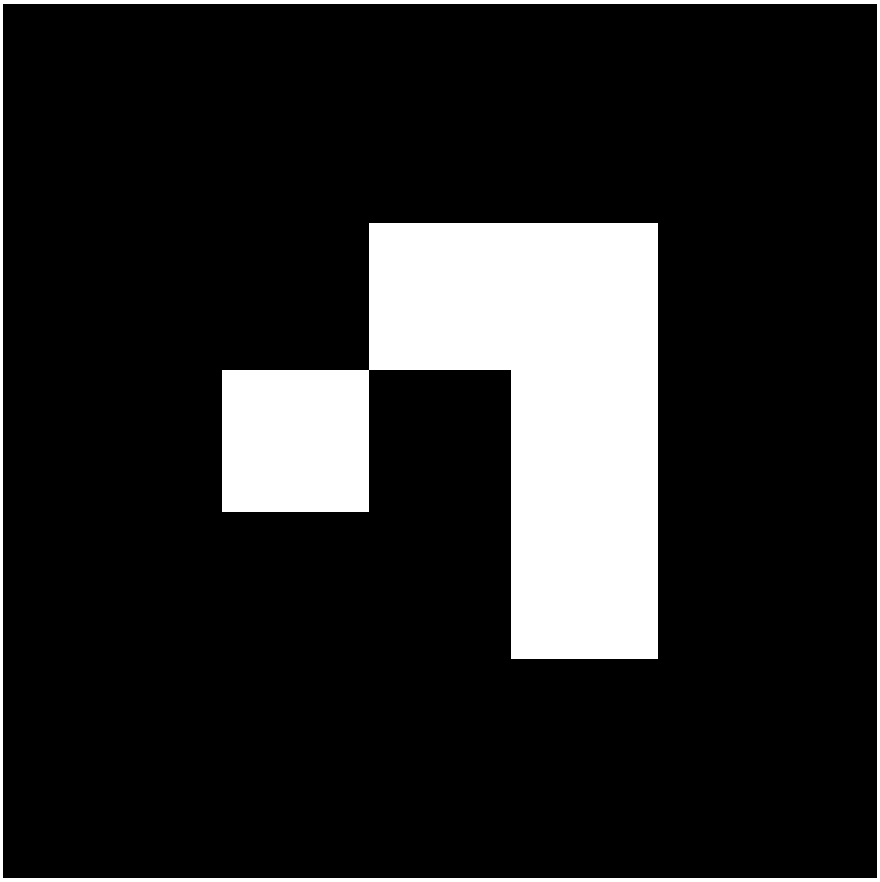

Using AR for device maintenance

Bachelor's thesis in Computer Science and Engineering

MARCUS MATHIASON
DAVID OLOFSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Bachelor's thesis 2019

Using AR for device maintenance

Marcus Mathiason, David Olofsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Using AR for device maintenance
Marcus Mathiason, David Olofsson

Supervisor: Fredrik Löfgren, Ericsson
Supervisor: Fang Chen, Department of Computer Science and Engineering
Examiner: Peter Lundin, Department of Computer Science and Engineering

Bachelor's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: One of several markers the developed application uses for identification.

Department of Computer Science and Engineering
Gothenburg, Sweden 2019

Exploring AR development for industrial applications
Marcus Mathiason, David Olofsson
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Currently, Ericsson and other actors are hard at work connecting the industry to local networks for increased oversight, flexibility and agility. This, in combination with stable optical identification libraries and powerful mobile devices creates an opportunity for an AR tool capable of extracting and visualizing valuable and relevant data.

This project aims to utilize this opportunity, exploring the possibilities of using augmented reality for device maintenance. More specifically identifying a malfunctioning device and the cause of the malfunction by accessing its data through AR.

Due to the difficulties in uniquely identifying a large number of devices through AR and the lacking documentation regarding this in most frameworks a large part of the projects time went to testing frameworks. This, combined with the projects development being limited to Linux, forced the project goal to be a proof of concept rather than a functional application. As such the project ended with a proof of concept application and a lot of experience regarding what tools shouldn't be used.

Keywords: AR, JavaScript, Marker based, Optical recognition, Industry 4.0

Sammanfattning

I dagsläget jobbar företag som Ericsson hårt med att ansluta industrier till lokala nätverk för ökad översikt, flexibilitet och smidighet. Detta, i kombination med stabila bibliotek för optisk identifiering och kraftulla mobila enheter skapar möjligheter för ett förstärkt verklighetsverktyg kapabelt att extrahera och visualisera värdeful samt relevant data.

Detta projekt har som mål att utnyttja denna möjlighet genom att utforska möjligheterna kring att använda förstärkt verklighet for enhetsunderhåll. Mer specifikt, att identifiera felaktigt fungerande enheter samt orsaken till funktionsstörningen.

Den största tiden av projektet gick åt till att välja rätt ramverk för att kunna lösa problemet på bästa sätt. Detta På grund utav svårigheterna i att unikt identifiera ett högt antal enheter genom AR och den bristande dokumentation som många frameworks har kring detta gick stor del av projektets tid till att testa frameworks. Det, i kombination med att utvecklingen begränsades till linux, tvingade projektets mål att vara ett bevis på konceptet snarare än en fungerande applikation. Därefter avslutades projektet med en enklare applikation som bevisar möjligheten och mycket erfarenhet om vilka verktyg som inte fungerar.

Nyckelord: AR, JavaScript, Marker baserat, Bildigenkänning, Industri 4.0

Acknowledgements

We would like to take this opportunity to thank Fredrik Löfgren from Ericsson for being our supervisor at the company as well as Francis Bonneau, also from Ericsson, for lending his experience in parts of the application development.

