RWTH AACHEN UNIVERSITY

FabQR

Using QR Codes in the FabLab Workflow

Bachelor's Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Christian Frohn

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Univ.-Prof. Dipl.-Ing. M.Arch Peter Russell

Registration date: June 26th, 2015
Submission date: July 26th, 2015

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, July 2015
Christian Frohn

Contents

Abstract **xv**

Überblick **xvii**

Acknowledgements **xix**

Conventions **xxi**

1 Introduction **1**

1.1 FabLabs 1

1.2 Motivation 4

1.3 Concept 5

1.4 Thesis Overview 8

2 Related Work **9**

2.1 Anonymous File Sharing 10

2.1.1 Dead Drops 10

2.2 QR Codes 11

2.2.1 Structure of QR Codes 11
2.2.2 Practical Use 12
2.3 Combination of QR Codes and File Sharing . 13
2.4 Interactive Fabrication 14
2.5 Related Hardware and Software 15
2.5.1 Raspberry Pi 15
2.5.2 openFrameworks 15
2.5.3 OpenMAX 16
2.5.4 PHP QR Code 16
2.5.5 ZXing 16
2.5.6 PNG libraries 16
2.5.7 PHPMailer 16
2.5.8 VisiCut and VisiCam 17
2.6 Sharing of Project Documentations and Knowledge 19
2.6.1 FabML 20
2.6.2 Comparison of Platforms 20

3 Own Work **23**

3.1 Requirements 24
3.2 System Overview 25
3.3 visicamRPiGPU 28
3.3.1 OpenGL Perspective Correction . . . 29

3.3.2 OpenMAX Modules 31
3.3.3 visicamRPiGPU Arguments 35
3.3.4 GPU Memory Split 36
3.3.5 File Locking 36
3.3.6 Signal Handler 37
3.3.7 Settings Header File 37
3.4 VisiCam Modifications 38
3.4.1 visicamRPiGPU Integration 39
3.4.2 Other VisiCam Improvements 40
3.5 VisiCut Modifications 41
3.5.1 Automatic VisiCam Images 42
3.5.2 Webcam Support 42
3.5.3 QR Code Detection 43
3.5.4 File Management 50
3.5.5 Projector Support 50
3.5.6 FabQR Upload 52
3.5.7 GUI Changes 54
3.5.8 Concurrent List Access 54
3.6 QR Code Readability 55
3.6.1 Test Arrangement 57
3.6.2 Measurement Results 58
3.7 FabQR Web Service 61

3.7.1 Installation Scripts 63
3.7.2 APIs 64
3.7.3 Project IDs 67
3.7.4 Data Representation 67
3.7.5 URL Structure 68
3.7.6 Security 69
3.7.7 Projector Support 70

4 Evaluation **73**

4.1 System Usability Scale 73
4.2 Results 74
4.2.1 User Study 74
4.2.2 Requirements 78

5 Summary and Future Work **81**

5.1 Summary and Contributions 81
5.2 Future Work 82
5.2.1 Performance Improvements 82
5.2.2 Projector Setup 82
5.2.3 Stand-alone FabQR Client 83
5.2.4 FabQR Network 83
5.2.5 Website Features 83
5.2.6 Long-term Evaluation 84

A First System Draft and Concept **85**

B QR Code Measurements **87**

C User Study **91**

Bibliography **95**

Index **99**

List of Figures

1.1 FabLab workflows 7

1.2 Interactive fabrication with QR codes 7

2.1 Dead drop inside a wall 10

2.2 Examples of QR codes and QR code structure 11

2.3 Approaches for the fabrication of an object . 14

2.4 Preview window in VisiCut 17

2.5 Material choice and positioning in VisiCut . . 18

3.1 Original system overview 26

3.2 FabQR system overview 27

3.3 Perspective correction 30

3.4 Data processing in visicamRPiGPU 33

3.5 Detection for multiple QR codes 44

3.6 Detection for rotated QR codes 45

3.7 Preview for QR code imported objects 49

3.8 Preview image export in VisiCut 51

3.9 FabQR upload confirmation dialog 52
3.10 FabQR upload dialog 53
3.11 FabQR configuration dialog 54
3.12 QR code on a smartphone display 55
3.13 Loss of QR code quality in camera images . . 56
3.14 Scheme of the test arrangement 58
3.15 Webcam image of the test arrangement . . . 58
3.16 QR code readability: Brightness 59
3.17 QR code readability: Side length 60
3.18 Project documentation on the FabQR website 63

