Computation Offloading Of Augmented Reality In Warehouse Order Picking

Creative Technology Bachelor of Science Thesis

Harald Eversmann

July, 2019

UNIVERSITY OF TWENTE

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Supervisor

Dr. Job Zwiers

Critical observer

Dr. Randy Klaassen

Client

Gerben Hillebrand

CaptureTech Corporation B.V.

[Page intentionally left blank]

ABSTRACT

A novel method is proposed to implement computation offloading in augmented reality (AR) technology, such that said technology can be of more use for industrial purposes, and in this case specifically, warehouse order picking. The proposed method utilises a wireless connection between the AR device in question and a server and lets them communicate with each other via video and MJPEG live streams. Experiments show promising results for the prototype, but not yet in terms of fully offloading the AR devices workload. It is expected that rising technologies like faster Wi-Fi connection can help in the successful conclusion of fully offloading AR devices.

ACKNOWLEDGEMENTS

The author would like to express his deep gratitude towards CaptureTech Corporation B.V. and in particular Gerben Hillebrand for the opportunity, knowledge, and resources to make this research project possible.

The author thanks Dr. J. Zwiers for his continuous supervision and assistance throughout the entire graduation project in question. Additionally, the author would also like to express his thanks to Dr. R. Klaassen for his role as critical observer during the process of this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Meaning**
AR	Augmented Reality
UX	User Experience
IoT	Internet of Things

I. **INTRODUCTION**

Augmented reality (AR) in consumer electronics as society knows it today has seen an explosive increase of use and development over the last couple of years. That is why warehouses see opportunities in the AR technology for the services they provide, as the organisations believe they can increase the efficiency of the order picking process through this technology.

In these warehouses, batch picking is a method of order picking in which multiple product requests together form a pick batch [1], [2]. Together with similar products other customers have requested, the order picker puts the requested product in the corresponding bin. When other customers have requested the same product, the order picker will pick the product as many times as requested in total, instead of picking them per client. This way of order picking can thus be seen as product-based, rather than order-based [2]. It is believed that augmented reality can help in making this process faster and more efficient.

CaptureTech Corporation B.V. (CaptureTech for short) is currently investigating if previously mentioned challenge is indeed possible. Specifically, they are intrigued by the opportunity to integrate wearable Augmented Reality into this process. CaptureTech is a relatively young company that specializes in the new generation of traceability in terms of company resource management. Effective key management, electronic locker solutions and web management is just a mere grasp of the challenges that keeps this dynamic company busy. They are constantly developing new systems that use state of the art tracking technology such as RFID, voice recognition, and the Cloud.

Previous work [3] concludes that the AR glasses currently need to do a lot of processing in order to support the process of warehouse order picking. This results in slow reaction time from the system and overheated AR technology. That is why the aim of this thesis is evaluating the processing speed and performance of the current AR technology and proposing a novel method that takes the processing off the AR device. This technique will use computation offloading, and is defined in research as a solution that utilizes nearby (mobile) devices and/or remote cloud for computational support [4]. Hence, the research question of this thesis is stated as follows:

RQ *"In what way can computation offloading for augmented reality devices be implemented in warehouse order picking applications?"*

To actually be able to answer the research question, there are still some things unclear that need to be clarified. First of all, the technology needs to be well enough developed, such that it is possible to really implement a system of sorts in the process of warehouse batch picking. This results in the first and following sub-question:

1.1 *"Is computation offloading technology currently fitting to be applied to augmented reality?"*

After a conclusion was drawn for the first sub-question, the first hurdle in a range of challenges is overcome. Computation offloading technology was indeed found to be fitting for AR technology to be offloaded. Afterwards, it became of importance to see how the deficiencies in AR can be improved. As processing power and battery life seem to be real problems, and these can be improved through computation offloading, the answer to this can be found by answering sub-question **1.2**:

1.2 *"What device is a viable option for offloading AR technology in the given context of warehouse order picking?"*

This sub-question can be answered through mainly literature and context research, which was done so extensively. The problem, together with the means to tackle this issue, are believed to be clear. Consequently, the communication between these devices need to be further specified, and this is where sub-question 1.3 comes into play:

1.3 *"How can the chosen device for computation offloading and AR technology communicate with each other?"*

After reviewing different kinds of communication technologies, Wi-Fi was found to be the most reliable and fastest of the lot, hence the choice was made to use this for the prototype. Once the communication was made, a novel method was proposed to increase the potential processing power and battery life of wearable AR technology by letting the offloading device do the hard work of recognising QR codes in the environment for 3D objects to be rendered on. This method has been evaluated through speed and distance experiments, from which it can be concluded that computation offloading is a promising technique for AR devices to gain more processing power, but the internet speed and heavy workload of image recognition both seem to be a bottleneck. However, technology is constantly in development, meaning that with some informed speculation this prototype cannot be deemed a failure. As more stable and faster wireless communication protocols emerge, together with the rise of new and better hardware solutions for image recognition, the proposed method can still see its successful conclusion in the near future and proof to be an effective technique for augmented reality to be implemented in industrial applications.

II. BACKGROUND

Augmented Reality

Augmented Reality in the order picking industry has been implemented in different ways, and this section will discuss the most relevant, but certainly not all kinds of implementation. This because it is simply impossible for the scope of this thesis to include all different solutions for it, since there are quite a few out there. Firstly, however, it needs to be determined what will be considered AR and what not. That is why, for this thesis, the following definition is used, based on multiple sources [5]–[8]: Augmented Reality is considered to be technology that overlays virtual objects into the real world for the user to see and interact with. AR technology uses the real world as a basis and is therefore supportive, not leading. Technologies that are defined as AR are thus all devices displaying virtual objects on top of the real world, whether it is through glasses, screens or other usable objects. In this thesis especially, the focus is on wearable AR technologies, as these tend to have less processing power and battery life than their non-wearable counterparts.

While there are definitely opportunities for AR out there, the technology still needs development in order to be really effective in industry. Palmarini et al. for instance highlight the technical limitations of the current state of AR in their review [7], mentioning how the current technology is not reliable and robust enough for industrial implementations. Additionally, M. Akçayır and G. Akçayır in their research [6] talk about a research that concludes that AR technology is considered to be complicated, and there were often technical issues encountered while using it. Next to that, it is mentioned that there are lots of different AR devices which can lead to misunderstanding and additional technological problems. Therefore, they conclude that the AR technologies should be developed to be *"smaller, lighter, more portable, and fast enough to display graphics"* (p. 2). In addition to that, Shi et al. [4] point out that, for wearable AR devices computational capability, memory storage, and power capacity are the biggest limitations when comparing these technologies with AR smartphone applications. However, Meulstee et al. [5] suggest that AR technology has seen a significant increase in development, and it can be expected that forms of augmented reality will soon see such an improvement that it will be more useful for industrial purposes. Because of the fact that it will probably get to such a state, the AR technology is even more worth researching.

From previously discussed situation it can be concluded that AR technology as of now might not be well suited to be implemented in industry. Nevertheless, AR technology is being improved rather quickly and will probably be capable of eventually replacing and/or enhancing certain parts of the process of order picking. That is why researching the implementation of this technology is definitely relevant and important. To improve the effectiveness and user experience of AR, speed, portability and comfort are mainly of importance. Furthermore, battery is currently still a limitation for the wearable devices. All these problems could be solved by implementing offloading [4], although it being in a more or less extent per issue.

User test

Since there already is a prototype of a solution for warehouse order picking [3], it is of importance that this prototype is evaluated. This evaluation was done through a user test executed at Idexx, a company that specialises in the manufacturing and distribution of products for different animal markets. The users that were asked to evaluate the system were professional order pickers in the industry, i.e. experts on the field of order picking and end users of the to be finalized product. Goal of this user test was to find out what part(s) of the process need(s) to be improved through offloading.

The user test was conducted as follows: first, the users filled in a consent form that can be found in appendix A. the users were asked to play out a scenario in which they start their day in the warehouse. The prototype consists of two parts: pairing and real-time feedback on the actual picking [3]. Hence, these two aspects were separately tested. First the pairing process was tested by letting the users play out a scenario where they are standing in the warehouse and "pairing" the lists with the bins. The users were given a list in the form of a QR code and they were told to scan the list and the according bin. Subsequently, the users were given a second scenario, in which the user puts the product into a bin, with the product being a carton box with a QR code on it. Due to lack of an actual order picking cart, a drawn version was made that represented the cart. The setup can be seen in figure 2.1.

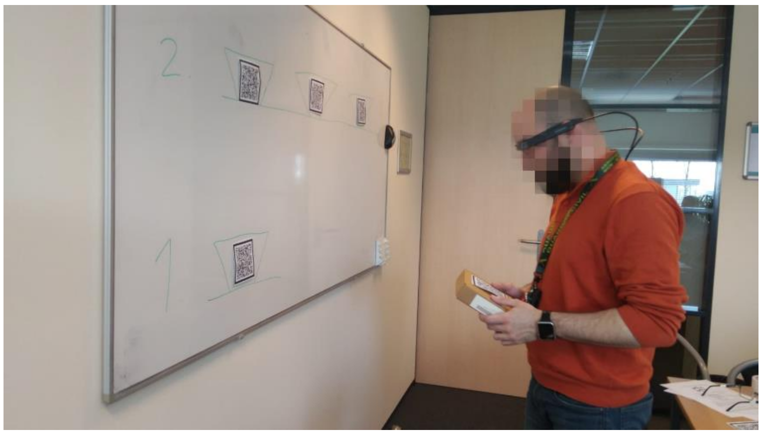


Figure 2.1: the test setup for the conducted user test at Idexx.

Since this setup was only for user testing, a "real" situation in which the markers are on the actual cart can be found in figure 2.2. After working through both scenarios, the users were asked to fill in a questionnaire which can be found in appendix B. The participants were asked to indicate how much they agreed with a total of 14 statements, ranging from the comfortability of the glasses to the digital display of the prototype. Six different aspects were evaluated through the user test: intuition, speed, performance, feedback, comfort, and potential. Each statement had to do with at least one of the aspects, and the results from the questionnaire were translated into a score for the different characteristics of the prototype. The following results were gained from the test, as can be found in figure 3.

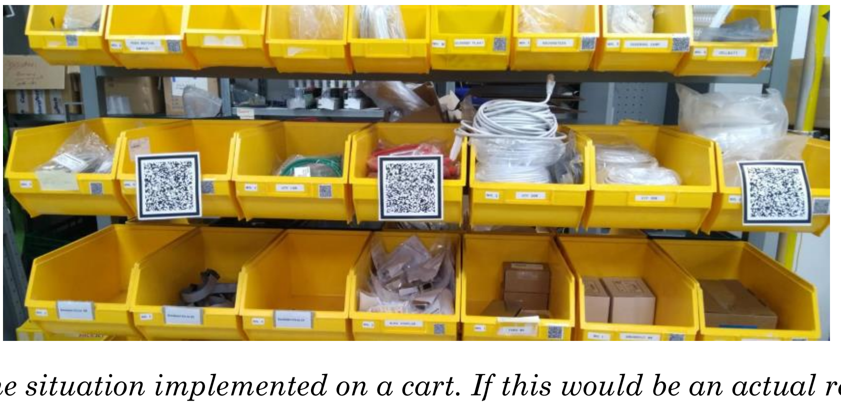


Figure 2.2: the situation implemented on a cart. If this would be an actual real situation, all bins would need an individual QR code.

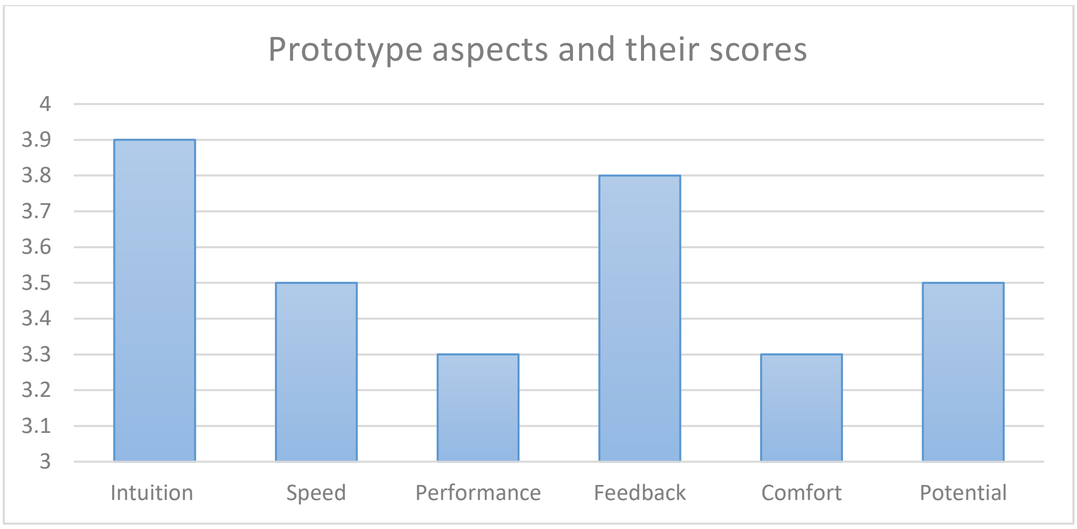


Figure 2.3: The aspects of the prototype and their respective scores.

As can be concluded from figure 2.3, performance and comfort seem to be the biggest deficiencies of the current prototype. Surprisingly, speed does not seem to be the lesser of the investigated aspects of the prototype, however this can be explained by the fact that the prototype is at a very initial state. Due to the fact that this solution is still in such an early phase, the hardware has rather easy tasks; The glasses only have to process the markers and the "pushing" of the virtual buttons. That is, there is no real tracking of the user's hands, there is no wireless connection with, for instance, a database, and the markers still need to be quite big for the glasses to recognize them. All of previously mentioned three aspects need to be improved and implemented in order to fully develop this product. These features bring a lot of additionally required processing power along, and it is not believed that the currently used Vuzix M300 glasses have the capacity to do all this. The assumption that the glasses indeed do not have the required processing power is supported by the literature review done earlier. To make sure the glasses can handle all the required processing, a technique called computation offloading can be implemented, such that some of the processing is done by a different device.

