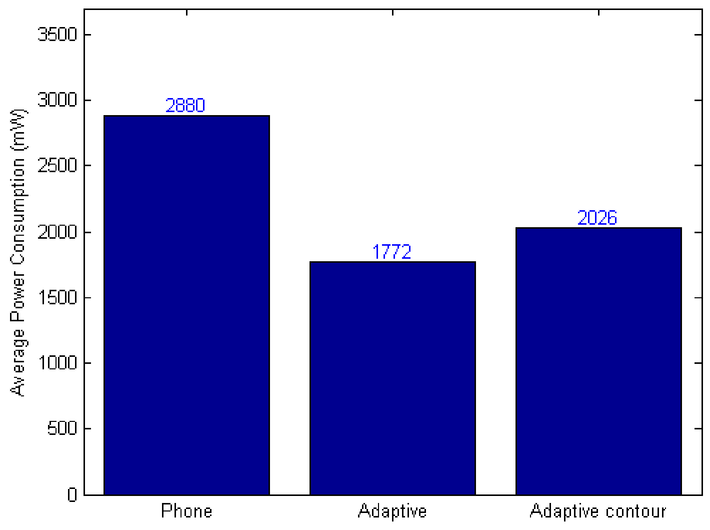


Figure 32. Adaptive square detection average power measurements

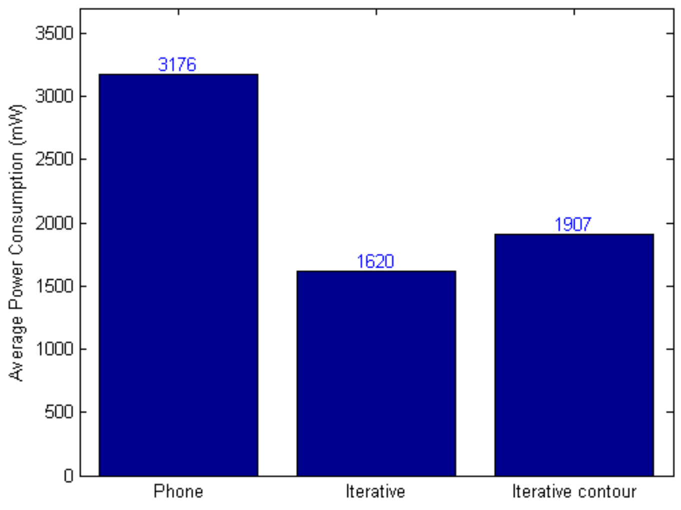


Figure 33. Iterative square detection average power measurements

7.4 Discussion

This Chapter analyzes the overall benefits of moving the image processing of augmented reality applications to a RACS. Improvement for the RACS server in the base station is also briefly discussed.

7.4.1 Overall benefits of cloud processing

Relocating the image processing from a mobile device to a server on either PC or RACS will improve the performance of the application. Although the color blob detection application will not be faster when the processing is relocated to the cloud but the power consumption of the application is reduced. A trade-off between speed and battery savings can be made, from the speed side the application will process the images rather slow when the server is used, but the power consumption will be lower so it can still be profitable to move the processing of the color blob the server to save energy of the device. But when the application needs to be time critical and the images needs to be processed fast it is not beneficial to move the processing to the cloud.

For the square detection methods it will be very advantageous to move the images processing to the server, because the use of the cloud server will speed up the application and will save a lot of energy of the device. Complex and high processing applications can be moved to the cloud to save energy and improve the processing speed of the application.

7.4.2 Improvement of the RACS base station

The base station where the current RACS is running is in the testing stage and is just a prototype. Currently the RACS contains multiple Virtual Machines that are each allocated only one core to process the application that is running on the Virtual Machine. To improve the processing speed of the RACS it could be profitable to allocate more cores to each VM. For example when a Virtual Machine can use two cores to process the application, OpenCV has the possibility to benefit of the multiple cores and speed up the application. So when more cores can be used the processing of the application will speed up the mobile application that cooperates with the server.

8 Conclusion

Mobile Edge Computing is an emerging technology for mobile users where data storage and processing capacity is placed at the edge of the mobile network . In this way data and services can be provided to the mobile user to increase the responsiveness and speed up at the edge of the network and thus increase the user’s experience. RACS is introduced as a solution from Nokia Solutions and Networks.

This Thesis deals with the trade-off of offloading image processing from mobile devices to the RACS. Several applications – on the mobile device and on the server – have been developed that cooperate with each other to relocate the image processing the RACS. Data will be streamed to the server from different locations to find the best trade-off.

Firstly server software has been developed that processes the images received from the applications. This server is deployed both on a PC and on the RACS. Secondly two road sign detection applications have been developed for a Samsung Galaxy S4 and a Google Glass. One application is created that will detect road sign based on blob color detection. The other application detects the square road signs and is more complex than the color detection application. The total execution times of the image processing are measured for all the applications, resulting that the color blob detection will be slower using the server and it will be very advantageous to move the image processing of the square detection application to the server.

Two different testing scenarios have been used to measure the execution times of the image processing. The first scenario loops twenty times using one complex image to measure the average processing time of the image. The second scenario uses a series of twenty different images with varying complexity.

The measurements show that the blob color detection cannot benefit from offloading to the RACS. However for the more complex square detection the speed up could reach up to 80%.