4.1 User study page 1 with data 75
4.2 User study page 2 with data 76

A.1 First system draft and concept 85

C.1 User study page 1 without data 93
C.2 User study page 2 without data 94

List of Tables

1.1 Countries and registered FabLabs 3

2.1 Comparison of different platforms 22

4.1 System Usability Scale results 74

B.1 QR code measurements: Printed 88

B.2 QR code measurements: Smartphone min. brightness 89

B.3 QR code measurements: Smartphone max. brightness 90

B.4 QR code measurements: Side length 90

C.1 User study data 92

C.2 User study personal data 92

Abstract

In the FabLab community the sharing of knowledge and documentation of projects play an important role for the development of new ideas and solutions. Although there are already several platforms for these purposes, many people still refuse to use them regularly. The FabLab community loses a lot of its potential because of this unpleasant circumstance.

This bachelor thesis is aimed at finding a solution for this problem by developing a new software system, which is based on the integration of QR codes in the FabLab workflow. In a first step the requirements for such a system are analyzed and defined.

The implementation of the software system considers these requirements in many different ways and it tries to fulfill the requirements as good as possible while paying attention to software and hardware constraints at the same time. A description of the implementation and its most important components and features is created as well.

In the evaluation of the software system a user study is conducted in order to examine whether the requirements were successfully met and to measure the usability of the system.

Finally a summary of all results and possible improvements for future developments are presented.

Überblick

In der FabLab Community spielen das Teilen von Wissen und die Dokumentation von Projekten eine wichtige Rolle für die Entwicklung von neuen Ideen und Lösungen. Obwohl es bereits einige Plattformen für diese Zwecke gibt, weigern sich viele Benutzer immer noch, diese regelmäßig zu verwenden. Die FabLab Community verliert durch diesen unangenehmen Umstand große Teile ihres Potenzials.

Ziel dieser Bachelor-Arbeit ist es, eine Lösung für dieses Problem durch die Entwicklung eines neuen Software-Systems zu finden, welches auf der Integration von QR-Codes in die Arbeitsabläufe in FabLabs basiert. In einem ersten Schritt werden zunächst die Anforderungen an ein solches System analysiert und definiert.

Die Implementierung des Software-Systems beachtet diese Anforderungen auf viele verschiedene Arten und Weisen und es versucht, die Anforderungen so gut wie möglich zu erfüllen und gleichzeitig dabei Beschränkungen durch Software und Hardware zu berücksichtigen. Eine Beschreibung der Implementierung und ihrer wichtigsten Komponenten und Funktionalitäten wird ebenfalls erstellt.

In der Evaluation des Software-Systems wird eine Nutzerstudie durchgeführt, um zu ermitteln, ob die Anforderungen erfolgreich erfüllt wurden, und die Benutzerfreundlichkeit des Systems zu messen.

Abschließend werden eine Zusammenfassung der Resultate und mögliche Verbesserungen für zukünftige Entwicklungen präsentiert.

Acknowledgements

Firstly I would like to thank my supervisor Dipl.-Inform. René Bohne for his ideas and valuable guidance. Of course I also want to thank Prof. Dr. Jan Borchers and Univ.-Prof. Dipl.-Ing. M.Arch Peter Russell for examining this bachelor thesis and giving me the possibility to write it at all.

Thanks to the whole chair for supporting me in writing this bachelor thesis in many different ways. Especially I am grateful for the help of the FabLab staff and the secretary staff. Thanks to Christoph Emonds for providing the web hosting for this bachelor thesis.

I want to thank my family and friends for their support as well.

Last but not least, I would also like to thank everyone, who participated in the user study for this bachelor thesis.

Conventions

Throughout this thesis the following conventions are used.

Text conventions

Definitions of technical terms or short excursus are set off in colored boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English.

Download links are set off in colored boxes.

File: myFile[a]

[a] http://hci.rwth-aachen.de/public/folder/file_number.file

Chapter 1

Introduction

Since this whole bachelor thesis deals with FabLabs, a general introduction to FabLabs is presented in the first part of this chapter. Afterwards the motivation for the FabQR system is given and the ideas and concepts behind the FabQR system are explained. This introductory chapter ends with a more detailed overview of the thesis.

1.1 FabLabs

As an important document the Fab Charter describes the different properties and characteristics of FabLabs. Most FabLabs see this document as a guideline, which mentions how a FabLab should be structured and what it should provide [The Fab Charter, 2012].