Offloading

Computation offloading for AR has seen a significant rise of attention over the last few years, and the problem of offloading has been tackled in various ways. As stated before, offloading in the context of augmented reality is considered to be a solution that utilizes nearby devices and/or remote cloud to support the computation and processing of different steps in the software system. This concept is especially used for real-time computationally heavy processes, for instance a machine learning application as marker tracking [9]. Computation offloading is said to improve the performance and reduce energy consumption of the system it is applied to [10].

As there are different ways of offloading, Lee et al. [9] identify four different categories in which methods fall: one-to-one, one-to-many, many-to-one, and many-to-many. These terms are very common in Internet of Things (IoT) applications, and are thus not only reserved for offloading, although it being in previously mentioned research that the terms indeed refer to said technique. As the four different terms might already suggest, this means that both the support and the supported devices can work in different quantities. Every term first names the supported device, i.e. the device that is being offloaded; Thereafter the quantity of the supporting devices is meant, i.e. the device that is offloading. One-to-many therefore means that one device is seeking computational support from multiple devices. A schematic showing the different concepts of IoT cross-device computational offloading can be seen in figure 2.4.

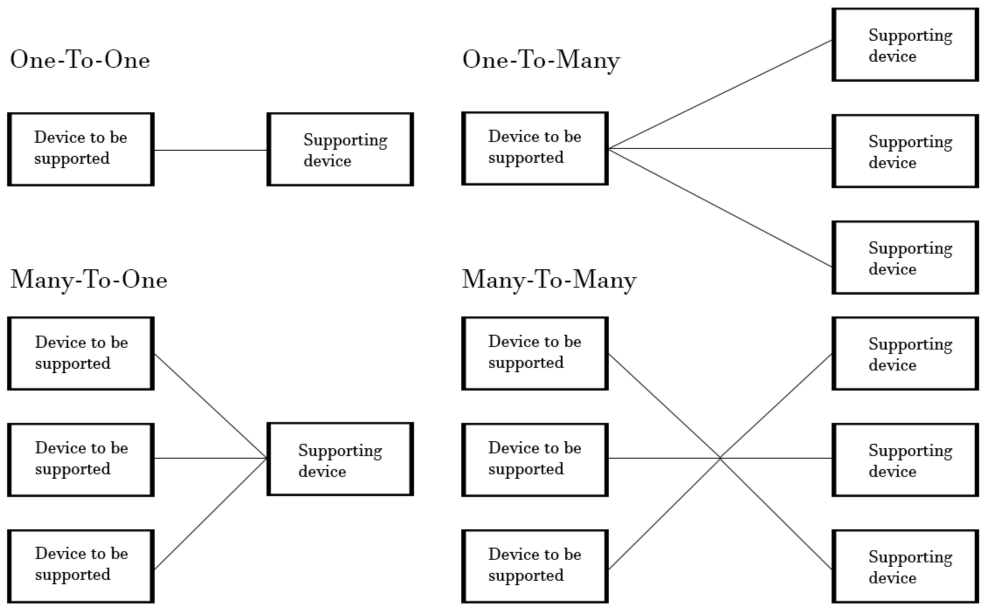


Figure 2.4: The previously mentioned four different IoT concepts in a model that shows how communication between the devices goes.

For this thesis, the main focus will be first on one-to-one communication/offloading to see whether this is effective. If it turns out to be less effective than hypothesized, the one-to-many concept might prove to be a better alternative.

As an example of offloading in AR, Ha et al. [11] developed a prototype named *Gabriel*. This prototype is based on the concept of cloudlets. As stated in the article, "*A cloudlet is a new architectural element that represents the middle tier of a 3-tier hierarchy: mobile device — cloudlet — cloud. It can be viewed as a data centre in a box*" (p. 4). The cloudlet works as follows: a mobile device that normally would do a lot of processing, is connected to the cloudlet. This cloudlet can be any device connected to the internet, like a laptop or a notebook. In turn, the cloudlet can optionally be connected to the actual cloud, where virtual machines get the processing commands from the cloudlet. The virtual machine(s) then send(s) the processed information to the cloudlet, which in turn sends it back to the mobile device. The cloudlet could also just do the processing by itself, if the device is strong enough to do so.

If the cloudlet is only connected to the device to be supported, it is called edge computing. Otherwise the process is referred to as cloud computing. Cloud computing is normally more powerful, while edge computing keeps the processed data closer to the end user [12]. As the concept of edge computing seems reliable to use and not that difficult to implement in any context, the preference goes out to this concept. On top of that, edge computing, without making the step towards cloud computing, seems to be more than sufficient for this prototype.

The mobile device can connect to different cloudlets, however the connection is always only with one, making it a one-to-one or many-to-one IoT solution. Nonetheless, if the cloudlets use the cloud to do some of the processing, all of these cloudlets are connected to the same cloud. This is in turn a Many-To-One concept. An image that visualizes the concept of cloudlets is shown in figure 2.5, as taken from the article by Bahwaireth and Tawalbeh [13]. Goal of the cloudlet is to *"bring the cloud closer"* [4] (p. 2). As a powerful, well connected and trustworthy cloud proxy or standalone, a cloudlet is said to be an excellent offload site for cognitive assistance technology, such as augmented reality [14].

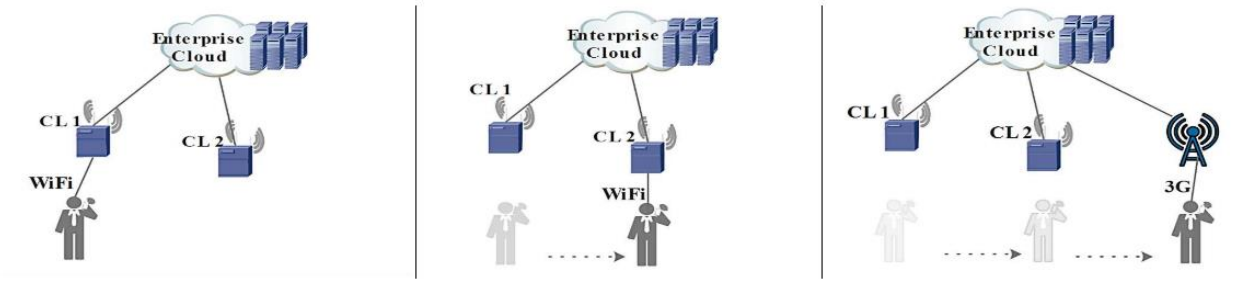


Figure 2.5: A model that visualizes the concept of cloudlets [13]. *Note how a 3G connection can also connect with the cloud, but the connection between device and 3G cannot provide any offloading on itself.*

Potential supporting devices

In order to make offloading effective for the context of order picking and see what opportunities lie in this field, a context has to be made first. As the context for this paper is warehouse order picking, there are a few things to keep in mind. First and most important is that the order picker that is using this device should be free to walk wherever they need to, together with their so-called "cart". On this cart, many different bins are placed that are used for different clients. Hence, every client has their own bin on this cart and the products are placed in the according bin. The cart is moveable, and should be, also after implementation of the proposed solution. That is why it may be more lucrative to implement the cloudlets across the warehouse, instead of implementing them on the carts, allowing for a non-depleting power source. This results in a different problem, however. Now the cloudlets are used by different users, and the users should be able to use different cloudlets. The corresponding IoT concept becomes Many-To-One, as the mobile devices should all be able to use offloading at the same place.

The Many-To-One concept extends the requirements for the cloudlet device to be used significantly, as many different people need to be able to use the computation offloading at the same time. According to Idexx, the company that was visited to sketch a context for this thesis, a maximum of nine people would be in the same area at the same time. Hence, the requirements for the processing power of the supporting device becomes nine times as high. Nevertheless, the prototype for this thesis will start with a one-to-one connection, as this should at least prove the concept to be effective. That is why for this thesis, initially only one cloudlet will be used.

Because of the fact that this thesis is focused on making a proof of concept, a device will be needed that is easy and reliable to use. The preference goes out to the Raspberry Pi 3 B+, as it is hypothesized to be a stable and easy-to-use microcontroller that may not have all the processing power that will ultimately be needed, but can proof that the concept of computation offloading works. If the concept is proven, more expensive devices with more processing power and compatibility can be looked into to increase the performance of the prototype.

Problem statement

As of now, people do see the benefit of augmented reality in order picking industry, and the prototype is well received. However, the current prototype is still missing some key functionalities that are required to fully implement this product into industry. This accumulation of process tasks calls for more processing power from the augmented reality device, which it simply does not have. Next to that, the speed of the prototype is not optimal already, and thus some form of computation offloading is desirable, if not needed.

That is why this project will revolve around finding a smart solution for the lack of processing power in wearable augmented reality devices. A prototype will be made in which the AR device communicates with an offloading device, in this case a Raspberry Pi model 3 B+. Both speed and performance will be tested for the recognition of QR codes. If time allows, this processing can be implemented into the current prototype for an AR supported batch picking system and evaluated through user tests to see if this helped the application to be faster and more stable.

III. IDEATION

Creative Technology design process

For the study Creative Technology at the University of Twente, a design process was constructed by Mader and Eggink [15]. In the paper that the two researchers wrote, it is stressed that the study Creative Technology has two major focuses: user centred design and the development of prototypes. To get the best result in both fields, Mader and Eggink developed a design process that is divided in four phases: ideation, specification, realisation and evaluation. In this study, previously mentioned design process was used in order to come up with the best argued iterations and results. Hence, for every single part a short evaluation was done to support the next choice of addition or alteration.

Stakeholders

To make a proper offloading system that can be of use for portable AR devices, and specifically for warehouse order picking, there are a couple of things to keep in mind. The most important things for a creative technologist to keep into account are practically always the stakeholders for the research. In this design research, a couple of stakeholders exist. CaptureTech Corporation B.V. is arguably the most important stakeholder, as they are currently looking into the possibility of implementing these AR devices in real world situations. However, in terms of design for this particular part of the prototype, the stakeholder to be most looked at has to be the end user: a professional warehouse order picker. The person that will be using the application to-be has to feel like the application is of considerable added value. It is of importance that the AR device is still very portable, and that the AR is providing actual real-time feedback, without noticeable latency. Next to that, the computation offloading should preferably not get in the way of starting up the rest of the application. In other words, the offloading part should ultimately be incorporated within the system and not be a standalone application. However, due to the time restrictions for this project, this might have to be done in further work.

Observations

To fully understand the context of warehouse order picking and construct properly grounded requirements for the prototype, a visit was made to IDEXX, as previously mentioned a company that among other things distributes animal pharmaceuticals. During this visit, an observation was made for both the process of warehouse order picking, and in particular batch picking, together with the environment in which this process happens. The state-of-the-art batch picking system works with a voice recognition concept, where order pickers confirm what they are doing to ensure a quite low error rate. The voice recognition concept works as follows:

- Every product is placed on a numbered shelf. The voice agent tells the order picker what shelf the product should be picked from, and what amount.
- The order picker confirms that they “pick” the product by saying the number together with “confirm”. By saying “confirm”, the system knows the message is finished.
- Then, the voice assistant tells the order picker in which bin the product(s) should be placed. This again happens with numbers, ranging from one to fifteen in this particular case.

- As expected, the order picker places the product in the correct bin and lets the assistant know they did it by saying the number of the bin and adding "confirm".
- The assistant continues by telling the order picker what the next product is.

This process of supporting order picking is, while working, not the fastest nor error-free. There seemed to be a considerable amount of time between the confirmation of the order picker and the system giving a next instruction. Hence, some time is lost in this process which does not necessarily have to be the case. This could, for instance, be solved through implementing a faster network or protocol, as the auditory agent needs to get the data from a database in order to generate an instruction. This is currently done through a Wi-Fi connection. With the rise of Wi-Fi 6 technology, that is said to be at least four times faster than the currently used Wi-Fi-5 [16], the amount of time between the order picker confirming their task and the agent giving a next one could be further diminished.

Despite the fact that the speed of the system can be increased, no extra visual feedback is currently possible with the device as is. Users of the current system need to check their own work, which results in easily missed flaws in the process. Next to that,

Development

The first step into making a prototype to prove the concept of computation offloading can be implemented has to deal with a choice: what to offload? For this, first the processes in augmented reality need to be identified.

One of the most important processes used in the current prototype [3] is image recognition. The kind of augmented reality technology that was applied uses a camera to identify visual markers or objects, such as a QR/2D codes or specialized markers, to determine positions of 3D rendered objects only when the marker is sensed by the device. This marker based technology thus fully relies on what the camera of the AR device sees. Image recognition generally takes up a lot of processing power, and it probably is the heaviest task for the device as of now. Furthermore, M. Kenny Williams in his work [3] mentioned the following:

"*In order for the prototype to become a usable product, the image recognition should be improved. At the moment, it is still too slow because picking processes happen quite fast. […] Moreover, the size of markers and the distance at which the user is standing also matters. The smaller the markers are, the better it would be for the warehouses, but it could also affect the speed of the image recognition"* (p. 54)

From the assumptions and the statement in previous work, it can be concluded that it is probably best to focus on the offloading of image recognition as a start. After implementation of this offloading, other parts should be easy to add and thus the prototype should be built in an easy adaptable way. In other words, the code that will be written for the prototype should be easily adaptable for different applications, and not too hard to combine with for instance high quality 3D rendering.