Marcus Mathiason, David Olofsson, Gothenburg, June 2019

Contents

List of Figures

1
Introduction

1.1 Background

The vision of future industries is constantly evolving to ensure improvements in efficiency and capacity. The latest and current trend, Industry 4.0 - Also known as *The fourth industrial revolution*, aims to develop 'The connected factory', meaning that everything on the factory floor is connected.[14] This can then be utilized to enable devices to communicate with each other and report their states to central locations, enabling more agile work flow compared to today's industries and more detailed control over the processes.

Ericsson is a company that specializes in mobile broadband Internet communications. Their idea of reaching a more efficient production is to "cut the cables" and focus more on using sensors to ultimately connect all devices in 'The connected factory' wirelessly, thus creating a more agile and flexible work flow.[13] By connecting all devices across the factory, extracting device data and adjusting based on that data will be possible.

1.2 Purpose

To make it easier to harvest the inter-connectivity brought by Industry 4.0 Ericsson wishes this report to research the production of an Augmented Reality application intended to use the data produced by all connected devices. In part gleaning the possibilities for such applications but also the challenges and pitfalls of its development.

1.3 Goal

The goal is to develop an AR application capable of identifying different devices as well as extracting valuable data concerning the identified device's state. With the device identified, the user will be shown a toolbar specific for that device allowing the user to see sensor data, interpretations of that data (such as "overheating" or "insufficient voltage") and maintenance instructions. Additionally an AR button can be pressed, redirecting the user to another view visualizing the device's status over time in the form of graphs.

In conclusion, the goal of this project is to develop av AR application that exemplifies what is possible with current technologies.

1.4 Delimitations

1.4.1 Device compatibility

The application was written for use on a modern smartphone, since it is a device capable of the task that most technicians already carry. Android was chosen since it is the dominating mobile operating system on the market [15]. Specifically the Google Nexus 5X was used, since it has the latest version of Android, good performance and is not overly expensive. In addition to this Android Studio has a built in emulator for this hardware, which makes development easier.

1.4.2 Integration

Since Augmented Reality rarely is entirely practical by itself it should be possible to combine the application within a more traditional user interface.

1.4.3 Backend

Due to not having access to any device data the application will be a proof of concept with hard-coded states for all identified devices, rather than using an Application Programming Interface to access device data.

1.4.4 Marker based

The application is intended to represent Augmented Reality in an industrial setting. Therefore it must have an efficient way of uniquely identifying a large number of devices at a distance and several of them at the same time. The easiest solution to this is marker based identification, since it is the easiest system for managing identification of multiple potentially otherwise identical devices. Other identification methods will not be considered for this report.

1.4.5 User Interface

Due to the project's focus on accessing data and the authors lacking experience in UI design a good user interface is not a priority for this project. Priority is instead given to identification in AR.

2
Technical Background

2.1 AR (Augmented Reality)

The concept of amplifying a user's reality with the assistance of an augmenting device such as a mobile phone and/or an augmented reality headset. Augmenting one's reality can mean very different things depending on what sense is being augmented. Following are three examples of sense augmentation.

2.1.1 Visual augmentation

To augment something visually means to overlay different points of interest onto a screen. This could, for example, be different kinds of tabs of information specific to a certain location.

Visual augmented reality is the key part in the concept '*Digital twin*'. Digital twin is the concept of copying a real life object and putting it into a virtual space, for example the augmented reality environment that can be simulated using an augmenting device with support for visual augmentation. The purpose of this is to enable the user to be able to inspect the object from different perspectives without actually having to be anywhere close to the real life object. This has several areas of use. For example, one can use this technology when educating new workforce around hazardous environments to avoid injuries while still letting the educates inspect objects up close.

By using the augmenting device's camera along with real life markers, this kind of augmented reality is relatively easy to develop with the right hardware.

2.1.2 Auditory augmentation

To augment something auditory means to enhance the user's perceived reality in forms of simulated sound. This opens up to several different possibilities when it comes to assisting the user. For example, this can potentially be used to simulate different environments to assist individuals with impaired vision. Additionally, this technology could also be used to make the experience of visiting museums and art galleries more captivating by simulating different types of sounds depending on where the user is located.

Thanks to sound being easy to overlay with only a vibrating membrane augmentation is arguably the oldest type or AR. It is exemplified by radios, phones, record players and other forms of listening equipment, making it date back at least to the late 1800s when audio recording was patented. [7]

2.1.3 Haptic augmentation

To augment something haptically means to simulate tactile feedback when interacting with augmented data. Racing video games are a great example for this. Most steering wheels support a force feedback mechanism which works as such; When the user collides with an in-game object, the wheels automatically turn as a result of the impact. The turning of the wheels will then transfer to the steering wheel, which will also be simulated onto the real life steering wheel. As a result, a virtual collision has caused a real life object to move accordingly.

2.2 Libraries

Libraries are collections of pre-written code that can easily be imported into other code projects to enable simpler and faster implementation of specific attributes.

2.2.1 Three.js

Three.js is a javascript library designed to simplify rendering 3D environments in javascript. It uses WebGL to accelerate the rendering with native hardware, greatly improving its efficiency, and has a large library of built in shapes and textures. [16]

2.2.2 jsARToolKit

ARToolKit is a C++ library designed for AR tracking and rendering. [3] It abstracts the advanced calculations and computer vision algorithms needed to track the users relative position. [1] Based on it jsARToolkit has been created, to fill the same purpose in web-applications. [4]

2.2.2.1 Barcode markers

ARToolkit uses markers for its spatial recognition, calculating the users position relative to the anchor based on its size and apparent shape. These markers are a hollow black square that can be filled with identifying data. One type of identifying data that can be used is serially encoded binary data in a grid of 3x3, 4x4, 5x5 or 6x6. [18] Though solutions exist that place a QR code in the marker instead these depend on an external library to identify the QR code, causing the identification to be disconnected from the marker identified. In situations with multiple markers that risks using the wrong marker. Markers with QR codes in them have other advantages, since the QR code can hold the URL to a web based AR application using the marker.