The Fab Charter also includes a definition for the term FabLab, which says:

FABLAB:
"Fab labs are a global network of local labs, enabling invention by providing access to tools for digital fabrication." [The Fab Charter, 2012]

Definition:
FabLab

FabLabs provide open access for everyone to tools, machines and knowledge for the digital fabrication of objects. In this context the digital fabrication describes that process in which digital schemes for a physical object are prepared with the help of computers and appropriate software. In a next step these schemes can be used to create the physical object by sending these schemes as input data to specialized machines and tools.

Usually these specialized machines are very expensive, which is the reason why sharing these machines and providing open access to them is in general beneficial for the users of FabLabs. Common examples for such machines are laser cutters, 3D printers and milling machines for PCBs (Printed Circuit Boards).

Sharing knowledge is important for the FabLab community.

Similar to the idea and movement of open source software, the users of FabLabs are encouraged to share their designs and schemes for physical objects freely with other users as open source hardware. Sharing knowledge and project documentations with other users in the FabLab community is that important that it is even mentioned as educational aspect in the Fab Charter [The Fab Charter, 2012]. Every user of a FabLab is supposed to share his knowledge and project documentations with other users of the FabLab community.

Mikhak et al. [2002] even describe the FabLab movement as a part of the digital revolution. After the phase of personal computation the FabLabs lead to the new phase of personal fabrication. This trend is compared with the historic development of computers and photography. In the beginning of photography only professional photographers had the possibility to take pictures. Similar to that, only very few people and organizations had access to large and expensive computers in the early stages of computers. In contrast to that, it is nowadays common for everyone to have access to photography and personal computers. The authors expect that FabLabs will promote a similar development for the field of personal fabrication.

The website fablabs.io [Fab Foundation, 2014] offers the service of a central register to FabLabs. FabLab managers can register their FabLab there and add some information such as available machines, contact information and public opening hours. For each FabLab the location is listed as well. Worldwide 547 FabLabs are currently registered in total. In table 1.1 some of the countries with the most registered FabLabs are shown.

Country	Registered FabLabs
United States of America	93
France	57
Italy	57
Germany	29
Netherlands	28
United Kingdom	23
Spain	20

Table 1.1: Countries and registered FabLabs
Source: fablabs.io [Fab Foundation, 2014]

It is noticeable that most of the registered FabLabs are located in the United States of America. There are two possible explanations for this. Firstly, the whole FabLab movement started in the United States of America and because of this it was likely to happen that many other FabLabs started there. Secondly, the territorial size of the countries plays a role in this context as well. Larger countries are likely to have more FabLabs in total.

Considering the fact that the FabLab movement started in 2001 [Mikhak et al., 2002] these numbers look very promising for the future of FabLabs and the concept of open access for everybody to tools, machines and knowledge for the digital fabrication of physical objects.

The amount of FabLabs grows remarkably for a relatively short period of time.

1.2 Motivation

Most FabLab users do not share knowledge and documentations of their projects.

Although there are a lot of FabLabs now, it is still noticeable that many FabLabs do not really actively participate in sharing their knowledge and project documentations with other FabLabs. Regarding this point, the network of FabLabs is currently in an unsatisfying situation.

Because of this unpleasant circumstance the network of FabLabs loses a lot of its potential. In many cases one FabLab can not find out, which new objects have been created at another FabLab and especially it is often unknown how they have been created. At the different FabLabs there is a lot of expertise, but this knowledge is almost never shared with other FabLab users.

There are many situations, in which the knowledge and expertise of other FabLab users is very helpful. For example, it is a common advice to merge identical or overlapping vectors of cutting schemes for the laser cutter. Often such graphic programs do not automatically merge these vectors, which means that the laser cutter tries to cut the same places multiple times. This does not only waste time, but it can also damage and break the final laser cut object.

Similar to that, there are some other special tips and tricks for other common machines in FabLabs as well. For the PCB milling machine it is usually useful to leave some small breakout tabs while milling the PCB because otherwise the PCB or even the milling machine might be damaged.

Not only the knowledge is unlikely to be shared, but also the documentation of projects is often not shared at all. This is the reason for the issue that many schemes for physical objects are created multiple times, although there might already exist another very similar scheme at another FabLab. Of course this leads to unnecessary inefficiency.

The aim of this bachelor thesis is to find a solution for these problems.