Another very important part of the prototype is rendering. Since the AR device needs to be able to show 3D objects, it needs to render them first. Depending on the hardware specifications, rendering can be heavier or less heavy for the device. As was concluded earlier, the hardware in the currently used AR device is not of the most satisfactory quality, and thus the device could use some offloading for this process. However, as of now the amount of 3D objects to be rendered simultaneously is not of substantial rate that offloading is really necessary in this part. This can be concluded due to the fact that the application does not start “lagging” increasingly when the device needs to render multiple objects.

Communication

Naturally, the communication for this computation offloading needs to be wireless, as otherwise the processing on other devices cannot be done without interfering with the portability of the AR glasses. Because of the fact that the offloading can only work when frames of the to be altered reality are sent at a reasonable “real time” pace, it is of importance that the data transmission speed is as high as possible. A protocol should be used that can transmit relatively much data in very little time, such that the application still gives real time feedback. A couple of different wireless communication technologies were evaluated as potential candidates for this project: Bluetooth; Wi-Fi; 4G; Zigbee. Of all these communication tools, Wi-Fi seems to be the best option as the maximum data transmission per second is the highest – the current standard 802.11ac namely has a maximum speed up to 1.3GBps [17]. The other standards currently have speeds that do not even come close to Wi-Fi, except 4G technology with a theoretical maximum transmission speed of 1GBps but with a much bigger gap between theoretical and actual transmission rates [18]. That is why the choice goes out to Wi-Fi for this prototype.

That being said, in the near future two upgraded technologies are expected: 5G, as the follow-up from 4G, which has a theoretical maximum of 1GBps [18] and will be upgraded to around 4GBps. The second upgrade is a new version of the 802.11 standard for Wi-Fi, 802.11ax. This Wi-Fi technology is said to have four streams of data, rather than one like the current 802.11ac [17]. The 802.11ac standard is mostly cited to have a maximum speed of 1.3GBps [17], while the 802.11ax will have a max of 3.5GBps. Combine that with the fact that it will have four times as much data streams available, and a 14GBps bandwidth is reached. Hence, this is a tremendous improvement compared to the current Wi-Fi technology.

With the upcoming advances in this field, the choice for Wi-Fi as communication tool for this prototype becomes even more sensible than before. The prototype will definitely benefit from the future transmission rates, because it can help the accuracy and pace of the application become better. Taking this in mind, it is not a failed prototype when it cannot deliver real time feedback just yet. When transmission rates increase significantly with the upcoming communication standards, the application could nonetheless come into its own if it is not yet of proper quality already.

Sending And Receiving Images To Process

The image recognition part of this prototype can only be incorporated if there is some transmission of images/frames. To do this, the VR device’s camera needs to be used. I.e. the device needs to send a video stream of what it is currently seeing to a server. On the server, the images can then be processed and sent back. To allow for sending this video stream, an Android application should be developed that can both send and receive images. To first focus on the image processing, the images were firstly sent with the help of an Android application called “IP webcam”. This application allows the device to send a live stream inside a Wi-Fi network using its own IP address. The live stream can then be found when navigating to this IP address. Through this medium, the server that will do the image processing can easily receive the frames. Moreover, the application makes it possible to stream in different qualities, which can be of great influence for the latency within the proof of concept. A lower quality image can for instance be sent over Wi-Fi faster than an image of higher quality, as the amount of data is simply a lot smaller.

Sooner or later, however, an application has to be developed for the AR device to receive the processed frames. The main reason for this is that the Android operating system – which runs on the AR device – does not allow multiple applications to use the camera. Next to that, the current *IP Webcam* application starts up a server to stream the camera. This is rather unnecessary extra work for the device as there is already a server running for the image recognition and processing. Naturally, the images to be sent cannot be the same images that are received, hence there will definitely be a delay, whether or not this is truly noticeable. For now, the *IP Webcam* can be used to develop a proof of concept which shows that image recognition can be done sufficiently on a different device than the one used for AR.

Offloading As Total Backend

For the proof of concept that is currently built, it is of importance to show the current frame together with the results of the scanned QR codes. When this prototype would be implemented in other applications, however, the response of the server could be totally different. For other applications, it would probably be more convenient to just send through the data about and the position of the QR codes, such that only the rendering of the 3D object remains. The following string results from the decoded frame if a single QR code is scanned with the text *"Hello :)"* encoded in it:

"Decoded(data=b'Hello :)', type='QRCODE', rect=Rect(left=119, top=157, width=184, height=200), polygon=[Point(x=119, y=351), Point(x=297, y=357), Point(x=303, y=164), Point(x=126, y=157)])"

If this would be sent through to the AR device, the device can render the objects on top of the QR codes by itself, knowing the position and data of the QR codes. This actually takes back the prototype a few steps in terms of complexity, which is why the choice was made to not focus on that part for the time being.

Nonetheless is this idea a good basis for further development, as it becomes much more implementable for other prototypes or actual AR applications. If in the prototype it is made possible to send through frames with overlay at a reasonable speed, it is hypothesized to send through positions and size of QR codes even faster. With the focus of this prototype being on speeding up image recognition and rendering of AR devices, together with the fact that sending through simple Strings is much easier than a live stream, it is hypothesized that deriving a solution like this from the future prototype will be relatively simple.

Image Recognition

Different languages, packages, and even devices can be used for image recognition. As OpenCV is a well-documented library of programming functions especially developed for real-time computer vision [19] and thus for image recognition, the preference goes out to using this library. If OpenCV turns out to be not specific enough for – in this case – QR codes, i.e. the library takes up a lot of time, other libraries can be looked at to improve efficiency.

The OpenCV library can be used in many different programming languages, with its primary interface being C++. There are bindings for OpenCV to run in the languages

Python, Java and MATLAB/OCTAVE. For the languages C#, Perl, Ch, Haskell, and Ruby, OpenCV wrappers have been made. This means that there is a lot of choice for the adaption of OpenCV. For rapid prototyping, Python is assumed to be a good foundation as the language allows for easy rapid prototyping due to the small nature of the code, rendering it perfect for a proof of concept. Additionally, there are a lot of different libraries for the language, which makes it rather convenient for executing different tasks for this prototype, or the same task but more efficiently. In other words, iterations can easily be made and the switch to another approach to this problem is rather doable if Python were to be used for the first prototype.

To scan a QR code that could be used in AR applications, OpenCV should know what to look for. The structure of QR codes can be found in figure 3.1 [20]. The most important parts to recognize are the blocks from 4.1, as these indicate that the presented square is a QR code. The data can afterwards be read from part 3, data and error correction keys. In other words, first the position and alignment of the QR code need to be determined, after which the data inside the code can be read.

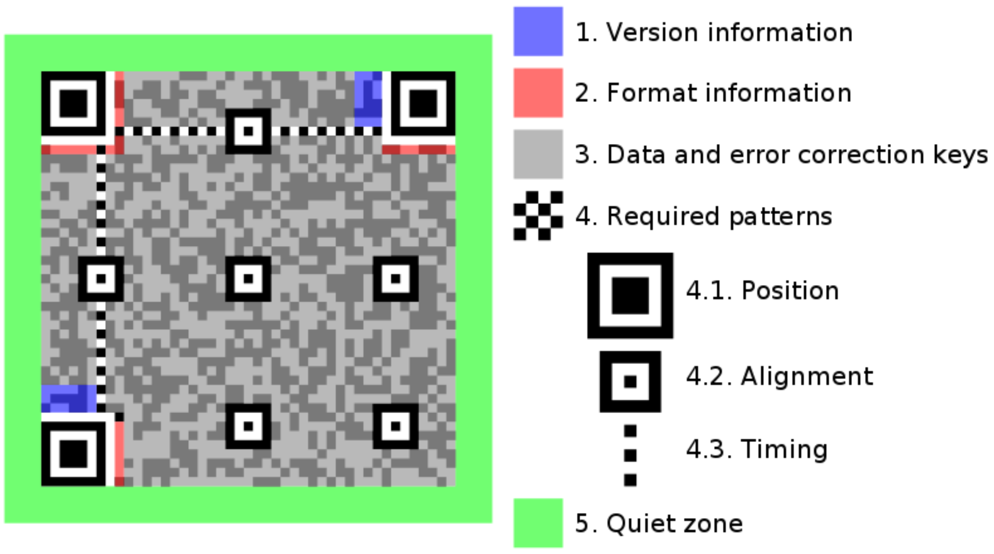


Figure 3.1: The structure of a QR code [20].

Fortunately, OpenCV has already a built in QR code detector, callable in Python with the function `cv2.QRCodeDetector()` [21]. After that, the currently loaded frame from the video stream can be called to scan by adding `.detectAndDecode(inputImage)` after `QRCodeDetector`, where `inputImage` is the current frame in this case – hence, `inputImage` is `cv2.imread(frameFromVideoStream)`. This function returns a couple of results: the data stored in the QR code, a “rectified” version of the QR code – especially handy when the code is distorted or not perfectly aligned for the camera – and the position. The data can be called when stored in a variable, and contains all the acquired data from one or multiple QR codes, depending on the amount of QR codes that is readable in the image.

User Experience

The ultimate goal of this prototype is to support an AR application in order to make it more user friendly. In other words, the focus of this project is not on user friendliness, but rather on achieving user friendliness through improved performance. Hence, while it might be the goal of this prototype, the most important part is getting it to work and proving that offloading is indeed possible via the used approach, together with evaluating if this would make the process faster. If the speed of image recognition with offloading indeed turns out to be more real time than without offloading, the prototype itself can be deemed more or less user friendly and ultimately successful. The rate of the prototype's user friendliness thus depends on how "real time" the image processing works. To give a more precise specification on what is considered real time, the following definition is used, as taken from the website TechTerms:

"When an event or function is processed instantaneously, it is said to occur in real-time. To say something takes place in real-time is the same as saying it is happening "live" or "on-the-fly." […] This means the graphics are updated so quickly, there is no noticeable delay experienced by the user." [22]

Whether or not this real time feedback can be achieved in the context for which this prototype is meant, depends on a couple of factors. Firstly, the internet that this prototype will be using to transfer frames needs to be fast enough to not already delay the process. If the Wi-Fi connection would not be fast enough, the current approach of offloading will simply not be sufficient as it will only slow down the AR application.

If sending the frames can be done quickly enough, however, the image recognition can become the next bottleneck. This can, nevertheless, be solved in numerous ways. How fast image recognition turns out to be namely depends on the speed of the code it is written in and the processing power of the device on which the program is running. Consequently, different iterations can be made in order to make this program run faster and decrease the delay.

The third hurdle in the process can become the processing of the AR device, but this is something that can unfortunately not be solved. The only option for making the AR device itself faster would be upgrading to better hardware. Hence, this is something that will not be focused on in this report, as this is considered to be solvable for anyone willing to implement the prototype in AR applications.

Interaction Diagram

For this prototype, there is limited interaction between the user and the application that will be running on the AR device. There is, however, a lot going on in the background that the user is not seeing/does not have to deal with. To visualise he concept that will be used in this thesis, an interaction diagram was made which can be found in figure 3.2. The interaction diagram was made with the online tool Sequence Diagram and can be found on the website https://www.sequencediagram.com. The commands for the resulting interaction diagram can be found in appendix C.

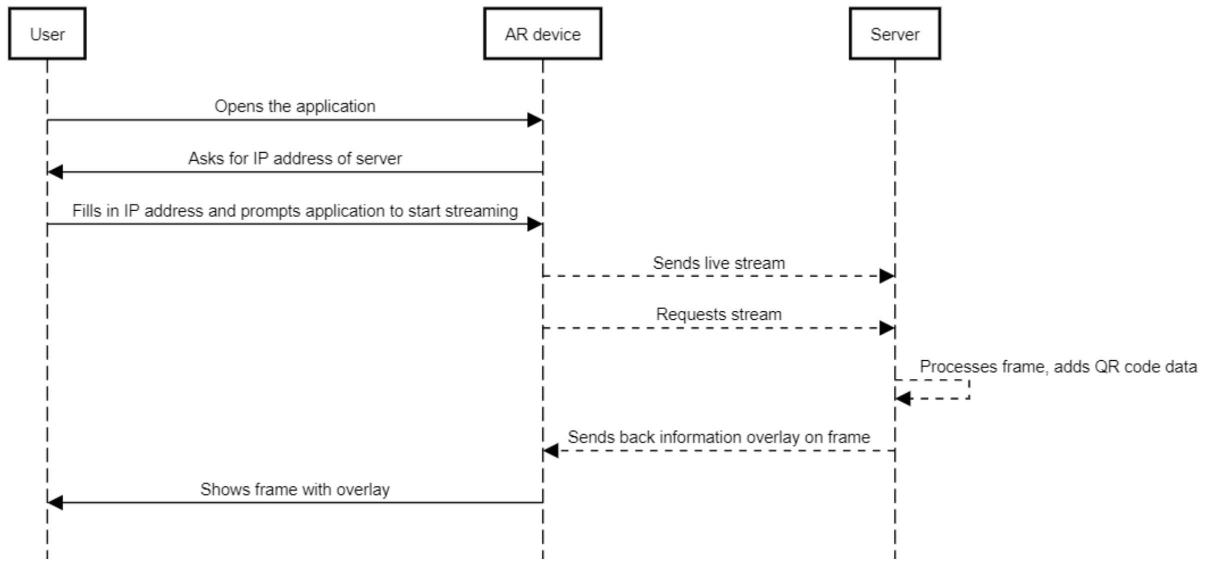

User
AR device
Server
Opens the application
Asks for IP address of server
Fills in IP address and prompts application to start streaming
Sends live stream
Requests stream
Processes frame, adds QR code data
Sends back information overlay on frame
Shows frame with overlay

Figure 3.2: Interaction diagram for the prototype.

Figure 3.2 clearly shows that both user input and feedback are very limited for this prototype. The main reason for this is that the prototype and even the end product for this concept is not supposed to be a standalone application to be used to just scan QR codes. It is rather supposed to be a supportive application that can ideally be adapted for different contexts, but is for now especially focused on warehouse order picking, and thus for instance small computers are used that can be swapped out if the context allows.