2.2.3 AR.js

AR.js is a JavaScript library developed by Jerome Etienne aimed towards creating a simple way of implementing augmented reality in a web browser. This makes it

easier to be able to utilize augmented reality, not only on augmented reality devices, but also other devices like mobile phones, tablets and computers.

AR.js is based on jsartoolkit5 and three.js. Three.js is used by AR.js for all the rendering as well as spatial calculations while jsartoolkit5 gives the marker tracking and some calculations.

The official AR.js GitHub repository features a web application that uses this library, showcasing the vital efficiency of the library, which is crucial when using a mobile phone for this kind of application that puts a lot of pressure on the processor.[9] This application can be seen in action in figure 2.1.

Figure 2.1: Augmented reality in a web browser using AR.js

3
Method

3.1 Setting the goal

The first thing done in the project was choosing the goal application. Ericsson had only given the proposal of "Exploring the possibilities of using AR for device maintenance", which is an extremely broad goal to work towards. Therefore, comprehensive delimitations needed to be made, while still complying with Ericsson's requirements, to have a clear goal to work towards. These can be read further into under 1.4.

3.2 Selecting tools

With an intended goal in mind tools were to be selected. Ericsson supplied the project with Linux laptops to develop on and access to android phones to test on. Based on the application requirements and the accessible tools frameworks and libraries were researched. This took a lot more time than what was originally anticipated. After having tested several different frameworks and libraries over the course of half of the project's set out time, one supporting the development environment and capable of building the application was found and chosen, AR.js.

3.3 Writing the application

The application was based on a simple example from the documentation so that the starting point was functional AR. Even so, a lot of debugging was needed to initially get it running, but by starting with a simple example the debugging was easier and quicker, making it possible to start integration earlier. From the functional example a simple application was gradually developed. Through continuous testing it was assured that the application always ran, enabling integration to be done parallel to the application development.

3.4 Implementing the application

The implementation of the application was split into two parts, permissions and integration. This was done to enable easier and more efficient simultaneous work on both parts. Additionally this allowed developers to specialize on their part, making development more efficient.

3.4.1 Permissions

Before AR could be integrated into the application it needed to be able to access the camera. Therefore camera permissions needed to be granted to the surrounding application and the AR application within it. For a standard Android application, only one simple line of code is needed in the application's xml document:

```
1 <uses-permission android:name="android.permission.CAMERA" />
```

Listing 3.1: Camera permissions for a standard Android application

For the application to be part of another bigger application, however, one may need to iteratively add additional code depending on how the different layers in the application are arranged and operate.[6] This was true for the surrounding application used in the project, requiring further code detailed in 4.1.

3.4.2 Integration

With the camera working the AR application was built into the surrounding application and menu entries added, to allow accessing AR in the same manner as any other feature in the surrounding application. Due to incompatibilities between the surrounding application and the AR application a wrapper was written that partially isolates them from each other.

4

Construction

4.1 Choosing the correct framework

Before work could start a framework with relevant operating system support, documentation, adequate features and good efficiency needed to be chosen.

4.1.1 Development compatibility

The first consideration for any framework is if it is possible to install. Due to delayed access to computers and diffuse documentation this took unexpectedly long in the project, causing several weeks to be spent on studying a framework unusable for the project due to it not supporting linux development. After that initial failure two potential alternatives were found, the ARCore framework and the artoolkit library. Due to artoolkit lacking accessible clear documentation ARCore was chosen.

4.1.2 Features

That choice was made with the assumption that ARCore had some sort of identifiable marker, since all summaries describe it as such. This turned out to be an oversight, since ARCore does not support markers per se but rather identifies images and places them within its native surface based tracking. [11] [10] Since this system has a hard limit both in number identifiable images (up to 1000 at a time) and identified images (up to 5 at a time) and additionally only recognizes an image after it has covered 25% of the screen it is greatly inferior to true marker based tracking for this case. The application is likely to need to show more than five device states at a time and may well be placed in an environment with need to recognize more than 1000 unique devices in a session, giving that this wasn't good enough. Since this was discovered after multiple weeks work on creating a basic ARCore application the projects time constraints forced goals to be reconsidered.

With that only artoolkit remained. It is marker based and has a type of marker that enables easy generation of millions of unique markers.[18] Additional research provided the javascript version of artoolkit, jsartoolkit, and AR.js, which connects it to three.js for fully web native AR. This library is quite well documented and has several working examples available, enabling development to start on a functional example.

4.2 Developing the application

Thanks to an incompatible local library creation of the application was started with a careful breakdown of an example program supplied by AR.js. This took additional time due to neither author having previous experience with javascript but yielded detailed insight into the structure of the framework which came of use later, when debugging integration. When the example was functional it was modded to recognize two specific markers and mark them as either red or green. Consideration was given to both adding the ability to tap a marker to open graphs related to it and to making the application recognize any marker of the right type. Due to tap detection requiring vector calculations performed through previously unexplored parts of three.js and arbitrary marker detection requiring modification of AR.js the choice was made to focus on integration onto the intended device and possibly develop more if time allowed.

4.3 Implementing the application

Upon starting implementation work was split up based on knowledge from previous parts of the development to minimize the need to learn before work could start.

4.3.1 Camera permissions

A major difficulty that arose when trying to implement AR as part of another, more comprehensive application was the granting of permissions to use the camera. Due to the projects application being placed an additional layer of isolation away from the operating system than most permissions had to be granted through the different layers of the surrounding application to ultimately grant the whole application access to read the camera input.

In a standard Android application, granting permissions to use the camera is as simple as one line of code in the application's xml document as shown by listing 3.1. In comparison, to additionally pass permissions through the webview which the outer application is built around, the amount of code needed increases drastically, as shown by Listing 4.1.

