CHALMERS
UNIVERSITY OF TECHNOLOGY

Designing a Tool for Assisting in the Setup of Optical Motion Capture Systems

Master's thesis in Interaction Design and Technologies
Division of Interaction design

EMMANUEL BATIS

MATHIAS BYLUND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017

Designing a Tool for Assisting in the Setup of Optical Motion Capture Systems

EMMANUEL BATIS
MATHIAS BYLUND

Department of Computer Science and Engineering
Division of Interaction Design
Chalmers University of Technology
Gothenburg, Sweden 2017

Designing a Tool for Assisting in the Setup of Optical Motion Capture Systems

Supervisor: Morten Fjeld, Computer Science and Engineering
Examiner: Staffan Björk, Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Division of Interaction Design
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2017

Designing a Tool for Assisting in the Setup of Optical Motion Capture Systems
Emmanuel Batis & Mathias Bylund
Department of Applied Information Technology
Chalmers University of Technology

Abstract

As optical motion capture systems grow bigger and more complex, the need for mobility when interacting with these systems becomes apparent due to the increasing spatial distances between system components. Seeing how these technologies rapidly evolve and how their usage is becoming more common, providing a way of assisting setup technicians is clearly valuable.

This thesis documents the design- and creation process of a tool that provides the aforementioned assistance. The tool enables its users to visualize the real-time output of every camera in a system. At the same time it provides a way of making adjustments on-the-fly without the need for excessive maneuvering, effectively reducing the amount of personnel needed for setting up the system.

Moreover, this tool acts as a technological statement which proves that it is possible to create cross-platform mobile tools that interact with an optical motion capture system in real-time and by that removes the need to maintain several applications for different platforms.

Keywords: optical motion capture, motion capture, mobile development, cross-platform development, real-time data, interaction design

Contents

List of Figures

List of Tables

Acronyms, Names and Concepts

API	Application Programming Interface	15
CPU	Central Processing Unit	73
DD	Double Diamond	74
DirectX	An application programming interface for multimedia purposes, especially game programming	73
IP	Internet Protocol	70
JPEG	Join Photographic Experts Group	64
LoC	Lines of Code	46
mocap	Motion Capture	1
MVVM	Model View ViewModel	53
OpenGL	Open Graphics Library	73
OS	Operating System	67
QTM	Qualisys Track Manager	9
RT SDK	Real-time Software Development Kit	45
SEQ	Single Ease Question	13
SDL	Simple DirectMedia Layer	51
TFS	Microsoft Team Foundation Server	29
UI	User Interface	20
UWP	Universal Windows Platform	45
Xamarin	Cross-platform framework for mobile applications	68
XAML	Extensible Application Markup Language	50

Android Mobile operating system developed by Google

AndroidTV Android TV is a smart TV platform developed by Google

BlackBerry A line of mobile devices

Cocos2d-x Open-source and cross platform free 2D game engine

Corona Labs Framework for building games/apps for all major platforms

iOS Mobile operating system developed by Apple

Kivy Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps

libGDX Cross-platform Java game development framework that provides a unified application programming interface.

Qt Cross-platform software development for embedded & desktop

React Native A framework for building native apps using the React framework

tvOS Operating system for Apple TV
Unity Cross-platform game engine

1

Introduction

1.1 Motivation

Motion capture technologies have come a long way during the past forty years, going from Calvert's potentiometer exoskeleton in the early 80's, to Acclaim's real-time solution for one-hundred simultaneous track points in ‘93 [62], and up to *Microsoft's* trackpoint-free *Kinect* sensor[13]. Current solutions provide a high degree of fidelity (depending on their price) and are used in a broad- and ever growing range of applications such as; sports (e.g. athlete analysis), aerodynamics, health (e.g. gait research), psychology, movie business and even video games. However, with better solutions comes more intricate software, complex setups and vast amounts of information; information that needs to be tracked, organized, queried and presented, sometimes even in real time. For this reason, pertinent research in the area of interaction design and motion capture systems is important in order to facilitate investigations which make use of this technology, thus increasing the amount of relevant Motion Capture (mocap) applications.

1.2 Context

Qualisys is a company based in Gothenburg that develops optical motion capture systems. Their technology has a wide range of applications in fields such as sports, engineering and medicine[1]. Each field poses different challenges to track motion data accurately and in way that provides useful data for analysis [42, 39, 41]. All of these fields share a need for tools that make the use of motion capture systems more user-friendly and intuitive.

The company currently has two mobile applications which are available in *Apple's App Store* and *Google play.* The *ViewFinder* application allows the user to stream a live camera's view to a mobile device, which is mainly used when setting up the cameras for the whole motion capture system. The other application serves as a remote control which is used to start/stop certain measurements in the software that is connected to the camera system.

1.3 Problem Definition

After a few years in the market, *Qualisys* has found that the *ViewFinder* application has not aged well; the graphical user interface (GUI) elements are outdated and the overall app structure is, by today's standards, subpar. On top of this, new cameras with more advanced features have been developed since the application was last updated, features that cannot be used as of now because they are not supported. This has lead the company to express a need for an updated application with new features and an improved user experience.

The current *ViewFinder* application was developed natively for two mobile platforms; *Android* and *iOS*, which makes occasional maintenance cumbersome for several reasons:

- Need to maintain two separate projects with entirely different toolchains.
- Even when a small change needs to be done, both projects have to be edited individually.
- It is complex to setup native development tools from scratch, let alone two of them.

1.4 Research Question

The following research question distills the aforementioned problem, narrowing the scope of this thesis and laying down the foundation for the results.

What should be considered when designing a mobile tool that aids with the setup stage of an optical motion capture system?

1.5 Proposal

To address this, an evaluation of currently existing tools will be performed and the output from this will be used to create a mobile proof of concept that we believe will be more user friendly and maintainable. More specifically, this work will start with an extensive assessment of the current state of motion capture systems and the interaction with these. Then a formative evaluation of existing tools will be performed to identify a potential room for improvement. These findings will then be used to ideate upon a solution that will then be prototyped and evaluated.

2

Background

This chapter will present the relevant background for this thesis as well as make the reader accustomed to the subject and prepared for the theory in the following chapter. Furthermore it will lay the foundation of the research and validate its relevance.

2.1 Related Work

Currently, a couple of tools are already available to aid the user interaction with a motion capture system. However the research of this thesis is only concerned with the setup stage, based on that, only two tools were identified to be of relevance. One such tool is the *ViewFinder* mobile application. It provides a set of features to assist users when setting up cameras so they can detect and calibrate them on the fly. The following features are supported by the application:

- Connecting to a motion capture system
- Viewing a specific camera output
- Modifying camera settings

The usual approach without the application is to consistently check a monitor connected to a computer that communicates with the system to see how well the camera setup works. This process demands cooperation and communication when done in pairs. In contrast, it can be a time consuming process when done without the aid of a partner due to excessive movement from cameras to computer monitor.

Vicon is another company that develops motion capture systems. The company has developed a tool similar to *ViewFinder* called *Vicon Control*. Apart from the features supported by the *ViewFinder* application it also supports additional features; *Qualisys* on the other hand provides additional applications for similar features. At the time of writing, *Vicon Control* is more up-to-date and the user rating indicates a better experience as well[33, 34]. A screenshot of each application can be seen in figure 2.1.

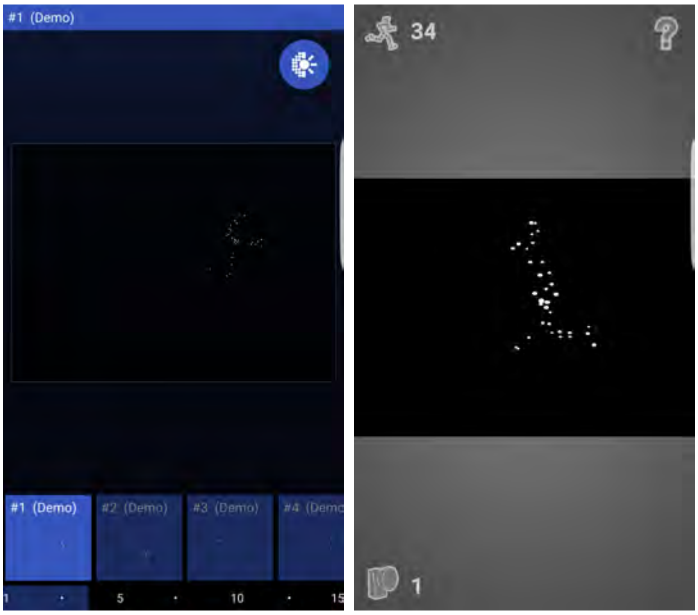


Figure 2.1: A side-by-side view of tools that aid in the setup stage of motion capture system. From left to right: Vicon, ViewFinder

2.2 Identifying Stakeholders

It is important to identify potential stakeholders to understand who a design is for. That is to say, all the people that might be affected with the success or failure of the end-result. Commonly there will be different groups of stakeholders with significantly different backgrounds and expectations, every stakeholder is important to take into account for a design to be successful. A couple of stakeholders were identified for this thesis and these were then grouped into the roles that they have from an organizational context. Additionally, their respective goals for the research were determined to better understand how they are affected by the product.

Table 2.1: Stakeholders of the ViewFinder application

Stakeholder	Goals and Interests
Support Technician	Improve efficiency during installations of the camera systems
Software Engineer	An intuitive and maintainable solution that communicates with the motion capture system.
The Customers	A good user experience when interacting with the camera system
Authors of this thesis	Explore potential ways of designing a tool that aids users during the setup stage of motion capture systems

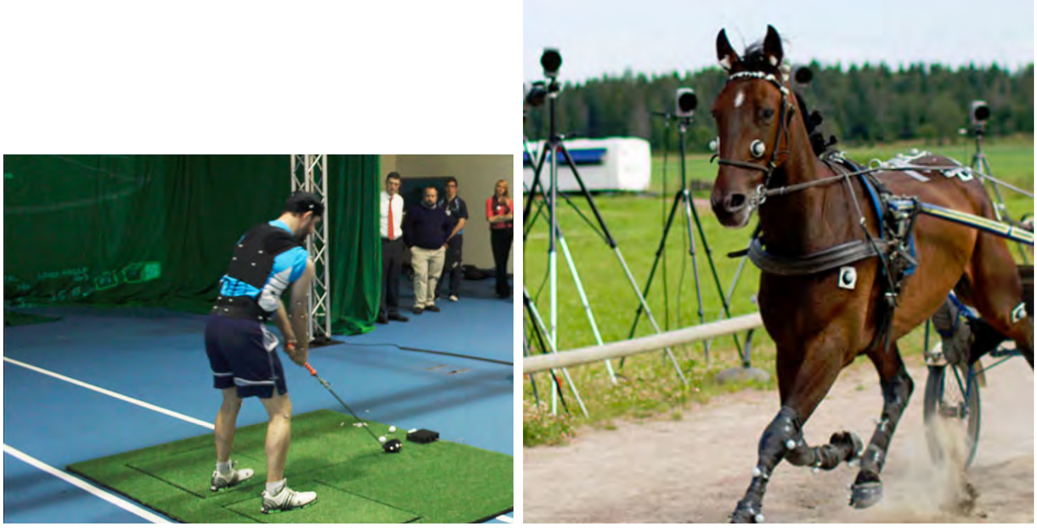


Figure 2.2: Some examples of how a motion capture system might be used.

2.3 Optical Motion Capture

Motion capture systems are used to track the movement of physical entities in space throughout a specific timespan [48]. These physical entities can be anything in the real world ranging from living beings such as humans and horses, to inanimate things like vehicles and even sea tides (see figure 2.2).

Various approaches and implementations exist when it comes to ways of doing motion capture, such as: optical motion capture, electromagnetic trackers, electromechanical devices, radio-frequency positioning and acoustic systems [48]. Qualisys' systems uses optical motion capture through the use of spherical passive markers, hence that will be the focus of this research.

These systems generally provide a high level of fidelity, even sub-millimetric resolution when operating a state-of-the-art system. A camera-rig is used to capture pictures of the desired target and computer vision algorithms are then used to interpret the data, identify markers and convert it into a 2D/3D virtual scene. Sensors track the markers by detecting a concentration of light in a specific spectrum. This light can be reflected on the markers (passive markers) or emitted from them (active markers) [48]. Due to this, it is possible for occlusion problems to arise. Optical motion capture precision is directly proportional to the amount of trackers and cameras used, therefore it can quickly become a very expensive solution if extreme accuracy is needed (see figure 2.3).

This following paragraphs will briefly go through each of the six stages[48] that comprises the marker-based optical motion capture process, with the goal of getting the reader acquainted with it (using a high level of abstraction). A visualization of this model can be seen in figure 2.4.

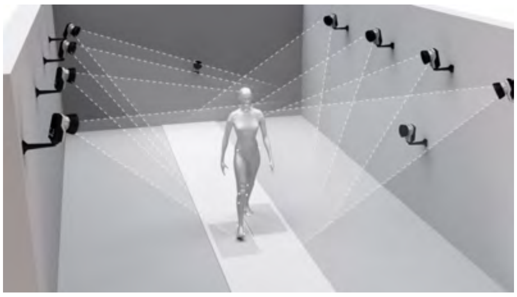


Figure 2.3: An example depicting a high-end gait analysis setup.

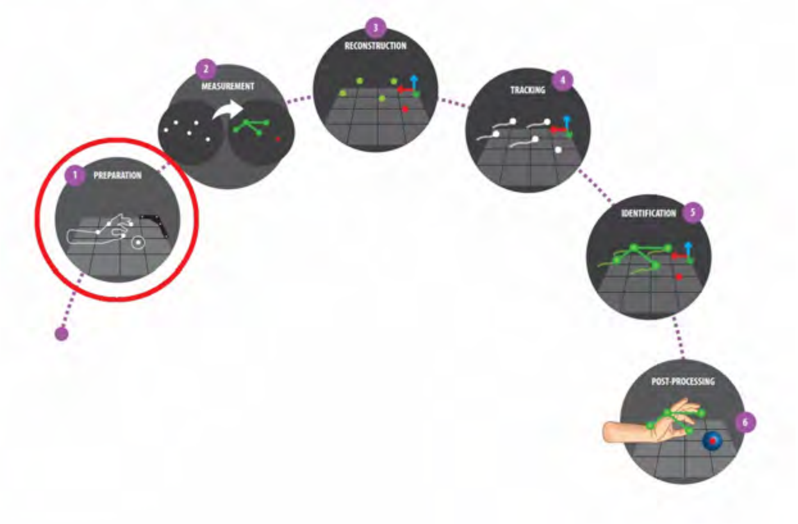
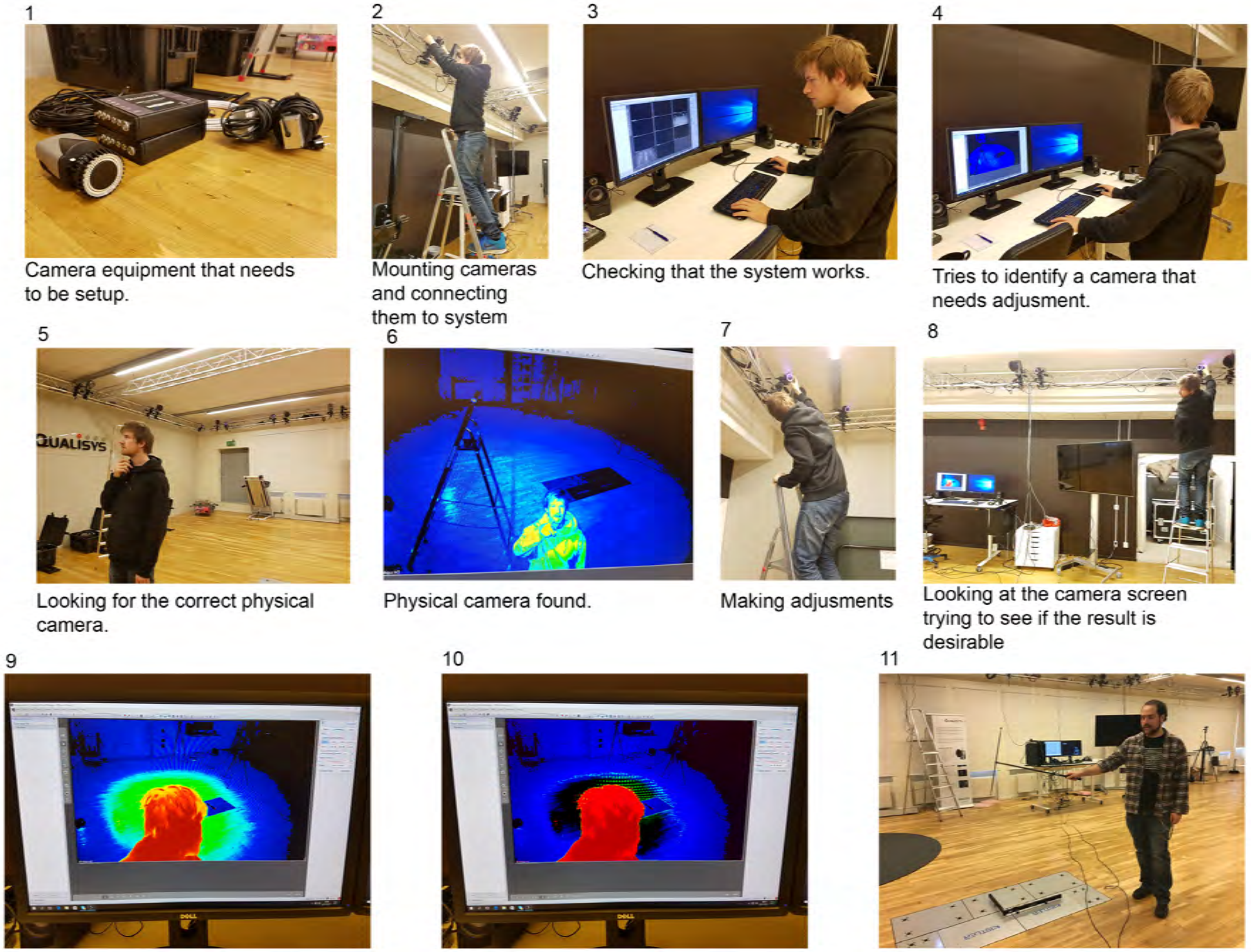


Figure 2.4: The six stages of optical motion capture, preparation being highlighted due to the importance it represents to this research work.

Preparation This step consists of setting up the measurement area and perform adequate calibrations on the equipment to ensure the best system performance under a specific environment. This is exemplified in figure 2.5. More specifically, this process includes:

- Setting up the camera rig (i.e. mounting cameras around the area, connecting them and making sure they aim in the right direction).
- Starting the software and connect to the mocap system.
- Adapting lighting conditions and camera settings such as exposure and contrast to adapt to surroundings and ensure tracking quality.
- Performing system calibration. Usually done manually by standing in the measure area and waving a special wand with markers attached to it. This provides the system with information that is indispensable to know where the cameras are located in relation to each other, as well as other important data such as focal length and lens distortion.

It is one of the main goals of this research team to design and implement an intuitive



1 Camera equipment that needs to be setup.

2 Mounting cameras and connecting them to system

3 Checking that the system works.

4 Tries to identify a camera that needs adjusment.

5 Looking for the correct physical camera.

6 Physical camera found.

7 Making adjusments

8 Looking at the camera screen trying to see if the result is desirable

9 Intensity output before adjustment

10 Intensity output after adjustment

11 Calibration of the system can commence

Figure 2.5: Exemplified scenario of the preparations stage of an optical motion capture system.

tool to help with this preparation phase, more specifically with camera placement and video calibrations.

Measurement In this stage sensor measurements are sampled and marker points are separated from the original image, this yields two-dimensional information of key points for each active camera in the set-up.

Reconstruction Using as an input the previous 2D marker information from every active camera in the rig (at least two are needed) and individual placement and orientation information for each camera (previously obtained with calibration in the preparation stage), the system transforms the points into a three-dimensional space by means of stereo triangulation[51].

Tracking Tracking consists of matching the reconstructed virtual information of the mocap subject at a given time with the respective information at future time instances. In other words, this stage follows (tracks) the movement of key points along a period of time.

Identification For practical purposes, each marker should be labeled and individually tracked throughout a motion capture session. This is trivial when active markers are used since they all have specific properties, but this stage requires additional advanced processing if passive markers are used.

Post Processing This last stage in the pipeline is entirely application-specific. For example, the resulting data could be exported to a certain file format that could then be used by film-makers to animate a virtual character in a blockbuster movie. It could also be used for advanced bio-engineering, linguistic research or even psychology studies within the field of animal biology. This stage could also be used to fix potential imperfections in the motion capture data if any of the previous stages were to alter the measurements in any way.

2.4 Qualisys Motion-Capture System

Qualisys develops marker-based optical motion capture systems with a wide range of hardware options and accessories to accommodate for very specific needs like precision, budget and flexibility. More specifically, they offer equipment such as cameras, calibration kits, markers (both active and passive) along with convenient marker-placing accessories, mounting gear and interfaces[1].

Qualisys currently offers a wide variety of cameras with different specifications that accommodate different needs for special purposes. Examples of these can be seen in figure 2.6.

The company also offers several software solutions that can be divided in the following categories:

Figure 2.6: In left-to-right order, Miqus Video and Oqus.

- Tracking software: PC suite that handles camera setup, system calibration and motion capture.
- Tracking software mobile companion: applications that connect to the tracking software to offer a *remote-control* experience and extend some functionality.
- Tracking software 3rd party integration: these integrate the real-time motion capture data with other popular software in the market.
- Developer tools: toolkits used to develop custom-made applications that make use of Qualisys' motion tracking protocol.
- Tracking analysis tools: used to analyze motion capture data for implementation-specific purposes such as running performance, gait and posture.

Some of these software solutions is of importance for this thesis research since they will be part of the foundation that the proof of concept will be built upon. The following subsections will introduce essential details about these software solutions.

2.4.1 Qualisys Track Manager

The Qualisys Track Manager (QTM) software provides a straightforward solution to capture and store data from a motion capture system. It also integrates significant functionality from other hardware solutions such as force plates, gaze-tracking glasses and electromyography (EMG) sensors. Motion capture cameras in a system are connected to a desktop computer running QTM, which interprets the data and transforms it as previously described in section **??**. QTM can track and present motion capture data in 2D, 3D and six degrees-of-freedom (6DOF); it offers both real-time streaming and playback, as well as a three-dimensional video overlay.

As of now, QTM is indispensable when setting up a mocap system because it is only through this software that the cameras can stream their information. It is worth mentioning that Qualisys' cameras can provide three different types of output, QTM can quickly change between these and so can the *ViewFinder* mobile application. These modes are:

- **Marker Mode** two-dimensional marker information
- **Marker Intensity Mode** color-coded visualization of current camera settings such as exposure and threshold
- **Video Mode** high definition video output

2.4.2 Real-time SDK

In addition to the QTM a real-time SDK has been developed that provides ways of communicating with a running QTM instance. The SDK enables developers to create applications that can discover nearby QTM hosts and connect to them. Furthermore it makes it possible to retrieve a wide array of data available to the connected QTM host. Moreover, it is possible to change the settings of connected cameras as well as retrieve events when the QTM host is updated. The SDK is available in the following languages: *C++*, *C#*, *JavaScript*, *Python*. Although, the features available for each language may vary.

3

Theoretical Foundation

Well-structured theories and solid arguments that provide a strong basis for this research are described in this chapter with the purpose of solidifying this work as an interaction design research. Some sections offer a more technical background with the aim of providing the necessary theory to follow and digest the rest of this document.

3.1 Substantiating Interaction Design Research

According to Zimmerman et al. research within the area of interaction design can be evaluated through four criteria: process, invention, relevance and extensibility. We strongly believe that our work has meaningful impact on each of these and this small section is dedicated to back it up.

Process It is this research team's mission to systematically document every step of the design and development process so that interaction designers conducting research along the lines of this specific area can reproduce it in its entirety.

Invention The nature of this work is not only theoretical, a working proof of concept will be developed along the way which will encompass all knowledge gained throughout every stage. The proof of concept developed along this work will not only improve the interaction experience within Qualisys' gamma of software solutions, it will also put to evidence the processes and methodology exercised to create it.

Relevance This kind of research is very relevant for today's motion capture industry because the incorporation of modern, mobile platforms in the mix is fairly new. Experimentation in this field is still being carried-out and novel uses and interactions are yet to be discovered. It of interest for Qualisys because it would mean a significant upgrade to their current solution, bringing mocap system-setup time down through the use of a more powerful, intuitive and responsive tool.

Extensibility As previously stated, everything related to the design and implementation of the research product is thoroughly described in this document, rendering other employees, researchers, students and/or enthusiasts able to study it and carry on with the work. From a more technical perspective, it is one of the main goals to provide a solution which is highly extensible and easily maintainable.