The input for an IP address will be necessary in the prototype, but this can be removed later on. The IP address namely changes while working on the prototype, as the server is booted and stopped numerous times on different networks. To change this in the code every single time seems rather redundant, especially when the application needs to be uploaded to an Android device every time. That is why, as far as the user experience goes, a very simple Android menu will be made in which the user can fill in the server's IP address. If the prototype were to be implemented in an AR application, this process would be fully working in the background, hence there would be no front-end part necessary.

Requirements

From previously discussed ideas, a list of requirements was setup for the prototype. The requirements for the prototype can be found in table 3.1. The requirements are classified in general requirements and prototype specific. Prototype specific requirements are part of the prototype that would not by and of itself be implemented in another AR application, as it is not of importance for offloading image recognition/rendering.

No.	**Requirement**	**Prototype Specific**
1	The AR device live streams camera frames which the server can access.	
2	The server decodes every frame of the live stream for QR codes.	
3	The server sends back decoded information of QR codes.	
4	The AR device is able to connect with the server in question	
5	The latency in the prototype is of such nature that the user experiences decoding QR codes as real time.	
6	The AR device shows decoded QR data to the user.	X
7	The user is able to connect with a server through its IP address.	X
8	The AR device's IP address is easy to access such that the server can be attenuated accordingly in a convenient matter.	X
9	The prototype uses small and relatively low-cost computers so that the warehouse can be filled with them without the costs getting too high	X

Table 3.1: A list of requirements for the prototype, classified in general and prototype specific necessities.

IV. IMPLEMENTATION

First Prototype

To see whether the proof of concept could work, a quick prototype was made in Python with the OpenCV for opening/editing the frames of the livestream and recognizing the QR codes. The *IP Webcam* application was used to have the live stream sent to the program, and the program simply showed the processed image, thus the images were not yet sent to a server. The code for this program can be found in appendix D. The output from the program can be found in figure 4.1.

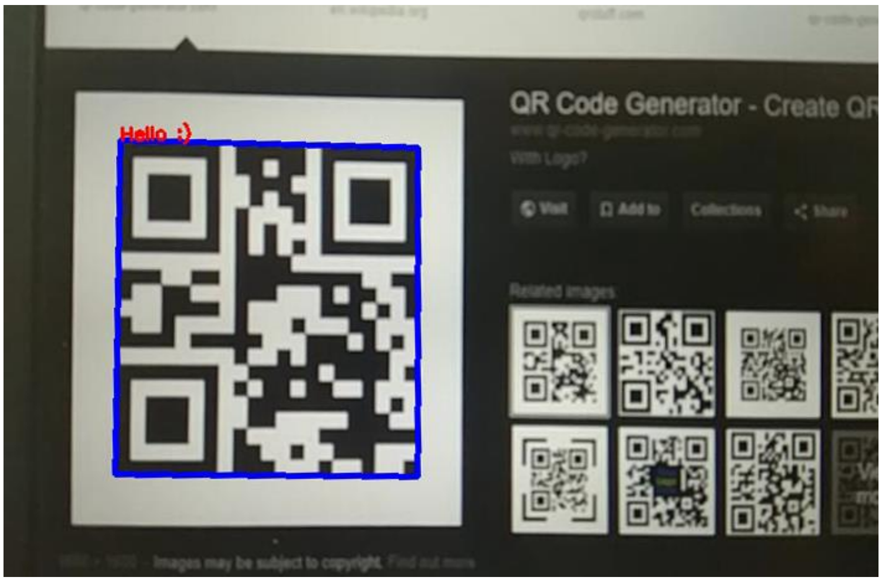

Hello :)
QR Code Generator - Create QR
Visit
Add to
Collections
Share
Related images
Images may be subject to copyright. Find out more

Figure 4.1: the result of the first prototype. The red text saying "Hello :)" is the data stored in the QR code. This could, otherwise, be an ID for a 3D object.

As can be seen in figure , the results seem to be sufficient to prove that image recognition over internet works. This first program, however, was measured to be very slow, at least too slow to work in real time, even without sending the images back over the internet to display them on the AR device. To show that this was indeed the case, the time between the processing of two frames was measured six times in a row with the help of the code that can be found in appendix E.

Each measurement was done with the same environment to be processed, namely the one that can also be found in figure 4.1. The video stream quality was set to a resolution of 640×360 pixels, which is believed to be a proper resolution for the Vuzix glassses' display as it has the exact same amount of pixels [23]. The Android device was held as still as possible, but since the actual situation would also be with a human carrying the device, it is believed that the relatively small movement of the camera should be no problem for the program. The results of the measurements for the program with OpenCV can be seen in table 4.1.

Measurement	Time between current and previous frame (s)
1	0.150447
2	0.120472
3	0.148361
4	0.160167
5	0.121904
6	0.129845
Mean	**0.138533**

Table 4.1: The measurements of time between two frames for the first prototype.

The Android device currently streamed the camera with a rate of 30 frames per second. Making the image processing actually real time would mean that the program should be able to process 30 frames per second. The mean time between two frames in this program is 0.14 seconds, which would mean that the program is able to process 1/0.139 = 7.19 frames per second. This is considered to be unacceptably slow for a real time working application. Additionally, the recognition did not seem reliable in a satisfactory fashion. The result happened to continuously flash, and this could mean that the 3D object that would ultimately be behind the QR code would be flashing as well. As earlier mentioned, this delay can be solved in numerous ways. First, the program was run on a Raspberry Pi 3 b+, which has a rather limited processing power. It is namely equipped with a quad-core 64-bit processor with a clock frequency of 1.4GHz [24] and 1 gigabyte of random access memory (RAM), which is considered to be a relatively low clock frequency. With a new Raspberry Pi model being in development, both processing power and RAM can be upgraded to achieve better results. Unfortunately the prototype could not be tested with this new model, as at the time of writing this new Raspberry Pi model 4 was not released yet. After running the program on a Lenovo Thinkpad P50, which has already a lot more processing power – namely a quad-core processor with 2.6 GHz and 16GB of RAM, as found in the devices system information, the delay was considerably diminished, with a mean time between frames of 0.0974 seconds. While this was indeed an improvement, the program was definitely still not fast enough to be considered real time. Next to that, to keep up with the video stream, the program tended to skip frames after processing a few. While the concept of skipping frames is reasonable to stay on track with speed, the delay together with skipping frames made the program rather rusty and unusable for real time image recognition. Moreover, the choice for a raspberry Pi was made for the context this prototype is made for, and thus the rest will also be done on this chosen device.

Lastly, it can be a factor that the QR code was shown on a computer screen, rather than showing a printed out one. In the context of warehouse order picking, it would be far more logical to print the QRs instead of using digital screens for the markers, and this could play a role in the programs ability to track and read the code. To make sure the programmes are not influenced by that and the results can be compared in a fair manner, all of the upcoming QR codes have been scanned from a digital screen.

Second Prototype

The most logical iteration to start with would be switching to another module or multiple other modules than just OpenCV, such that more specialized modules are used for specific tasks in the program. For QR code recognition, the ZBar library could be used, an open-source C barcode reading library with bindings for C++, Python, Perl, and Ruby [25].

Implementing ZBar was the first iteration – a rather easy one, that is, and already made the program reasonably faster. ZBar was used primarily for the QR code detection instead of OpenCV, and the code for this program can be found in appendix F. Both results were generated with the same parameters – QR code, camera and processing device –, except for the use of ZBar and a little alteration in the drawing of the box around the QR code. The latter was mainly done to make the code somewhat simpler, and is not expected to have that much influence on the program as is. Figure 4.2 shows the resulting image when incorporating the ZBar library.

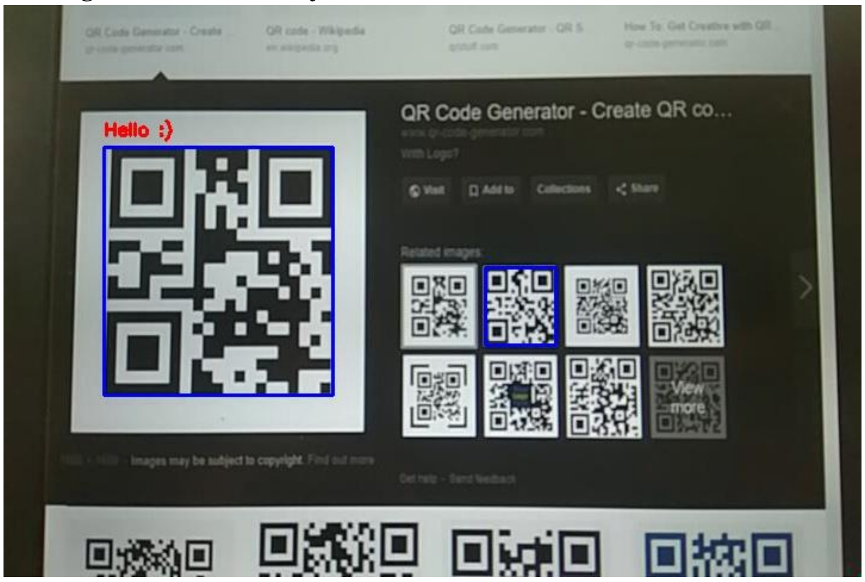


Figure 4.2: The resulting image with the ZBar library implemented.

In figure 4.2, it can immediately be noticed that the image seems to be more clear than the image made with solely the OpenCV library. This could be the case because of ZBar having a better/more powerful image processing tool built in, or because of the camera focusing better this time. Additionally, even a second QR code that is a lot smaller than the target was detected, although it does not show data stored in the QR code. This seemed odd at first, as the program is prompted to show all data from recognized QR codes. To identify the reason behind the code being recognized but data not being showed, the QR code was scanned with multiple programmes, devices and cameras. The QR code turns out to simply not have any data in the form of readable text stored inside of it. The QR code in question can be found in figure 4.3.

Figure 4.3: The QR code without any data, as seen in figure 8.

To see if this implementation was faster than the previous program, the same process of measuring as before was applied. The results of this test can be found in table 4.2.

Measurement	Time between current and previous frame (s)
1	0.06953
2	0.09444
3	0.110449
4	0.102992
5	0.129372
6	0.128882
Mean	**0.105944**

Table 4.2: The time between two frames measured for the program with ZBar implemented.

In table 4.2 it can be found that the mean time between two frames was, while still relatively high, already a bit lower than the previous measurement. With this program, 1/0.106 = 9.43 frames per second are achieved.

Although this amount of frames per second is still quite low to be considered real time, it can be concluded that the program is showing the correct data for the recognized codes at a faster rate than before. Take that together with the fact that even smaller codes are scanned than before, it can be concluded that the ZBar library is a more powerful and fitting library for recognizing and scanning QR codes than OpenCV. That being said, the program is still not fast enough to be used in real time, and that is why at least one further iteration will be necessary in order to make the program more usable.

Third Prototype

Using trial-and-error as problem solving method, different libraries were added and deleted in order to get to a more desirable result. As a result, the *imutils* library was implemented, and specifically the *Videostream* module. The imutils package is a series of convenience functions for image processing, and the Videostream module especially is well suited for processing frames from, naturally, a video stream. This especially helped with starting and processing the live stream from the Android camera, as the module seems faster than the OpenCV module. On top of that, if the stream would still not be fast enough, the choice could be made to not send the whole frame but rather a white image with an overlay of the scanned QR codes. The possible new interaction diagram with the seemingly small change can be found in figure 4.4. As the white image does not change other than the overlay it gets, it is less work for the program to process this “frame”. An example of a resulting image that will be sent to the AR device can be found in figure 4.4.

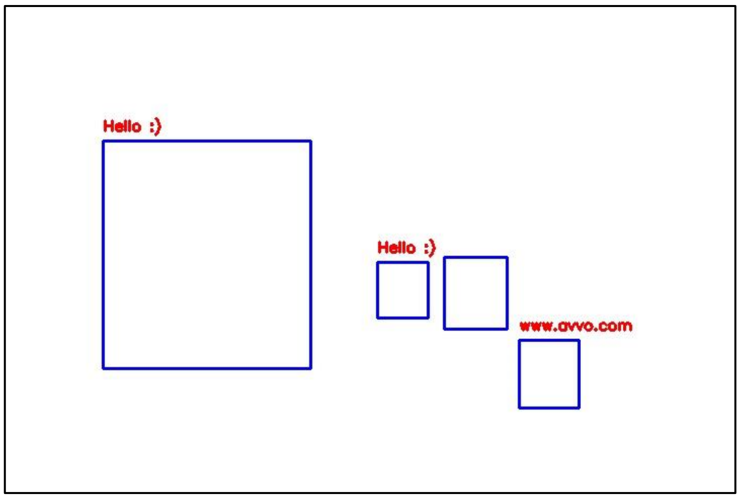


Figure 4.4: The resulting image with a "transparent" background. Now the program only has to change the resulting pixels, and not the ones from the frame.

This way, there remain two bottlenecks for the prototype, but as mentioned earlier, only one that can be solved within the prototype. The hypothesized solvable bottleneck being data transmission speed, the unsolvable being the AR devices hardware.