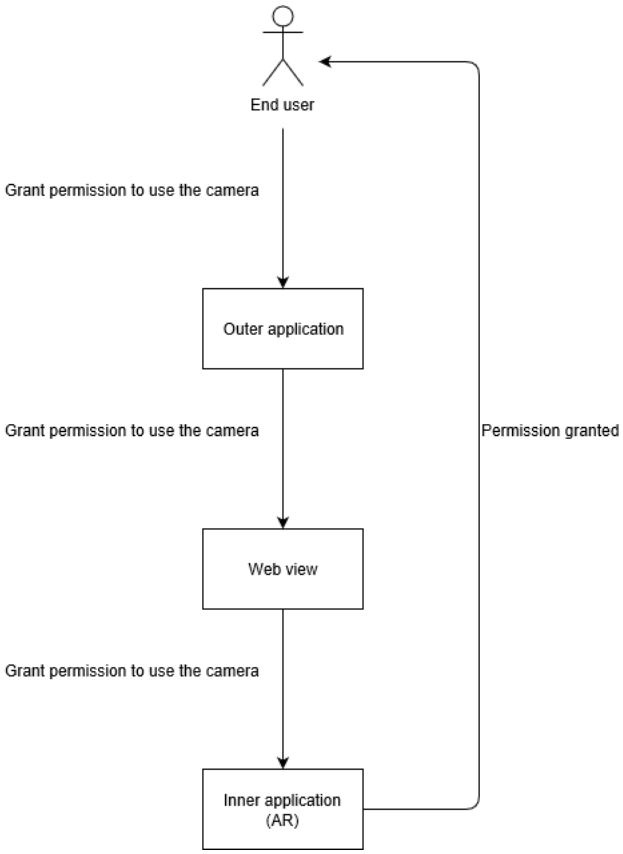

Figure 4.1: The different layers of permissions needed to read camera output

For this project there were only two layers of applications permissions had to be passed through, so the previously mentioned code snippets were enough to be able to use the camera in the full application, both inside and outside the AR view.

```
webView.setWebChromeClient(new WebChromeClient() {
            // Grant permissions for cam
            @Override
            public void onPermissionRequest(final PermissionRequest request) {
                System.out.println("onPermissionRequest");
                PermissionRequest mPermissionRequest = request;
                final String[] requestedResources = request.getResources();
                for (String r : requestedResources) {
                    if (r.equals(PermissionRequest.RESOURCE_VIDEO_CAPTURE)) {
                        // In this sample, we only accept video capture request.
                        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(MainActivity.this)
                                .setTitle("Allow Permission to camera")
                                .setPositiveButton("Allow", new DialogInterface.OnClickListener() {
                                    @Override
                                    public void onClick(DialogInterface dialog, int which) {
                                        dialog.dismiss();
                                        mPermissionRequest.grant(new String[]{PermissionRequest.RESOURCE_VIDEO_CAPTURE, PermissionRequest.RESOURCE_AUDIO_CAPTURE});

                                        System.out.println("Granted");
                                    }
                                })
                                .setNegativeButton("Deny", new DialogInterface.OnClickListener() {
                                    @Override
                                    public void onClick(DialogInterface dialog, int which) {
                                        dialog.dismiss();
                                        mPermissionRequest.deny();
                                        System.out.println("Denied");
                                        //Log.d(TAG,"Denied");
                                    }
                                });
                        AlertDialog alertDialog = alertDialogBuilder.create();
                        alertDialog.show();

                        break;
                    }
                }
            }

            @Override
```

```
40              public void onPermissionRequestCanceled(PermissionRequest
    request) {
41                  super.onPermissionRequestCanceled(request);
42                  Toast.makeText(MainActivity.this,"Permission Denied",
    Toast.LENGTH_SHORT).show();
43              }
44          });
```

Listing 4.1: Camera permissions source code [12]

4.3.2 Integration

The sample application used as a base could run in a native web browser by default but needed to run inside the surrounding application. In most android applications this could be achieved with a web-view, as detailed above, but the application to integrate into was a locally hosted web application already running in a web view. This removed the need for a web view but required integrating the AR view into the web application itself.

The first step was to create an internal web page on which the AR would be shown and linking it into the menu. The page was easily created along with the theme and menu from the surrounding application but when the AR code was added problems started showing. The code executed but the camera stream was placed at the end of the HTML document, placing it under the menu and the themes background. No matter what settings used in the initialization of the AR objects this behavior would not change. After a day of reading the source code for AR.js it became clear that the library assumed AR applications were to run on the entire web page, since the initialization read the size of the browser window and set that as the size of the AR application rather than using the entered configurations.

Because of this the AR code needed to be housed on its own separate web page, without any menu or theme. In that shape it worked as intended but the user could be stuck in AR after entering it, since no menu existed to navigate with. To handle that a page with the menu and theme of the surrounding application acted as a wrapper for the AR, embedding the AR page in an iframe. This is an HTML function which treats the embedded page as a separate browser window, thus causing the AR.js library to read the correct size when reading the size of its browser window.

5
Results

The project has successfully created a very integrable proof of concept AR application embedded as a part of a more comprehensive application developed by Ericsson.

5.1 Final goal reached

The original goal, *to develop an AR application capable of identifying different devices as well as extracting valuable data concerning the identified device's functioning state* along with a series of other functions was reconsidered relatively early, since the process of researching what frameworks worked best for the intended purpose took much longer time than what was originally anticipated. Instead, the goal was set to develop a proof of concept application as a starting ground for future work to be made upon, enabling to others reach the application concept stated in the original goal. The new goal was successfully met and an application implemented as a view in another, more comprehensive application Ericsson has developed.