3.2 Interaction Design Research Contribution

Besides process documentation, contributions from this research will also come in the form of artifact research. Wobbrock and Kientz summarizes artifact contributions as something that "arises from generative design-driven activities (invention)". The first part of this thesis will evaluate existing relevant tools to get an understanding of mocap-assisting technologies and their current state. Next, a tool will presented based on user- and system needs and eventually evaluated in a holistic fashion. Throughout this work; mock-ups, sketches and prototypes will be made to substantiate features and design choices. This process will be iterative and part of re-framing the problem in an attempt of making the *right* thing, which according to Zimmerman et al. is the real skill a designer possesses.

3.3 Target Users

It could be argued that optimizing the created proof of concept for experts would be relevant since the majority of the people that will have use of the final product will have technical knowledge about the subject. However, it needs to be taken into account that the usage of the tool won't be extensive, and thus the learning curve should not be too steep. Based on that, the aim is to optimize for the perpetual intermediate. At the same time, emphasis should be put on easing the load for the users that want to become experts. [43] introduces a threefold approach when optimizing for intermediates that we believe is valuable for this research:

- To rapidly and painlessly move beginners into intermediates
- To avoid putting obstacles in the way of intermediates who wants to become experts
- Most of all, to keep perpetual intermediates happy as they move around the middle of the skill spectrum

3.4 Evaluation Paradigms

Evaluation is an important part of interaction design research to elicit what works in a given context. Preece et al. describes four paradigms of evaluation that approach it from different perspectives. During this research combination of all four approaches will be used.

Quick and Dirty Quick and dirty evaluation essentially means getting fast input that is potentially valuable. The documentation process should not be time-consuming and it usually takes the form of noted key points or rough sketches. This type of evaluation usually occurs during informal talk where presence is key to understand the users and enables the designer to put emphasis on what is really going on. The evaluation method will be used extensively during the initial phase of the research, and more sparsely during later phases but still very much present.

Usability Testing Usability testing attempts to carefully measure how users perform certain tasks. These tasks are usually well defined and performed in a controlled environment. Data is generally gathered by taking a lot of notes, logging behavior as well as recording both audio and video. The observational data can then be used to analyze what users are doing and how much time they spend doing a task. Part of the goals for this research is to investigate how to go about creating unobtrusive interfaces, by that usability testing will be a major part when attempting to validate artifacts throughout the research.

One method that will be used during usability testing is the Single Ease Question (SEQ). It is a method used in questionnaires with the aim to get perception-metric results [30]. It is intended to be answered by a user shortly after completing a specified task. It works by grading the ease of use of a task using a seven-point rating scale which goes from *very difficult* to *very easy*. Obtaining task-performance satisfaction data from the evaluation stage of this thesis is valuable due to the iterative nature of the design process. Furthermore, it is imperative to provide an optimal user experience.

Field Studies Field studies should be carried out in natural settings, which can be anything from a work place to a bench in a park. Emphasis is put on understanding the natural behavior of the user and how it might be improved. The studies are usually performed using qualitative research techniques such as interviews and observations. [58] introduces two approaches to field studies; outsider- and insider-observer. The former depends on doing observations without interrupting the flow of the users. Data is then analyzed which might produce qualitative or quantitative results. The latter involves the researcher taking on the role as someone who is performing the task in its natural context.

Predictive evaluation The final evaluation method that will be introduced is *predictive evaluation*. In contrast to the other methods it depends primarily on expert evaluators and not the users. Experts commonly base their predictions on heuristics and perform analysis using evaluation models. The final data output are expert reviews and quantitative data models. Compared to other methods predictive evaluation is more efficient with a relatively low-cost. However, there is a risk of being lead astray when using heuristics and care must be taken into which heuristics to use.

3.5 Mobile Computing

As will become evident in future chapters, during the design process it was decided that the proof of concept would be a mobile application. Concepts relevant to mobile computing will hence be introduced in the following sections.

3.5.1 Mobile Interaction Design

Formal interaction design studies within the spectrum of mobile platforms are still very young, nevertheless this area has been subject to a really fast growth in the past decade due to important advancements in technology and in the way in which we interact with it. User-interface guidelines and interaction standards are becoming more common as the tools for mobile software development become easier to use and open up to a new set of designers and programmers from different backgrounds. As both hardware and software evolve, our understanding of how to incorporate these gadgets in our daily life becomes more apparent. Back in 2007 with the introduction of the *iPhone*[57] and the advent of an online application marketplace, mobile software used to be treated just like any other legacy desktop PC software; they were costly, standalone experiences that provided very specific functionality. Nowadays we do not develop such applications, we now focus on *complementing* and *enhancing* a user's traditional workflow[53]. In other words, mobile applications are commonly used as tools with limited functionality that extend, simplify and boost other already-existing services and software. Mobile platforms are useful as workflow-enhancing tools when dealing with:

Dynamic workplace provides portability by being wireless.

Limited maneuverability relatively small compared to desktops or even laptops.

Physical impediments by providing stripped-down, simpler software that can be operated by other means than classic computing (keyboard and mouse).

Out-of-the-box interactions through novel motion-enhanced gestures, touchscreens and even GPS.

They can extend and improve previously developed functionality through faster and more intuitive ways of interaction, mobile platforms can also offer new interesting ways of interaction by means of motion gestures, speech and even tactile feedback. Because of all these reasons motion capture companies have started to invest in mobile applications that can be used to assist with complex tasks. The mobile application abstracts and isolates particular functionality from a rich mocap desktop suite; this in turn provides a more intuitive interaction by letting the user concentrate on the task at hand. The research team considers this to be a strong argument in favour of using a mobile application as means of aiding with a motion capture system's setup.

3.5.2 Cross-platform Development

Software development for modern mobile platforms (i.e. Blackberry, iOS, Android) has come a long way since the release of the first Blackberry *smartphone* back in 2002[57]. Since then, development tools and environments have evolved a lot, becoming cheaper, more accessible and relatively user-friendly. Each major vendor offers a separate set of toolchains and software development kits to build applications for their own platform. This means that in order to create a program that is - for example - meant to run on *Android* and *iOS*, different project instances, codebases, IDEs and even operating systems need to be used. This makes it highly impractical when writing, updating and testing features because both ends need to be edited separately.

There are several frameworks and libraries that try to mitigate this by offering a common layer of abstract code on-which developers can build shared components, drastically reducing development time among other difficulties. To fulfill the goal of easing the maintainability of our proof of concept, one of these frameworks will be used. The following section provides an insight to some of the more popular solutions at the time of writing.

To ensure that our proposed solution is maintainable, extendable and relevant, a rigorous evaluation of several different frameworks was carried-out to conclude how well they fulfill certain criteria, the most important being the quality and robustness of the support for most modern platforms. The evaluated frameworks are introduced in the following sections. However, the actual evaluation will be presented in chapter 5.

Xamarin
Xamarin is an open source cross-platform SDK owned by *Microsoft.* Applications developed using *Xamarin* are written in C#, usually in combination with either *Visual Studio* or *Xamarin Studio.* One of the key features of the framework is that you can share most of the codebase with other platforms while still being native at the core. Instead of restricting developers from using platform-specific Application Programming Interface (API) calls, it encourages an architecture where platform specific calls are part of the development. This provides developers the ability to make full use of the platforms but keep repeated code to a minimum.

In addition to the SDK they develop a cross-platform UI-generation tool called *Xamarin.Forms* that can be used to share even more code among the platforms. It is a well suited approach to UI creation when platform-specific functionality is of less importance.

To streamline the development process and automate the testing process *Xamarin* offers a service called *Xamarin Test Cloud.* The service is based on a framework called *Calabash* that let's developers test anything from the user interface down to the core logic. The fee of using the service depends on the amount of devices and device hours being used[2].

Qt

Pronounced "cute", it is a cross-platform framework used by several leading companies and organizations. It is designed to let developers create multi-platform applications and graphical user interfaces [3], making it an attractive choice for software developers of embedded systems. In contrast to the other evaluated frameworks, *Qt* applications are written in the language *C++.* The framework provides several individual modules to make development of applications more convenient. *Qt Creator*, which is the officially supported IDE, provides tools to optimize build steps, run environments, UI design tools and others [25]. In addition, *Qt Lite* was introduced as a new configuration system to be a part of the *Qt 5.8* release that enables developers to define in greater detail what modules to include in a build, thus enabling developers to decrease build size [26].

Qt provides a *QML*(Qt Meta Language) module which can be used to write the front-end code of the applications. It is a declarative language with JSON(JavaScript Object Notation)-like syntax that focuses on the visual aspects of components and their interaction with each other [28, 24].

React Native

Facebook develops and maintains an open source framework called React Native that lets developers create cross-platform applications using *JavaScript* and *React.* The final build is compiled into native code. A part from writing code in *JavaScript* developers can also write native code, something that is useful when optimization of features inside an application is needed. Motivations behind the framework is to provide an efficient way of writing cross-platform applications using a language that are well known to most developers. It is a relatively young framework and as of now there exists no official documentation about usage of the framework in combination with 3D graphic libraries, but there are third-party libraries available.

Unity

With industry-leading multi-platform support, *Unity* is an interesting contender amongst the candidates. It is a game engine that supports both 2D- and 3D-graphics, and as the name suggests, it is mainly used to quickly develop video games through the use of highly-abstracted layers. Although *Unity* focuses on games there is nothing that prevents developers from using the engine for other purposes such as cross-platform mobile development, but it has to be kept in mind that its modules and libraries are mainly focused on interactive media. *Unity*'s engine cannot be modified nor extended, but extra functionality and behaviours can be implemented through the use of scripts which are written in *C#* or *JavaScript*, making it attractive to developers whose proficiency and language preference may vary. A major drawback comes from the fact that final application file sizes tend to be quite big compared to other frameworks due to the engine been compiled in its entirety even when some functionalities are not used.

Kivy

Kivy is an open source framework written in Python. It is free to use under the MIT License and *LGPL 3* for the previous version. The engine used in *Kivy* supports hardware-accelerated graphics and thus boasts the ability to create efficient and fluid user interfaces[14]. Compared to other frameworks it aims to be agnostic when it comes to look and feel, the motivation behind this is that it will reduce the need for maintenance and possible bugs[56]. It is a comparably small framework and community-driven project and by that does not have the same resources for development and maintenance of the framework as the others mentioned above[15].

Others

Some additional frameworks were evaluated, besides the frameworks already mentioned, but got scrapped early on since the did not meet the criteria. They are briefly mentioned in the list below accompanied with a motivation of why they were discarded.

- Cinder: *cinder* is a *C++* library which is mainly used for *creative coding*, a term which is used to describe a field that combines art & design with programming[59]. It is open-source and it has a growing community focused on creating new and experimental technological experiences. It does not support *Android* officially, although due to its open-source nature some developers are working on this.
- Xojo: uses the *Xojo* language (not supported by QTM real-time SDK), doesn't support *Android* builds.
- *Cocos2d-x*: fast multi-platform game engine with small footprint, written in *C++*. Doesn't support native UI calls and the widgets included are limited.
- *libGDX*: supports plenty of languages (i.e. *Java*, *Kotlin*, *Groovy*, *Clojure* & *Scala*) and platforms, but just as *Cocos2d-x*, it is mainly oriented towards game development and UI functionality is very limited.
- *Corona Labs*: it is a free and powerful tool when it comes to multi-platform development, as it currently supports *iOS*, *tvOS*, *Android*, *AndroidTV*, *Kindle*, *Windows Phone*, *Mac* and *Windows* with concurrent live builds. However, code is written using the *Lua* scripting language (not supported by real-time

SDK).

3.5.3 Platform Architecture

When working with cross-platform development it is valuable to understand what is going on under the hood. This enables developers to make informed decisions on an architecture that targets several platforms. The platforms targeted in the scope of this thesis are *iOS* and *Android* since they are the most prominent mobile platforms to date. Their respective architecture will be introduced and briefly described in the following sections.

Android
The *Linux Kernel* layer provides low-level functionalities such as threading and low-level memory management. It provides an abstraction of underlying complexities that concern hardware, something that in most cases are unnecessarily close to the hardware for standard mobile application development. There is yet another layer on top of the previously mentioned that rarely needs to be touched unless very specific functionality is needed, this layer is called the *Hardware Abstraction Layer*(HAL). Its main purpose is to expose device functionality to the higher level *Java API Framework* and does this by providing multiple library modules that targets several different Android devices. Developers are usually working in using the *Java API Framework* layer which provides building blocks for development that reaches the entire feature set of the Android OS. There is one last additional layer in the Android stack for the *System Applications* such as messaging, email and calendar that is there to provide developers with functionality for common use cases[4]. An overview of this architecture can be seen in figure 3.1.

iOS
The bottom layer is known as the *Core OS* layer and it contains low-level features which are closely coupled with the hardware. An example of this is the *Accelerate* framework which provides fast calculations in the areas of linear algebra, image-processing and digital signal processing [9]. The *Core Services* layer sits on top of this, providing system services such as location, cloud storage, social media and networking [10]. The next layer is the *Media* layer which is there to provide developers with functionality to more easily create an application that sounds and looks good [12]. Finally there is a layer called *Cocoa Touch* which is the most prominent framework when building *iOS* applications. It provides an API for touch interactions and gestures amongst other features. The *iOS* guide recommends looking into this layer before going further down in the architecture stack[8].

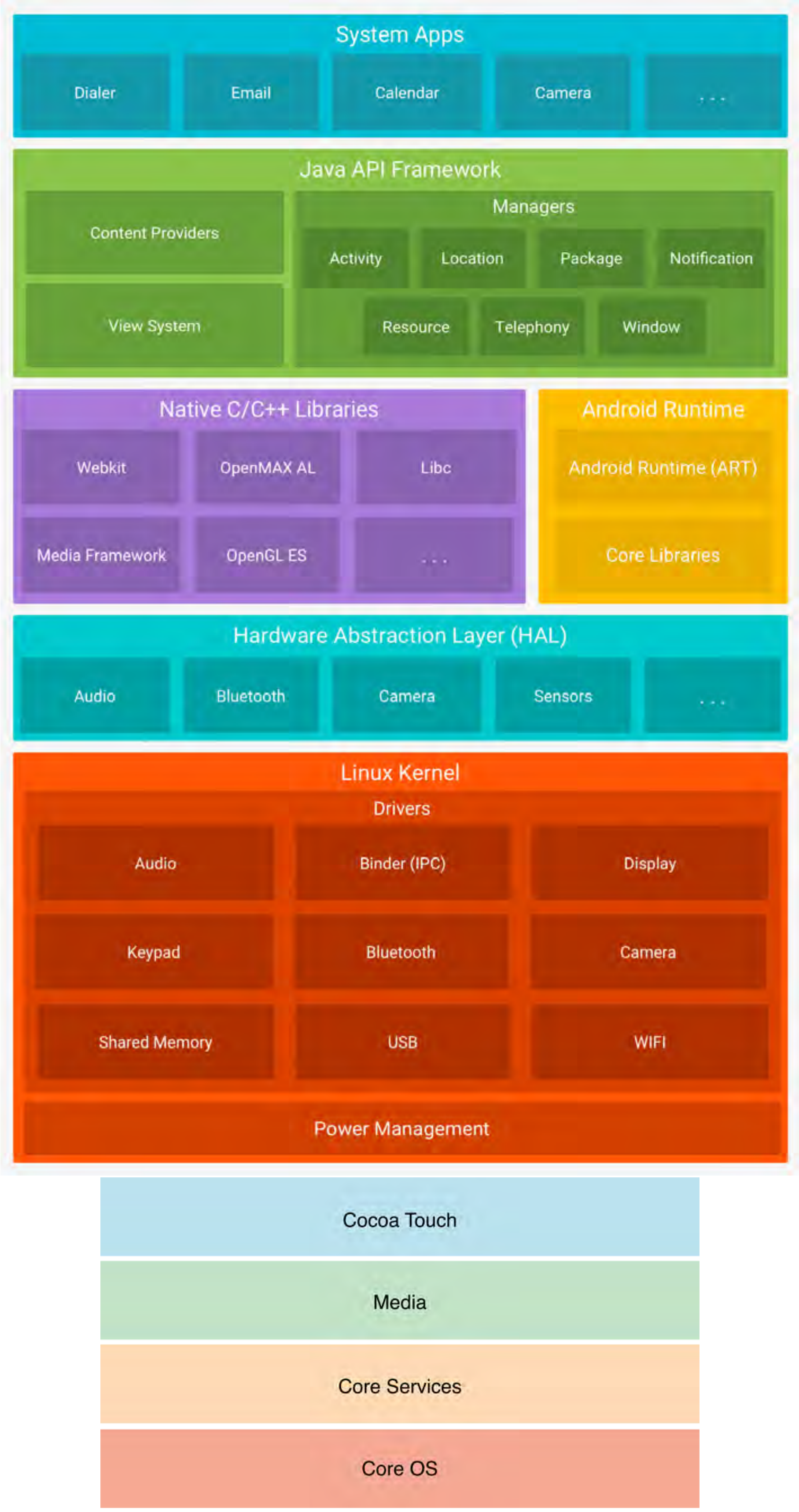


Figure 3.1: An overview of the layered architecture of both Android (on top) and iOS (below). Red indicates close-proximity to hardware.

3.5.4 Interaction Design Challenges

Cross-platform mobile development does not come without its set of problems, the most relevant within the field of interaction design being the fact that there are different guidelines and implementations for each platform, these include: color guidelines, application flow, fonts and UI-control elements amongst several others[53, 19, 11]. It can be tempting for designers to get carried away with native User Interface (UI) elements and components, altering the application structure and completely changing the way a user interacts with the system across different platforms. Although it is virtually impossible to offer the exact same experience on multiple operating systems, some care needs to be taken to structure interface elements in a similar manner. Bottom line is, the user experience on one platform should not be better or worse than on others [53].

4

Methodology

Methods relevant for this thesis will be introduced in this chapter. Part of the methodologies will revolve around evaluating the current state while others are there to aid in exploring new ideas, iterating upon the ideas that seem valuable as well as evaluating possible design solutions.

4.1 Interaction Design Activities

[58] identifies four basic activities for interaction design, the necessary methodology to carry out each individual activity is described throughout this section. Figure 4.1 shows an overview of these activities.

4.1.1 Identifying Needs and Establishing Requirements

Interaction design is about designing for people, and in order to successfully do this such people - the target group - and their needs need to be thoroughly analyzed. This phase provides the designer with the necessary information to create a set of requirements on which the rest of the design-development cycle will be based.

Observations

It is important to understand human behavior in interaction design research, and by that find a design solution that solves an actual problem. Observations are one method of doing ethnographic research to understand human behavior. Observations can be done directly or indirectly, as an observer or participant; it is not uncommon that the ethnographer switches between these roles to get an even broader understanding[61].

Researchers can take several approaches when doing observations and most of them consist of documenting events, behavior, reactions and other aspects that might be of interest for the research. There is no optimal approach so it is important to be able recognize suitable and efficient ways in accordance to the context. Since this

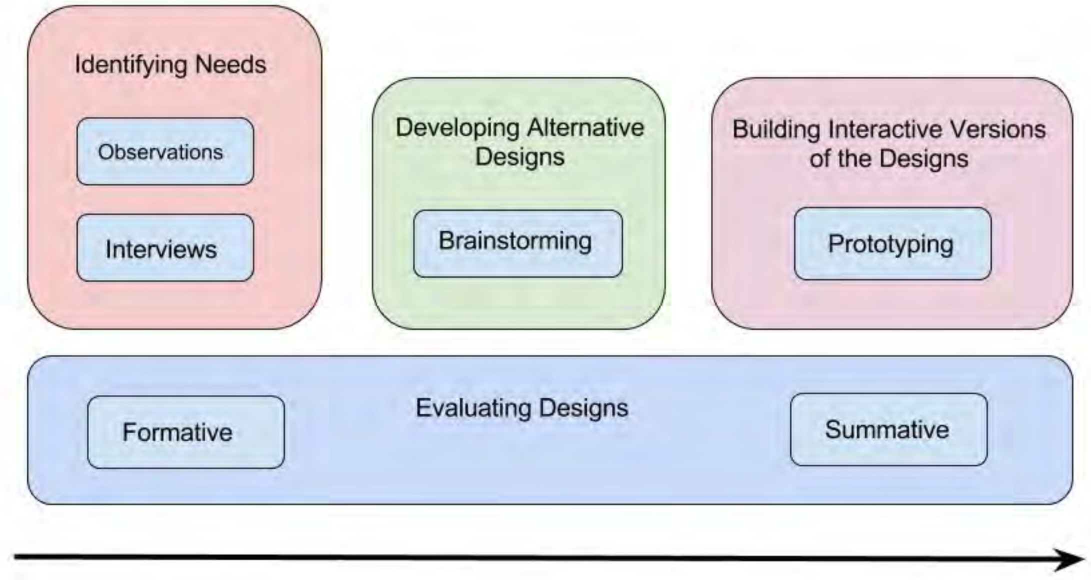


Figure 4.1: An overview of the design process.

research approaches experts and possibly intermediate users the observations had to gather detailed and qualitative data. With that in mind, the following ways of documenting the process were used:

- Notes: To recognize patterns and phenomena
- Audio- and video-recording: To analyze reaction, behavior and speech
- Screen-casting: Recording of a mobile device's screen to further identify key behavioural aspects.

[60] describes a technique when taking notes during observations. The technique works by taking notes on human action and make ticks on each repeated action. Actions are then categorized according to how many times they occur. One tick means that it probably isn't something of significance, two ticks might indicate that a pattern is emerging and three ticks that a phenomena has occurred. Phenomena are something that with high certainty will occur more times and thus should be given more attention.

Audio and video recording are tools that can be used to make in-depth analysis of a situation. During live observations there is a possibility that events of importance are missed. In those cases, an audio or video recording might be proven invaluable. It is important however to not use this as a substitute for doing real observations in a natural environment. There are several reasons for this; for example, one may move around asking questions and perceive reactions more clearly, while performing live observation than watching it on a recording afterwards[61].

To support the methods mentioned above and get more in-depth data a method called *think aloud* was incorporated during observations. *Think aloud* works by prompting the subject to constantly speak their thoughts when performing a set of tasks. It is important that the room facilitates this behavior so the subject can talk without being disrupted or feel uncomfortable. A think aloud session is typically recorded and then later on transcribed and analyzed [47]. During the observations done for this thesis, we made a habit of taking notes and asking questions to clarify behavior when necessary to get a more full picture of what was going on. Each session was followed up with a brief and concise semi-structured interview.

Interviews
Interviews were performed in a semi-structured fashion [58]. They were formulated based on a set of guidelines with relevant questions, the interviewee was then asked to elaborate on each question to get a better understanding of their intention behind the answer. During each interview there was an interviewer, one or several interviewee(s) and a note-taker to ease the load of the interviewer so that full focus could be put on the interviewee(s).

4.1.2 Developing Alternative Designs

Part of the work consisted of exploring and developing ideas. This is commonly referred to as the *ideation phase*, a word that [64] use to describe "generation of many possible solutions". The main ideation phase should take place after the problem has been clearly defined. It is only possible to come up with relevant solutions with a clear understanding of the problem. This understanding can be achieved through field studies, observations and evaluation of the current state.

Brainstorming
Brainstorming is the act of having intensive discussion about problem solutions and(or) generation of ideas [7]. [54] argues that brainstorming can be an art; something that one continuously can become better at. Building on those ideas, the work done here will embrace previous findings on what makes up a good brainstorming session and apply them in practice. We will use the following list, based on our own experiences as well as other researchers within the field [54], to guide us during our brainstorming sessions. It is our belief that this will help us gather more valuable ideas.

We aim to:

- Have clear problem statement
- Encourage all ideas
- Number ideas
- Visualize the flow (e.g. using post-its)
- Be creative and visual

We aim to avoid:

- Letting the boss speak first
- Employing a turn based strategy
- Only letting experts talk
- Getting too silly
- Documenting everything

KJ Technique
The *KJ-technique* (also called affinity diagram) named after its inventor *Jiro Kawakita*, will be used in combination with methods for idea generation such as brainstorming, with the aim of reaching consensus of top priorities more efficiently through evaluation of ideas that encourages collective thinking[17]. A fundamental part of the technique is that there should be no discussion between the group members until the very end of the session. The technique consists of several steps that will be listed in chronological order below:

1. Determine a focus Question
2. Organize the Group
3. Put Opinions onto Sticky Notes
4. Put Sticky Notes on the wall
5. Group Similar Items
6. Naming Each Group
7. Voting for the Most Important Groups
8. Ranking the Most Important Groups

For the work of this thesis there will be no facilitator, even though that is something that is recommended[17]. This is mainly due to the amount of people available for the work of this thesis and partly due to our belief that if we are able to obtain more people for a session then it is better if they are joining the idea generation process instead of facilitating a session.

4.1.3 Building Interactive Versions of the Designs

After the ideation phase, a somewhat clear vision of the actual product should be identified. The design activity of building interactive versions of a design consists of developing an artifact with a specific level of fidelity. This level of fidelity depends on the current state of the design. The creation of this artifact can be iterative and build upon previous results once they have been evaluated.