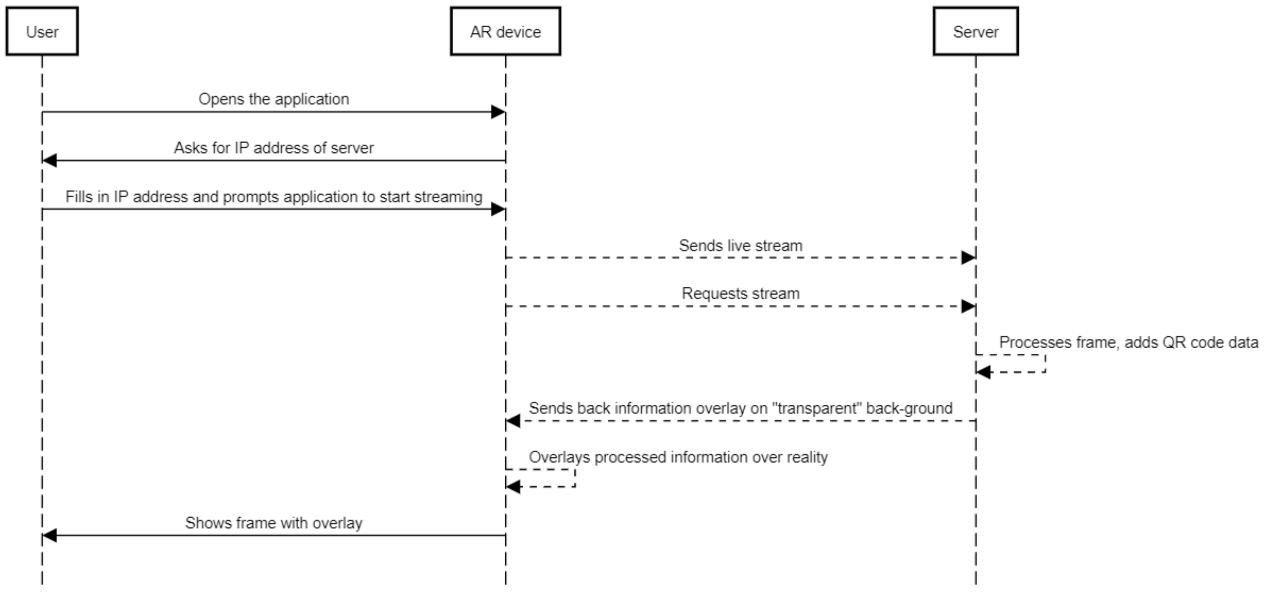

User
AR device
Server
Opens the application
Asks for IP address of server
Fills in IP address and prompts application to start streaming
Sends live stream
Requests stream
Processes frame, adds QR code data
Sends back information overlay on "transparent" back-ground
Overlays processed information over reality
Shows frame with overlay

Figure 4.5: The new interaction diagram with a seemingly small change, but with big impact on the speed of the program.

In figure 4.5, it can be seen that the AR device would now have an extra task to do: overlaying the result onto the current frame. It is hypothesized that this might turn out to be a problem, in which case this can be approached from two directions. The first option would be to simply continue with the prototype as is, assuming that more powerful AR devices are out there or still being developed. The other option would be looking into ways of taking this task off the AR device again, and trying different – more powerful – devices for the image processing part. For now, just sending through the frame with overlay from the server seems sufficient, in order to take as much processing off the AR device as possible. The code used to make this prototype can be found in Appendix G.

After implementing Imutils and the transparent background, values were measured for the time between two frames. These values can be found in table 4.3.

Measurement	**Time between current and previous frame (s)**
1	0.082027
2	0.066694
3	0.070003
4	0.088558
5	0.078358
6	0.075223
Mean	**0.076811**

Table 4.3: The time between two frames measured for the program with Imutils and a transparent background.

Without the transparent background, the program achieved the following results, as seen in table 4.4.

Measurement	**Time between current and previous frame (s)**
1	0.068021
2	0.10131
3	0.130878
4	0.097624
5	0.103178
6	0.034753
Mean	**0.089294**

Table 4.4: The time between two frames measured for the program with Imutils, without a transparent background.

With the white background instead of the frame, the program became somewhat faster, but not remarkable. This can be deducted from the values in table 4.3 and 4.4, with an average difference of 0.012 seconds per frame, which would mean a difference of less than 2 frames per second. Adding the processing the Android device now has to do, it is hypothesized to not be worth the two frames per second extra. Consequently, the choice was made to not implement the transparent background in the program.

With the highest amount of frames per second being roughly 13.0 (situation with transparent background), the program as of now cannot be deemed real time. In practice, this amount of frames per second will never be sufficient to implement the offloading in AR applications. However, with the rise of better Wi-Fi technology and possibilities to upgrade the hardware tremendously, not all hope is lost. By practicing some informed speculation about Wi-Fi technology getting better in relatively little time, the prototype can still come into its own. Furthermore, the context of the tests, while all the , may not have been ideal. The tests were all done on one wireless network, namely the University of Twente's Eduroam network. It could very well be that different networks give a better result due to their higher bandwidth and lower traffic intensity. For that reason, the prototype in its present state will not be deemed a failure, but rather one that can benefit from better circumstances with superior technologies that are out there already, or will be released in the near future.

Setting Up The Server

Now, it is time to make a server that was able to send back a livestream. For setting up something like this, Python has multiple options. For this prototype, a Flask server was setup. As the developers themselves state [26], Flask is a microframework for Python which is based on *Werkzeug* – a web application library - and *Jinja 2*, a templating language for Python. It allows for static pages and does not use up too much processing power, which of course will be needed for live image processing in this prototype. That being said, the framework is rather limited but flexible enough for this project.

To stream frames over the internet, `yield` was used within the server response. Yield is a keyword that can be used like a return, except it does not just return the frame once. It creates a generator that constantly runs, until the webpage is closed or the server does not have any frames left to show. To actually "stream" the frames, a multipart content type was used, to indicate that there are multiple frames to be received by the client. Setting up a Flask server and making sure it responds with the stream resulted in the webpage as can be seen in figure 4.5. To make sure the server is not only running on localhost but can be accessed from other devices too, the server's IP address was first retrieved by connecting to Google's public DNS, and retrieve the IP address through the socket. The code for the resulting server can be found in Appendix H.

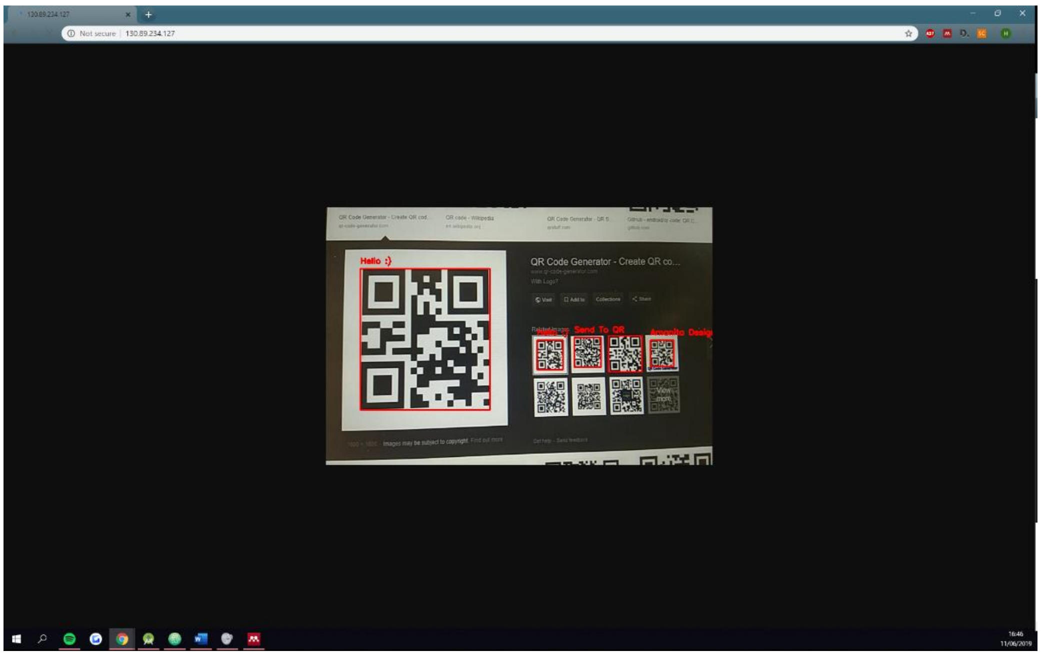


Figure 4.6: The response from the Flask server in a browser.

As can be seen in figure 4.6, no additional/unnecessary data was sent from the server, which results in a fully optimized data transmission. This way, no redundant information needs to be sent, encoded, or decoded and the program will be using its processing power in the most effective way possible for this prototype. As the stream is basically a constantly renewed image, or JPEG, the server is sending through a so-called MJPEG or Motion JPEG stream. This now only needs to be implemented in an Android application, and the offloading chain is complete.

Android Application

Because of the fact that the Vuzix M300 glasses runs on the Android mobile operating system, an application was made in Android Studio for the second part of this prototype. This developed application has the sole purpose to connect to the server and show the video stream that it is returning. As of now, it is not a necessity to develop a streaming option for the camera to a server, as the currently used *IP Webcam* app does this already. Nevertheless, if the prototype were to be further worked out, having the application to have a live stream of itself could be convenient. This could, for instance, be used for testing if sending a white image with overlay and letting the AR device process the image would be faster. Additionally, putting all essential tasks in one application instead of two could make the process substantially faster. The code for the complete Android application as is can be found in appendix I.

The application has a simple main screen, or "activity". There is one input element, where the IP address of the server can be filled in. This was currently necessary as the server was iterated and improved on different locations, making it more convenient to make the IP address of the server an input rather than something that needs to be changed in the code. When the IP address is filled in, the user can click on the "GO!" button, after which the device will switch to a different activity where the stream is shown. This stream is being received and shown through an MJPEG stream library, called *mjpeg-view*. There is one other button, called *"Find IP"*. When the user presses this button, the Android device's IP address is found and shown on the screen at the place where now the text *"IP address"* is shown. This was found to be helpful when setting up the server, but will not be very necessary in later developments of this product. The Android menu and the stream activity on a smartphone can be seen in figure 4.7 and 4.8 respectively.

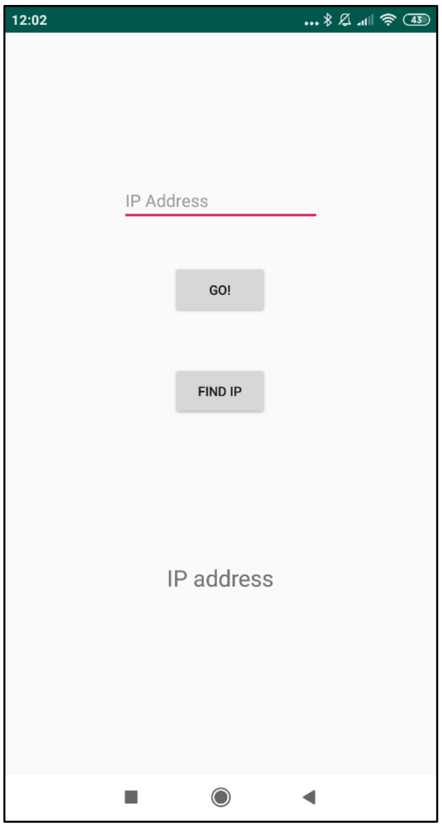


Figure 4.7: The main menu of the Android application as seen on a smartphone.

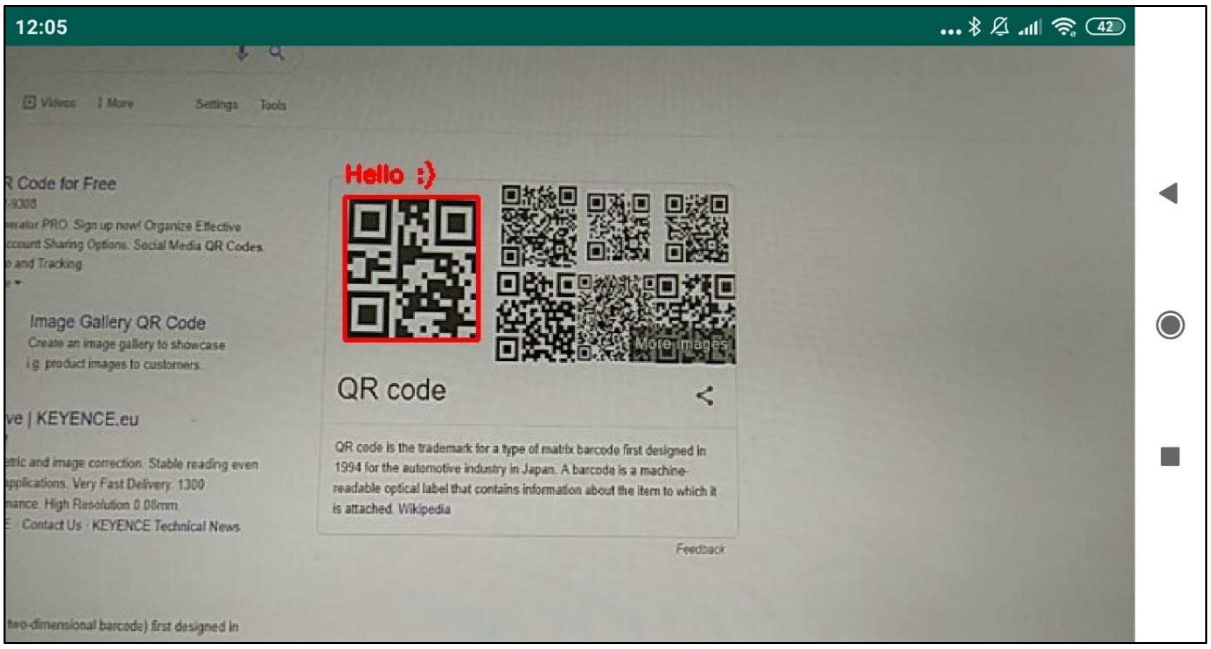
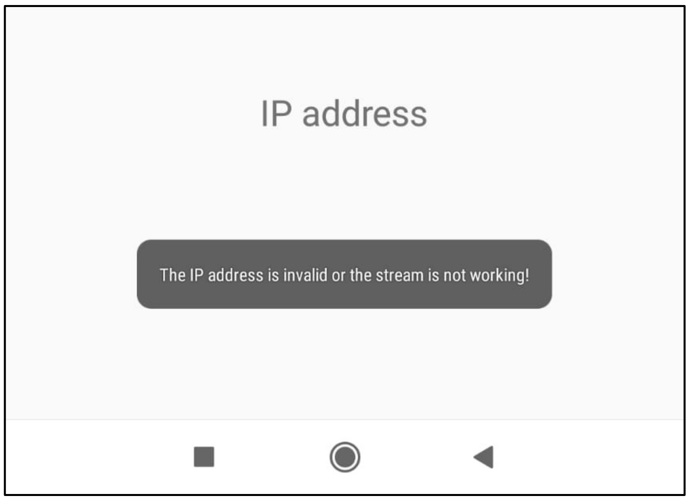


Figure 4.8: The streaming activity as shown on an Android device.