5.2 Functionality

The AR application can identify barcodes that are registered in the application. Once the application has identified the barcode in the camera feed, it will overlay a colored circle on top of the barcode. The colored circle's purpose is to visualize the funcitonal status of the device linked to that specific barcode. The circle is intended to change colors between red and green depending on whether the device is malfunctioning or not. Simply put; red signifies a critical malfunction. Green signifies a device that is not in need of maintenance.

The application is shown in action by figures 5.1 and 5.2.

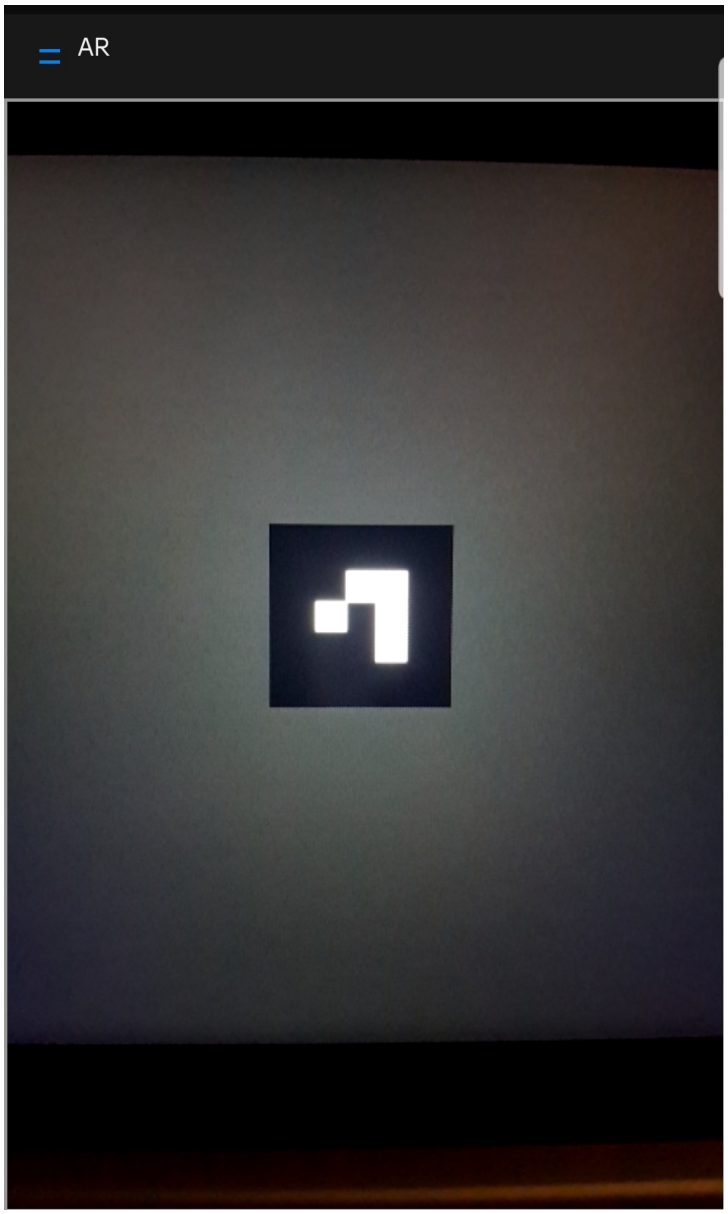

Figure 5.1: Bar code not recognized in the application

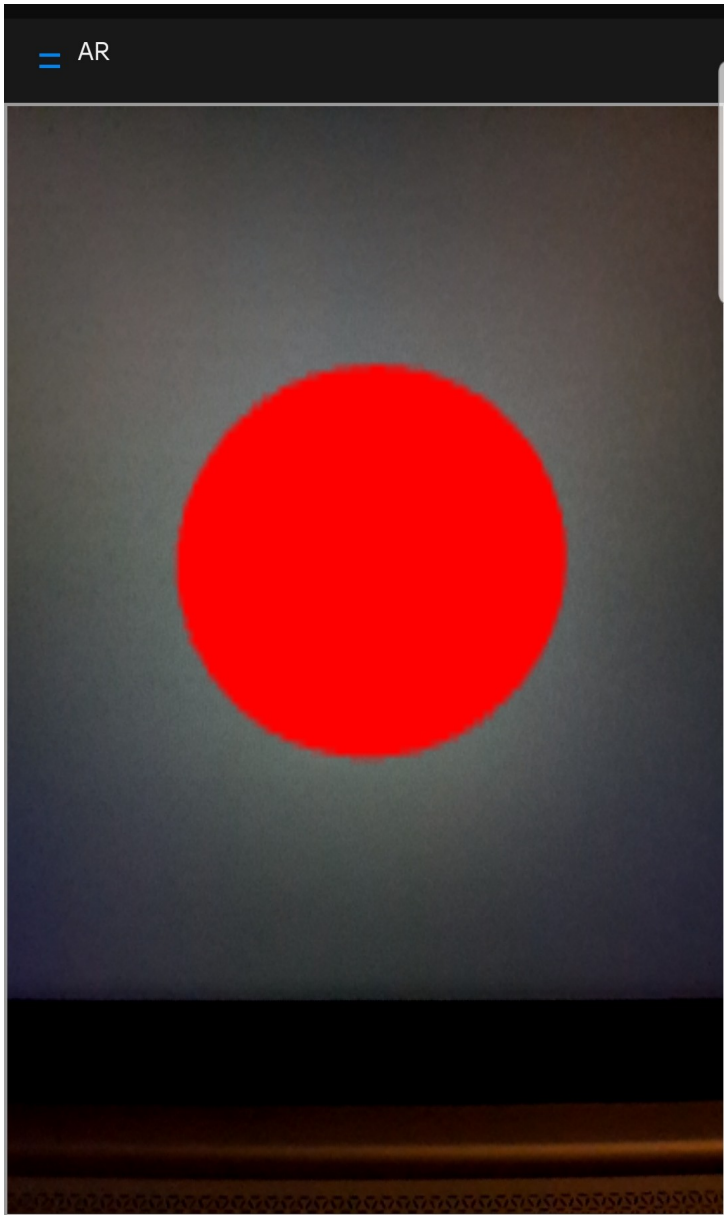

Figure 5.2: Bar code recognized in the application

6
Discussion

6.1 The project's contribution

At the end of the project, a functioning proof of concept and basis has been created for an AR interface to manage a large number of devices. This application signifies that the technology is mature enough for practical large scale applications as well as efficient enough for use on mobile devices. Research and testing found less well suited frameworks and libraries for large scale industrial AR as well as some suited libraries, artoolkit and those based on it. As such future implementations could use the code and research herein as a starting point.