Prototyping
Envisionment of ideas make design work visible to ourselves, and to others[40]. Prototyping is commonly used to create concrete but not fully fledged implementations

of a system or an idea and thereby envision it. One of the skills of a designer is to find a suitable representation when doing envisionment. A good representation is according to [40] "accurate enough to reflect the features of the system being modelled, but simple enough to avoid confusion". This work will be on the more technical side of interaction design, thus some techniques will be more suitable than others. The following list will introduce prototyping techniques and motivate why we believe they are suitable.

Low-fidelity Fast prototypes will be viable in some cases, such as trying out and do user tests of an interaction.

Sketching Quickly visualizing ideas is an effective way to do prototyping for most designers. In this work it will be anything from user interfaces to system architecture.

Mock-ups Wireframes to create early design and test with the users

High-fidelity Prototyped applications of critical features to figure out how the system should be designed

The goal is to create an environment and a resource that enables developers to create a novel and intuitive experience when interacting with motion capture systems. To do this, high-fidelity prototypes will be created to determine feasibility, usefulness and areas of improvement. This will also help to determine requirements of the final system.

Designs can always be improved and the same goes for software architecture; it is a wicked problem and there exists no optimal solution.

4.1.4 Evaluating Designs

In this design activity the usability and acceptability of the produced solution(s) are measured against specific criteria. This phase determines if the product is indeed worthy of further development. This stage is also very important when working with an iterative design process as it will yield critical results that will then be plugged to another round of ideation and development. *[...] without evaluation, designers cannot be sure that their software is usable and is what users want* [58].

What to evaluate depends on the context [58]. The context for this work is mobile real-time interaction with motion capture systems and a bi-goal is how to design a software system that supports multiple mobile devices without affecting interaction negatively.

To guide our evaluations we will be using the *DECIDE* framework proposed by Preece et al.. It is an abbreviation of six things that should be taken into consideration before evaluation, they will be explained in the following list.

- **D**etermine the goals to guide and emphasize on what is truly important
- **E**xplore and dissect questions to make the evaluation more specific
- **C**hoose an evaluation technique to add structure
- **I**dentify practical issues that must be addressed
- **D**ecide how to deal with ethical issues that might arise
- **E**valuate how data should be collected, interpreted and presented

According to Preece et al., there are two types of evaluation. The first one is **formative evaluation** which occurs during the early stages of the design process. This is used to evaluate current product usability. It helps to understand in what degree the product meets the user's needs. At the end of a design process iteration there is a **summative evaluation** to asses the result. Any type of evaluation can always be performed during any stage of the design process, this ensures some level of quality and helps the designer to focus on the problems at hand. During the work of this thesis, a major formative evaluation will be performed to evaluate the current state and will be an important part of the process when defining the problem. A later summative evaluation will be carried to assure that artifacts (proof of concept) and potential solutions are adhering to user needs.

4.2 Planning the Design Process

There exists a vast amount of different design processes that organize and further delve into the previously described design activities[58], this reaffirms the fact that there is no one way to approach design. Interaction design is essentially about understanding the users and the actual problem, and then find a way to create a design that takes the problem to a preferred state.

Our design process will take inspiration from the Double Diamond (DD) Design Process that was introduced by *The British Design Council*. The process stems from qualitative research on design approaches of major companies with leading corporate designers. Based on their expertise in theories and practices of design management, they analyzed their findings and found striking similarities, something that eventually came to be the DD process[44].

Even though our process will be inspired by the process mentioned earlier it is important to emphasize that our process will not be static but rather dynamic. There is no obvious way when it comes to how design should be approached and having some flexibility is good to be able to adapt to changing requirements and knowledge about the domain. This has been acknowledged before in research about what should be expected from research through design and [50] states that "..theory by necessity under-specifies design activities".

The DD design process is structured in a way that addresses design challenges in four phases:

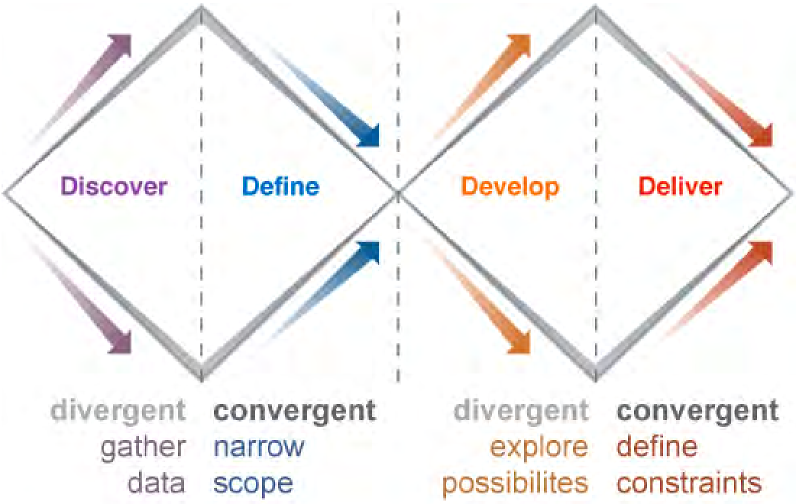


Figure 4.2: Two major stages of the different phases in the double diamon design pattern.

- Discover/Research
- Define/Synthesize
- Develop/Ideate
- Deliver/Implement

Furthermore, the phases in the list can be grouped into two major stages. Both of these have a converging phase in the beginning and a diverging one in the end. The focus of the first stage is to figure out how the problem should be addressed and what the questions that should be asked before proceeding towards an actual solution are. The second stage is about doing the *right* thing, to accomplish this it is important to explore possible solutions through ideation and other methods of exploration. Finally a design solution is evaluated based on the defined problem. A model of these stages can be seen in figure 4.2.

The following subsections describe the phases in more detail as well as relate to how they will be integrated as a part of our design process. How the final design process came to be will be explained in greater detail in chapter 5.

Discover/Research

Consists of exploring and dissecting the challenge in order to get a better understanding of what lies ahead, identifying obstacles and gathering as much information as possible within related fields of study. Two types of research are done in this phase, we present these along with the fields concerning our work:

- **Primary:** Interaction design, motion capture systems, mobile development and cross-platform development.
- **Secondary:** Xamarin platform and in-house QTM tracking software along with its real-time protocols.

Define/Synthesize

After getting all the relevant findings from the previous step, it is important to summarize them, find similarities between them and discard those that are not sufficiently relevant. After this, the research question will become more clear which in turn will make it easier to find a valuable answer. Clear areas become more apparent and it gets easier to identify problems and ask pertinent questions. Thus, inherently setting the foundation for the ideation phase.

Develop/Ideate

During this phase, ideas regarding possible design solutions are generated, developed and iterated upon. Exploring and evaluating different paths is very important for it enables a team to make thoughtful choices.

Our approach was to ideate on possible features and prioritize them according to how valuable they were as well as how critical they were to the final product.

Software critical features were to be prototyped as working applications to ensure expected functionality. It could be argued that this actually belongs to the previous phase. However, we saw it as an important part during the *Ideation Phase* so that we could quickly jump back to the *Define Phase* and reframe the problem if necessary.

Deliver/Implement

Based on ideas and prototypes from the *Ideation Phase*, sufficient data should have been gathered in order for the actual product implementation to begin. All realizations made during the previous phases should act as a guide on the design of the final result.

5

Design Process

The purpose of this chapter is to give an understanding of the choices that were made and how the final design solution came to be. Furthermore it will explain the design process that was revealed during the work of this thesis as well as the gained insights.

5.1 Evaluation of Frameworks

Several cross-platform frameworks were introduced in section 3.5.2; these were evaluated to figure out which one was more suitable for the proof of concept. The criteria that were used to evaluate said platforms is listed below.

- Real-time SDK compatibility
- Developing environment
- Compatible platforms
- Deployment process
- Testing
- Look & feel options
- Access to platform-specific APIs
- Filesize
- Access to 3rd party libraries
- Compatibility with Microsoft Microsoft Team Foundation Server (TFS)
- Cost-licensing

The selected framework and environment should preferably be familiar for developers of the motion capture system used with the proof of concept. To keep the need for maintenance to a minimum it is important to aim for correctness, with that in mind, possible testing frameworks for each candidate were also evaluated. Automation of processes such as publishing and deployment was desirable as well.

Making the application look aesthetically pleasing was not part of the criteria. However, flexibility for look and feel specification was considered. Using cross-platform UI code for native look and feel provides a way of automating the graphical interface using autonomous UI generation which in turn reduces the development time and might result in higher productivity [55]. Moreover, UI generation can provide a well tested set of modules that works well and at the same time be adaptive to the context. Even though native components may be adaptive there are cases were a more customized look is required.

Furthermore, to reduce storage, load- and download-time the file size of the application should not be unnecessarily large. Finally, suitable libraries were evaluated that might ease the load of development time and add reliability due to support from communities.

Based on all this, Xamarin was the framework which was deemed the most suitable. It provides a way of sharing a single C# code-base, which is supported by the real-time SDK. Additionally, it compiles into native code. -This translates to a small application filesize and good runtime performance. It is important to notice that the Xamarin framework can be managed within Visual Studio, which is a recommended IDE when working on a Windows PC. This makes integration with Microsoft TFS more accessible. Finally, the license cost for Xamarin application development is dependent on the Visual Studio licensing, thus it may vary depending on customer needs[35].

5.2 Requirements Specification

This section aims to provide an insight into this work's requirement gathering process. [58] distinguish five different types of requirements (functional, data, environmental, user & usability), table 5.1 shows an abstract set of these which were identified in the early stages of this work. The results and thresholds for the respective criteria and how well they are fullfilled by each framework is provided in appendix F.

5.2.1 Elicitation

A user-centric elicitation approach was done to find and transform needs to formal requirements. Instead of having a focus on what the users should be able to do with the aid application the aim was to understand what the user was doing with the old application and find out how this experience could be improved further. During elicitation a couple of techniques were utilized to find out the requirements for our work, these techniques will be briefly introduced in the following sections as well as the findings from using them.

Observations
Besides eliciting requirements from the previous *ViewFinder* application through

Table 5.1: ArqusFinder application requirement overview

Requirement	Description
Functional	Application must connect to a QTM host through a LAN and access every camera's stream and settings, it must also be able to tweak said settings.
Data	Application must present specific real-time motion capture information based on the user's input (i.e. marker positional data, marker intensity feed and camera stream feed).
Environmental	The application is intended to run on a controlled environment (e.g. a studio) where a motion capture system has already been connected to a QTM instance.
User	Intended users must be acquainted with the system (this includes both hardware and software), after all, it is the application's goal to make a mocap technician's job easier.
Usability	Application must be as intuitive and non-intrusive as possible, it must offer clear functionality through a clean interface to promote user flow.

feature analysis, there was a need to evaluate how it worked in-action with users. The aim was to find out what the users used the application for and how well they were able to perform their intended tasks with the aid of the application. Prior to observations a couple of preparations were done. Firstly the camera system was intentionally disarranged to ensure that the subjects had a reason for performing certain tasks. Secondly the subjects were asked if they were okay being video-recorded during the observation to enable the authors to go back and analyze the sessions in a later stage. Additionally an attempt to record the mobile device screen was done but due to technical difficulties the authors decided not to present nor employ the data in this work.

Finally the subjects were prompted to do *Think Aloud* as they performed their tasks so that their rationale behind their actions could be studied as well. During observations one observer would take care of the recording whilst the other made notes on anything potentially interesting. Furthermore, the observers asked the subject to clarify actions when what was said and what was done was not sufficient to fully grasp the action. A technique to recognize phenomena and patterns were employed during the observations to find recurring behaviour which is described in 4.1.1, this technique resulted in a couple of patterns listed below.

- Looking at cameras to identify their order
- Going back and forth to the QTM desktop application
- Putting down mobile device to move cameras
- Moving to pick-up and start interacting with application
- Changing camera focus and tweaking settings while in intensity mode
- Corroborate with QTM after tinkering with a camera with the help of the ViewFinder application

Figure 5.1: A focus group session with discussions surrounding QTM's features and the real-time SDK.

The material was carefully studied and discussed post-observations and a couple of requirements and scenarios were formalized based on the findings from the observations. These findings will be presented in chapter 7.

Interviews

Semi-structured interviews were conducted following the observations. These interviews were based on the authors findings from previous evaluations of the *ViewFinder* application. The formalized questions worked primarily as a catalyst for discussion regarding how the application could be improved but also as a way for the authors to confirm that their findings were in-line with the users own thoughts towards the application. A major goal with the interview was to let the users elaborate on aspects of the *ViewFinder* that were hard to observe and discuss what these aspects meant for the application. Worth emphasizing is that these interviews did not provide any results on their own but rather worked as a complement to the result from the observations conducted previously. Additionally a lot of informal interviews with key stakeholders were had to discuss relevant subjects for this work.

Focus Groups

Exhaustive meetings were held several times with key members and stakeholders. These meetings were mildly-structured as there were no formal guidelines nor were they handled by a facilitator, but they were always flowing through a main discussion theme. These sessions were used to discuss matters such as the current application's features, lack of features, strengths and weaknesses. Other matters included cross-platform mobile development and the challenges that this represented to them and their teams. During these sessions the stakeholders would extensively talk about how their software suite had evolved and how it was important to incorporate new functionalities and handle new cases through the use of modern technologies. Without a doubt these focus groups were extremely important to define the project scope and describe the key features and requirements of both the application and the toolkit.

Figure 5.1 depicts a typical focus group session.

Output
All together, these different ways of gathering information provided the team with lots of feedback which could prove to be useful. It was the team's job to filter-out the remarks which did not fit in the scope of this research work. The following is a semi-curated list with some comments and observations that were gathered by the research team at different stages throughout the whole elicitation process.

Interaction observations

- Previously existing application (ViewFinder) is not used often because it crashes a lot; reliability is of the essence.
- ViewFinder tries to address the problem where there is only one technician working with the system.
- Some people do not use ViewFinder very often because it takes some time to setup, mainly because of networking.
- Changing camera settings is not obvious at first because these options are hidden, users had a hard time figuring out that they needed to press a specific icon to bring-up the settings drawer. There was even a specific case in which a user was so unaware of this that he used the desktop application to change the settings for the first camera in the test.
- Operating the application whilst standing on a ladder and physically manipulating a camera is no easy task; users typically placed a tablet on the ladder's last step and looked at it while operating a camera with both hands.
- On a user's words, *it's OK because it doesn't have a lot of buttons and clutter, but more information or hints about some features would be great.*

Technical observations

- Current ViewFinder does not offer a way to exit demo mode, users need to restart application.
- A camera's ID can be changed from the desktop QTM application, but not from ViewFinder.
- Interface feels outdated by today's standards.
- It is not possible to directly set a specific value for a camera setting, meticulously operating a slider becomes hard when values range from 0 to 30 thousand.
- When ViewFinder crashes, it will sometimes crash the running QTM instance as well.
- Due to the application's time of release, it is not compatible with some of the newest camera features.

It is important to notice that even before the *Ideation Phase* officially began, lots of interesting thoughts and schemes were already being discussed. Each interview and focus group session yielded plenty of ideas and discussions; whether they were realistic or not in terms of time and scope was still not for debate and so they always seemed relevant. Some discussions regarding voice-activated commands and audio feedback were held a couple of times and because of this recurrence we decided to incorporate related inquiries on some interviews, the response was mixed as some technicians were very interested in this, others thought it could be annoying and others just did not care. Another interesting idea that was brought to the table several times was to make use of the built-in smartphone rumble functionality to provide fast feedback concerning actions such as system calibration and measurement capture; this was inspired by another thesis work (Franjcic et al.[49]) in which visual and haptic feedback was used to help with the system calibration process. Other concepts that never made it to the *Ideation Phase* included controlling the system through smartphone motion-gestures and a wireless dongle with communication scripts to streamline application-host connection.

All this information was then used to produce use cases and scenarios on which the research team would ideate and design a tailored solution.

5.2.2 Use Cases

After numerous interviews and focus group sessions, we went back to the documented observations and unveiled very interesting workflow patterns and behaviours, we wrote these down along with some descriptions and documented them as *contextual tasks* [52], for they depict special tasks that occur under certain circumstances. These contextual tasks were analyzed and transformed in order to convey more generic cases, therefrom *use cases* were generated. These recognized use cases would then be used to analyze and further understand how work flows in a more generic environment. The following list contains the identified tasks, while elaborated use cases can be found under appendix G:

- **UC1** Connect to QTM host
- **UC2** Select camera and view real-time data
- **UC3** Change stream mode
- **UC4** Change camera settings
- **UC5** Browse Cameras

Use cases are great for understanding what should be done in reality, mapping user profiles and setting development goals and milestones which help determine specific functional and non-functional requirements. The complete list of such requirements can be reviewed on appendix B.

5.2.3 Scenarios

Scenarios were created based on previous findings to create a more vivid idea of the problems that the proof of concept should aim to solve. They are primarily based

Table 5.2: Scenarios

Scenarios	Description
Ladder	The actor climbs up a ladder to move a camera into position. Before making adjustments to the camera position the actor has to put down the mobile device somewhere. During adjustments the actor will repeatedly look at the device and climb down to interact with it when needed.
Camera Rotation	Actor decides it would be best to physically rotate the camera for functional purposes. Actor then needs to rotate the camera view in the application.
Camera Identification	Actor looks at camera rig and application, needs to Identify camera order and correspondence.
Camera Focus	The actor needs to adjust the focus of the camera. To do that it is important to be able to view the intensity of the marker sphere inside the camera and update the settings accordingly until a desirable result is achieved.

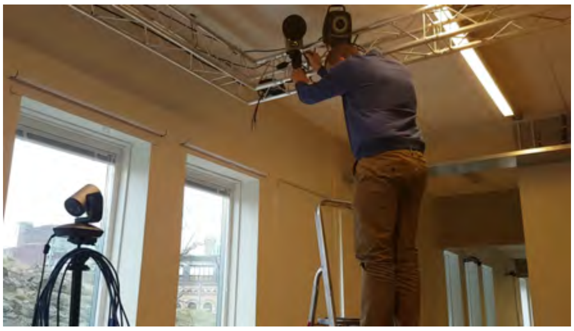


Figure 5.2: Image depicting one of the scenarios - the ladder scenario - identified during observations.

on the patterns that were recognized, although adapted in a more generalized way to include additional supportive tasks. The scenarios are described in table 5.2 and an example of one such scenario can be found in figure 5.2 with the intention of making the scenarios more concrete to the reader.

5.3 Ideation

After all functional and non-functional requirements were elicited and documented, the time had come to make use of all that information, officially giving way to the *Ideation Phase*. When it comes to doing idea generation, one of the most important things to be taken into account is to stay on-track with the objective. Therefore the very first step taken by the research team was to briefly discuss and agree upon two very important aspects; a design problem and a goal. Once formulated, these were written and displayed prominently in the work environment, reminding the participants to remain focused on the goal of the task at hand. When it came to managing the session and setting it up, the group did not have a facilitator (as discussed in section 4.1.2) and so, a session program was produced beforehand and a visible stopwatch was set to ensure that the everything went according to plan.

5.3.1 KJ-Technique Brainstorming

To kick-start the ideation phase, the *KJ technique* was chosen between several others because the research team was well acquainted with it, it had been used more than once throughout the master's programme and it never failed to deliver. The *KJ technique* is useful not only for the development of new ideas, but also to establish a common mindset amongst the participants, which is exactly what the research group wanted at this point. The session started by setting the environment (i.e. a whiteboard on which to place the sticky notes), procurement of tools (i.e. sticky notes and markers) and setting a timer with a 40-minutes session goal. For practical purposes, each member used a different marker color and every sticky note was labeled with a number. It is important to notice that every idea was accepted at this point no matter how far-fetched, as long as it aimed to fulfill the ideation goal. The timer started and the participants began writing down ideas on the sticky notes and silently placing them on the whiteboard without any particular order. After 40 minutes, the idea generation stopped and the silent grouping of these according to pattern identification commenced; the timer was restarted and it was set to another 40 minutes. After this phase was completed, we broke silence and began going through the clustered sticky notes, discussing our reasoning behind its classification and agreeing upon a name for a category. Twenty-five idea groups were created and with further discussion we re-organized the sticky notes accordingly; figure 5.3 depicts the result.

5.3.2 Further Ideation

After the traditional *KJ technique*, the team proceeded to iterate through every single generated idea and tried to further extend and build upon it. In some cases new ideas ware found during this process and these were then added to a fitting group. The results of this exercise are depicted in figure 5.4.

5.3.3 Ordinal Categorization

After a successful brainstorming session, an overwhelming number of ideas will have been brought to the table, and most of them will not be worthwhile to investigate further. Kelley reinforces this statement, "... brainstormers may generate a hundred

Figure 5.3: KJ-technique brainstorming result

Figure 5.4: Building on previously-generated ideas.

Figure 5.5: *On the table's center,* ideas that were voted as valuable. *At the right-side of the table,* pile of ideas that could be re-visited later on.

or more ideas, ten of which may be solid leads.". From the previous step a total of around 80 ideas were generated. The next step was to employ a strategy to quickly filter out the least interesting ideas and by that work more with ideas with higher potential. It was important for the researchers that the selection was a group effort and thus a democratic approach was employed. The technique can be seen as a combination of two tools for selecting ideas presented by *CreatingMinds*, namely *Voting* and *Negative Selection.* Each group member got to categorize all ideas based on a colloquial ordinal scale described in the list below. Based on the results from this strategy some ideas were removed, some were to be further evaluated and some were selected as solid leads, this selection will be further detailed in the next section.

- *I love this idea* - Highest
- *This could be cool* - High
- *I don't know about this* - Medium
- *No way José* - Low

5.3.4 Ranked Categorization

When each idea had been categorized, according to the ordinal scale presented in section 5.3.3, they were sorted based on their average result. This was doable since the research team consisted of only two people and thus the combinatorial space was fairly low. We classified ideas according to their combined result and from that 8 relevant piles of ideas were discovered. The piles where opinions lined up were categorized accordingly, the ideas that received a low score were discarded, and those where opinions differed were further discussed and evaluated.

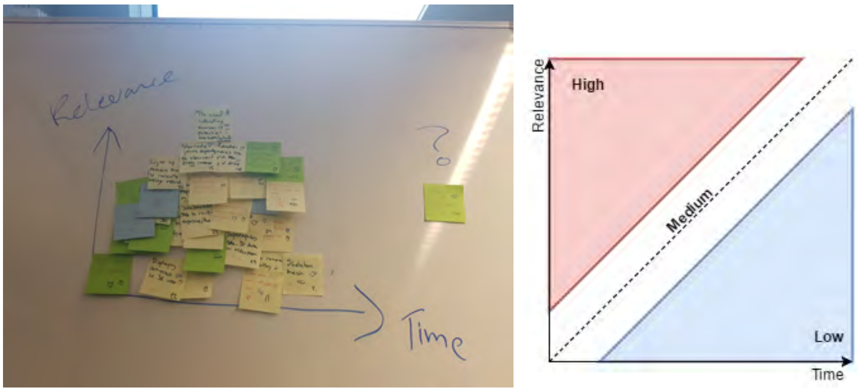


Figure 5.6: *Left-side*, ideas distributed in a graph according to believed relevance and time-to-realise idea. *Right-side*, the model used to determine which ideas to prioritize

5.3.5 Further Discussion

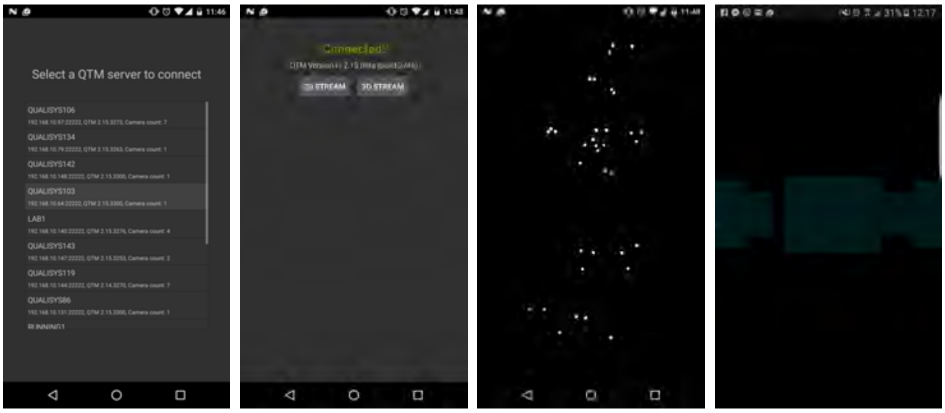
Due to difference in opinions regarding some ideas there was a need for further discussion. Firstly, to ensure that an idea was of significance the researchers were given the opportunity to speak up about the ideas and convey why they believed or did not believe them to be good ideas. The researchers were then given 20 points to divide between the ideas that they wanted to keep. Depending on the received score the ideas were grouped with the other prioritized ideas, the higher the score the higher the priority, the ideas without a score were put in an "idea stack" that was kept in case ideas needed to be revisited in the future. Figure 5.5 shows the ideas that were kept after this step.