In figure 4.8, one might notices how both the status bar (top of screen) and the navigation buttons (right side of screen) are still visible. Since the prototype is just supposed to show that offloading can be done through streaming back and forth, it was deemed unnecessary to remove these screen obtrusions. Like mentioned before, the front-end of the application is not of the utmost importance, since the code for image processing would be used purely as a background process.

The streaming activity can be closed with the back button (arrow on the right side of the screen in figure 4.8) on the phone, or with the back button of the Vuzix glasses, after which the streaming automatically stops. In case the user fills in an invalid IP address or no IP address at all in the main activity, a message pops up that says *"This IP address is invalid or the stream is not working!"*. This way, the user gets feedback on their error and knows to either check the filled in IP address or the server. The server can, for instance, be checked by navigating to the corresponding IP address through an internet browser. The error message and the way it is presented can be found in figure 4.8, a cropped screenshot of the application.



The IP address is invalid or the stream is not working!

Figure 4.9: The error message that pops up if no connection can be established between the device and the server with the given IP address.

Now, the error message as seen in figure 4.9 currently pops up when there is no response from the server. That being said, if there is a response but there is no MJPEG stream, the application would theoretically continue with trying to show the stream in the corresponding activity. This could be avoided by first letting the application check on what the server is responding, but this would slow down the process of getting the stream to work in the first place. In practice, the user is hypothesized to fill in the correct IP address, and even if the user makes a typographical error, the chance of another server running on exactly that IP address is very small. And that is if the user even filled in a proper formatted IP address with the typographical error. These reasons combined is why the choice was made to not let the application check for this particular situation.

V. EVALUATION

Possible Evaluations

To evaluate how well this prototype works in a working order picking environment, different aspects of the situation need to be taken into account. First of all is that the user should be able to see a 3D rendered object from a reasonable distance, for instance at least an arm's length. That is because the application for which this prototype was made is meant to increase order picking efficiency, and so no time should be lost by actively looking at the AR markers/QR codes. Hence, a user should be able to walk to the different bins and immediately see what bin the product should go into.

The next important variable of the prototype is the stream quality. The *IP Webcam* Android application that was used for his proof of concept allows for different settings in terms of stream quality, which can make a substantial difference when it comes to transmission speed. As expected, the lower the quality of the stream, the faster the images can be processed and sent back. A lower quality, however, could mean that the program experiences more difficulty recognizing and decoding the QR codes. Additionally, a lower stream quality means a lower resolution for the user to see, which could cause confusion or frustration while looking through the screen of the AR glasses. Finding a proper equilibrium for this variable could very well elevate the prototype to a higher level, making the possible application and/or further development for this concept more likely.

As Mark Kenny Williams mentioned in his thesis about the AR supported batch picking system, the size of the markers is an important factor in both the recognition and user friendliness of the prototype [3]. In his report, he states the following about the QR codes that he used:

"[…] the size of markers and the distance at which the user is standing also matters. The smaller the markers are, the better it would be for the warehouses, but it could also affect the speed of the image recognition." (p. 54)

What can be taken away from this citation is that preferably the QR codes need to be as small as possible to let them fit on the order picking cart without them getting in the way of the worker. As can be seen in figure of the corresponding previously conducted user test, the QR codes of the previous prototype were rather large. The image recognition of that prototype was acceptable but not excellent, with the maximum possible distance for image recognition being heavily dependent on the light intensity [3].

Consequently, different markers can be tested for this new form of image recognition to see if, with the new prototype, it is possible to make small enough QR codes with a reasonable fast image quality stream, such that these codes are not in the way of the order picking process anymore.

Lastly, different hardware options could also be a variable to test. Better hardware could, for instance, mean faster image processing and last latency. Finding a proper computer for the server side would make the prototype even better. However, since the choices for this prototype were made based on the context where it is meant for, an order picking warehouse, this is a factor that is preferred to not deviate from the hypothesis too much. In other words, looking at different kinds of hardware is preferred to only do so if evaluation deems it necessary.

Test Setup

It was decided to test different video resolutions for different sizes of QR codes, and the maximum distance at which these QR codes were still readable. As earlier concluded, the system as is will not be fast enough to implement in real time, and while the program can benefit from better and more advanced technologies, testing the speed of the current prototype was found to be redundant. By testing different QR code sizes and different video qualities, a conclusion can be drawn for at least the performance of the image recognition.

The test involved the following setup: the prototype running on an Android device, with three QR codes in front of it. These three QR codes were identical except for their sizes: the first one being 13x13cm, the second one 6.5x6.5cm and the last and smallest 3.25x3.25cm. In other words, three QR codes were used where the second and third ones had half the dimensions of their respective predecessor. The QR codes as used in the test can be found in figure 5.1. The QR code is the same as the one constantly used in chapter IV, with the text *"Hello :)"* behind the code.

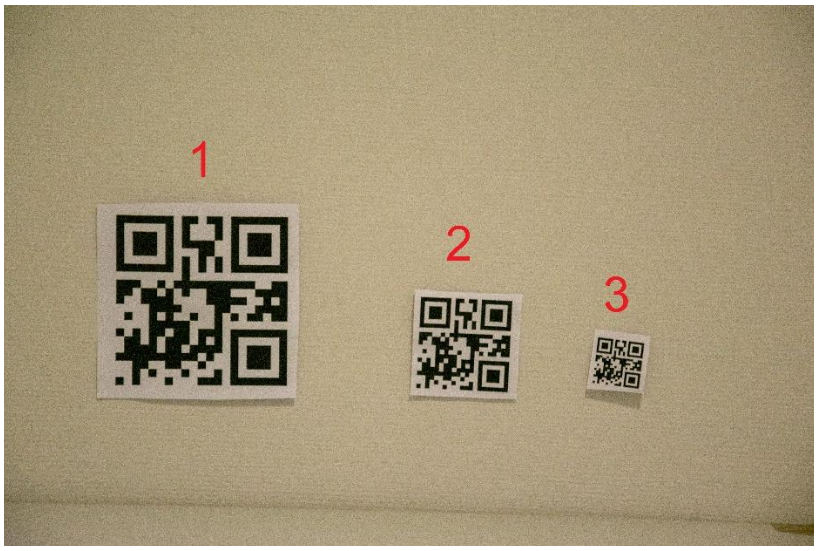


Figure 5.1: The QR codes as used in the evaluation.

These QRs were hung up on the wall and a tape measure was installed next to them. Then, the Android device was placed in front of the QR code, measuring the maximum distance the resolution allowed the server to recognize the QR. This could then be seen in the Android application. If the QR code's result started flashing in the application after the Android device was not moved for at least three seconds, the distance from the QR code was measured and deemed to be the maximum. After this was done for all three QR codes individually, the resolution of the stream was increased by one step. The *IP Webcam* application allowed for 19 different resolutions, which means a total of 57 measurements were conducted. A couple of tables were used in order to get the Android device on a proper height without needing a person to hold it, such that human motion did not play a role in this test. The Android device was chosen to not be the AR glasses specifically, because a device with a bigger screen made the observation of the flashing results more reliable, especially with the Android device put on a table. It was confirmed, however, that the application can run on the Vuzix M300 glasses without any complication. There is no difference between the used Android device and the AR glasses, since both cameras allow for a 1080p video stream [23], [27]. In figure 5.2, the complete test setup can be found.

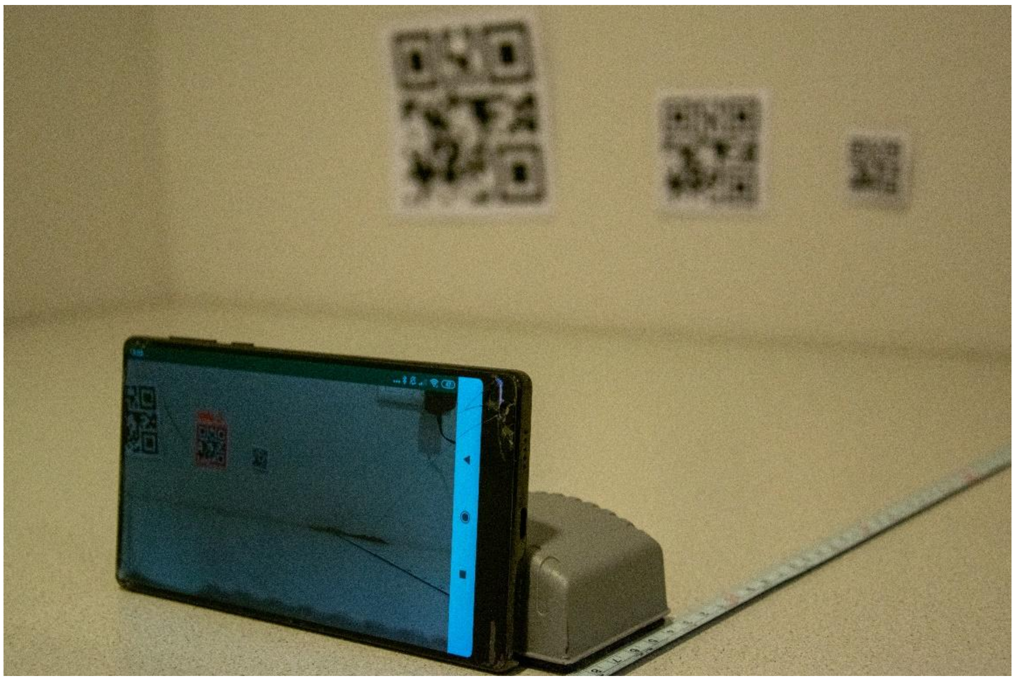


Figure 5.2: The complete setup with an Android device (left) and tape measure (right).

Collected Data

By measuring the maximum distance per QR code per video resolution, the data as shown in table 5.1 were acquired. Every distance in the table is given in centimetres, and the distances were rounded off to the nearest 0.5 centimetre.

Video Resolution	QR 1 (cm)	QR 2 (cm)	QR 3 (cm)
176×144	50	31.5	16
240×160	79	45	20.5
320×240	103	61	28
352×288	124	68	32.5
384×288	124	68	32.5
480×320	135	80	38
480×360	135.5	80	38.5
576×432	151.5	91	41
640×360	170	93.5	45
640×480	178	100.5	48
640×640	201.5	107.5	51.5
720×480	214	114	52
768×432	214	114	52
800×480	214	114	52
864×480	214	114	52
1280×720	219.5	120	58.5
1280×960	220.5	122	58.5
1440×1080	220.5	122.5	58.5
1920×1080	225	124	60

Table 5.1: The maximum distances the QR codes were still readable for the program per resolution.

The results as found in table 5.1 were plotted in a graph, with the resolution on the horizontal axis and the distance on the vertical axis. The graph can be found in figure 5.3.

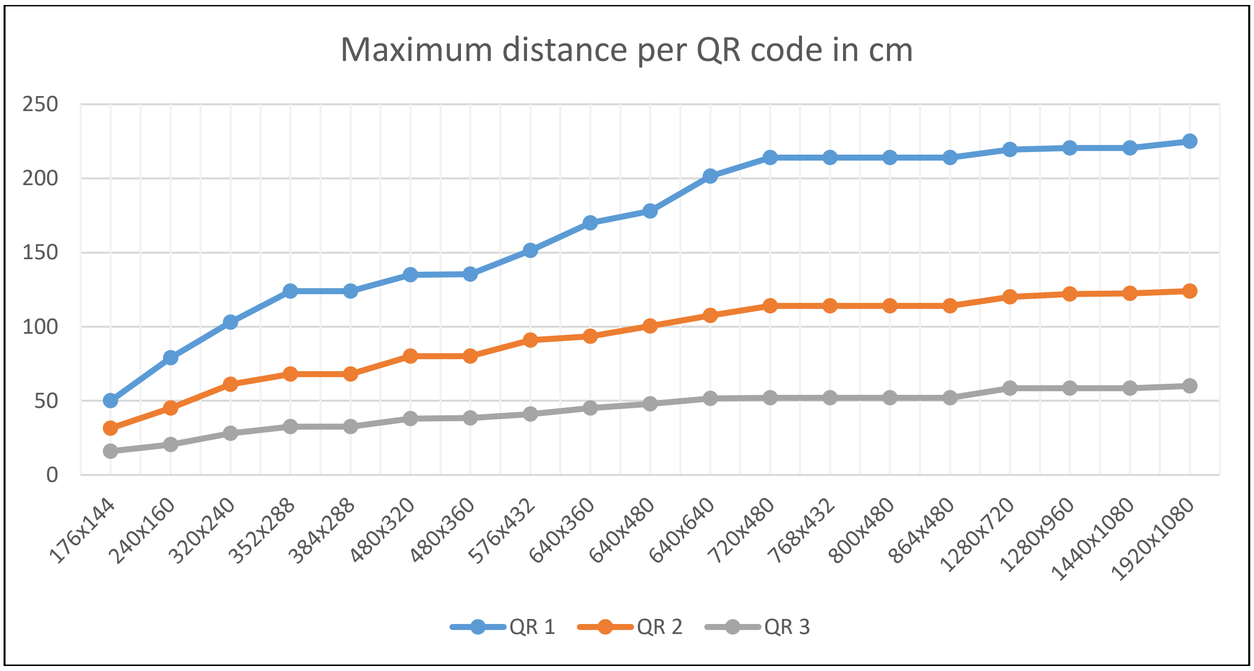


Figure 5.3: The graph that results from the collected data.