1.3 Concept

The key elements in the concept of the FabQR system are QR codes. Basically the idea is to use QR codes for the identification of projects and created physical objects in FabLabs.

With the help of QR codes the project data can easily be exchanged and there is the possibility to create and share a short documentation for each new project and physical object. In FabQR this concept is implemented for the use case of the laser cutter.

Unfortunately, sharing of knowledge and project documentations is currently not an essential part of the FabLab workflow. Most people just visit the FabLab, bring their prepared project data on a USB flash drive, create their project and leave the FabLab again. This workflow is completely contradictory to the guideline, which expects every user of a FabLab to actively share project documentations and knowledge.

It is a common misconception that shared knowledge and project documentations must not mention any difficulties of the creation process of a FabLab project. But in fact it is actually the other way around. Particularly such difficulties should be mentioned because other users in the FabLab community can learn a lot from the difficulties and especially the solutions for the difficulties of other FabLab users. In the context of FabLabs people are often able to learn from their own mistakes as well as the mistakes of other FabLab users.

FabQR changes the workflow in FabLabs and includes sharing of knowledge and documentations of projects.

With the FabQR system this workflow is supposed to be changed completely. Users of the FabLab upload their project data to a server of the FabLab over the Internet. As a response the users receive a QR code, which identifies the uploaded project data and is valid for a limited amount of time. The upload has the advantages that the users need to prepare their project data more carefully before visiting the FabLab and users can not forget their project data because it is already stored on the server of the FabLab.

The users can print that QR code directly or they can receive it as an email attachment. With the QR code the users visit the FabLab and show their QR code to a computer with a camera, which then loads the identified project data directly into the program.

After building the project, users are actively asked by the software to create a project documentation and share it together with the related knowledge on the websites of the FabLab. On the website the entered information of the users is shown and a new QR code is generated for permanent identification of this shared project.

The basic concept of FabQR can be applied to other machines in the FabLabs as well.

Although the implementation of FabQR mainly deals with the laser cutter as use case, the basic concept of identifying project data with QR codes to incorporate the sharing of knowledge and FabLab project documentations into the FabLab workflow can be applied to other tools and machines in FabLabs as well.

For the laser cutter the QR codes can additionally be used to create an interactive fabrication process as well. Users can place and rotate their QR codes right on top of the material, which they want to cut. With a camera the position and the rotation of the QR code are detected. The associated scheme for laser cutting is directly loaded at that position with the correct rotation in the laser cutter software. By physically manipulating the QR code in reality the users have the possibility to place and rotate their laser cutting schemes interactively in the virtual preview of the laser cutter program.

In the figure 1.1 the unmodified workflow is compared with the modified workflow of the FabQR concept.

Figure 1.2 shows the concept how QR codes can be used for an interactive fabrication process with the laser cutter.