5.3.6 Modeling Relevance

In the end of the *Ideation Phase* there was a couple of ideas that were deemed relevant and valuable for the research. However, one last prioritization technique was needed to determine what ideas to explore first. For this a graph model was used to plot the ideas according to their relevance for thesis-work research and how time consuming an idea might be to realize, the graph is depicted in 5.6. A linearly increasing slope from the origin was traced, those below the line were branded Low-, those near the line Medium- and the ones above the line High-priority ideas.

5.4 Early Prototyping

This section covers the team’s prototyping process and outlines the different ways in which this phase was conducted.



Select a QTM server to connect

QUALISYS106
192.168.10.97:22222, QTM 2.15.3273, Camera count: 7

QUALISYS134
192.168.10.79:22222, QTM 2.15.3263, Camera count: 1

QUALISYS142
192.168.10.148:22222, QTM 2.15.3300, Camera count: 1

QUALISYS103
192.168.10.64:22222, QTM 2.15.3300, Camera count: 1

LAB1
192.168.10.140:22222, QTM 2.15.3276, Camera count: 4

QUALISYS143
192.168.10.147:22222, QTM 2.15.3253, Camera count: 2

QUALISYS119
192.168.10.144:22222, QTM 2.14.3270, Camera count: 7

QUALISYS86
192.168.10.131:22222, QTM 2.15.3300, Camera count: 1

RUNNING1

Connected!!

QTM Version [illegible] 2.15 [illegible]

Figure 5.7: Overview of critical prototype developed with Xamarin, running on Android.

5.4.1 Critical Prototyping

Due to the cross-platform framework evaluation that had to be performed during the early stages of this research work, a small, barely functional quick solution had to be developed. This was done in order to test and evaluate a specific set of features and requirements. For example, software-architecture practicalities, front-end development, network connectivity, data streaming, platform deployment and performance. This prototype was developed over the course of two weeks and was successfully deployed and tested on both iOS and Android devices (see figure 5.7). It offered the following functionality:

- Displaying and refreshing server list.
- Connecting to a server.
- Selecting between 2D and 3D data visualization.
- 3D scene with touch gesture-based navigation.
- Carousel display of a set of cameras and their respective 2D data.

This greatly helped the team to estimate feature-development times, familiarize with the toolset's workflow, identify possible slowdowns, plan for architectural software modality and test the builds' performance. This exercise was also very important because it helped the team to finally settle down for a cross-platform development framework, *Xamarin*. After finalizing and evaluating the results, the team moved from this software engineering approach to a more interaction design-oriented one by dissecting the previously identified requirements and features and designing UI and navigational elements to help make sense of it all.

5.4.2 Low-fidelity

When the critical parts had been prototyped it was time to take a step back and start prototyping on possible design solutions for the toolkit. This was approached

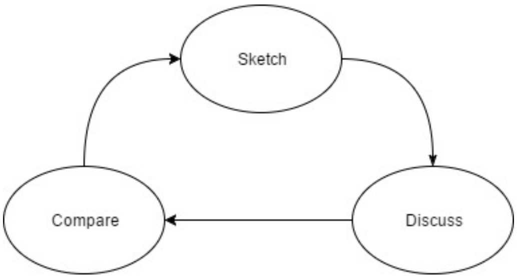


Figure 5.8: The feedback loop that was engaged during low-fidelity prototyping.

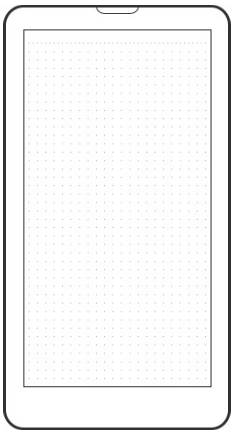


Figure 5.9: The template used when making low-fidelity prototypes of the user interface

in a way similar to the one during the *Ideation Phase*. Starting with a diverging phase to find possible design solutions to the prioritized ideas that we already had, ending with a converging phase to establish the ideas that made sense and needed further exploration. Early on the aim was to engage in a creative feedback loop, as visualized in 5.8. This enabled the group to work with ideas efficiently and iterate when necessary. The team members would start by sketching design solutions to some of the ideas without any input from the rest of the group, when an idea had been envisioned to a certain degree it was discussed in the group and eventual findings were noted and compared with other findings. The idea would then either be dropped if it felt irrelevant, iterated upon if further investigation was needed or kept as an potentially valuable design solution.

Later on, the sketches were drawn inside a generic smartphone template that can be seen in 5.9. This was done to get a feel for the proportion and the used space. The aim was to get as much relevant information on the screen as possible without making the interface feel cluttered. Moreover it created a narrative that more clearly could convey the intention of the user interface elements.

The prototypes started to converge with only slight variations so redoing a full page for each prototype seemed redundant. Therefore the group started approaching each UI-element individually which resulted in the creation of design modules that could be moved around. This provided flexibility and reuseability to the design process,

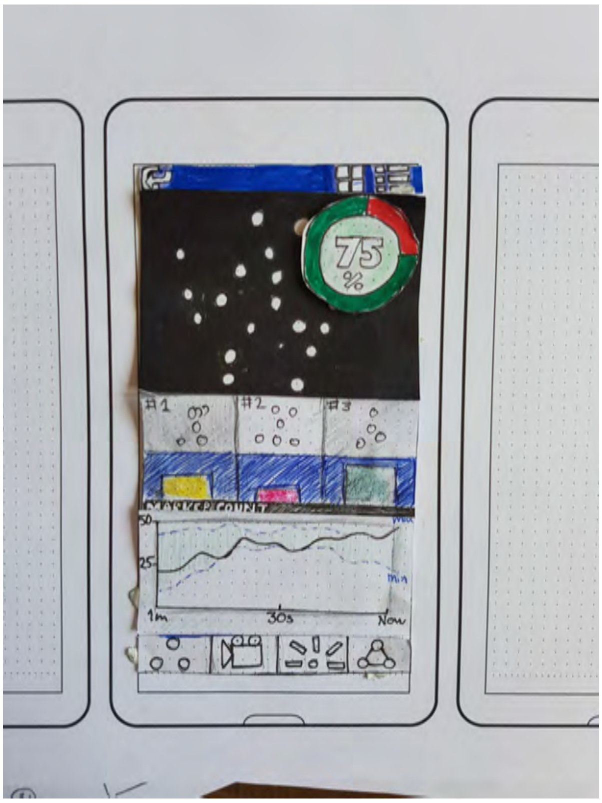


Figure 5.10: Low-fidelity modules assembled together to prototype a potential screen inside the mobile tool.

something that proved useful later on. Additionally it emphasized the functional aspects of each element which was valuable during comparison of different design solutions. The modules were defined by an evaluation of the wireframe sketches earlier where the subjectively good parts were picked by the team. The design solutions were then refined and drawn as modules instead. Finally similar modules were compared against each other to determine their pros and cons. Figure 5.10 depicts one of these UI module evaluation sessions.

5.4.3 High-fidelity Wireframe

After crafting and reviewing several paper prototypes, a few interface distributions were identified and the next step would be to digitally prototype them by means of high-fidelity wireframe software. Two different software suites were used, one by each member of the team.

- *Axure RP* is used to create prototypes without any coding. It enables designers to create dynamic content with conditional flows, furthermore it supports diagramming of user flow [5].
- *Mockplus* is an easy to use tool for rapid prototyping that support user interaction with dynamic content [21].

The goal with the creation of digital wireframes was to get a concrete idea of how the *ViewFinder* application could be redesigned to fulfill the imposed requirements. The prototypes helped in establishing good user flow by providing interactive components and navigational logic. A couple of tasks were created to validate how well the prototypes supported the use cases, namely:

- Go to camera X and set the exposure to X in intensity mode.
- Go to camera X an set the flash time to X in video mode.
- Go to camera X and then go directly to camera Y.

The prototypes were evaluated and redesigned until all tasks were supported. This iterative process was carried along with feedback from the core team at *Qualisys*. Furthermore, they revealed the importance of a context aware navigational flow and an hierarchical model was created to support this, see 5.11. During discussion with domain experts the model was redesigned to support one extra layer of navigation to support a dashboard for multiple tools that may aid when interacting with a motion capture system. The rationale behind this was to have all tools available for easy access and relieve the users of having to install several applications.

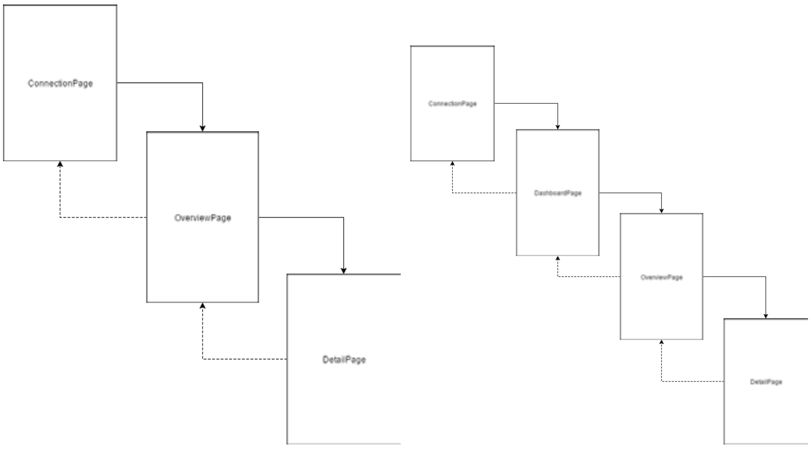


Figure 5.11: Model of the navigational flow. Left: Initial model, Right: Redesigned model after discussion with domain experts

Apart from the findings mentioned above the prototypes also helped in confirming that the design met the experts expectations, both in form and functionality. It also worked as a tool to rapidly prototype the layout of the pages and the color to some degree. The final high-fidelity prototypes can be seen in 5.12.

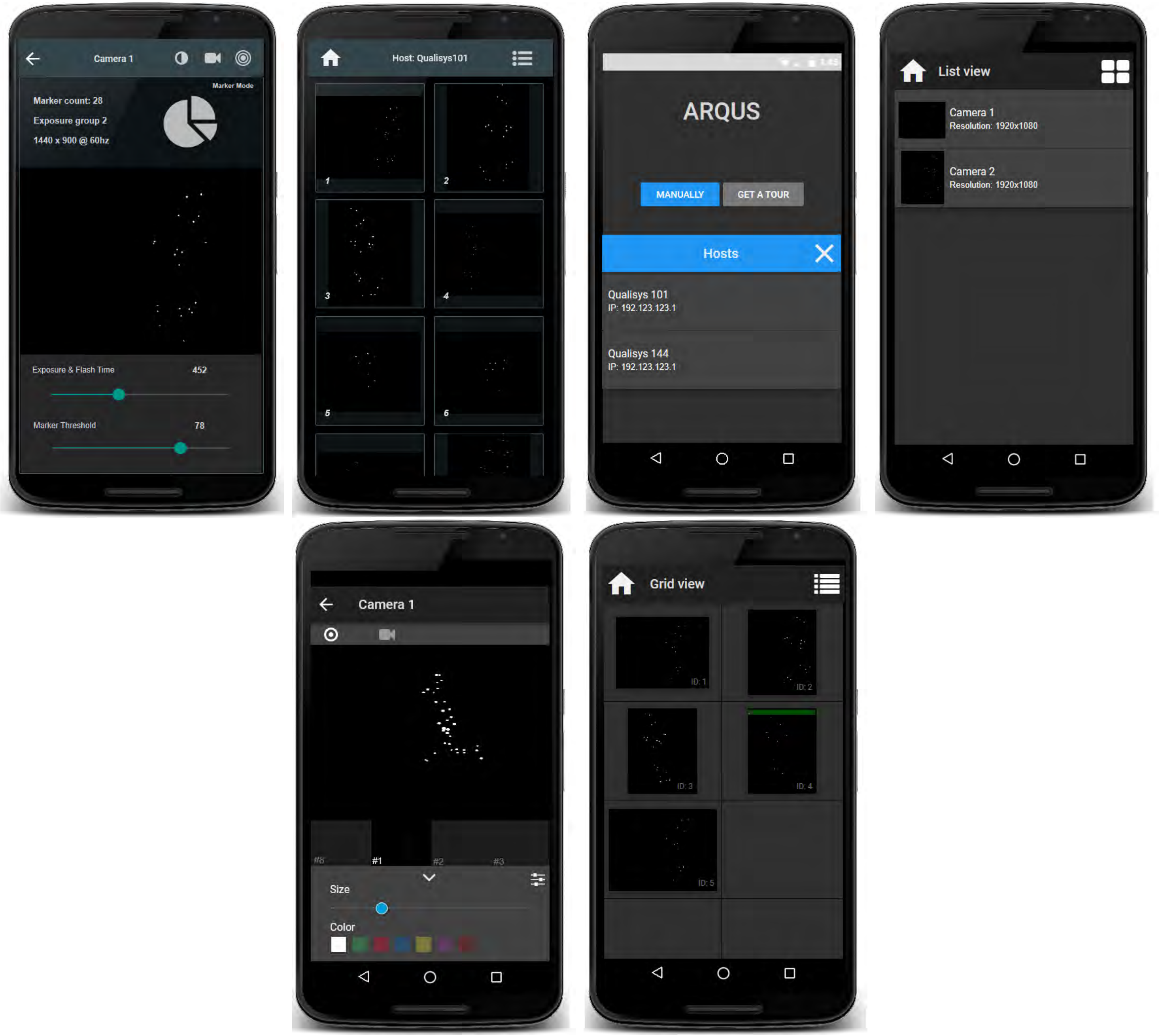


Figure 5.12: A few screenshots of the prototypes created with Axure.

6

Results

This chapter goes through some technical procedures that were taken to implement a more complete prototype based on the previous iterations. Underlying technologies, patterns and architectural decisions that were made during development are presented here, as well as the different application page layouts. These results are then evaluated in chapter 7.

6.1 Development Environment

Based on the foundation gained from previous stages the team started the development of *ArqusFinder*. The motive during this stage of development was not to create a production-ready application but rather explore how such an application might be constructed. The primary objective was to develop the application with a single codebase and to be able to deploy to different platforms, of course without leaving aside core functionality, performance and usability. The chosen main target platforms were *iOS* and *Android*. This decision was based on the current mobile-platforms market share [53]. After all the necessary research and technical decisions (i.e. middle-ware, development tools and platforms) were done, the team proceeded to set up the development environment; working with *Xamarin* meant that *Visual Studio* could be used to organize, build and deploy the project. On top of this, version control was hosted on the company's TFS. The team then proceeded to incorporate *Qualisys'* Real-time Software Development Kit (RT SDK) and to run a couple of test builds. The research team tried to integrate a Universal Windows Platform (UWP) build into the testing but compilation was unsuccessful; this was already expected as some networking components (.NET sockets) used in the RT SDK are not available on this platform. Some attempts to solve this were made by incorporating a different cross-platform network library but due to its early stages and the lack of documentation, these attempts were dropped as they were consuming a lot of time. A meeting was held and it was agreed that the team would focus on developing a fully-functional build for both *iOS* and *Android*.

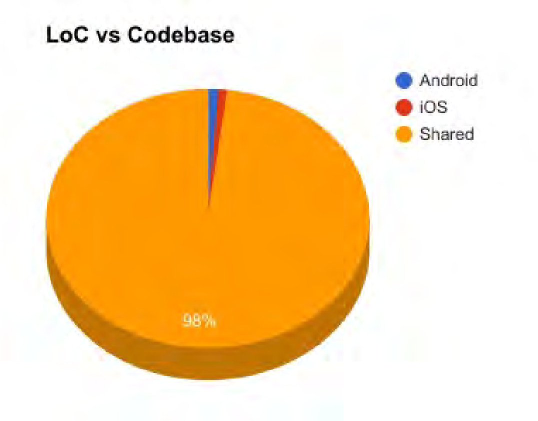


Figure 6.1: Pie chart demonstrating platform-specific- versus shared lines of code.

6.1.1 A Shared Codebase

An important aspect of the research was to determine how much code that could be shared without having unreasonable negative impact on performance nor reliability. This was evaluated by continually building for respective platforms, thus ensuring that they worked properly.

A deliberate decision to use *Xamarin.Forms* was made even though the user interface would be highly specific with a lot of custom views. It can be argued that this might be a sub-optimal choice since the need for native specific user interfaces is very high. However, *Xamarin.Forms* provides ways to implement platform specific user interfaces through *effects* and *custom-renderers* and even though this is added complexity to some degree it was deemed a worthwhile risk to more thoroughly investigate how much code that could be shared.

This approach enabled us to share both code concerning the logic as well as the interface. It posed a couple of interesting challenges such as: native- versus custom look & feel, logging and error handling, form size and platform specific features.

When the application's proof of concept was done, the research team proceeded to quantitatively evaluate the shared codebase, analyzing the percentage of shared- over native Lines of Code (LoC). It is important to notice that *Xamarin* only provides an interface to a platform's specific code. When a solution is built, a special compilation stage will generate platform-specific code which will be then compiled into a device. This code comparison only accounts LoC that were written by the team. The results of this comparison can be appreciated in both figure 6.1 and table 6.1.

Due to the project's nature, a 3D graphics library needed to be used in order to accommodate camera views and marker positions in virtual canvases. At the same time, these three-dimensional elements were meant to be combined with a 2D interface, so having something that could easily co-exist with *Xamarin.Forms* was a huge plus.

Table 6.1: Lines of code written for each platform

Codebase	LoC
Android	51
iOS	52
Shared	4582

6.1.2 Graphics API

Xamarin offers several library options for programming 2D and 3D graphics applications, mostly oriented towards game development. As of now, there are currently two powerful and officially supported 3D libraries:

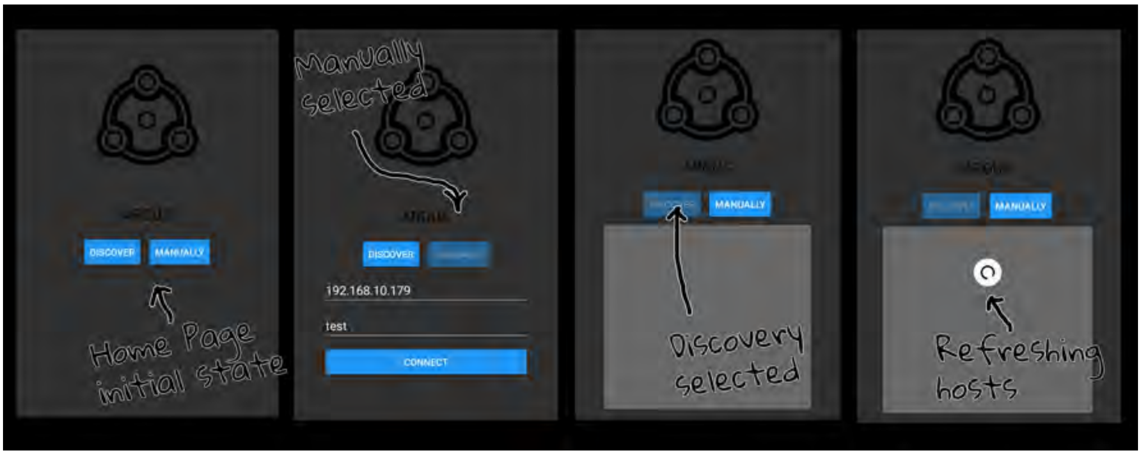
- **MonoGame** is an open-source implementation of Microsoft's XNA framework that can be used to develop games for iOS, Android, Mac OS X, Linux, Windows, Windows RT, and Windows Phone [37].
- **UrhoSharp** is a .NET binding of the Urho3D game engine which is also a cross-platform solution for Android, iOS, Windows and Mac; it can render to both OpenGL and Direct3D [32].

It was an obvious choice for the team to quickly settle down for *UrhoSharp* since it offers a specific version of the library that can be easily used together with *Xamarin.Forms* called *UrhoForms*. Needless to say, only the modules related to computer graphics were used. Furthermore it had support for *Microsoft's HoloLens* which is something that could be interesting to investigate further in the future, this will be discussed in greater detail in chapter 9.3.

With the *UrhoSharp* library selected the team proceeded to create a rapid prototype to try different features and learn how they could be coupled with data from the RT SDK. One member of the team worked on displaying 2D marker information from several cameras on various canvas objects in a scene, then a carousel-type of navigation was implemented with swipe touch-screen gestures. In the meantime, the other member worked on displaying markers in a three-dimensional scene along with different gestures to perform navigation (i.e. rotation, translation and zooming). This gave the research team important insights into some of the most useful components of the engine, as well as a general sense of the kind of structures, classes and data-types that were to be implemented. All this information was vital to estimate development times and evaluate viability of other potential libraries.

6.2 ArqusFinder

This section will go through each major feature of the ArqusFinder application, both technical and layout design approaches are presented. As previously stated, chapter 7 will evaluate these through usability tests. A video presenting an overview of the



192.168.10.179

test

Figure 6.2: Annotated screens of the home page.

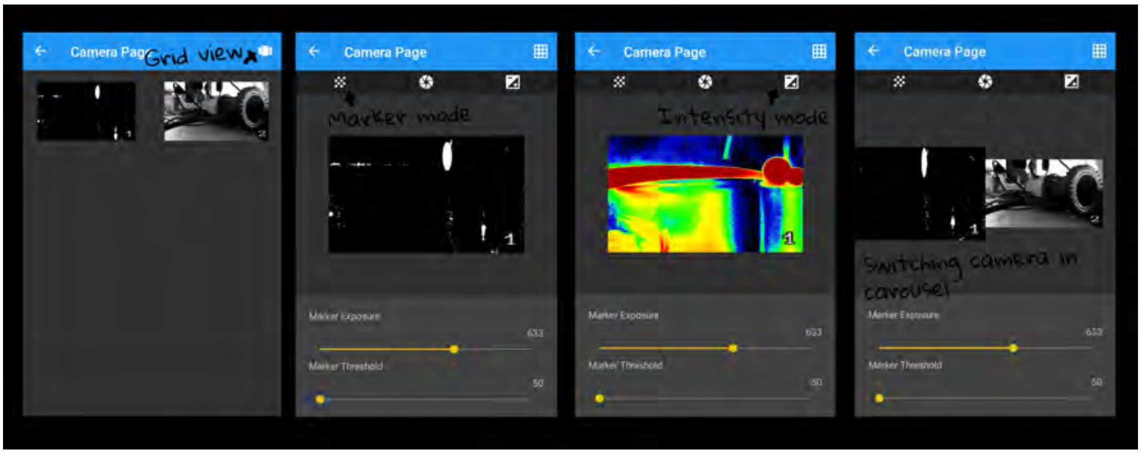
application is available for the interested reader[46].

6.2.1 Connection Page

The *Connection Page* was naturally placed as the home page since connecting to a QTM host is a necessity for the application to work. This page had to contain at least two features, namely: discovery of nearby QTM hosts and connecting manually to a host. Additionally a button to start a tutorial of the application was added. The final prototype contained a welcome screen that would show nothing except the possible options and the logo of the application. The rationale behind the screen was to let the user feel comfortable with the application before being prompted to perform tasks. From this screen, the user is able to decide to either connect manually or do a discovery of nearby QTM hosts. An optional quick tutorial was planned to be shown here, however the idea was not elaborated since it was deemed not valuable enough for a prototype, but it is an important feature that could be incorporated in future development of the application.

When selected, the discovery view provides a list of nearby QTM hosts that the user may select. During refresh of nearby hosts, a loading spinner is shown to the user to indicate that the application is still responsive and doing work in the background. In contrast, when manual connection is selected, input fields are given to the user instead. These fields are to be filled with the IP-address as well as a password -if applicable- to establish a connection with a running QTM host.

In the case a password is needed to be able to fully control a QTM host, a field is provided in which a user can enter it using a native alpha-numeric keyboard. This field can be ignored to connect to a QTM host without the ability to control the settings. A screenshot of this page can be seen in figure 6.2.



Marker Exposure 633
Marker Threshold 50

Marker Exposure 633
Marker Threshold 50

Marker Exposure 633
Marker Threshold 50

Figure 6.3: Annotated screens of the camera page.

6.2.2 Camera Page

The *Camera Page* offers a detailed view of each individual camera in the system as well as ways to switch the current streaming mode for that specific camera and a couple of sliders to change its settings (settings drawer). The controls to switch modes were added to a bar (mode toolbar) which was located at the bottom of the screen (it was later moved above), selecting a new mode changes the stream in the application code and updates the camera screen in the *Urho* application accordingly. The most distinctive feature of this view is the carousel-type navigation which is performed by the *UrhoSurface* component in the center of the page. This 3D scene takes care of rendering every camera stream to different canvases and of navigating through these, as well as other gesture-based interactions which translate into zooming and panning. Figure 6.3 provides an annotated screenshot of this page.