Discussion

The results show that there is a clear relation between the size of the QR code, the used resolution and the maximum distance at which the QR code is recognizable. In both the graph and the table can be seen that QR code 1 is definitely the best recognizable one, and seems to have the biggest growth over higher resolutions. That being said, QR 1 is 13x13cm, which could be too big for an order bin to not hinder the order picker that would be in the process of order picking. QR code 2 is relatively well recognizable, at least at an arm's length starting at a resolution of 480×320 pixels. As earlier mentioned, the AR glasses' screen has a resolution of 640×360 pixels, and if that exact resolution is used the second QR code is recognizable from almost a metre away. This is arguably far enough for the order picker to know where the product needs to end up without having to stand too close to it, as the maximum distance of 93.5 cm is more than an average human arm's length. That distance is namely said to be around 65 cm [28]. With the AR application's focus being on error preventing rather than speeding up the process, this distance is considered adequate for QR code recognition, especially since it is believed to not slow down the process of order picking in any way.

It is interesting to see that starting from a resolution of 720×480 pixels, the maximum distance does not increase that much anymore. This means that every resolution higher than 720×480 does not add that much to the maximum distance while the amount of processing power required increases quite a bit. Consequently, it might not be worth to use resolution higher than 720×480 pixels.

Evaluation Conclusion

From the concluded test and discussion afterwards, it can be concluded that computational offloading has the potential to help out AR devices struggling with image recognition due to their limited processing power. In general the prototype proved to be promising but not ready for implementation just yet. The question, however, is not if the prototype will ever be ready, the question is how long it will be before technology advances far enough for computational offloading to become faster and more effective.

A QR code of 6.5×6.5cm is considered to be big enough for the prototype to recognize it at a reasonable distance. Video resolution might be a matter of preference, but the AR device's resolution of 640×360 pixels could already be enough. With a QR code of this size, it is assumed that the code can be placed on the bin without hindering the user in the process of order picking.

There are, however, a few aspects of this test that need to be taken into consideration. Firstly, the measurements were done with a tape measure and the human eye, which means this test could have some deviations if done multiple times due to measurement insecurities. Furthermore, while the test was conducted under controllable circumstances without natural occurrences like daylight interfering, the light that was used might not be of the exact same intensity and frequency as in a warehouse. That is why in practice the achieved results could very well deviate from the results of the test.

VI. CONCLUSION

What can be concluded from literature research, realization and evaluation is that offloading is a powerful tool that, while not perfect, continues to benefit from emerging technologies. The current prototype might not be ready to be implemented for AR applications just yet, but can in the near future be the exact kind of technology augmented reality and other image recognition reliant technologies need. As the prototype is only a proof of concept, and the aim of this thesis was to find a smart solution to the limiting processing power and speed of AR devices, the prototype is regarded proof that computation offloading can be implemented in AR technology.

The speed and performance of the prototype depend on many different factors. Wi-Fi, marker size, light intensity and video resolution all influence the different parts of this product. That being said, all these factors can be humanly controlled and do not have to be a problem as such, with the exception for Wi-Fi technology. The biggest problem with the current prototype is regarded to be the limiting bandwidth of state of the art wireless internet technology. It is only to be expected that as time goes by, this limitation will naturally become less of an issue. Moreover, hardware also gets increasingly better, with processing power and portability of hardware developing. It is up for debate if the growth of processing power and flexibility of hardware is going to outrun Wi-Fi technology in its application potential. If this were indeed the case, the prototype could be considered redundant, as computation offloading would not be necessary. Since this is not expected to become reality, however, the prototype can be considered a success provided that internet technology develops further.

The choice for using Wi-Fi was deduced to indeed be the best way of communication, especially since the prototype as is still struggles to achieve a proper amount of frames per second. The bandwidth of Wi-Fi with the 802.11ax standard could tremendously help the prototype come more into its own than it currently does. This technology is expected to be around soon, which could elevate the proof of concept to a whole new level.

To give a concise answer to the research question "*In what way can computation offloading for augmented reality devices be implemented in warehouse order picking applications*", the following explanation was formed: Computation offloading for augmented reality devices in warehouse order picking applications can be a very effective way to overcome the limiting processing power and speed of AR devices, given that Wi-Fi technology adequately evolves in the near future. With the help of Raspberry Pi's a network of offloading devices can be built, such that the amount of data hops remains small but AR devices can get the computation support they need in order to work both in real time and in a reliable manner.

VII. DISCUSSION AND FUTURE RESEARCH

The proposed method of computation offloading is considered promising, however there is still a lot that needs to be improved. In the realization and evaluation parts, already some of its current limitations were highlighted. Some of these limitations can be solved by further research, other ones are simply a matter of time before they improve at the hand of technological developments.

The prototype is implementable for AR devices and applications, and while not fast enough, future research is highly encouraged to test this prototype in a real AR based environment, like the AR Supported Batch Picking System. This way, maybe new obstacles are disclosed which need to be overcome before applying this technique to industrial augmented reality applications.

Furthermore, the prototype currently works with an external Android app that allows for live video streaming. This background process should preferably be built into the developed application to optimize processing power and connection speed. Additionally, Future work could also focus on optimizing the speed of the supporting device. This could be achieved by effectively using multithreading, or in other words dividing work over multiple processor cores. Because of the fact that the used Raspberry Pi model has multiple cores, there might be a lot to gain in terms of processing power.

Lastly, the current AR batch picking system is not ready to be implemented in warehouse order picking just yet. Next to implementing computation offloading fully into the prototype, other parts of the product like implementing a database would be of great value for further development of this proof of concept.

IV. REFERENCES

[1] J. P. van der Gaast, R. B. M. de Koster, and I. J. B. F. Adan, "Optimizing product allocation in a polling-based milkrun picking system," *IISE Transactions*, vol. 0, no. 0, pp. 1–15, 2019.

[2] T. Le-duc and K. J. Roodbergen, *Design and Control of Warehouse Order Picking : a literature review René de Koster , Tho Le-Duc and Kees Jan Roodbergen REPORT SERIES*, no. January 2006. .

[3] M. K. Williams, "Augmented Reality Supported Batch Picking System," 2019.

[4] B. Shi *et al.*, "Offloading Guidelines for Augmented Reality Applications on Wearable Devices," pp. 1271–1274, 2016.

[5] J. W. Meulstee *et al.*, "Toward Holographic-Guided Surgery," *Surgical Innovation*, vol. 26, no. 1, pp. 86–94, 2019.

[6] M. Akçayır and G. Akçayır, "Advantages and challenges associated with augmented reality for education: A systematic review of the literature," *Educational Research Review*, vol. 20, pp. 1–11, 2017.

[7] R. Palmarini *et al.*, "A systematic review of augmented reality applications in maintenance," *Robotics and Computer-Integrated Manufacturing*, vol. 49, no. July 2017, pp. 215–228, 2018.

[8] Z. He *et al.*, "Progress in virtual reality and augmented reality based on holographic display," *Applied Optics*, vol. 58, no. 5, p. A74, Feb. 2019.

[9] H. Yoon and C. Shin, "Cross-Device Computation Coordination for Mobile Collocated Interactions with Wearables.," *Sensors (Basel, Switzerland)*, vol. 19, no. 4, p. 796, 2019.

[10] A. Toma and J.-J. Chen, "Computation offloading for embedded systems," 2013, no. December 2015, p. 1650.

[11] K. Ha *et al.*, "Towards wearable cognitive assistance," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services - MobiSys '14*, 2014, pp. 68–81.

[12] Z. Lamb and D. Agrawal, "Analysis of Mobile Edge Computing for Vehicular Networks," *Sensors*, vol. 19, no. 6, p. 1303, 2019.

[13] K. Bahwaireth and L. Tawalbeh, "Cooperative Models in Cloud and Mobile Cloud Computing," no. October 2017, 2016.

[14] I. Fajnerová *et al.*, "Could Prolonged Usage of GPS Navigation Implemented in Augmented Reality Smart Glasses Affect Hippocampal Functional Connectivity?," *BioMed Research International*, vol. 2018, 2018.

[15] A. Mader and W. Eggink, "A Design Process for Creative Technology," *Proceedings of the 16th International Conference on Engineering and Product Design Education; University of Twente*, no. September, pp. 1–6, 2014.

[16] "5G vs. Wi-Fi | Latest Standards Compared and Why We Need Both | Digital Trends." [Online]. Available: https://www.digitaltrends.com/mobile/5g-vs-wi-fi/. [Accessed: 17-May-2019].

[17] "What is 802.11ax Wi-Fi, and what will it mean for 802.11ac | Network World."

[Online]. Available: https://www.networkworld.com/article/3258807/what-is-802-11ax-wi-fi-and-what-will-it-mean-for-802-11ac.html. [Accessed: 12-Jun-2019].

[18] "How Fast Is 5G Vs 4G?" [Online]. Available: https://thedroidguy.com/2019/05/how-fast-is-5g-vs-4g-1084299. [Accessed: 17-May-2019].

[19] K. Pulli *et al.*, "Realtime Computer Vision with OpenCV," pp. 1–17.

[20] "File:QR Code Structure Example 2.svg - Media Commons." [Online]. Available: https://commons.wikimedia.org/wiki/File:QR_Code_Structure_Example_2.svg. [Accessed: 21-May-2019].

[21] "QR Code Scanner using OpenCV 4 (C++ &amp; Python) | Learn OpenCV." [Online]. Available: https://www.learnopencv.com/opencv-qr-code-scanner-c-and-python/. [Accessed: 21-May-2019].

[22] "Real-Time Definition." [Online]. Available: https://techterms.com/definition/realtime. [Accessed: 22-May-2019].

[23] Vuzix, "M300 Smart Glasses Hands-Free Mobile Computing," 2016.

[24] "Raspberry Pi 3B+ Specs and Benchmarks - The MagPi MagazineThe MagPi Magazine." [Online]. Available: https://www.raspberrypi.org/magpi/raspberry-pi-3bplus-specs-benchmarks/. [Accessed: 28-Jun-2019].

[25] "Index of downloads: zbar." [Online]. Available: https://linuxtv.org/downloads/zbar/. [Accessed: 24-May-2019].

[26] "Welcome | Flask (A Python Microframework)." [Online]. Available: http://flask.pocoo.org/. [Accessed: 11-Jun-2019].

[27] "Xiaomi Mi Mix - Full phone specifications." [Online]. Available: https://www.gsmarena.com/xiaomi_mi_mix-8400.php. [Accessed: 19-Jun-2019].

[28] "Average Male and Female Dimensions." [Online]. Available: https://www.firstinarchitecture.co.uk/average-male-and-female-dimensions/. [Accessed: 20-Jun-2019].

APPENDIX A: QUESTIONNAIRE

Please read the following statements and indicate how much you agree with each statement. (1 = Strongly disagree, 5 = Strongly agree)

1. The system is intuitive and I understood it immediately. (Intuition)
2. The icons, as shown during the process, are clear in what they are representing. (Intuition)
3. The system fits well in the regular process of batch picking. (potential)
4. The system is fast enough to support the process of batch picking. (Speed, performance)
5. The markers are well enough recognized by the system. (Speed, Performance)
6. The size of the markers does not bother me. (Performance)
7. The arrows that indicate the correct bin give a good guidance. (Intuition)
8. The counter that indicates the amount of products to be picked is clear and intuitive. (Intuition, feedback)
9. The confirmation when successfully picking all the products is clear. (feedback)
10. The error indication when putting items in the wrong bin is clear enough to prevent me from making mistakes. (intuition, feedback)
11. The accuracy of the device while tracking my hands was good. (Speed, performance)
12. Augmented reality could be a good technology to support batch picking. (potential)
13. The augmented reality glasses are comfortable to wear during batch picking. (comfort)
14. The fact that the screen of the augmented reality glasses only uses half of my field of view is a good thing. (comfort)

Do you have any remarks on the study or the product? (optional)

APPENDIX B: CONSENT FORM

Consent Form for the study on User Experience Of Augmented Reality In The Context Of Warehouse Order Picking

Please tick the appropriate boxes/mark what is applicable

Taking part in the study

I have read and understood the study information dated 08-04-2019, or it has been read to me. I have been able to ask questions about the study and my questions have been answered to my satisfaction.	☐
I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and that I can withdraw from the study at any time, without having to give a reason.	☐
I consent to being photographed during the study.	Yes No
I consent to it that taken photographs of me can be used in the study report or relevant presentations, guaranteed that I am not recognizable in any way.	**Yes No**

Use of the information in the study

I understand that information I provide will be used for a Bachelor of Science thesis on the user experience of augmented reality and how to improve this.	☐
I understand that personal information collected about me that can identify me, [e.g. my name or where I live], will not be shared beyond the study team.	☐

Future use and reuse of the information by others

I give permission for the survey answers that I provide anonymously to be archived in a Bachelor of Science Thesis, so it can be used for future research and learning.	☐
I agree that the data I provide anonymously may be shared with other researchers for future research studies that may be similar to this study or may be completely different. The information shared with other researchers will not include any information that can directly identify me. Researchers will not contact me for additional permission to use this information.	☐

Signatures

______________________________	_______________________	08/04/2019
Name of participant	Signature	Date

I have accurately read out or shown the information sheet to the potential participant and, to the best of my ability, ensured that the participant understands to what they are freely consenting.