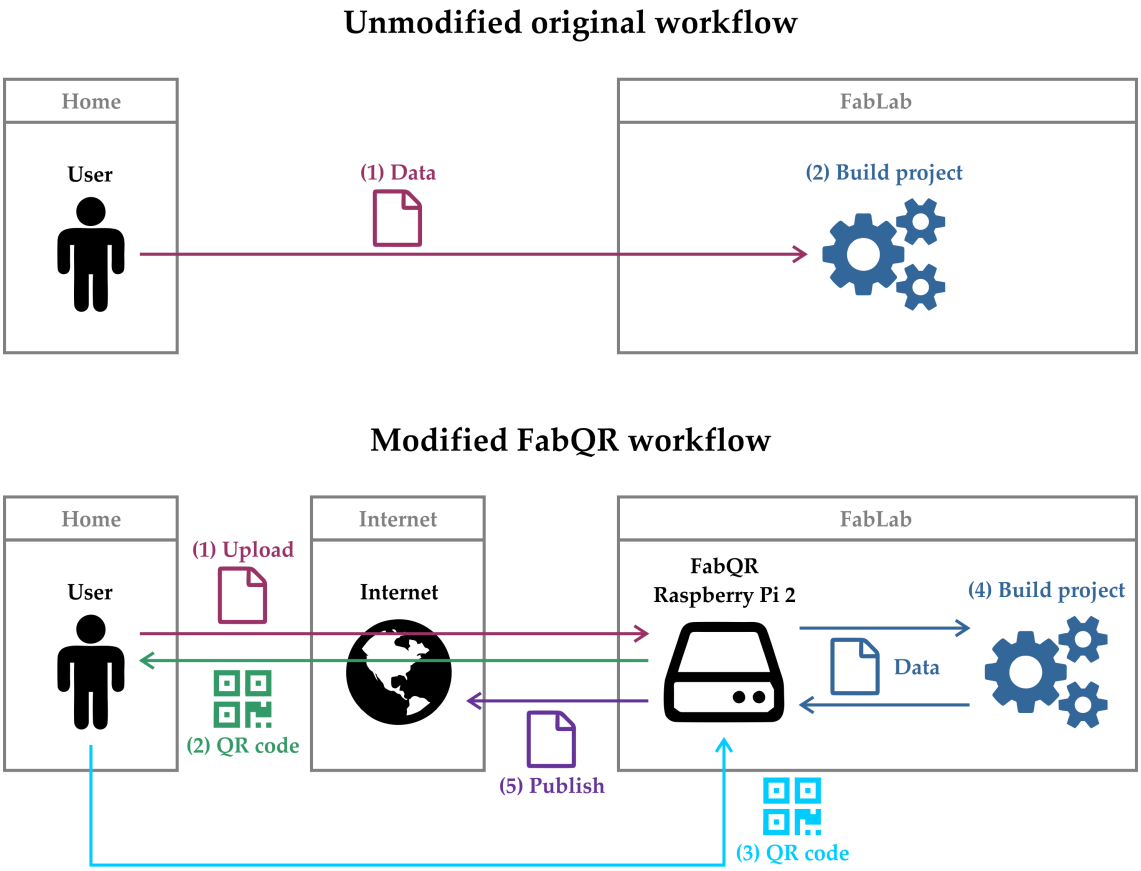


Figure 1.1: FabLab workflows
Includes icons from Font Awesome [Gandy, 2012]

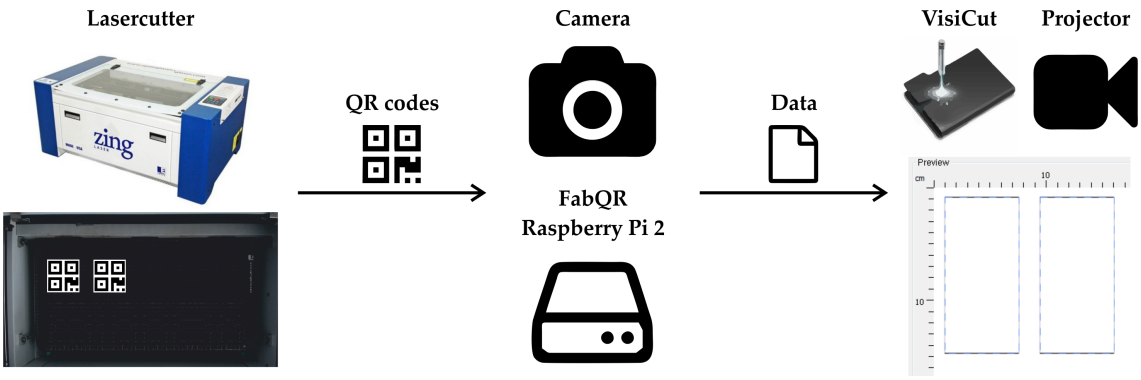


Figure 1.2: Interactive fabrication with QR codes
Includes icons from Font Awesome [Gandy, 2012], VisiCut [Oster, 2011]

1.4 Thesis Overview

A more detailed overview of the following remaining chapters of the bachelor thesis is given here.

- In chapter 2 "Related Work" different types of related work for this bachelor thesis are presented. Interesting characteristics of other systems, which are useful for the development of the FabQR system, are mentioned in this chapter. Important literature for the basic elements of the FabQR system is examined in this chapter as well. Many different libraries and software projects in different programming languages are used to create the FabQR system. The software projects, which are used for the FabQR system, are also listed in this chapter of the bachelor thesis.
- Chapter 3 "Own Work" begins with the requirements for the software system FabQR. Based on the different types of related work a combination of different characteristics and properties of other systems is chosen for the FabQR system. After the requirements the details of the implementation of the FabQR system are described and a documentation of the different changes and new software parts is created.
- In chapter 4 "Evaluation" the user study for the FabQR system is presented. The results of the questionnaire are analyzed and the answers of the participants are presented. After that it is checked, whether the requirements of the FabQR system are fulfilled by the implementation.
- Chapter 5 "Summary and Future Work" summarizes the results of the bachelor thesis. Additionally, it gives an outlook on possible future developments.