6.2.3 Camera Carousel

Another design idea from the *Ideation Phase* was to use a carousel to display the camera screens in a circular fashion. This was implemented using a mathematical model of a circle with a set of points along the circumference to represent potential positions of the camera screens. The currently selected camera is at the first point of the circle and the neighboring camera screens on the points beside it, an example of this can be seen in 6.4 where the neighbouring screens in this scenario would be the two at position 2 and 4. Given an ID the carousel returns the coordinates for placement of the camera screen. During a swiping gesture an offset is added to the positions. On a touch release the closest camera screen to the original position of the first point of the circle will be determined and selected as the current camera updating the circle model accordingly. The circle lies in the x- and y-axis of the application and thus the camera screens comes in from the left and the right of the device screen, this is common behavior for a carousel and the team saw no reason

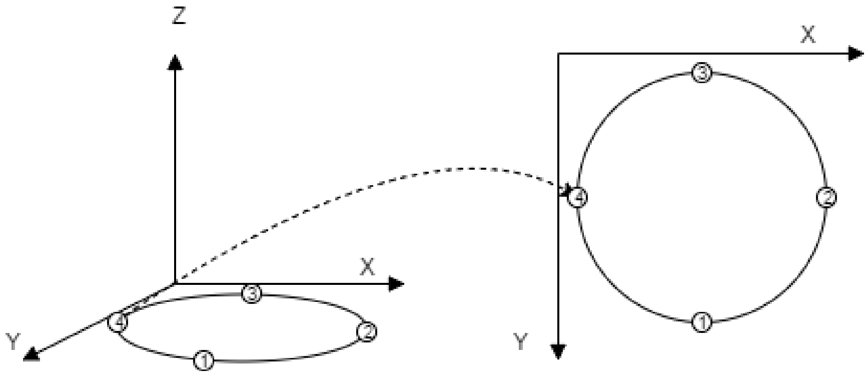


Figure 6.4: The circle with numbers represents a potential position for a camera screen where the circle with number one represents the currently selected camera. Left: Circle in a 3D-coordinate system, Right: Circle in a 2D-coordinate system

to break this pattern.

6.2.4 Camera Grid-Overview

Based on the results from the *Ideation-* and the *Design-Phase*, displaying a grid with an overview of every camera was a key aspect of the app navigation and general flow. This grid view primarily shows all the camera views in the similar way to how they are displayed inside the QTM application. Ordering inside the application is closely related to how they are ordered inside QTM in relation to their respective camera ID's where the first camera in a set starts at the top-left-corner and the last camera in the bottom-right-corner, a visual representation of this can be seen in 6.5. The *Grid-Overview's* primary functionality is to provide overview an of all the cameras so the one of greatest concern can be selected quickly by tapping that screen. The view is also scroll-able so that a larger set of cameras can be visualized without the camera views being to small due to resizing to fit them all in the screen.

The cells in the grid view contains canvases portraying respective camera information, this being markers, intensity feed or video feed. Because of the way *Xamarin.Forms'* components are built, it is technically challenging and naïve to try and display marker information with said components, that is why the team opted for a more sophisticated approach. Crafting a 3D Urho scene in which to display such markers and rendering it to a render texture would provide *Xamarin.Forms* with an Image object that could then be displayed as a cell. This technique is fast because only references to streams of memory are used, it is also easily maintainable and neatly organized because it does not use a lot of code, as it makes use of preexisting *Xamarin* classes. Sadly, this could not be successfully implemented due to memory management constraints and poor Xamarin documentation regarding image streaming. After this, the research team decided to create the *Grid View* within an Urho scene and render this through an *UrhoSurface*, an *UrhoForms* viewport that can be managed within **Xamarin.Forms** and Extensible Application Markup

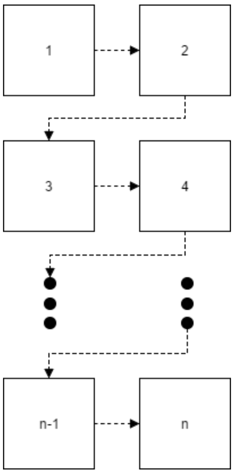


Figure 6.5: Order of the cells in the grid, the top-left-corner is the first screen and the bottom-right corner is the last screen.

Language (XAML). This gave the team more freedom to customize certain actions like animations and gesture behaviours. Of course nothing comes without a few disadvantages, the most important one being not using *Xamarin's* pre-existing framework and writing a significant amount of code that could otherwise be omitted. Unfortunately, this implementation proved to be very unstable as the navigation between the *Grid View Page* and the *Camera Detail Page* was prone to failure; the application would randomly crash due to an unhandled exception within the Urho library, a synchronization error regarding Simple DirectMedia Layer (SDL) threads was all we could get from a bloated stack trace which made it difficult to tackle. After several days of trying to address this and with the pressure of time, a strategic decision was made to drop the *Grid View* from the prototype and concentrate on delivering a fast and stable build with a *Camera Page* in which the whole camera set could be browsed by means of a carousel. After this, the research team found a way to accommodate both grid mode and carousel mode in a single *Urho* context, which was used for this prototype iteration. The grid- and carousel-mode will be referred to as *Grid Screen Layout* and *Carousel Screen Layout* from here on out.

6.2.5 Top bar

A top bar was added with the main purpose of providing interactive elements for navigation. Apart from a conventional back arrow an additional *Screen Layout* toggle button was added to toggle between the *Grid Screen Layout* and the *Carousel Screen Layout*. The two different states of the top bar when in the *Camera Page* can be seen in figure 6.6.

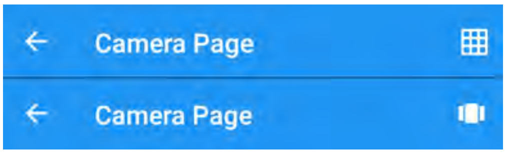


Figure 6.6: The top row is visible during a carousel screen layout and the one below when in grid screen layout.

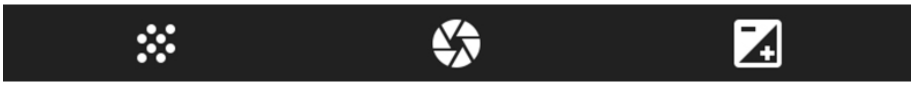


Figure 6.7: Application's mode toolbar

6.2.6 Mode Toolbar

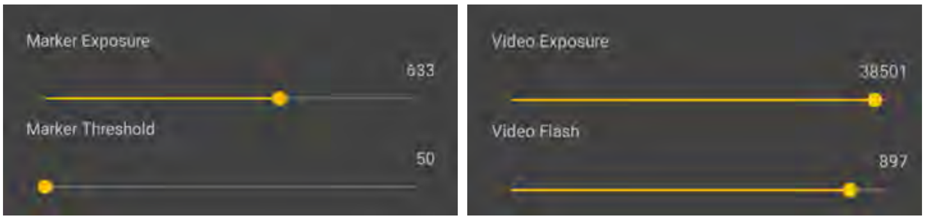
Changing the stream mode is handled by a set of buttons with callbacks which in turn send a command to the QTM host with the current camera ID and the new stream mode (markers, video or intensity mode). When either video or intensity mode are selected, a thread starts buffering the received camera images (compressed JPEG format), these images are then decoded and used to create texture objects that are then fed to the respective camera canvas. It should be noted that the icons used for such buttons were not made specifically for this purpose, they are generic images retrieved from *Google's material design icons* [18] and were hand-picked to fit the buttons' purpose as close as possible.

6.2.7 Settings Drawer

Besides switching modes, the application also needs to change the settings of the cameras depending on the currently selected mode. For this a *Settings Drawer* was created that changes layout depending on context, the relationship between available settings and context is described in figure 6.8. This drawer is composed by a couple of sliders which represent and set a specific camera setting. These sliders communicate in both directions; when they are locally set, they send a command package specifying the type of setting to be changed along with its new value. The application also listens to changes done at server side, an *EventListener* object is always listening to event packets of the *SettingsChanged* type and it runs on a separate thread.

6.2.8 Graphic Profile

The application got a dark theme applied to it to keep it consistent with the application that it stems from. It takes heavy inspiration from the design guidelines provided by *Google*. On another note, instead of defining colors and the general *look & feel* of graphical elements for each platform using the native API the group



Marker Exposure

633

Marker Threshold

50

Video Exposure

38501

Video Flash

897

Figure 6.8: Left is the settings drawer when in marker or intensity mode. Right is the settings drawer when in video mode.

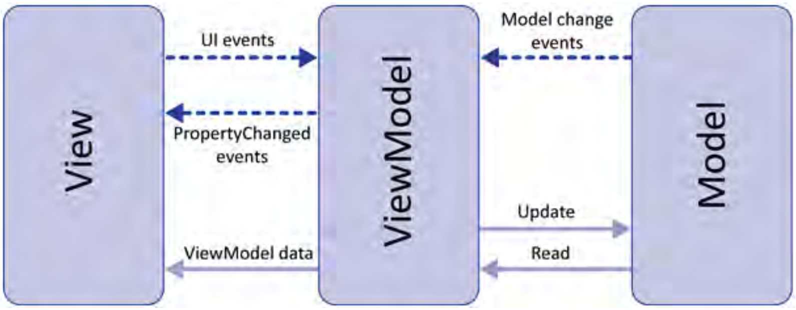


Figure 6.9: An overview of the MVVM architecture. (Courtesy of Microsoft

decided to move as much of this code to the shared code base. How these elements were defined can be seen in listing E.1 in appendix E.

6.2.9 Application Architecture

We decided to employ an Model View ViewModel (MVVM) pattern to attain more flexibility in the application as well as separate responsibility. The hypothesis was that this would support the removal of the View for another later on, which could prove useful when working with support for a platform that is not supported by *Xamarin.Forms*.

The reader is encouraged to read more about the MVVM pattern in the references since it is not really part of the result. However it will be briefly introduced to the reader in the list below and an overview of the architecture can be seen in 6.9.

- **View** only concerned with how it looks
- **ViewModel** represents interactions with the datamodel and reflects changes made
- **Model** is the datamodel of the application

After some initial experiments with the pattern the team recognized that the *Xamarin.Forms* support for an MVVM architecture was not fully adhering to the requirements of the ArqusFinder application. Thus a couple of MVVM frameworks were evaluated to determine if the team could avoid having to implement this behavior themselves. The *Prism framework* was picked as it seemed the most suitable

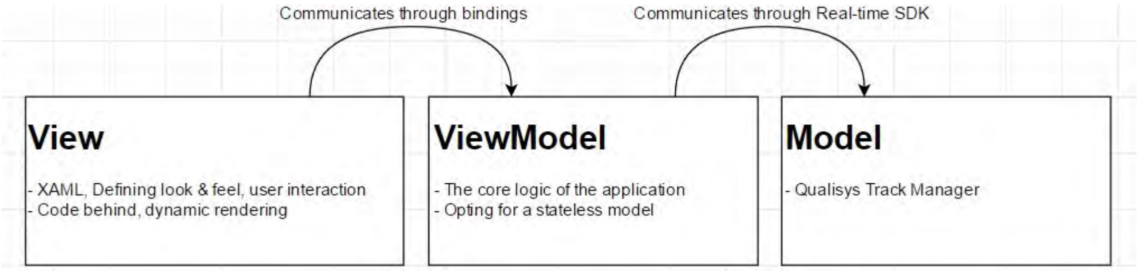

Communicates through bindings
Communicates through Real-time SDK
View
- XAML, Defining look & feel, user interaction
- Code behind, dynamic rendering
ViewModel
- The core logic of the application
- Opting for a stateless model
Model
- Qualisys Track Manager

Figure 6.10: An overview of the MVVM architecture.

for the purposes of the *ArqusFinder* application.

To make it more clear to the reader, XAML-files and the associated code behind will be briefly introduced before going any further with application development details. XAML is a language for visual representation that declares graphical components, their formatting and the layout of the components. A XAML file is usually accompanied with a code behind file that provides the functionality for the *View*[36]. To adhere to the pattern of MVVM the purpose of the code behind became mainly a way for *Xamarin* to compile the XAML file into native UI components (how this works is out of the scope for this thesis). With the use of bindings the *View* is updated when properties change in the *ViewModel*, this is in contrast to using a code behind approach where the actual values of the component would be mutated instead.

With the use of the *Prism framework* the team managed to quickly move most of the code from the code behind to the *ViewModel*. There were however some cases were this was not possible such as when an XAML component needed to react on an event during run-time. For example, an *UrhoSurface* had to initialize the application when the *View* appeared due to the fact that *XAML* has no concept of what *Urho* is and thus does not know how to start the application. This could have possibly been done in the *ViewModel* but would couple the *ViewModel* with the *View* thus pattern breaking the MVVM pattern. Thereby, it was decided that this would be done in the code behind as it was reasoned to be part of the *View*. The architecture of the proof of concept related to the MVVM pattern can be seen in figure 6.10.

Error handling was a big problem during development and the exceptions thrown in the native code became hard to debug. In some cases (just as stated in section 6.2.3) the application would crash due to threads that would not be destroyed properly and the only indication of this would be stack traces of references to missing objects.

Catching exceptions would improve this to some degree but overall debugging was quite time consuming and cumbersome. To address this the team decided to experiment with a service released by *Microsoft* called *Mobile Center*. It acts as a platform for automated-build, continuous integration, event tracking, analytics, tests on several real physical devices and crash reporting amongst other services for mobile applications to ensure high-quality mobile applications [20]. Moreover the event tracking needed to have fine granularity since it was going to be used during summative evaluation to analyze how the users interacted with the application. After some simple tests it was concluded that the Mobile Center event tracking was lacking that granularity and was only able to visualize daily data at its finest granularity. Due to that further research was made on platforms that would be more suitable for the event tracking that was needed for the summative evaluations. The team ended up using Kibana which is able to visualize data at granularity of milliseconds and has support for several different visualizations. Kibana depends on Elasticsearch that is used to store the data. The data gathered as well as the visualizations of that data will be presented in section 7.3.

7

Summative Evaluation

Summative evaluation was carried out based on the developed prototype to determine how well it worked with actual users. Two different approaches were made for this evaluation, these will be presented in the following sections along with the gathered data received through comments and questionnaires. Additionally, the user data that was gathered during these tests will be presented in the final section of this chapter.

These were short and fast tests that generally took from five to ten minutes to complete, they were aimed to provide feedback on very specific system interactions so that key features could be evaluated. This was very important for the team because it would not only yield quantitative data, but also feature-specific comments and in some cases requests for specific functionality. The team approached users with different technological backgrounds to get a diverse view of how well the application was received. A running instance of *ArqusFinder* was readied and a set of four tasks were given to the users. They were asked to familiarize themselves with the application so that they could subsequently perform the tasks and rate them by using the SEQ scale. The following is a list with descriptions of said tasks:

- **Task 1** Navigate to camera 2.
- **Task 2** Ensure that camera 4 is running in intensity mode.
- **Task 3** Switch every camera to marker mode, except camera 4.
- **Task 4** Lower the threshold of camera 1.

This task-set was created to support the use cases that were elicited during the early stages of this research work, described in section 5.2.2. The following table demonstrates how these relate to each other.

Table 7.1: The tasks that support each use case is presented in this table.

Use case	**Task**
UC1	Task 1, Task 2, Task 3, Task 4
UC2	Task 1, Task 2, Task 3, Task 4
UC3	Task 2, Task 3
UC4	Task 4
UC5	Task 1, Task 2, Task 3, Task 4

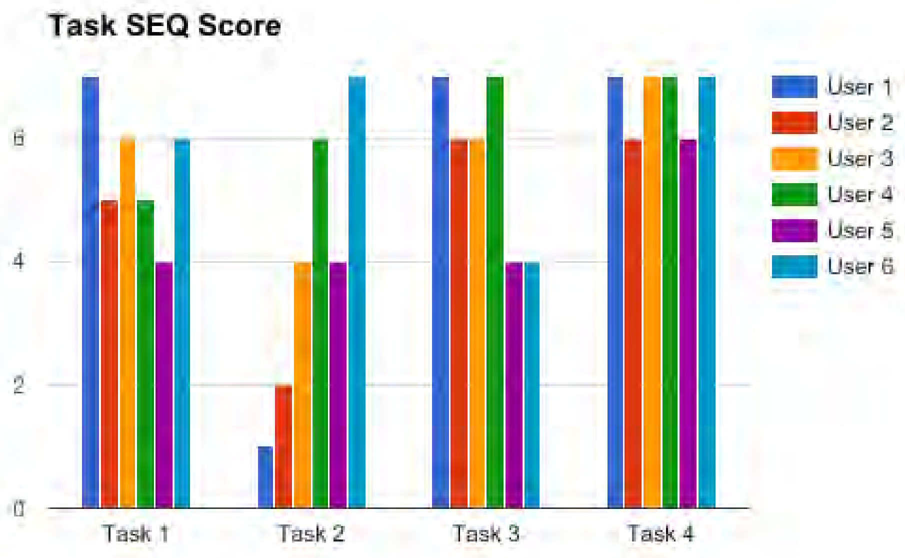


Figure 7.1: A bar chart visualizing the score for each task. Score is the y-axis where higher means easier to complete. The different tasks are placed on the x-axis. The colors indicate the different users, moreover the users appear in order where the leftmost is User 1 and rightmost User 6.

7.1 Single Ease Question Results

The results from the SEQ are presented in table 7.2 and figure 7.1. Performing these tests on users from different backgrounds naturally meant that ratings for one specific task would somewhat vary. It is apparent that some tasks were perceived as being easier to complete than others according to the users. Overall, task 4 was the least difficult to complete compared to the other tasks according to the answers. In contrast task 2 was rated the most difficult on average. There might be several reasons for this, something that is evaluated further in the next section. The two other tasks score almost the same on average and overall most of the tasks were perceived as being moderately to very easy.

Table 7.2: User score of each task. Column represents the different tasks and rows the different users.

	Task 1	**Task 2**	**Task 3**	**Task 4**
User 1	7	1	7	7
User 2	5	2	6	6
User 3	6	4	6	7
User 4	5	6	7	7
User 5	4	4	4	6
User 6	6	7	4	7

7.2 Observations

Besides the users filling the SEQ forms with task ratings, the research team carried a more qualitative approach at the same time, taking notes based on on-the-spot observations using a pre-defined table; this table was filled with the features (UI elements) that were most important for navigation and interaction. Observations on these features are written in the following sections. The pre-defined table can be seen in appendix D.

Login Page
The *Login Page* was considered as a feature in its own even though it provided two ways of connecting to a QTM host. This was intentional since what was interesting in this page was the synergy between the two modes and how to switch between them. During the tasks most people attempted to connect manually, some of those users noted that it would be more convenient if the application provided a numerical keyboard instead of a full keyboard layout when entering the IP address since the only symbols they really needed were either numerical or a dot. In the cases were they entered invalid information they would receive no response, which made the users uncertain of the current state of the application. The users that attempted to make a discovery of nearby QTM hosts had trouble figuring out how to perform a host refresh, the result of this was that most users ended up connecting manually in the end.

Carousel Layout
Based on the reactions of the users it seemed like the carousel was a nice-to-have feature and it was being used by most users without any explanation. However, there were two users that did not discover this feature until they were told that it actually existed. One user felt that it was a bit to sensitive which in turn made it hard to select the intended camera. Another user proposed that it would be nice to have a view with only one camera instead of the currently implemented views that were either carousel- or grid-view. A combination of panning and zooming would

make other cameras appear in the background which seemed like an odd behavior to those users that manage to get the application into this state. Furthermore, only half of the subjects attempted to use gestures.

Grid Layout

Most users wanted the ability to change stream mode of several cameras at the same time. Some wanted to change all the cameras streaming mode at once while others thought it would be nice to be able to select a set of cameras to update. During the tasks some users also tried to use the back button in the top bar to go back to the carousel view, this was not the buttons intended purpose and it would navigate the users back to the home screen instead. One user did not notice the grid layout throughout the whole test. Another user had concerns regarding the situation where even more cameras would be available and saw a potential scalability issues.

Mode toolbar

Almost every user made it clear that the icons used for this iteration's toolbar were not comprehensible at first glance, it generally took a few tries for them to understand how they were related to the different camera stream modes. Some users suggested that adding text to said buttons would improve interaction, especially if someone were to stop using the application for a month or two. One of the main problems that recurred throughout every session was the delay when changing to either video- or intensity mode; this prototype did not have any loading indication and it proved to be very misleading as some users even thought the application had crashed. They were also concerned with the lack of a current mode indicator; the research team thought it would be enough to visualize the specific stream mode, but these tests proved it wrong. One of the most significant comments from one of the engineers was that this toolbar was not consistent with the order in which the items were presented in the QTM software, a seamless, overlooked matter that can nevertheless break the user flow. This same person also pointed out a potential problem; the mode toolbar is placed just above the *Android* navigation bar which could, in some instances, cause navigation mishaps. The team later decided to move the toolbar to the top of the view.

Settings Drawer

All users had trouble interacting with the sliders in the settings drawer. It was not uncommon that users had to try multiple times before succeeding in modifying the settings according to their intentions. Most users seemed to think that the responsive area was to small and thus easily missed. Two users expressed interest in being able to hide the settings drawer to not obscure the view of the cameras during times when no changes to the settings had to be made. The users had no problem in understanding how to interact with the input components even though the interaction in itself was a problem. Finally, a user - with expert knowledge in the motion capture software that the application communicates with - explained that the threshold should be in percentage and not numbers as it were in the current iteration of the application.

Additional notes
One user questioned why the backdrop of the marker screen was green. The same user also asked "What is a Camera Page?" when looking at the top bar that was displaying the title of the current page. Two other users noted that the spheres would go outside the screen when close to the marker screen edge.

7.3 Evaluation of usage data

Quantitative usage data was gathered during the usability tests and relevant data is presented in this section. Data was gathered for each user interaction based on the idea that this would provide a deeper insight into users behavior. Visualizations of the data can be seen in figure 7.2. The visualizations give an indication that users mostly toggle to the grid view and thus uses that view to select a camera instead of toggling back to the carousel. The data also makes it clear that the most used screen mode during the tests was the video mode.

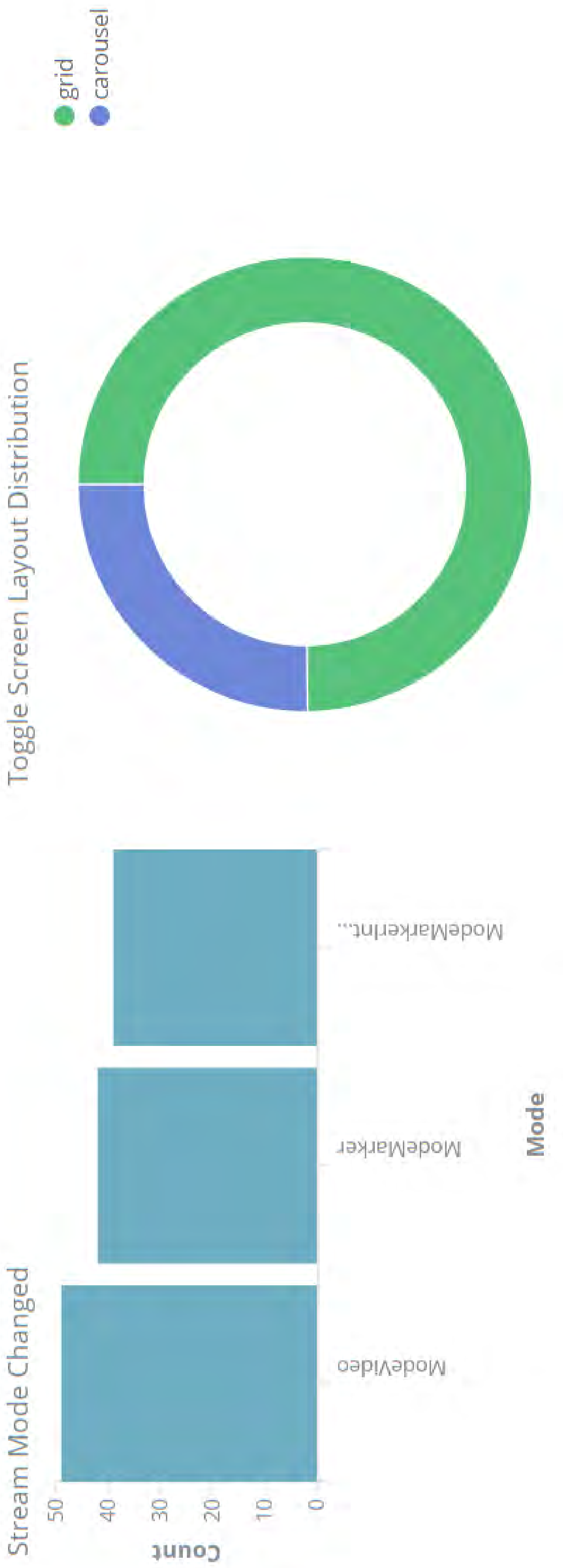


Figure 7.2: Usage data gathered during the user tests

8

Discussion

This chapter will discuss the approach taken during this research. Moreover, the evaluation of the end result will be analyzed and dissected to discover important insights regarding the design solutions with the goal of highlighting factors of importance when designing a tool to aid during the setup stage of motion capture systems.

8.1 Formative Evaluation

As described in chapter 1, our tool is intended to aid technicians when doing an initial setup of the motion capture system. These technicians need to assist different companies in all sorts of scenarios. This means that the setup environment is highly dynamic. For practical purposes, this project's formative evaluation tests were performed in a somewhat controlled environment, quite different from what a real-life scenario would look like as the camera rig, along with cables and interfaces were already setup. Furthermore, the technicians had a ladder available to their disposable which eased the interaction with the mobile device since they could conveniently place it on top of ladder. Doing evaluation out on the field may have procured different, interesting results.