6.2 Possible applications

Whilst this project aimed for a simple colour based interface, the same libraries could be used to create dynamic time-tables for bookings of rooms or buses, where a user could read the latest information in AR by pointing their phone at a marker outside the room or at the bus stop. Every application where large amounts of centralized data relevant to specific locations exists is suitable for similar solutions.

An example where this could be well used is on the electric scooters being rented out in many larger cities. Current solutions use QR codes that are scanned before you are told if the scooter is reserved or how much battery it has. [17] With an application similar to the one in this project the battery state and whether the scooter is free could be presented more quickly in AR.

6.3 Difficulties

The major challenges for industrial AR are initially finding a scalable library or framework and a suitable device for the application. Thereafter an efficient method of accessing relevant data and presenting that data in a user friendly fashion.

6.3.1 Framework

Several different frameworks were tested during the project, each with different outcomes. This section will go over the attempts of utilizing the tested frameworks as well as what caused them to not be suitable for the project's scope.

6.3.1.1 Vuforia

Due to a large number of advanced features and good device support, Vuforia was the first alternative researched. Due to lacking support for the available development environment it ended up not being the one used. Vuforia requires Unity3D on Windows or MACOS which conflicted with the available development environment. The available environment was Linux based which meant that official support would not be available. There were workarounds available but a decision was made to refrain from using such since Vuforia would also make it difficult to integrate with existing mobile applications. For a project developed on a supported platform intending to create a standalone application it may be better suited.

6.3.1.2 ARCore

The large advantage of ARCore in this project was its integration in Android Studio, since the application to integrate with was written in Android Studio. After research, it was discovered that ARCore does not have support for markers, only images that must be quite distinct from one another to not be misidentified. Since this would limit the number of identified devices another solution was needed.

6.3.2 Device

A device must be reasonably priced while having sufficient processing capabilities to render the AR in real time. Modern mobile phones or tablets are an attractive choice for this but AR headsets would create a more immersive experience, improving ease of use. Future works needs to consider their application and the technical state in their time to select between writing for a phone or an AR headset, since prices are likely to change and different use cases require different hardware.

6.3.3 Access to data

For AR to be useful it needs to present some data. Accessing that data may be difficult, if it is real-time data from a distributed network, since the application cannot present data it doesn't have. This will most likely require caching and some manner of centralizing service to avoid too high latencies in the application. In addition to that, creating the useful data in the first case may be difficult. Even for a simple case of presenting temperature data from five entities there must be sensors for each as well as a device that serves that data. Even static data may well be too large for local storage, requiring some external service to hold it. As such, a practical implementation needs a solution to querying device data quickly with minimal resource usage to integrate the data well into AR.

6.3.4 Presentation of data

Presenting the data in a way that is usable is one of the biggest challenges in AR development. Whilst this project aimed to simplify AR data as much as possible to make it easier to use future cases may well require the effort to find better methods of data presentation. Adding the needed data in a readable manner without causing the application to be confusing and difficult to use may thus be a challenge for others going forward.

6.3.5 State of art

The developed application aims to be able to display data for several devices at the same time in a universally integrable manner. This is in comparison to other similar applications where few markers can be known simultaneously and the code is difficult to integrate. Enabling more markers to be recognized makes device status extraction easier, since you can identify the faulty device among others. It will make work faster, simpler and more agile, which goes hand in hand with the vision of Industry 4.0.

6.4 Future work

Since this project concludes it possible to write an application for industrial AR the next step towards adoption is a specific implementation of it. Either through an application or an optimized library that just reads AR markers as their identifying integer and simplifies placing 2D objects over them, perhaps written in C++ using the underlying "artoolkit5" rather than AR.js for support on specialized hardware and better performance over the universal support given by JavaScript.

6.5 The project's role in society

The project is from the ground intended to be aimed towards technicians performing maintenance on devices in a '*connected factory*'. The concept is that, with the help of the resulting application, technicians are easily able to extract valuable data from connected devices in the factory. By pointing the device with the application on a device with an attached barcode marker, data will be fetched from a time series database. This data will then be handled by an algorithm that concludes whether or not the device currently is critically malfunctioning or not, which will be visualized with the help of a colored circle being colored red or green, depending on the state of the device. With a simple tap on this circle, the user would then be redirected to another view in the application showcasing how the device in question has operated over a set time interval and advice on what kind of maintenance the device may be in need of with instructions on how this maintenance would be carried out.

This concept would greatly assist technicians in forms of flexibility, time spent looking for malfunctions as well as carry load since the only device needed to seek

out malfunctions has gone from tools to measure all different parts of the device in question to only one powerful tool - The mobile phone. Therefore this technology has potential for reducing downtime in industrial facilities and reducing the need for measuring devices, potentially reducing costs and improving resource efficiency where implemented. This, in turn, is likely to make a noticeable reduction in the environmental impact of industrial facilities.

Bibliography

[1] *artoolkit*. URL: http://www.hitl.washington.edu/artoolkit/.

[2] Chromium developers. *Chromium*. URL: https://www.chromium.org/Home (visited on 05/08/2019).