______________________________	_______________________	08/04/2019
Researcher name	Signature	Date

Study contact details for further information

Harald Eversmann, h.y.n.eversmann@student.utwente.nl

Contact Information for questions about your rights as a research participant

If you have questions about your rights as a research participant, please contact the Secretary of the Ethics Committee of the Faculty of Electrical Engineering, Mathematics, and Computer Science at the University of Twente by ethics-comm-ewi@utwente.nl.

APPENDIX C: SEQUENCE DIAGRAM COMMANDS

The first interaction diagram:

```
User -> AR device: Opens the application

AR device -> User: Asks for IP address of server

User -> AR device: Fills in IP address and prompts application to
start streaming

AR device --> Server: Sends live stream

AR device --> Server: Requests stream

Server --> Server: Processes frame, adds QR code data

Server --> AR device: Sends back information overlay on frame

AR device ->User: Shows frame with overlay
```

The second interaction diagram:

```
User -> AR device: Opens the application

AR device -> User: Asks for IP address of server

User -> AR device: Fills in IP address and prompts application to
start streaming

AR device --> Server: Sends live stream

AR device --> Server: Requests stream

Server --> Server: Processes frame, adds QR code data

Server --> AR device: Sends back information overlay on "transpar-
ent" background

AR device --> AR device: Overlays processed information over reality

AR device -> User: Shows frame with overlay
```

APPENDIX D: FIRST PROTOTYPE

```
#import OpenCV
import cv2

#Get the live stream
cap = cv2.VideoCapture('http://130.89.233.238:7777/video')

#if a QR code is found, put a square around it and show the data it has
def display(im, bbox):
    n = len(bbox)
    for j in range(n):
        cv2.line(im, tuple(bbox[j][0]), tuple(bbox[ (j+1) % n][0]), (255,0,0), 3)
        text = "{}".format(data)
        cv2.putText(im, text, tuple(bbox[0][0]),
              cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 255), 2)
while(cap.isOpened()):
    ret, inputImage = cap.read()
    inputImage = cv2.resize(inputImage, (720,480))
    # Import the QRcodeDetector from OpenCV
    qrDecoder = cv2.QRCodeDetector()
    # Gain all data from the detected QR codes, including position
    data,bbox,rectifiedImage = qrDecoder.detectAndDecode(inputImage)
    if len(data)     > 0:
    # Display QR code location and data
        display(inputImage, bbox)
    # press q in the window to close it
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    # Show the resulting frame, with or without QR code
    cv2.imshow('frame', inputImage)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

APPENDIX E: CODE FOR MEASURING TIME BETWEEN FRAMES

```
import time

currentTime = float(time.time())

counter = 0

…

      if counter % 2 == 0:

            currentTime = float(time.time())

            difference = previousTime - currentTime

            print(difference)

      previousTime = currentTime
```

APPENDIX F: SECOND PROTOTYPE WITH ZBAR

```
import cv2
from pyzbar import pyzbar

#Get the live stream
cap = cv2.VideoCapture('http://130.89.233.238:7777/video')

# Display barcode and QR code location
def detect_qr(frame):
    barcodes = pyzbar.decode(frame)
    # loop over the detected barcodes
    for barcode in barcodes:
        (x, y, w, h) = barcode.rect
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
# Gain all data from the detected QR codes, including position
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        text = "{}".format(barcodeData)
# Put the data as text above the QR code
        cv2.putText(frame, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0,
255), 2)
while(cap.isOpened()):
    ret, inputImage = cap.read()
# Resize the image to have a clear overview
    inputImage = cv2.resize(inputImage, (720,480))
    detect_qr(inputImage)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    cv2.imshow('frame', inputImage)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

APPENDIX G: THIRD PROTOTYPE WITH VIDE-OSTREAM AND WHITE BACKGROUND

```
from imutils.video import VideoStream

import cv2

from contextlib import closing

import requests

from pyzbar import pyzbar


# Open the livestream

vcap = VideoStream('https://130.89.235.71:7777/video').start()

def get_image():

    while(True):

        # Capture frame-by-frame

        frame = vcap.read()

        transparent = cv2.resize(transparent, (720, 480))

        frame = cv2.resize(frame, (720, 480))

        if frame is not None:

            # Detect qr code and draw a box around it + show data of the qr code

            barcodes = pyzbar.decode(frame)

                # loop over the detected barcodes

            for barcode in barcodes:

                (x, y, w, h) = barcode.rect

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

                barcodeData = barcode.data.decode("utf-8")

                barcodeType = barcode.type

                text = "{}".format(barcodeData)

                cv2.putText(frame, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 255), 2)

            #Display the resulting frame at a normal size

            cv2imshow('frame', frame)

cv2.destroyAllWindows()
```

APPENDIX H: IMAGE PROCESSING SERVER CODE

```
import socket, select

import requests

from flask import Flask, render_template, send_file, Response

from imutils.video import VideoStream

import cv2

from contextlib import closing

import requests

from pyzbar import pyzbar


app = Flask(__name__)


s= socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

s.connect(("8.8.8.8", 80))

ipadr = s.getsockname()[0]

s.close()

# Open the livestream

vcap = VideoStream('https://130.89.235.71:7777/video').start()

def get_image():

    while(True):

        # Capture frame-by-frame

        transparent = cv2.imread('transparent.png')

        frame = vcap.read()

        transparent = cv2.resize(transparent, (720, 480))

        frame = cv2.resize(frame, (720, 480))

        if frame is not None:

            # Detect qr code and draw a box around it + show data of the qr code

            barcodes = pyzbar.decode(frame)

                # loop over the detected barcodes

            for barcode in barcodes:

                (x, y, w, h) = barcode.rect

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

                barcodeData = barcode.data.decode("utf-8")

                barcodeType = barcode.type

                text = "{}".format(barcodeData)

                cv2.putText(frame, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 255), 2)
```

```
            #Display the resulting frame at a normal size

            webFrame = cv2.imencode('.jpg', frame)[1].tobytes()

            yield(b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' + webFrame +
b'\r\n\r\n')

@app.route('/string')

def string():

    return "Wow ik doe het."


@app.route('/')

def index():

    # if a normal request is made, initiate the image processing

    return Response(get_image(), mimetype='multipart/x-mixed-replace; bound-
ary=frame')

cv2.destroyAllWindows()


if __name__ == '__main__':

    app.run(ipadr, port=80, debug = True)
```

APPENDIX I: ANDROID APPLICATION

App build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.helloworld"
        minSdkVersion 23
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    testImplementation 'junit:junit:4.13-beta-3'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
    implementation project(':volley')
    implementation project(':libstreaming')
    implementation 'com.android.support.constraint:constraint-layout:2.0.0-beta1'
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.github.perthcpe23:mjpegviewer:1.0.7'

}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.hardware.camera2.full" />


    <application
        android:noHistory="false"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <service android:name="net.majorkernelpanic.streaming.rtsp.RtspServer"/>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".Test_Activity"
            android:screenOrientation="landscape"/>
    </application>

</manifest>
```

MainActivity layout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/ipadrEditText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="132dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:ems="10"
        android:hint="IP Address"
        android:inputType="text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/currentTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Your IP address"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/findipButton" />

    <Button
        android:id="@+id/findipButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="108dp"
        android:layout_marginEnd="8dp"
        android:text="Find IP"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/GoButton" />

    <Button
        android:id="@+id/GoButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="100dp"
        android:layout_marginEnd="8dp"
        android:text="GO!"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/ipadrEditText" />

</android.support.constraint.ConstraintLayout>
```

MainActivity Java

```
package com.example.helloworld;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;


public class MainActivity extends Activity implements View.OnClickListener {
    public static String ipadr;
    public String url;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText ipadrEditText = findViewById(R.id.ipadrEditText);
        final TextView currentTextView = findViewById(R.id.currentTextView);
        Button testButton = findViewById(R.id.GoButton);
        testButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ipadr = ipadrEditText.getText().toString();
                url ="http://" + ipadr;
                RequestQueue queue = Volley.newRequestQueue(getApplicationCon-
text());
// Request a string response from the provided URL to check if the IP address is
valid.
                StringRequest stringRequest = new StringRequest(Request.Method.GET,
url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {
                                Toast.makeText(getApplicationContext(), response,
Toast.LENGTH_LONG).show();
                                Intent Stream = new Intent(MainActivity.this, Cam-
era_Activity.class);
                                startActivity(Stream);
                            }
                        }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        Toast.makeText(getApplicationContext(), "The IP address is
invalid or the stream is not working!", Toast.LENGTH_LONG).show();
                    }
                });
                queue.add(stringRequest);
            }
        });



        //Button for finding own IP address, linking to Utils.java
        Button findipButton = findViewById(R.id.findipButton);
        findipButton.setOnClickListener(new View.OnClickListener(){

             @Override
             public void onClick(View v) {
                 try {
                     String ip = Utils.getIPAddress(true); // finding IPv4
                     currentTextView.setText(ip);
                 } catch (Exception e) {
                     currentTextView.setText("error finding IP address");
                 }
             }


        });
    }

    @Override
    public void onClick(View v) {

    }
}
```

Camera stream Activity

```
package com.example.helloworld;

import android.os.Bundle;

import com.longdo.mjpegviewer.MjpegView;

public class Camera_Activity extends MainActivity {
    MjpegView viewer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_camera);
        viewer = findViewById(R.id.mjpegview);
        viewer.setMode(MjpegView.MODE_FIT_WIDTH);
        viewer.setAdjustHeight(true);
        viewer.setUrl(url);
        viewer.startStream();
    }
    @Override
    public void onStop() {
        super.onStop();
        viewer.stopStream();
    }
}
```

Utils.java, used for finding IP address

```
/*
Full credits for this part of the code go to Toaster on StackOverflow
https://stackoverflow.com/questions/6064510/how-to-get-ip-address-of-the-device-
from-code
With this code, the device can find its own IP address.
*/
package com.example.helloworld;

import java.io.*;
import java.net.*;
import java.util.*;
//import org.apache.http.conn.util.InetAddressUtils;

public class Utils extends MainActivity {

    /**
     * Convert byte array to hex string
     * @param bytes toConvert
     * @return hexValue
     */
    public static String bytesToHex(byte[] bytes) {
        StringBuilder sbuf = new StringBuilder();
        for(int idx=0; idx < bytes.length; idx++) {
            int intVal = bytes[idx] & 0xff;
            if (intVal < 0x10) sbuf.append("0");
            sbuf.append(Integer.toHexString(intVal).toUpperCase());
        }
        return sbuf.toString();
    }

    /**
     * Get utf8 byte array.
     * @param str which to be converted
     * @return  array of NULL if error was found
     */
    public static byte[] getUTF8Bytes(String str) {
        try { return str.getBytes("UTF-8"); } catch (Exception ex) { return null; }
    }

    /**
     * Load UTF8withBOM or any ansi text file.
     * @param filename which to be converted to string
     * @return String value of File
     * @throws java.io.IOException if error occurs
     */
    public static String loadFileAsString(String filename) throws java.io.IOExcep-
tion {
        final int BUFLEN=1024;
        BufferedInputStream is = new BufferedInputStream(new FileInputStream(file-
name), BUFLEN);
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream(BUFLEN);
            byte[] bytes = new byte[BUFLEN];
            boolean isUTF8=false;
            int read,count=0;
            while((read=is.read(bytes)) != -1) {
                if (count==0 && bytes[0]==(byte)0xEF && bytes[1]==(byte)0xBB &&
bytes[2]==(byte)0xBF ) {
                    isUTF8=true;
                    baos.write(bytes, 3, read-3); // drop UTF8 bom marker
                } else {
                    baos.write(bytes, 0, read);
                }
                count+=read;
            }
            return isUTF8 ? new String(baos.toByteArray(), "UTF-8") : new
String(baos.toByteArray());
        } finally {
            try{ is.close(); } catch(Exception ignored){}
        }
    }

    /**
     * Returns MAC address of the given interface name.
     * @param interfaceName eth0, wlan0 or NULL=use first interface
     * @return  mac address or empty string
     */
    public static String getMACAddress(String interfaceName) {
        try {
            List<NetworkInterface> interfaces = Collections.list(NetworkInter-
face.getNetworkInterfaces());
            for (NetworkInterface intf : interfaces) {
                if (interfaceName != null) {
                    if (!intf.getName().equalsIgnoreCase(interfaceName)) continue;
                }
                byte[] mac = intf.getHardwareAddress();
                if (mac==null) return "";
                StringBuilder buf = new StringBuilder();
                for (byte aMac : mac) buf.append(String.format("%02X:",aMac));
                if (buf.length()>0) buf.deleteCharAt(buf.length()-1);
                return buf.toString();
            }
        } catch (Exception ignored) { } // for now eat exceptions
        return "";
    }

    /**
     * Get IP address from first non-localhost interface
     * @param useIPv4   true=return ipv4, false=return ipv6
     * @return  address or empty string
     */
    public static String getIPAddress(boolean useIPv4) {
        try {
            List<NetworkInterface> interfaces = Collections.list(NetworkInter-
face.getNetworkInterfaces());
            for (NetworkInterface intf : interfaces) {
                List<InetAddress> addrs = Collections.list(intf.get-
InetAddresses());
                for (InetAddress addr : addrs) {
                    if (!addr.isLoopbackAddress()) {
                        String sAddr = addr.getHostAddress();
                        //boolean isIPv4 = InetAddressUtils.isIPv4Address(sAddr);
                        boolean isIPv4 = sAddr.indexOf(':')<0;

                        if (useIPv4) {
                            if (isIPv4)
                                return sAddr;
                        } else {
                            if (!isIPv4) {
                                int delim = sAddr.indexOf('%'); // drop ip6 zone
suffix
                                return delim<0 ? sAddr.toUpperCase() : sAddr.sub-
string(0, delim).toUpperCase();
                            }
                        }
                    }
                }
            }
        } catch (Exception ignored) { } // for now eat exceptions
        return "";
    }

}
```