Since the research team was not well acquainted with the hardware setup back when formative evaluations were performed, the only thing that could be done to counter this optimal setup was to slightly rotate some cameras relative to their respective bases. These tests paved the way for some interesting results, but the technicians seemed to run through the tasks fairly easily. The research team cannot help but wonder if the validity of the gathered observations were compromised due to familiarity with the environment.

Edge cases such as doing setup in the desert or even under water were not considered. This was mainly to the group not having knowledge of these. It might have been interesting to consider some of these edge cases since they call for mobility.

Finally it is worth mentioning that the formative evaluation could have been more elaborate. Only two experts in the field were evaluated a part from the evaluation

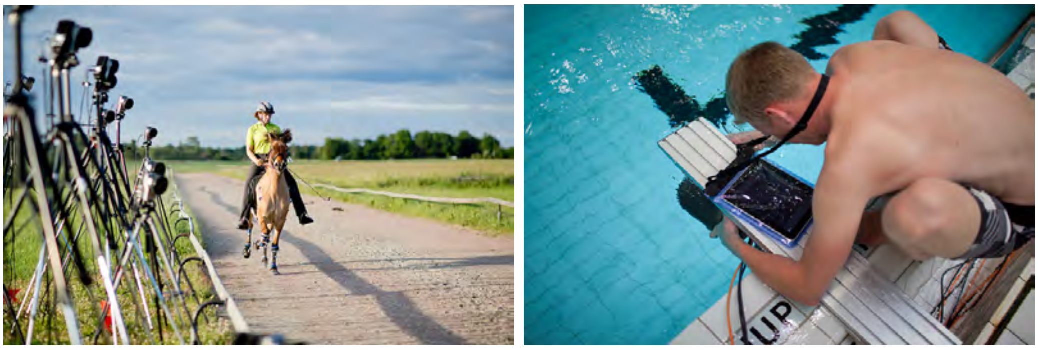


Figure 8.1: A few examples of common setups outside of a studio

done by the team itself. Doing a more substantial amount of tests with different users might have been valuable.

8.2 Cross-Platform Development

Early on, the team realized that doing cross-platform development comes with a lot of trade-offs. There are usually ways to target the native APIs but this in turn means less shared code. In the end it is all about finding a good intersection and do native specific coding when necessary. The goal for this thesis work was to share as much code as possible, thus a lot of decision were made based on this. As stated in the results the application managed to have a shared code base of about *98% of the total code base*. This shows that it is possible to share the majority of code between *iOS* and *Android* with today's frameworks and by that improve maintainability. It is important to note that the written native code (2% of the code base) in this project was updated no more than two times, plus the written methods were fairly small. This further confirms that it is possible to develop for several platforms without excessive need to maintain each platform individually.

That is not to say that cross-platform development came without any obstacles and another approach might have been more efficient to find an answer to the research question. Doing cross-platform development is essentially added complexity on top of the native API and by that diverts the goal of researching*what should be considered when designing tools that assist during the setup stage of an optical motion capture system*. However, cross-platform was something that was important for the stakeholders to increase maintainability and thus this was done to adhere to their needs.

The research team can only assume that working with constantly changing code within a cross-platform framework could get messy. No data was gathered with regards to this but it is definitely something worth considering.

At the end of the development of the *ArqusFinder* application a distinction was recognized between the core logic and the user interface logic, a speculative model

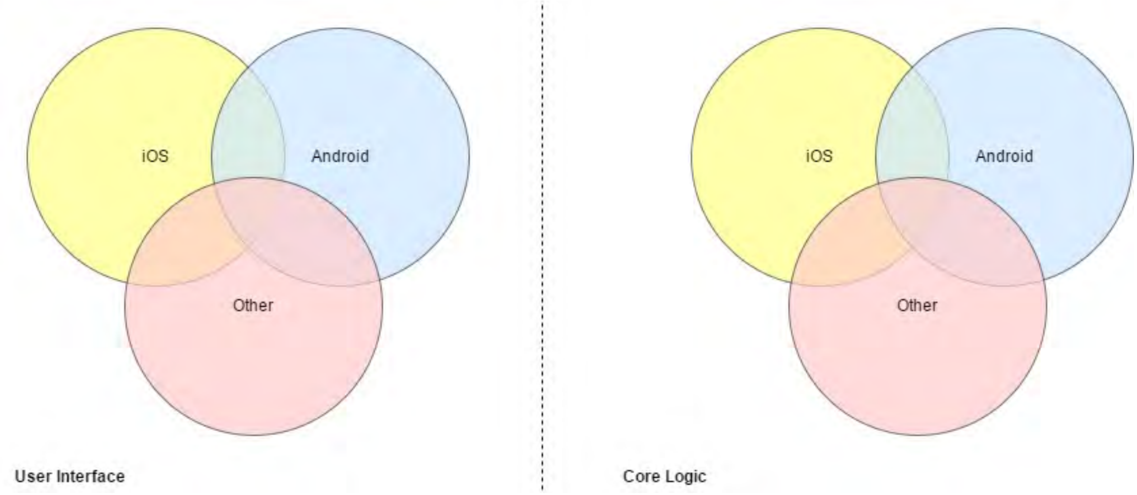


Figure 8.2: Intersection of platforms and the distinction made between core logic and user interface.

of this distinction can be seen in figure 8.2. We believe that it is important to be aware of such distinction, as it will aid to create a more flexible solution during development. The distinction can be closely related to doing front-end (User Interface) and back-end (Core Logic) development and thus will be referred to as such from here on.

Viewing them separately also enables developers to tackle the different challenges that come with doing front-end and back-end development. These challenges will be discussed in detail in the subsections 8.2.1 and 8.2.2.

8.2.1 A Shared Back-End

The back-end of the application was concerned with everything that did not have to do with how the components were to be presented. This included how the state was handled and kept consistent between the components as well as the core logic such as mathematical computations and data manipulation. During development it was decided the application would aim to reflect the state of QTM at all times and thus it would act as the model of the application. This worked fairly well and it was easy to share nearly all of the code since the majority of the computations would be done on the QTM side. However, this was not effortlessly implemented, and it will be discussed in the coming sections.

Real-time computation
One problematic area was getting raw image data from the image stream received from QTM. The images would be encoded in a Join Photographic Experts Group (JPEG) format to keep the data size small during network transfer. A consequence of this was that the image needed to be decoded when displayed in the 3D scene.

There are a couple of light-weight image processing libraries out there, but it is hard to find a library with support for both *Android* and *iOS*. In the end the team settled with a library that managed to decode the images in a lower resolution with reasonable speed. Doing this natively might have made the decoding process more efficient, but due to time constraints this was not further investigated.

3D Engine

Another area that introduced obstacles was the cross-platform 3D engine used for the application. During navigation the application would randomly crash due to some SDL-threads used by the engine not being destroyed properly. A lot of time were put into solving this and in the end the team managed to get it somewhat stable, though it still would crash on rare occasions. It does seem like the library is going through a lot of updates at the time of writing and there has been several improvements to the library since the beginning of this work giving the indication that this might not be a problem in the future [22].

Overall having a shared 3D environment has proven very valuable. It has eased development through reduction of repetitive code and assured a consistent experience between the platforms. Code becomes far more maintainable than having separate code for each platform. We believe that given time, this might be a good approach for doing cross-platform 3D development.

Networking

Throughout the whole development phase, several issues were encountered that were related to the way QTM's real-time protocol was implemented. One of the biggest challenges came when the team realized it was noticeably easy to cause a complete crash in the host's QTM instance. The research team did not consider this scenario during the early stages of development and therefore strange behaviour occurred whenever the host suddenly stopped sending network packets. Luckily, the amount of crashes were diminished significantly when one of the company's engineers located and fixed the problem on the host side. The cause for these severe software malfunctions lied on the research team not deliberating upon a network connection's inherited reliability constraints, it was always taken for granted that the host-end and the protocol worked perfectly. From the beginning it was known that the networking solution provided by the RT SDK was not compatible with UWP (as previously mentioned in section 6.1), this lead the research team to plan an architecture based on networking interfaces and *dependency injection* to be able to provide a different socket solution depending on the running platform. This approach was abandoned after a few days due to time constraints and more importantly, the fact that redesigning a pre-existing RT SDK is not valuable for this research work. The application is built in a way that requires four different clients connected to a QTM host, three handle in-bound data and one handles out-bound data.

- **MarkerStream** Streams marker-based positional information.

- **ImageStream** Streams camera image-based information.
- **EventListener** Listens to events sent by QTM.
- **SettingsService** Sends camera-specific settings to QTM.

Regretfully, this was not thoroughly analyzed beforehand, and for a while the application behaved oddly. The research team was baffled by this since debugging and identifying open-connections was not an easy task; a considerable amount of work had to be put into solving this by manually going through the code and ensuring that only the needed connections remained open. It was not until the team analyzed the host's network connections that this was proved to be working.

8.2.2 A Shared Front-End

The goal for the front-end was to create a similar experience amongst the platforms without breaking platform specific design conventions. Sharing this part of the code was suitable for common use cases which in turn enabled the team to quickly create a functional interface. One approach that worked well for the team was to create shared style resources for color and general look & feel of interactive components, see listing E.1 for implementation details. This was done in contrast to relying on the underlying style API's for each platform, which made styling confusing due to a multitude of different style definitions. An example of this can be seen in listing E.2.

The 3D engine used for the application proved to be really useful when opting for nearly identical visuals. There are two-sides to this though, on one hand it works really well for getting a similar experience across platforms, on the other hand it provides no way of taking advantage of predefined style conventions that the users are used to. However, being a 3D engine and not a framework for developing cross-platform mobile applications it is fair to say that it is out of the scope for what the engine should support. It was a challenge to make the 3D part of the application look consistent with the standard mobile interface components. Due to time constraints no investigation was done regarding how those styles could be shared between them both, thus repetitive code was used to declare a consistent color scheme.

It is also worth mentioning that working with a highly abstracted super-set of interface components -such as the one used in this proof of concept-, translates to the native platform-specific ones, and thus solid knowledge about these is still imperative. Some implementation-specific features that could normally seem trivial can be easily missed and cause major usability problems. As an example; if one desires to use a slider with natural numbers and steps, using an integer variable to store and set this value will work with *iOS* but not with an *Android* slider. It will still require the precision of a double variable if the slider range grows bigger. Different platforms have unique interface components and it is virtually impossible to try and use them through a shared front-end file. Unless one decides to create such interface component on another platform, there is no way around this. One could also interpret a unavailable component as something similar on a different Operating System (OS), but this will not work all the time. Bottom line is, describing interfaces for different platforms through a shared code base can increase design complexity depending on the project.

Animations can be created using the front end framework, it supports basic animations such as translation, position and color. For more advanced animation features the native API's will have to be targeted. In this work it was decided to not care about specific native animations that users might expect to have so that focus could be on sharing code instead. This might have lowered the user experience of the application although we believe this was a worthwhile trade off to investigate how a user experience can be shared between platforms. It would have been interesting to research how a shared color theme and a set of shared animations could be combined with native styles and animations given more time. The user experience is fundamentally different between platforms but we do believe that some of the front-end can be shared without for that matter impacting the native behavior that they are used to.

8.2.3 Architecture

As stated in chapter 6 an MVVM architecture was used. This was decided not only to create a separation of concern but also make the application adaptable to changes of the *View*. Using a shared front-end was partly an investigation to determine if it was a feasible approach, even though the application would require somewhat customized interactive components. With that in mind it was of importance to be able to adapt in case the shared front-end approach had to be abandoned. By following the MVVM pattern we believed that we would be able to go from a shared front-end to a native one without making any changes to the back-end. It never got to that point though so this has not been confirmed, but it proved to be useful when making iterative changes to the *View* since the back-end could be left untouched.

During discussion about application state, it was suggested that it could be modelled after the connected QTM host with the goal to stay in sync at all times. This essentially meant that there would be a mixture of state, one set of states from the QTM side and another set of states from the application side. This proved to be more complex than expected and created a high risk of out-of-sync states in certain parts of the application. A particular symptom of this problem was the discrepancies between the camera settings stored in the application and the ones managed by QTM. The research team was surprised to find out how easy it was for the state to go out of sync. Extra logic had to be written to ensure that this wouldn't happen. However, this approach did provide a consistent state between the mobile application and QTM when set properly.

In hindsight it might have made more sense to handle a lot of the data received from QTM as a stream of events instead of imperatively acting on it. This could

Table 8.1: An overview of some of the most popular frameworks and their release dates (in no particular order)

Framework	Release
Xamarin	2013
Qt	1995
React Native	2015
Unity	2005
Kivy	2011
Cinder	2010

have made the application more readable since its closer to the nature of how QTM actually sends the data.

To ensure a proper adherence to the MVVM pattern, a framework was used that provided convenient functionality. It removed a lot of boilerplate code and did save a lot of development time that could be used to investigate things more closely related to the research of this thesis. It is not certain that this was the right approach due to a higher application size, as well as it coupled the application to the framework to some degree. The framework made it easy to stick to the pattern and made things like navigation easier to manage.

8.2.4 Constantly Evolving Frameworks

Most commercially available cross-platform development solutions are still young and developing (as can be seen in table 8.1) [38, 27, 29, 31, 16, 6], and a whole book can be written when it comes to analyzing their key features, advantages and disadvantages. However, there is one specific characteristic that the research empirically learned, a major factor when considering any of these frameworks; continuous development. For good or for bad, these tools are constantly evolving. For instance, *Cross-platform framework for mobile applications (Xamarin)'s* current shared front-end is missing plenty of features, it is not uncommon to navigate through some files and find methods with comments along the lines of *to be implemented.* On the other side, developers of these tools are implementing new features building upon user-feedback and needs; some even have public bug databases where users can report platform errors. Because of this software's fast-evolving nature, it is common to find that official documentation sections and articles are outdated, referring to deprecated classes and objects that no longer behave as they used to. When thinking of implementing a complex feature, it is highly recommended to do an extensive documentation and forums research fist, as some required components that are generally taken for granted might not work as expected.

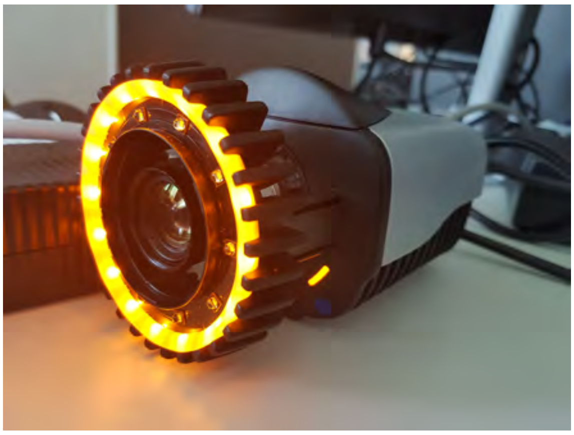


Figure 8.3: A Miqus camera with its LED-ring lit.

8.3 Summative Evaluation

This section will discuss our *user test approach* during the *summative evaluation* of the proof of concept. Findings and their implications on how the application can be improved are also discussed. These results are curated by the specific UI components that they relate to, similarly to how the results are presented in section 6.2.

8.3.1 User test approach

The proof of concept was never tested with a full camera setup due to technical difficulties and time constraints. Even though that would have provided a better understanding and a more elaborate evaluation of how well it performed in real-life scenarios, it was decided that a test with the full system would have been too unreliable and thereby would have affected the output of the test negatively. Instead of doing a test with a full camera system, a smaller setup was created using four cameras, which is not a common scenario. With that in mind, the goal was not to evaluate how well it worked in a real-life scenario, but rather evaluate how the application worked in general when interacting with a set of cameras. This did provide valuable input for creating an even better user experience and what previously had been only speculations and assumptions were now backed by real user data. On the other hand, important aspects of identified scenarios were neglected. One such scenario was the *Ladder Scenario* introduced in chapter 5 which could have provided information regarding how the application worked from a greater viewing distance. It would also enable evaluation of how well the users are able to locate the cameras using new features such as lighting the LED-ring when a camera is selected as seen in figure 8.3.

8.3.2 Observations

Observations made during summative evaluation will be discussed below. Each component will be discussed individually but also in combination where applicable. The purpose with this section is to reflect on how well the components performed and make informed suggestions on how the application can be improved based on the knowledge gained from evaluations.

Login Page
Both presentation and usability in the *Login Page* are extremely important as it is the very first interaction that the user will have with the system. A highly debated matter within the research team has been related to the way in which the application will handle the protocol's *master mode* selection -this mode lets the user change QTM's settings from the application. Accessing this mode requires a password, but showing a password field to a user if this is not going to be used can affect usability. An idea is to use a checkbox that will either show or hide this password field, but this means that if the user wants to access master mode, an extra touch is required to show such field. Another approach is to let the user connect normally to a server and then prompt for master mode through a modal dialogue; this has the disadvantage of modal excise.

When it comes to connecting to a host, there are two ways of doing this (as described before). Unfortunately, the host discovery routine was not working as expected, plus the UI component's pliancy had not been brushed enough so the users did not know how to operate it. This led the vast majority of the testers to manually enter a host's Internet Protocol (IP) address. This manual approach is another important aspect that must not be overlooked. Using a standard on-screen alphanumeric keyboard proved to be tedious, so one obvious approach is to just use a numerical keyboard. This can help mitigate some writing weariness, but the writers of this work wonder if there is a better way of writing IP addresses on a mobile device. For example with an application that knows what an IP-address looks like.

As of now, there is no proper error handling when the user attempts to do one of the following:

- Writing an invalid IP address and trying to connect to it
- Attempting to connect to a computer which is not running QTM

An easy solution would be to show a dialogue with a very specific cause for the problem and a suggestion for how to solve it. Unfortunately, due to time constraints this had to be overlooked, but it is definitely a matter for future work.

Carousel Layout
A lot of users seemed to appreciate the *carousel layout* and most of them find

it to be intuitive, the fact that it is a common interactive component in mobile application nowadays probably plays a big part in this. However, there are rooms for improvement of the carousel in the final prototype. For example, there was no interpolation when the user ended a swiping gesture in the carousel, thus the camera screens would pop into position upon finger release. This was not really something that the users noticed during evaluation, but having it would certainly make the animations feel softer which in turn might have made the application feel more responsive and user friendly.

The evaluation showed that some users perceived the carousel as being too sensitive when making a swiping gesture. Which in some cases would result in a user selecting a camera by mistake. In retrospect this was to be expected since no heavy thought process besides intuition played apart when the sensitivity was defined for a swipe gesture.

Grid Layout

Several users expressed an interest in being able to change the mode of more than one camera at a time when in the *grid layout*. This was also discussed during the ideation phase but was abandoned with the reasoning that it would make the application more complex. Based on this feedback, it might be worth reevaluating this feature since the users seem to be very open for the idea. This also indicated that it is important to recognize that the application should not only be designed with intermediates in mind, but also experts.

Settings Drawer

Users felt that the settings drawer did obstruct the view of the camera screens. This might partly be due to the drawer taking up too much screen space, but it might also have to do with the fact that it was static, meaning that there was no way of hiding it from view. Letting the user hide the settings drawer will enable to view the camera screen using more of the display. This is important since they will be far away from the device in certain scenarios (presented in section 5.2.2). The risk when hiding interface elements is that users might not know where to look for them [23]. This could for example be mitigated by adding hinting animations of where the drawer is located when opening the camera page or adding a button that opens the drawer. This is further exemplified in figure 8.4. An even simpler approach would be to add a button to toggle the drawer with the disadvantage of taking up more screen space.

There has been discussion around hiding the drawer when entering landscape mode as well. This was never implemented and thus has not been evaluated but it is still believed to be a feature in need for further investigation.

Some users commented that they wanted to be able to add settings values directly. A potential solution to answer this need could be to integrate an input field where a user can manually enter a value.

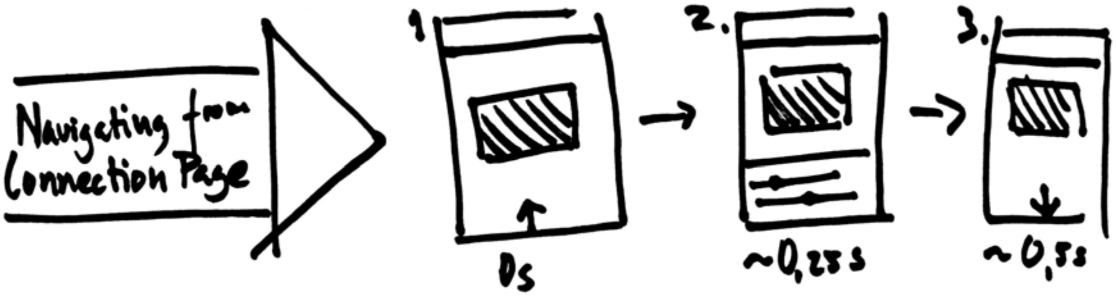


Figure 8.4: Storyboard depicting an affordance hint animation of the settings drawer

Mode toolbar

While carrying out the usability tests there was a situation that was unexpected, oddly never accounted for in the design process. During one of the tasks which required a user to change a camera's stream mode to *intensity mode*, two of the six testers hit a wall because as of then, they were not familiar with all the features and data that the company's mocap cameras can produce (one of the users was fairly new in the company). The target user-base was always assumed to have a wide technical knowledge with regards to these systems, problems arise when said users are still not well-acquainted with them. This made the research team start wondering about practical issues such as *usage frequency*; how likely would it be for users to estrange from the iconography used in the application after not using it for a couple of weeks? These concerns brought-up interesting ideas such as incorporating text to the mode switch buttons and/or displaying a *tip of the day* during loading screens.

Merging Page Functionality

Combining both *carousel layout* with the *grid layout* was not something that was considered from the beginning; they both display what camera x is seeing at a given time but they do so with different scopes (different number of cameras). They also serve different purposes, as a user can manipulate specific camera settings through the *carousel layout* only. As previously stated in section 6.2.3, the grid view was dropped for a while due important stability issues. The decision to bring it back and combine both pages into one came right before the formative test sessions for the following reasons:

- A considerable amount of work was put into designing and developing the grid view.
- The research team wanted to test the grid view's usability.
- It was significant to this research to find out which one was used the most.
- It was even more important to find out how these two views interacted with each other in the form of usage and flow.

From a user's perspective, there is virtually no difference between navigating through

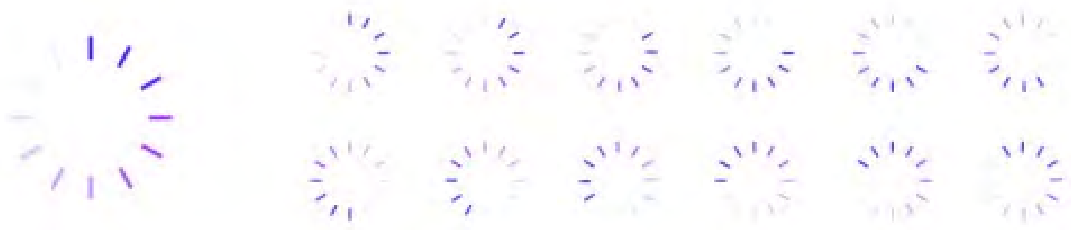


Figure 8.5: Infinite loading spinner example

two pages or one (besides loading times), as this behaviour can be emulated. There is however, a significant contrast when it comes to actual implementation.

Technical Pros

- Since both views are being handled inside the same 3D scene, the Central Processing Unit (CPU) has an easier time since there is no need to create and destroy objects and surfaces every time a view gets navigated to/from.
- Helps solving low level threading issues between graphics context (Open Graphics Library (OpenGL) or An application programming interface for multimedia purposes, especially game programming (DirectX)) and Xamarin's navigation stack.
- Navigation becomes visibly faster.

Technical Cons

- Mixed logic gets confined into one single view-model class.
- Becomes easy to mix-up functionality and produce bugs.
- Requires more dedication to get it right (as in clean, functional and understandable).

Based on the usage data gathered during the user tests it is clear that the grid view were used by the majority of the users. This data can be visualized in appendix C.

8.4 General Usability

Regarding application usability, there are still some previously-discussed things that need to be addressed. For example, a user needs to know that things are happening under the hood, especially during a CPU-heavy operation, otherwise the system "hangs" and it is then considered unresponsive. As of now there are no loading screens or indicators of any kind to communicate the application's status to the user. This has been proven to be a serious problem since the user is bound to get irritated and tap everything on the screen to corroborate that the application is indeed frozen. This leads to strange behaviour, actual freezes and even crashes. It is a top priority to increase performance and responsiveness, but some loading will still need to be done, which should be addressed with the use of loading indicators in the form of spinners (such as the one seen in figure 8.5).

As previously discussed in section 8.3.2, user-input validation is not being performed by the proof of concept. This leads to severe stability issues when a non-valid entry is produced, ranging from system hangs to sudden crashes. A layer which takes care of this and handles the situation accordingly still needs to be developed and tested. Better error handling needs to be implemented to make the application more fault-tolerant. In the cases where this is not possible, a message should be displayed to the user explaining to them what happened and the best way to proceed.