[3] Daqri. *artoolkit5*. URL: https://github.com/artoolkit/artoolkit5.

[4] Daqri. *jsartoolkit5*. URL: https://github.com/artoolkit/jsartoolkit5.

[5] A. Z. David Herman Luke Wagner. *asm.js*. Aug. 18, 2014. URL: http://asmjs.org/spec/latest/.

[6] A. developers. *Request App Permissions*. URL: https://developer.android.com/training/permissions/requesting (visited on 06/15/2019).

[7] T. Edison. *Phonograph or Speaking Machine*. Feb. 19, 1978. URL: https://patents.google.com/patent/US200521A/en (visited on 16/15/2019).

[8] J. Etienne. *AR.js*. URL: https://github.com/jeromeetienne/AR.js.

[9] J. Etienne. *AR.js Official Github*. URL: https://github.com/jeromeetienne/AR.js/blob/master/README.md (visited on 06/13/2019).

[10] Google. *ARCore overview*. URL: https://developers.google.com/ar/discover/ (visited on 06/13/2019).

[11] Google. *Recognize and Augment Images*. URL: https://developers.google.com/ar/develop/java/augmented-images/ (visited on 05/07/2019).

[12] A. GUPTA. *Camera permissions source code*. URL: https://stackoverflow.com/a/49403612 (visited on 05/15/2019).

[13] E. Josefsson. *Cellular networks deliver industrial IoT connectivity – cut the cables for flexible production*. Apr. 2018. URL: https://www.ericsson.com/en/blog/2018/4/cellular-networks-deliver-industrial-iot-connectivity--cut-the-cables-for-flexible-production.

[14] KPMG. *Industry 4.0*. URL: https://www.youtube.com/watch?v=IMmnSZ7U1qM (visited on 06/15/2019).

[15] C. Miller. *Latest Gartner data shows iOS vs Android battle shaping up much like Mac vs Windows*. Aug. 18, 2016. URL: https://9to5mac.com/2016/08/18/android-ios-smartphone-market-share/ (visited on 05/15/2019).

[16] mrdoob. *WebGLRenderer*. URL: https://threejs.org/docs/#api/en/renderers/WebGLRenderer (visited on 06/15/2019).

[17] voi. *How to VOI*. URL: https://www.voiscooters.com/en/how (visited on 05/14/2019).

[18] J. Wolf and erindaqri. *marker_about*. Dec. 9, 2015. URL: https://github.com/artoolkit/artoolkit-docs/blob/master/3_Marker_Training/marker_about.md.

A

Appendix 1

```
<!DOCTYPE html>
<html><head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Watchdog</title>
    <meta name="viewport" content="width=device-width, user-scalable=no
    , minimum-scale=1.0, maximum-scale=1.0">

    <!-- Importing libraries, currently offsite dependencies -->
    <!-- three.js -->
    <script src="https://jeromeetienne.github.io/AR.js/three.js/
    examples/vendor/three.js/build/three.js"></script>
    <!-- ar.js -->
    <script src="https://jeromeetienne.github.io/AR.js/three.js/build/
    ar.js"></script>
  </head>
  <body style="margin : 0px; overflow: hidden; font-family: Monospace;"
    >
    <div style="position: absolute; top: 10px; width:100%; text-align:
    center; z-index: 1;"></div>

    <!-- The actual AR script -->
    <script src="./src/ar-view.js"></script>
  </body>
</html>
```