In another scenario different modes for when to stream the data to the RACS have been evaluated for both the color blob and square detection. Streaming the image to the server and receiving the processed image from the server for both the color blob and the square detection application gave the best results. The modes that return only the contours require less data traffic resulting in smaller delays with the server but the data conversion of the received contours in the application takes a lot of time what makes it overall slower than just streaming the full image.

The average power of the phone is also measured using the Monsoon Power Monitor to measure the average power of a detection session of the application. The moving of the processing to the server will result in a reduced energy consumption for both the square and color blob detection application. So also for speeding up and save energy it will be advantageous to offload image processing from the mobile device to the RACS.

The RACS can be even more improved by allocating more cores per Virtual Machine such that the image detection with OpenCV can benefit from using multiple cores.

8.1 Future work

This Thesis proposed an outline for the trade-off of offloading image processing to the RACS. Based on these results the development of the applications can be continued by reducing the received and sent conversion times of the images. The most optimal way off offloading would be to send the image to the server and only receive the contours because this has the smallest delay with the server. Furthermore the RACS can be expanded by locating more cores to each virtual machine so applications can take benefit of multiple cores to speed up the processing.

The offloading of the data processing to the RACS can be adapted to other fields than augmented reality applications. It can also be adapted to virtual reality applications that require more and more CPU and power from the phone. For example Oculus Rift applications for mobile devices can move the processing of the data to the RACS so only the data that needs to displayed on the screen needs to be streamed from the server. The latter can reduce battery usage and speed up the applications.

Bibliography

[1] "liquid apps." [Online]. Available: http://networks.nokia.com/portfolio/liquid-net/intelligent-broadband-management/liquid-applications. [Accessed: 26-Feb-2015].

[2] a D. Hwang and E. Peli, "An augmented-reality edge enhancement application for Google glass," *Optom. Vis. Sci.*, vol. 91, no. 8, pp. 1021–1030, 2014.

[3] K. Kim, "Mobile Cloud Computing," *IEEE COMSOC MMTC E-Letter*, vol. 6, no. 10, pp. 22–23, 2011.

[4] M. Goyal and S. Singh, "Mobile Cloud Computing," vol. 3, no. 4, pp. 517–521, 2014.

[5] a Sprintson, S. Y. El Rouayheb, and C. N. Georghiades, "Robust Network," *Main*, pp. 378–383.

[6] H. M. Patel, Y. Hu, P. Hédé, I. B. M. J. Joubert, C. Thornton, B. Naughton, I. Julian, R. Ramos, C. Chan, V. Young, S. J. Tan, and D. Lynch, "Mobile-Edge Computing," no. 1, pp. 1–36.

[7] N. Networks, "Nokia Networks Intelligent base stations."

[8] D. Pompili, A. Hajisami, and H. Viswanathan, "Ad Hoc Networks Dynamic provisioning and allocation in Cloud Radio Access Networks (C-RANs)," vol. 30, pp. 128–143, 2015.

[9] "IaaS." [Online]. Available: http://nl.wikipedia.org/wiki/Infrastructure_as_a_service. [Accessed: 01-May-2015].

[10] Nokia Solutions and Networks, "Increasing Mobile Operators' Value Proposition With Edge Computing (white paper)," pp. 1–6, 2013.

[11] M. S. Programming, "MEC Mobile Edge Computing," 2015.

[12] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," *IEEE Internet Comput.*, vol. 16, no. 4, pp. 45–53, 2012.

[13] A. Wessels, M. Purvis, J. Jackson, and S. (Shawon) Rahman, "Remote Data Visualization through WebSockets," *2011 Eighth Int. Conf. Inf. Technol. New Gener.*, pp. 1050–1051, 2011.

[14] "WebSocket protocol," pp. 1–71, 2011.

[15] "OpenCV." [Online]. Available: http://opencv.org/. [Accessed: 11-Mar-2015].

[16] A. Kaehler and B. Gary, *Learning OpenCV*. 2013.

[17] A. Lorsakul and J. Suthakorn, "Traffic sign recognition using neural network on opencv: Toward intelligent vehicle/driver assistance system," *... Intell. Serv. Robot. Springer*, pp. 1–19, 2007.

[18] T. F. Chan, S. H. Kang, and J. Shen, “Total Variation Denoising and Enhancement of Color Images Based on the CB and HSV Color Models,” *J. Vis. Commun. Image Represent.*, vol. 12, no. 4, pp. 422–435, 2001.

[19] “Autobahn websockets.” [Online]. Available: http://autobahn.ws/python/websocket/programming.html. [Accessed: 11-Mar-2015].

[20] “Numpy.” [Online]. Available: http://www.numpy.org/. [Accessed: 11-Mar-2015].

[21] “Module time.” [Online]. Available: https://docs.python.org/2/library/time.html. [Accessed: 11-Mar-2015].

[22] “Benchmark.” [Online]. Available: http://en.wikipedia.org/wiki/Benchmark_%28computing%29. [Accessed: 03-May-2015].

[23] “Geekbench 3.” [Online]. Available: http://www.primatelabs.com/geekbench/. [Accessed: 03-May-2015].

[24] “Monsoon Power monitor.” [Online]. Available: https://www.msoon.com/LabEquipment/PowerMonitor/. [Accessed: 11-May-2015].

[25] T. Kämäräinen, “Design , Implementation and Evaluation of a Distributed Mobile Cloud Gaming System,” 2014.