8.5 Usage data

The usage data that was gathered during the formative evaluation did provide useful insights that can be used in a future iteration of the application, but there is room for improvement when it comes to how this data is gathered. It would be useful to add data to each user interaction to make it possible to do a distinction between each user session. This could help when analyzing aspects regarding user flow and interactions. This was not taken into account before the formative evaluation and thus it was only possible to determine user flow by manually looking at the time span.

8.6 Looking Back at the Design Process

As mentioned in chapter 3 we sought out to follow a design process that took inspiration from the Double Diamond (DD) design process. This worked well for this thesis work and we managed to get a strong foundation of what needed to be developed early on. Emphasizing ideas and evaluation of the state during the early stages enabled us to get a variety of potential design solutions to work with. Furthermore, it created discussion material that could be used before and during development, which in turn made the work adaptable to quick changes.

The development was done by iteratively creating prototypes. This was an appropriate approach early on and most of the critical parts of the code were implemented with ease. It is however important to state that this approach created a lot of technical depth due to a *quick and dirty* implementation mindset. A result of this was that changing one part of the code would affect other parts, in other terms, *spaghetti code*; something that was time consuming and the cause for lots of frustration. However, it did provide a development platform that encouraged experimentation since breaking the code was to be expected. Furthermore, it gave the group a clear idea of what worked and what did not, enabling the research group to avoid such implementations during development of a more refined application.

8.7 Changing Research Angle

This research work's objective was not always to explore how a mobile application can aid the setup stage of an optical mocap system. The original proposal had a much more technical approach and it was aimed to design and develop a software toolkit, built upon a cross-platform framework that would integrate Qualisys' real-time SDK. This toolkit was then to be evaluated by means of a successor application to *ViewFinder*, our now proof of concept *ArqusFinder*. The research team was always confused about this software toolkit as there was not a clear distinction between the RT SDK, the toolkit and the applications to be developed using the toolkit. With hard time constraints, background research, an extensive formative evaluation of the previous application and a prototype yet to be designed and developed, it was agreed by both ends (Qualisys team and research team) that the research angle should be changed to a multi-platform mobile application one.

9

Conclusions and Future Work

The motivation behind this thesis work in general was to research how tools for interaction with motion capture systems might be designed. In particular it was about designing a mobile tool to aid users during the setup stage of a motion capture system, as the current solutions were lacking in usability and maintainability. The question this thesis sought to answer was *What should be considered when designing a mobile tool that aids with the setup stage of an optical motion capture system?* and the following section will conclude our findings.

9.1 Conclusions

The proposed approach to answer the research question was to create a proof of concept of a more maintainable and user friendly tool than what was currently available. This research work aimed to investigate the implications of creating the proof of concept and abstracting specific considerations from these findings. Three main areas for consideration were revealed by the end of this work and are presented in the following sections.

9.1.1 Mobile Development

Different evaluations performed throughout this work substantiated our belief that a mobile approach was necessary for providing simple, fast and intuitive interactions when working with the setup of a motion-capture camera system. These evaluations made it clear that users were accustomed to use a mobile device - such as a smart phone or tablet - as a tool during the setup stage of motion capture. Interactions between a motion capture system and mobile platforms never revealed to be lacking in any way, therefore no need for a more novel approach was elicited. Moreover, formative evaluations made it evident that users will usually be looking at the mobile device screen from afar while making modifications to the system. Based on this, it is important to utilize the screen size as much as possible to conform to user needs.

Different mobile devices behave differently, and their performance can greatly vary. When working with complex systems such as a motion capture one, it is indispensable to constantly perform rigorous testing on multiple devices. This can reveal important hardware limitations and differences which must be taken into account to ensure a good user experience.

9.1.2 Cross-platform Development

A mobile cross-platform framework was utilized with the belief that this would improve maintainability. The proof of concept proved that it is possible to create a mobile, multi-platform tool that aid in the setup of a motion-capture system with the help of a single developing environment. We strongly believe that working with this approach greatly increases maintainability, as it was empirically discovered that building for different platforms from a single code-base is by far faster than taking the native approach. Most of the testing can also be done using a shared code base, although testing on all platforms using physical devices is still a necessity due inconsistent behavior between platforms. Another good reason is that all necessary add-ons can be managed from the same environment, making it easy to share libraries amongst targets (if available), or using specific ones for some.

Furthermore a common environment to design both the user interface as well as user interactions was employed. This approach enables designers to come up with a general look & feel that can be applied to well-known native user interface elements, thereby reducing the need for repetitive code.

Some of the down-sides of targeting several platforms when designing a look & feel in a shared environment is that it might be hard to target very specific native elements. This can be resolved using custom extensions that target specific native APIs, at the cost of having more code. Although, finding a corresponding interface for the other platforms might not be possible in some cases. Platform-specific resources and UI design guidelines still need to be taken into account as ultimately, this shared interface code is interpreted and compiled into native elements.

9.1.3 Real-time Systems

Apart from being mobile, the application also needs to handle real-time data to provide an interactive experience with the system. Some of the data demands heavy computing which can cause severe stability issues and drastically decrease usability. This needs to be tackled with a set of optimizations spanning different application levels; computer graphics, networking and memory usage are some examples of potential bottlenecks that must always be considered. With the use of such optimization, our proof of concept managed to get close to the performance of the native predecessors. This shows that performance-wise, a cross-platform framework that compiles into native code is viable when working with real-time data that demands heavy computing.

9.2 Contribution

This work's contribution is a mobile application artifact that demonstrates that it is possible to develop a cross-platform application that communicates with an optical mocap system in real-time. This same artifact proves the mocap industry that tools for interacting with motion camera systems can still be improved. Subsequently, as a result of the design-driven approach of this research, a blueprint for both design and implementation of this type of interactive tools has been extensively documented. This is expected to aid future researchers in the field of real-time motion capture and mobile computing.

9.3 Future Work

There is still a lot that can be done to create an even better tool to aid the user during the setup stage of a motion-capture system. Testing the design solution with real-life scenarios should be done to gain extra insights and valuable information regarding user-system interaction to see how this could be improved. More specifically attention should be put into how the user, the application, the camera system and the physical components play together. Direct physical interaction with a camera rig is a subject that still needs to be addressed, especially with regards to safety. There must be a better way than standing on top of a ladder with a handheld device on one hand and a camera in the other; unfortunately this topic was out of the scope of this thesis.

Additionally, handling system scalability is one of the most challenging aspects when assuring stability and performance; the proof of concept has been tested not more than a couple of times with a real setup consisting of ten cameras, unfortunately the results were not positive. Thus investigating potential optimizations is of great importance to improve user experience.

Further field research could be done to investigate how a tool can help in cases where there is an even greater need for portability such as setting up motion-capture systems outdoors or even underwater as seen in figure 8.1.

Finally, the mobile market is a fast-evolving one with new devices and software being released with each turn of the year, innovative forms of interaction are constantly introduced such as augmented reality. It is our belief that further research on how these novel technologies could be used with a motion capture system is relevant.

Bibliography

[1] Qualisys | motion capture system. `http://www.qualisys.com/`. Accessed: 2017-01-25.

[2] Mobile app testing on hundreds of devices | xamarin test cloud. `https://www.xamarin.com/test-cloud`. Accessed: 2017-01-30.

[3] About qt | qt wiki. `https://wiki.qt.io/About_Qt`. Accessed: 2017-01-30.

[4] Platform architecture | android developers. `https://developer.android.com/guide/platform/index.html`. Accessed: 2017-05-24.

[5] Prototypes, specification, and diagrams in one tool | axure software. `https://www.axure.com`. Accessed: 2017-04-27.

[6] Cinder's github repository. `https://github.com/cinder/Cinder/tags`. Accessed: 2017-05-16.

[7] Collins dictionary. `https://www.collinsdictionary.com/dictionary/english/brainstorming`. Accessed: 2017-02-01.

[8] Cocoa touch layer. `https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html#//apple_ref/doc/uid/TP40007898-CH3-SW1`, . Accessed: 2017-05-24.

[9] Core os layer. `https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1`, . Accessed: 2017-05-24.

[10] Core services layer. `https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW5`, . Accessed: 2017-05-24.

[11] Design principles - overview - ios human interface guidlines. `https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/`, . Accessed: 2017-05-24.

[12] Media layer. `https://developer.apple.com/library/content/`

documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW4, . Accessed: 2017-05-24.

[13] Kinect for xbox one. http://www.xbox.com/en-US/xbox-one/accessories/kinect. Accessed: 2017-01-25.

[14] Kivy: Cross-platform python framework for nui development. https://kivy.org/#home, . Accessed: 2017-01-30.

[15] Kivy: Cross-platform python framework for nui development. https://kivy.org/#organization, . Accessed: 2017-01-30.

[16] Kivy's github repository. https://github.com/kivy/kivy/tags?after=1.0.6, . Accessed: 2017-05-16.

[17] The kj-technique: A group process for establishing priorities ux articles by uie. https://articles.uie.com/kj_technique/. Accessed: 2017-02-10.

[18] Material design - icons. https://material.io/guidelines/style/icons.html#, . Accessed: 2017-05-05.

[19] Introduction - material design - material design guidelines. https://material.io/guidelines/, . Accessed: 2017-05-24.

[20] Mobile center | mobile app development | visual studio. https://www.visualstudio.com/vs/mobile-center/. Accessed: 2017-05-29.

[21] Our vision, values and promises. https://www.mockplus.com/about. Accessed: 2017-04-27.

[22] Urhosharp.forms. https://www.nuget.org/packages/UrhoSharp.Forms/. Accessed: 2017-05-26.

[23] What you need to know about popular ux naigation patters. http://www.freshform.com/blog/popular-ux-navigation-patterns/. Accessed: 2017-05-17.

[24] Qml applications. http://doc.qt.io/qt-5/qmlapplications.html. Accessed: 2017-01-30.

[25] Qt - product | the ide. https://www.qt.io/ide/. Accessed: 2017-01-30.

[26] Introducing the qt lite project–qt for any platform, any thing, any size - qt blog. http://blog.qt.io/blog/2016/08/18/introducing-the-qt-lite-project-qt-for-any-platform-any-thing-any-size/. Accessed: 2017-01-30.

[27] History of cute qt. http://www.masteringqt.com/2013/06/history-of-cute-qt.html. Accessed: 2017-05-16.

[28] Qt qml 5.8. http://doc.qt.io/qt-5/qtqml-index.html. Accessed: 2017-01-30.

[29] React Native's github repository. https://github.com/facebook/react-native/tags?after=v0.3.4. Accessed: 2017-05-16.

[30] Measuringu: 10 things to know about the single ease question (seq). https://measuringu.com/seq10/. Accessed: 2017-05-10.

[31] Unity - editor version release dates. http://web.archive.org/web/20141015144227/http://docs.unity3d.com/Manual/ReleaseDates.html. Accessed: 2017-05-16.

[32] An introduction to urhosharp. https://developer.xamarin.com/guides/cross-platform/urho/introduction/. Accessed: 2017-05-02.

[33] Control - android apps on google play. https://play.google.com/store/apps/details?id=com.vicon.control. Accessed: 2017-01-30.

[34] Viewfinder - andoid apps on google play. https://play.google.com/store/apps/details?id=se.qualisys.magnum.viewfinder. Accessed: 2017-01-30.

[35] Xamarin pricing. https://store.xamarin.com/. Accessed: 2017-05-24.

[36] What is xaml? https://msdn.microsoft.com/en-us/library/cc295302.aspx. Accessed: 2017-05-25.

[37] Introduction to game development with xamarin. https://developer.xamarin.com/guides/cross-platform/game_development, . Accessed: 2017-05-02.

[38] Xamarin delivers tool for building native mac os x apps with c#. http://www.zdnet.com/article/xamarin-delivers-tool-for-building-native-mac-os-x-apps-with-c/, . Accessed: 2017-05-16.

[39] R.P.M Fonseca K. Siewert J.A.A Martins A.L.C. Fujarra, R.T GOncalves. Optical motion capture as a technique for measuring the water wave elevation. Number 4, 2009.

[40] D. Benyon, P. Turner, and S. Turner. *Designing Interactive Systems: People, Activities, Contexts, Technologies*. Addison-Wesley, 2005. ISBN 9780321116291. URL https://books.google.se/books?id=iWe7VkFW0zMC.

[41] Luc Berthouze and Margaret Mayston. Design and validation of surface-marker clusters for the quantification of joint rotations in general movements in early infancy. *Journal of Biomechanics*, 44(6):1212 – 1215, 2011. ISSN 0021-9290. doi: http://dx.doi.org/10.1016/j.jbiomech.2011.01.016. URL //www.sciencedirect.com/science/article/pii/S0021929011000467.

[42] Wallace AM Colborne GR, Hadley NR. A novel method for defining the greyhound talocrural joint axis of rotation for hinged transarticular external skeletal fixation. *Veterinary and Comparative Orthopaedics and Traumatology*, (4):298–303, 2013.

[43] Alan Cooper, Robert Reimann, and Dave Cronin. *About Face 3: The Essentials of Interaction Design.* John Wiley & Sons, Inc., New York, NY, USA, 2007. ISBN 9780470084113.

[44] The Design Council. Eleven lessons: managing design in eleven global brands | the design process. 2007.

[45] CreatingMinds. Tools for selecting ideas. `http://creatingminds.org/tools/tools_selection.htm`. Accessed: 2017-03-01.

[46] Mathias Bylund Emmanuel Batis. Arqusfinder demo - password: arqus. URL `https://vimeo.com/219486756`.

[47] Marsha E. Fonteyn, Benjamin Kuipers, and Susan J. Grobe. A description of think aloud method and protocol analysis. *Qualitative Health Research*, 3(4): 430–441, 1993. doi: 10.1177/104973239300300403. URL `http://dx.doi.org/10.1177/104973239300300403`.

[48] Zlatko Franjcic. *Towards improving performance and user-friendliness of optical motion capture systems.* Technical report L - Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, no: Lic. Institutionen för tillämpad informationsteknologi (Chalmers), Chalmers tekniska högskola, 2015.

[49] Zlatko Franjcic, Pawel W. Wozniak, G. Kasparavičiute, and Morten Fjeld. Wavi: Improving motion capture calibration using haptic and visual feedback. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2016*, pages 254–265, 2016. ISBN 978-145034408-1.

[50] William Gaver. What should we expect from research through design? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 937–946, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208538. URL `http://doi.acm.org/10.1145/2207676.2208538`.

[51] Gutemberg B. Guerra-filho. Optical motion capture: Theory and implementation. *Journal of Theoretical and Applied Informatics (RITA*, 12:61–89, 2005.

[52] Julie A. Jacko. *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications.* CRC Press, 2012. ISBN 9781439829431.

[53] B. Cameron & W. Josh. *Essential Mobile Interaction Design: Perfecting Interface Design in Mobile Apps.* Addison-Wesley Professional, 2014. ISBN 978-0321961570.

[54] T. Kelley. *The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm.* Crown Publishing Group, 2007. ISBN 9780307423863. URL `https://books.google.se/books?id=yjgO7Og_qbsC`.

[55] Richard Kennard and John Leaney. Towards a general purpose architecture

for {UI} generation. *Journal of Systems and Software*, 83(10):1896 – 1906, 2010. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2010.05.079. URL `//www.sciencedirect.com/science/article/pii/S0164121210001597`.

[56] Richard Larkin. Kivy showcase: a short exploration of how kivy is changing the world. `https://www.youtube.com/watch?v=kXLQ_7GGMnM`, October 2015.

[57] Mahesh Panhale. *Beginning Hybrid Mobile Application Development*. Apress, 2016. ISBN 978-1-4842-1315-5.

[58] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 0471402494.

[59] K. Rijnieks. *Cinder - Begin Creative Coding*. Packt Publishing, 2013. ISBN 9781849519564.

[60] Dan Saffer. *Designing for Interaction: Creating Innovative Applications and Devices*. New Riders Publishing, Thousand Oaks, CA, USA, 2nd edition, 2009. ISBN 0321643399, 9780321643391.

[61] Douglas Schuler and Aki Namioka, editors. *Participatory Design: Principles and Practices*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1993. ISBN 0805809511.

[62] D. J. Sturman. A brief history of motion capture for computer character animation. *SIGGRAPH 94, Character Motion Systems, Course notes 1*, 1994.

[63] Jacob O. Wobbrock and Julie A. Kientz. Research contributions in human-computer interaction. *interactions*, 23(3):38–44, April 2016. ISSN 1072-5520. doi: 10.1145/2907069. URL `http://doi.acm.org/10.1145/2907069`.

[64] John Zimmerman, Jodi Forlizzi, and Shelley Evenson. Research through design as a method for interaction design research in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 493–502, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: 10.1145/1240624.1240704. URL `http://doi.acm.org.proxy.lib.chalmers.se/10.1145/1240624.1240704`.

A

Appendix: Brainstorming Ideas

High priority

1. Light-up camera that is currently being viewed
2. Incorporate calibration haptics (magic wand)
3. Hololens to visualize tracking data
4. Color coding joints depending on marker count (average)
5. Pie chart indicating number of potential markers/currently tracked
6. Move in 3D space with gyroscope and accelerometers
7. Grid with cameras
   (a) Re-order cameras in the grid with drag-and drop
   (b) Able to switch stream mode from each camera
8. Speech recognition
9. Viewport interaction
   (a) 3D view: add small buttons of transformation interaction

Medium priority

1. Calibration
   (a) Calibration indicator good-bad
2. Navigation/UI
   (a) Tabbed view
   (b) Navigation bar on bottom
   (c) Floating button for changing stream mode
   (d) Swipe up/down to change stream mode
   (e) Drawer menu
   (f) Change orientation by holding and rotating two fingers
   (g) Round slider
   (h) Button to switch current 2D view to 3D
3. Layout
   (a) Dual layout mode with 2D camera feed over 3D or viceversa
4. Settings and Configuration
   (a) Side bars/edges to configure exposure/threshold (visualize and/or modify)

Low priority

1. Viewport interaction
   (a) Focus button/focus gesture
2. Spatial interactions
   (a) Aim phone to camera and get its feed
3. Networking
   (a) Automatic server refresh
4. Visualization
   (a) Draw bones between joints
   (b) Superimposing 3D data on video stream
   (c) Displaying cameras in 3D view
   (d) Skeleton Mesh
5. Navigation/UI
   (a) Vicon-like camera scroll
6. Calibration
   (a) Mocap volume real-time preview

B

Appendix: Requirements

B.1 ArqusFinder Requirement Description

Functional Requirements

- FR1. Should be able to change camera mode without changing it for all cameras.
- FR2. Component interaction should be isolated from other components.
- FR3. Should be able to change camera view in camera-detail-view
- FR4. There should be a way of rotating the camera view
- FR5. A way to identify a Miqus camera with the aid of the ID-ring
- FR6. Slider components for camera settings should work smoothly and without any noticeable lag.
- FR7. Slider components for camera settings should make the current changing value obvious.
- FR8. Device should not go idle when using the application
- FR9. Application should offer an overview of all the cameras
- FR10. If no measurement has started there should be an obvious way of doing that
- FR11. User should be able to zoom in and out of camera view
- FR12. User should be able to pan camera view
- FR13. Should be able to manually connect to host through IP specification
- FR14. Application should display all hosts running in the LAN

Non-functional Requirements

- NFR1. Error-handling should be done without disrupting the users flow

C

Appendix: Usage data

C.1 Screen Layout Toggled Data

Name	**CameraScreenLayout**	**Count**
CameraPageViewModel	grid	92
CameraPageViewModel	carousel	31

C.2 Message Data

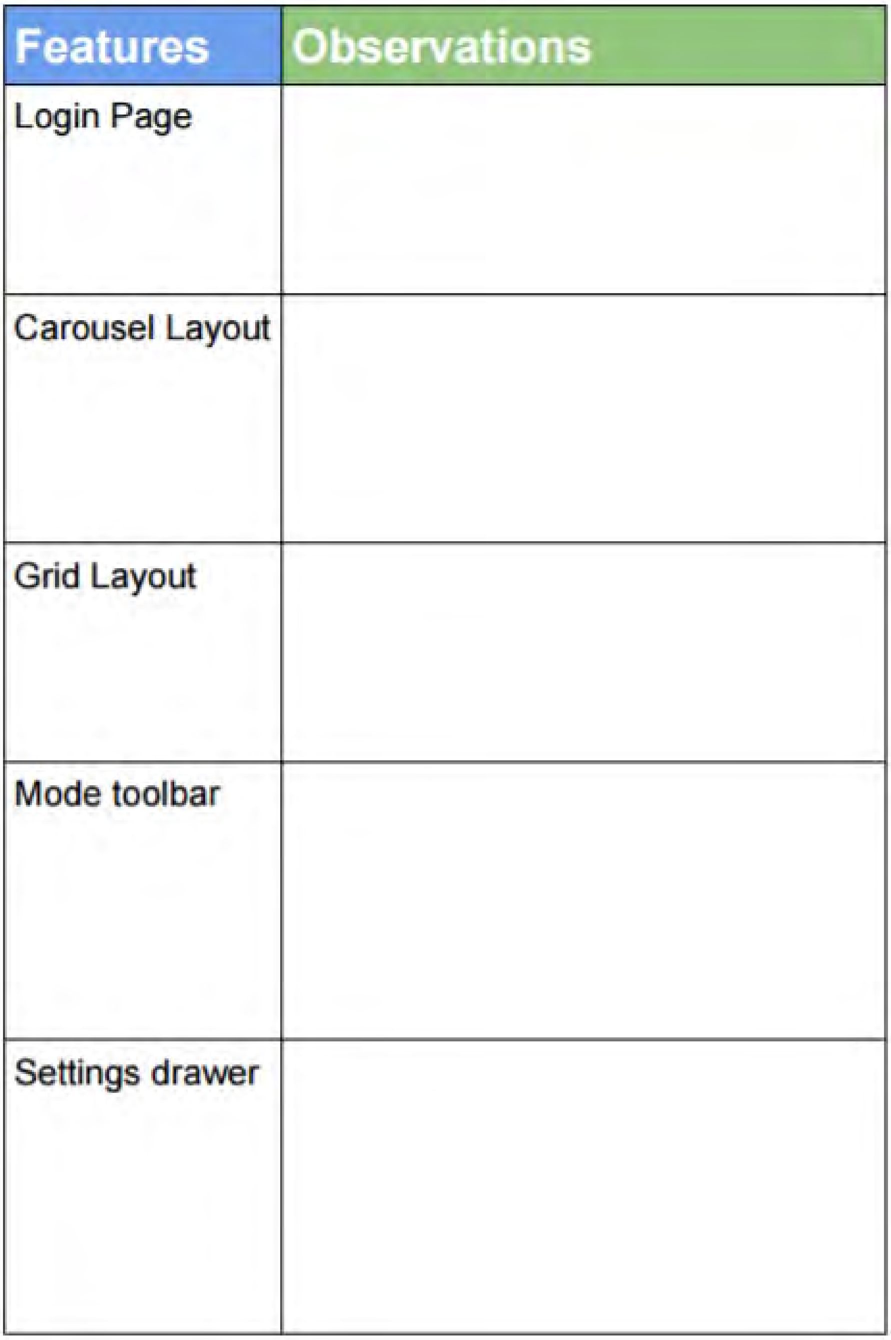
Name	**Message**	**Count**
CameraScreen	STREAM_DATA_SUCCESS 1	80
CameraScreen	STREAM_DATA_SUCCESS 2	80
CameraScreen	STREAM_DATA_SUCCESS 3	78
CameraScreen	STREAM_DATA_SUCCESS 4	78
CameraScreen	STREAM_MODE_CHANGED1	40
CameraApplication	SET_CAMERA_SELECTION	543
CameraApplication	SET_CAMERA_SCREEN_LAYOUT	39
Camera	STREAM_MODE_CHANGED4	50
Camera	STREAM_MODE_CHANGED1	33
Camera	STREAM_MODE_CHANGED3	28
Camera	STREAM_MODE_CHANGED2	17
Camera	STREAM_MODE_CHANGED13	1
App	CONNECTED	84
App	DISCONNECTED	44
CameraPageViewModel	SET_CAMERA_SCREEN_LAYOUT	123

C.3 Stream Mode Data

Name	**NewMode**	**Count**
Camera	ModeVideo	49
Camera	ModeMarker	42
Camera	ModeMarkerIntensity	39

D

Appendix: Evaluation table



Features	Observations
Login Page	
Carousel Layout	
Grid Layout	
Mode toolbar	
Settings drawer	