Listing A.1: html code for the AR application

```
// For advice on how to draw 2d sprites and handle multiple markers this example is good.
// https://github.com/jeromeetienne/AR.js/blob/master/three.js/examples/measure-it.html

// For some reason this code produces lower resolution AR than the example above.
// That needs fixing

//////////////////////////////////////////////////////////////////////////////////
//    Init
//////////////////////////////////////////////////////////////////////////////////

// init renderer
var renderer  = new THREE.WebGLRenderer({
    antialias: true,
    alpha: true
});
renderer.setClearColor(new THREE.Color('lightgrey'), 0)
renderer.domElement.style.position = 'absolute'
renderer.domElement.style.top = '0px'
renderer.domElement.style.left = '0px'
document.body.appendChild( renderer.domElement );

// array of functions for the rendering loop
var onRenderFcts= [];

// init scene and camera
var scene = new THREE.Scene();

//////////////////////////////////////////////////////////////////////////////////
//    Initialize a basic camera
//////////////////////////////////////////////////////////////////////////////////

// Create a camera at origo of AR space
var camera = new THREE.Camera();
scene.add(camera);

//////////////////////////////////////////////////////////////////////////////////
//          handle arToolkitSource
//////////////////////////////////////////////////////////////////////////////////

var arToolkitSource = new THREEx.ArToolkitSource({
    // to read from the webcam
```

```
42      sourceType : 'webcam',
43  })
44
45  arToolkitSource.init(function onReady(){
46      onResize()
47  })
48
49  // handle resize
50  window.addEventListener('resize', function(){
51      onResize()
52  })
53  function onResize(){
54      arToolkitSource.onResize()
55      arToolkitSource.copySizeTo(renderer.domElement)
56      if( arToolkitContext.arController !== null ){
57    arToolkitSource.copySizeTo(arToolkitContext.arController.canvas)
58      }
59  }
60  //
        ////////////////////////////////////////////////////////////////////////////////

61  //          initialize arToolkitContext
62  //
        ////////////////////////////////////////////////////////////////////////////////

63
64  // create atToolkitContext
65  var arToolkitContext = new THREEx.ArToolkitContext({
66      //cameraParametersUrl: THREEx.ArToolkitContext.baseURL + '../data/
        data/camera_para.dat',
67      cameraParametersUrl: 'https://jeromeetienne.github.io/AR.js/data/
        data/camera_para.dat',
68
69      detectionMode: 'mono_and_matrix',
70      matrixCodeType: '3x3',
71  })
72  // initialize it
73  arToolkitContext.init(function onCompleted(){
74      // copy projection matrix to camera
75      camera.projectionMatrix.copy( arToolkitContext.getProjectionMatrix
        () );
76  })
77
78  // update artoolkit on every frame
79  onRenderFcts.push(function(){
80      if( arToolkitSource.ready === false ) return
81
82      arToolkitContext.update( arToolkitSource.domElement )
83
84      // update scene.visible if the marker is seen
85      scene.visible = camera.visible
86  })
87
88  //
        ////////////////////////////////////////////////////////////////////////////////
```

```
89 //          Create a ArMarkerControls
90 //
    ////////////////////////////////////////////////////////////////////////////////

91
92 //By adding separate "THREE.Groups" that we link to the markers the
    groups can
93 //move independently to eachother, enabling multiple visible markers.
94
95 //Adding a visible object to a group will draw it on the marker that
    that group
96 //is bound to.
97
98 //The example at the top of the document additionally adds those groups
     to a
99 //group, since this enables measuring their relative distance. We don't
     need
100 //that here though.
101
102 //
    ////////////////////////////////////////////////////////////////////////////////

103 //    markerRoot1
104 //
    ////////////////////////////////////////////////////////////////////////////////

105
106 //Create a object group for the marker
107 var markerRoot1 = new THREE.Group
108 markerRoot1.name = 'marker1'
109
110 //Add it to the overall scene
111 scene.add(markerRoot1)
112
113 //Declare the marker
114 var markerControls = new THREEx.ArMarkerControls(arToolkitContext,
    markerRoot1, {
115     type : 'barcode',
116     barcodeValue: 20,
117     // patternUrl : THREEx.ArToolkitContext.baseURL + '../data/data/
    patt.kanji',
118 })
119
120 // create a colored orb over the marker
121 var geometry  = new THREE.CircleGeometry( 1, 64 )
122 var material  = new THREE.MeshBasicMaterial( { color: 0x00ff00 });
123 var mesh  = new THREE.Mesh( geometry, material );
124
125 // Draw the gizmo onto the marker
126 markerRoot1.add( mesh );
127
128 //
    ////////////////////////////////////////////////////////////////////////////////

129 //    markerRoot2
130 //
```

```
    ////////////////////////////////////////////////////////////////////////////////

131
132 //Create a object group for the marker
133 var markerRoot2 = new THREE.Group
134 markerRoot2.name = 'marker2'
135
136 //Add it to the overall scene.
137 scene.add(markerRoot2)
138
139 // Declare the marker
140 var markerControls = new THREEx.ArMarkerControls(arToolkitContext,
    markerRoot2, {
141     type : 'barcode',
142     barcodeValue: 5,
143     // patternUrl : THREEx.ArToolkitContext.baseURL + '../data/data/
    patt.kanji',
144 })
145
146 // create a gizmo
147 var geometry  = new THREE.CircleGeometry( 1, 64 )
148 var material  = new THREE.MeshBasicMaterial( { color: 0xff0000 });
149 var mesh  = new THREE.Mesh( geometry, material );
150
151 mesh.rotation.x -= Math.PI / 2 ;
152 //mesh.position.y += 1;
153
154 //Draw it onto the marker
155 markerRoot2.add( mesh );
156
157 //
    ////////////////////////////////////////////////////////////////////////////////

158 //    add an object in the scene
159 //
    ////////////////////////////////////////////////////////////////////////////////

160
161 // To add the data properly we need to draw inspiration from the
    example at the top.
162
163 // It shows how a canvas is added in AR and drawn on to produce 2d text
164
165 // Most of all we need an optimised way to handle hundreds of markers,
    I (work@sidju.se)
166 // can probably make a merge request to the library for a fix than
    enables that optimisation.
167
168
169 // add a torus knot
170 // var geometry = new THREE.CubeGeometry(1,1,1);
171 // var material = new THREE.MeshNormalMaterial({
172 //     transparent : true,
173 //     opacity: 0.5,
174 //     side: THREE.DoubleSide
175 // });
```

```
176 // var mesh = new THREE.Mesh( geometry, material );
177 // mesh.position.y  = geometry.parameters.height/2
178 // scene.add( mesh );
179
180 // var geometry = new THREE.TorusKnotGeometry(0.3,0.1,64,16);
181 // var material = new THREE.MeshNormalMaterial();
182 // var mesh = new THREE.Mesh( geometry, material );
183 // mesh.position.y  = 0.5
184
185 //Note that the object is simply added to a coordinate in AR space.
186 //This is because
187
188 //For multiple markers you need to draw on the marker, using a "THREE.
    group"
189 //scene.add( mesh );
190
191 // Makes the mesh rotate at a constant speed
192 // Helpful to understand the rendering loop
193 //onRenderFcts.push(function(delta){
194 //    mesh.rotation.x += Math.PI*delta
195 //})
196
197 //
    ////////////////////////////////////////////////////////////////////////////////

198 //    render the whole thing on the page
199 //
    ////////////////////////////////////////////////////////////////////////////////


200
201 // render the scene
202 onRenderFcts.push(function(){
203     renderer.render( scene, camera );
204 })
205
206 // run the rendering loop
207 var lastTimeMsec= null
208 requestAnimationFrame(function animate(nowMsec){
209     // keep looping
210     requestAnimationFrame( animate );
211     // measure time
212     lastTimeMsec  = lastTimeMsec || nowMsec-1000/60
213     var deltaMsec = Math.min(200, nowMsec - lastTimeMsec)
214     lastTimeMsec  = nowMsec
215     // call each update function
216     onRenderFcts.forEach(function(onRenderFct){
217   onRenderFct(deltaMsec/1000, nowMsec/1000)
218     })
219 })
```

Listing A.2: Javascript code for the AR application