E

Appendix: Style Implementation

Listing E.1: The theme of the application defined in the shared code base.

```
App.xaml
<Style x:Key="AppBar" TargetType="Frame">
    <Setter Property="BackgroundColor" Value="#212121" />
</Style>

<Style x:Key="Card" TargetType="Frame">
    <Setter Property="BackgroundColor" Value="#424242" />
</Style>

<Style x:Key="Background" TargetType="StackLayout">
    <Setter Property="BackgroundColor" Value="#303030" />
    <Setter Property="VerticalOptions" Value="FillAndExpand" />
</Style>

<Style x:Key="PrimaryText" TargetType="Label">
    <Setter Property="TextColor" Value="#FFFFFF" />
    <Setter Property="Opacity" Value="1" />
</Style>

<Style x:Key="SecondaryText" TargetType="Label">
    <Setter Property="TextColor" Value="#FFFFFF" />
    <Setter Property="Opacity" Value="0.7" />
</Style>

<Style x:Key="DisabledText" TargetType="Label">
    <Setter Property="TextColor" Value="#FFFFFF" />
    <Setter Property="Opacity" Value="0.5" />
</Style>
```

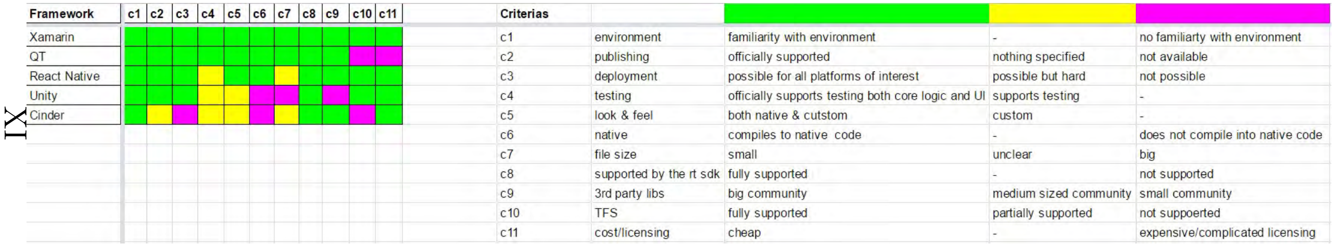
Listing E.2: An Android style.xml example file that defines styling for a theme.

```
---------------------------------------------------------------
style.xml
```

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <style name="MyTheme" parent="MyTheme.Base">
  </style>
  <style name="MyTheme.Base"
    parent="Theme.AppCompat.NoActionBar">
    <item name="colorPrimary">
        @color/primary
    </item>
    <item name="colorPrimaryDark">
        @color/primaryDark
    </item>
    <item name="colorAccent">@color/accent</item>
    <item name="windowActionModeOverlay">true</item>
  </style>
</resources>

---------------------------------------------------------------
color.xml

<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <color name="primary">#2196F3</color>
  <color name="primaryDark">#1976D2</color>
  <color name="accent">#FFC107</color>
</resources>

---------------------------------------------------------------
```

F

Appendix: Framework Evaluation Criteria

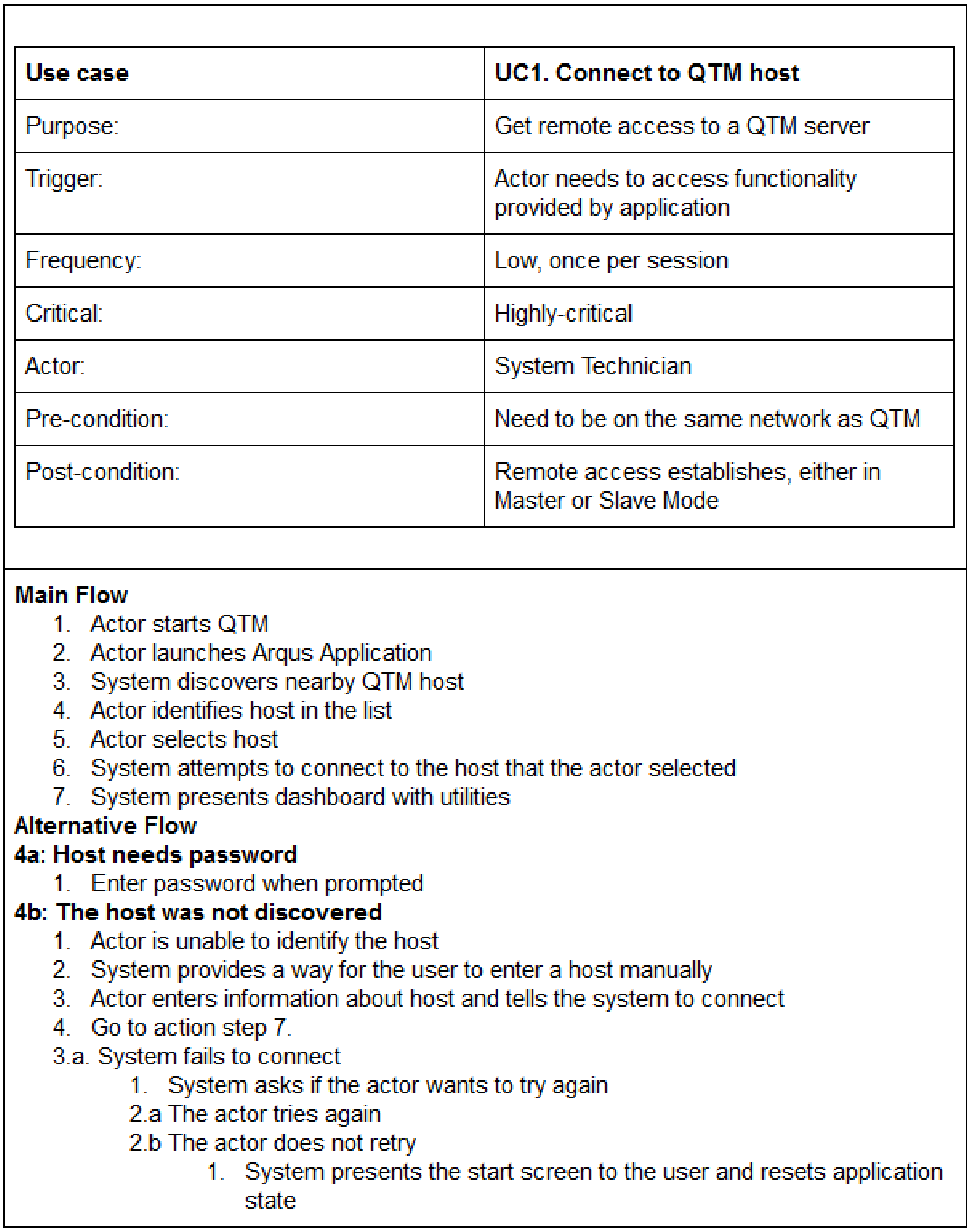


Framework	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11
Xamarin	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
QT	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪
React Native	🟩	🟩	🟩	🟨	🟩	🟩	🟨	🟩	🟩	🟩	🟩
Unity	🟩	🟩	🟩	🟨	🟨	🟪	🟪	🟩	🟪	🟩	🟩
Cinder	🟩	🟨	🟪	🟨	🟨	🟪	🟨	🟩	🟩	🟪	🟩

Criterias		🟩	🟨	🟪
c1	environment	familiarity with environment	-	no familiarty with environment
c2	publishing	officially supported	nothing specified	not available
c3	deployment	possible for all platforms of interest	possible but hard	not possible
c4	testing	officially supports testing both core logic and UI	supports testing	-
c5	look & feel	both native & cutstom	custom	-
c6	native	compiles to native code	-	does not compile into native code
c7	file size	small	unclear	big
c8	supported by the rt sdk	fully supported	-	not supported
c9	3rd party libs	big community	medium sized community	small community
c10	TFS	fully supported	partially supported	not suppoerted
c11	cost/licensing	cheap	-	expensive/complicated licensing

G

Appendix: Use cases



Use case	UC1. Connect to QTM host
Purpose:	Get remote access to a QTM server
Trigger:	Actor needs to access functionality provided by application
Frequency:	Low, once per session
Critical:	Highly-critical
Actor:	System Technician
Pre-condition:	Need to be on the same network as QTM
Post-condition:	Remote access establishes, either in Master or Slave Mode

Main Flow

1. Actor starts QTM
2. Actor launches Arqus Application
3. System discovers nearby QTM host
4. Actor identifies host in the list
5. Actor selects host
6. System attempts to connect to the host that the actor selected
7. System presents dashboard with utilities

Alternative Flow

4a: Host needs password

1. Enter password when prompted

4b: The host was not discovered

1. Actor is unable to identify the host
2. System provides a way for the user to enter a host manually
3. Actor enters information about host and tells the system to connect
4. Go to action step 7.

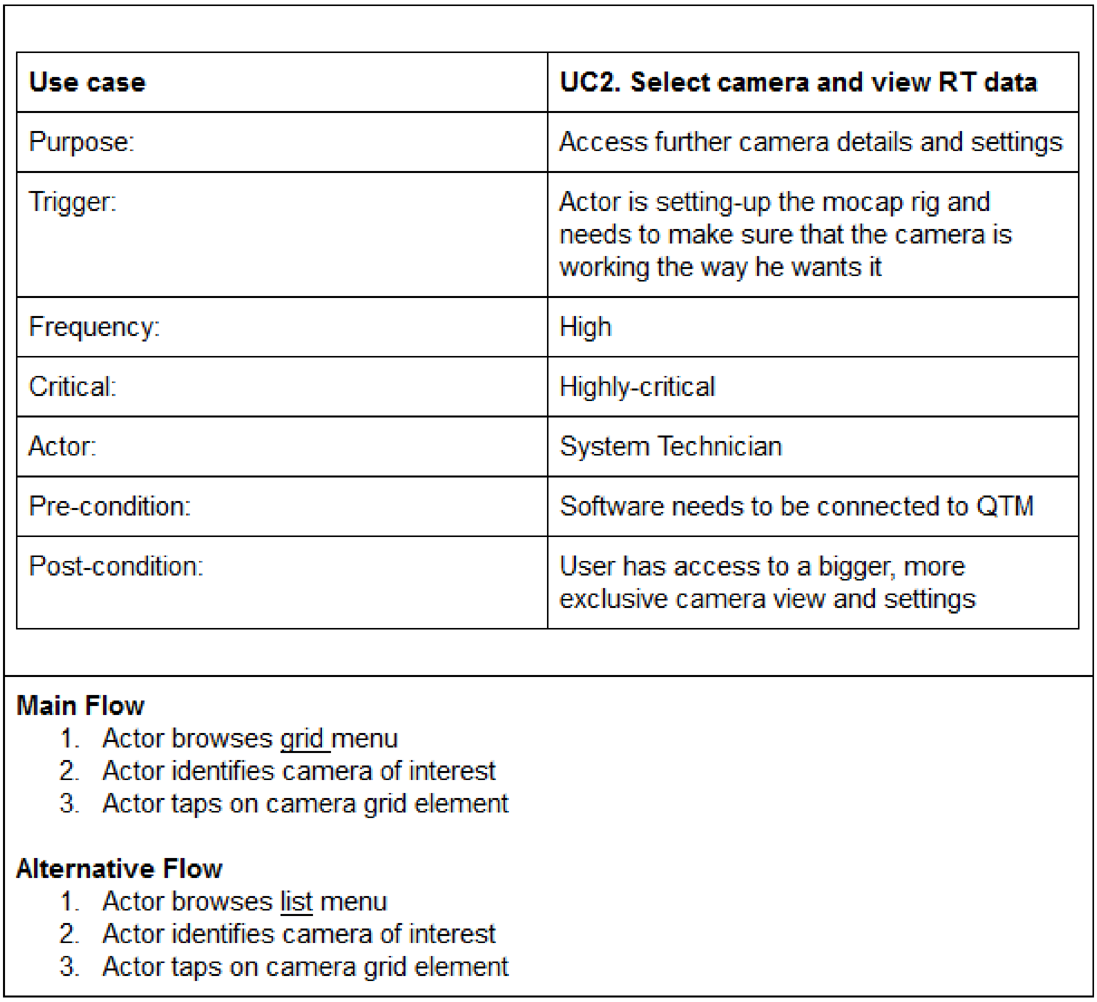
3.a. System fails to connect

  1. System asks if the actor wants to try again

  2.a The actor tries again

  2.b The actor does not retry

    1. System presents the start screen to the user and resets application state



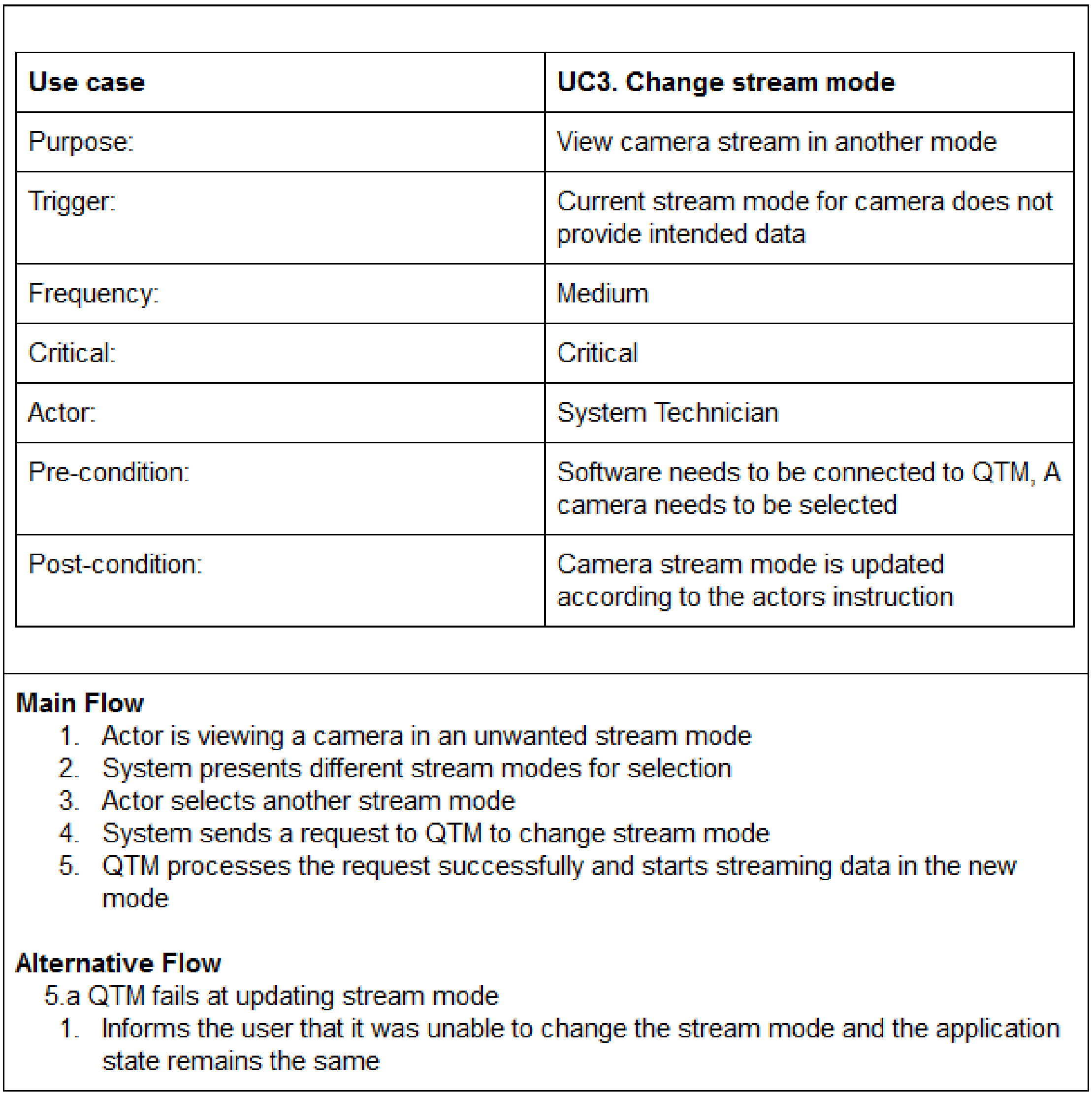
Use case	**UC2. Select camera and view RT data**
Purpose:	Access further camera details and settings
Trigger:	Actor is setting-up the mocap rig and needs to make sure that the camera is working the way he wants it
Frequency:	High
Critical:	Highly-critical
Actor:	System Technician
Pre-condition:	Software needs to be connected to QTM
Post-condition:	User has access to a bigger, more exclusive camera view and settings

Main Flow

1. Actor browses grid menu
2. Actor identifies camera of interest
3. Actor taps on camera grid element

Alternative Flow

1. Actor browses list menu
2. Actor identifies camera of interest
3. Actor taps on camera grid element



Use case	**UC3. Change stream mode**
Purpose:	View camera stream in another mode
Trigger:	Current stream mode for camera does not provide intended data
Frequency:	Medium
Critical:	Critical
Actor:	System Technician
Pre-condition:	Software needs to be connected to QTM, A camera needs to be selected
Post-condition:	Camera stream mode is updated according to the actors instruction

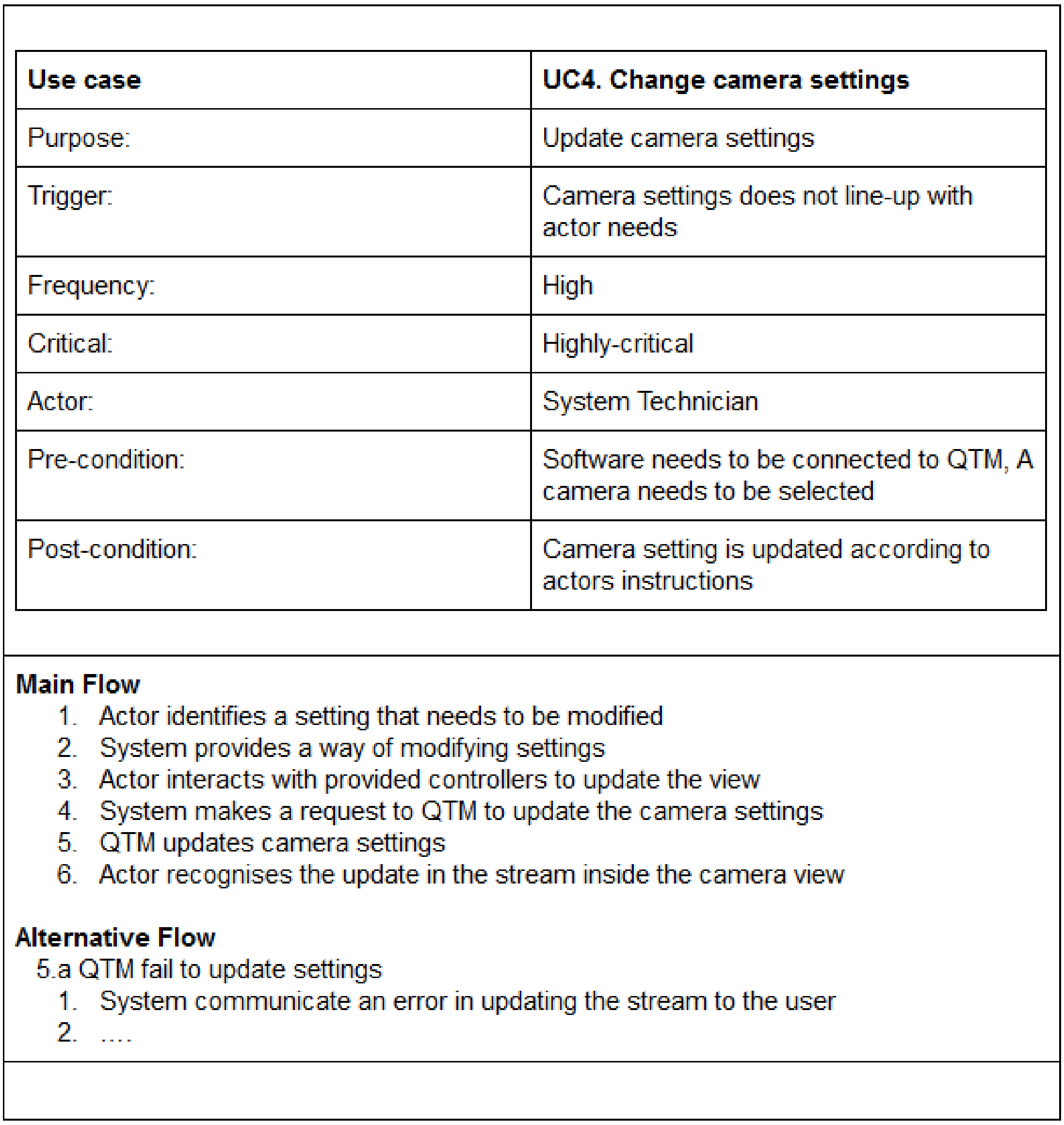
Main Flow

1. Actor is viewing a camera in an unwanted stream mode
2. System presents different stream modes for selection
3. Actor selects another stream mode
4. System sends a request to QTM to change stream mode
5. QTM processes the request successfully and starts streaming data in the new mode

Alternative Flow

5.a QTM fails at updating stream mode

1. Informs the user that it was unable to change the stream mode and the application state remains the same



Use case	**UC4. Change camera settings**
Purpose:	Update camera settings
Trigger:	Camera settings does not line-up with actor needs
Frequency:	High
Critical:	Highly-critical
Actor:	System Technician
Pre-condition:	Software needs to be connected to QTM, A camera needs to be selected
Post-condition:	Camera setting is updated according to actors instructions

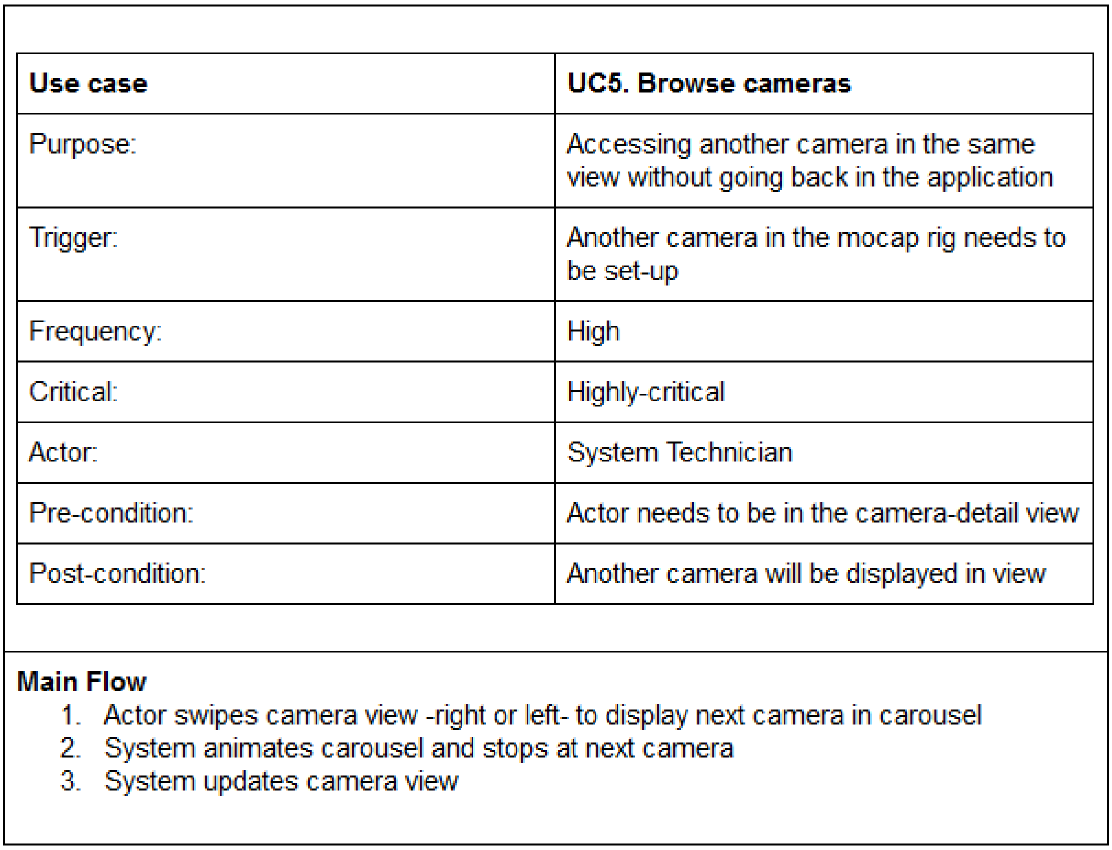
Main Flow

1. Actor identifies a setting that needs to be modified
2. System provides a way of modifying settings
3. Actor interacts with provided controllers to update the view
4. System makes a request to QTM to update the camera settings
5. QTM updates camera settings
6. Actor recognises the update in the stream inside the camera view

Alternative Flow

5.a QTM fail to update settings

1. System communicate an error in updating the stream to the user
2. ….



Use case	**UC5. Browse cameras**
Purpose:	Accessing another camera in the same view without going back in the application
Trigger:	Another camera in the mocap rig needs to be set-up
Frequency:	High
Critical:	Highly-critical
Actor:	System Technician
Pre-condition:	Actor needs to be in the camera-detail view
Post-condition:	Another camera will be displayed in view

Main Flow

1. Actor swipes camera view -right or left- to display next camera in carousel
2. System animates carousel and stops at next camera
3. System updates camera view