Chapter 2

Related Work

In this chapter the different types of related work for FabQR are presented. There are several topics, which are interesting in the context of FabQR. The first part of this chapter deals with anonymous file sharing. The theoretical backgrounds and the practical use of QR codes are explained as well. Since a combination of these aspects is used in FabQR, similar projects and concepts, which also use this combination, are mentioned. In FabQR the QR codes can be used to create an interactive fabrication process and therefore other methods for interactive fabrication are presented as well. For this FabQR system several hardware and software aspects are relevant. A FabQR integration for the laser cutter software projects VisiCut and VisiCam is developed in the context of this thesis. At the end of this chapter the sharing of project documentations and related knowledge in the FabLab community are addressed and several platforms for this purpose are compared with each other.

2.1 Anonymous File Sharing

This chapter deals with the project Dead Drops and explains anonymous file sharing.

2.1.1 Dead Drops

Dead Drops are USB flash drives in public spaces, which create an anonymous offline file sharing network.

The basic idea of Dead Drops [Bartholl, 2010] is to create a public accessible and offline file sharing network, which can be used anonymously by everybody. The network has these properties because it is established by incorporating USB flash drives permanently into walls and buildings in public spaces. Everyone is allowed to use these drives without any authentication. Users can copy and store files by connecting their own devices and they interact physically with the USB flash drives.

Figure 2.1: Dead drop inside a wall
Source: Dead Drops [Bartholl, 2010]

This project shows that file sharing is nowadays common in everyday life. The concept of using a service anonymously has the advantage that people can use the service quickly and easily. FabQR also considers the concept of physical interaction for a better user experience.

2.2 QR Codes

QR codes are an essential part of FabQR. The theoretical backgrounds of QR codes and the practical use of QR codes are explained in this chapter.

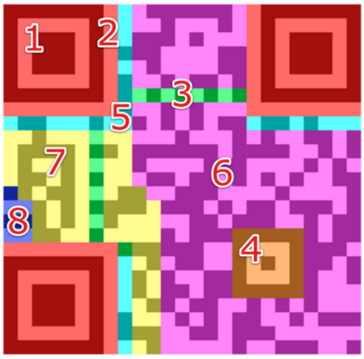
2.2.1 Structure of QR Codes

The special structure of QR codes combines several beneficial properties of matrix codes.

QR (Quick Response) codes were established by DENSO Wave Incorporated in 1994. In contrast to one-dimensional barcodes, QR codes use two dimensions for data representation. There are several other types of two-dimensional matrix codes, but only QR codes are able to combine all positive properties such as high machine readability, small physical size and the capability of storing large amounts of data [DENSO ADC, 2011].

QR Code Security

Sample QR Code



(1) Finder Pattern
(2) Separators
(3) Timing Pattern
(4) Alignment Pattern
(5) Format Information
(6) Data
(7) Error Correction
(8) Remainder Bits

Figure 2.2: Examples of QR codes and QR code structure
Source: Based on [Kieseberg et al., 2010]

Two examples of QR codes are depicted in figure 2.2. Although there is a lot of distortion in the right QR code due to the structural information, it is still readable, which is a big advantage of QR codes. As explained by Kieseberg et al. [2010], most structural elements of QR codes are supposed to ensure high readability and resistance against distortions. QR codes use error correction techniques in such a way that even very distorted QR codes can be readable. There are four levels of error correction: L (7%), M (15%), Q (25%) and H (30%). The percentage values indicate how much error correction information is added to the QR code and with a higher error correction level less original data can be stored in a QR code of the same size.

Because of all these positive aspects QR codes are useful in the context of FabQR and are better suited than other matrix codes.

2.2.2 Practical Use

Nowadays QR codes are commonly used for various purposes.

Today QR codes are common in many different areas. Often there are industrial applications, which track a product or an item with QR codes. Therefore, QR codes are frequently used in logistics, transportation and fabrication processes. With increasing numbers of smartphone users, QR codes have recently become popular in the branches of advertising and marketing as well [DENSO ADC, 2011].

QR codes identify FabLab users and FabLab project files in FabQR.

2.3 Combination of QR Codes and File Sharing

There are several services, which already use QR codes for file sharing and synchronization purposes. For instance, the programs Sync [BitTorrent, Inc., 2015] and SuperBeam [LiveQoS, 2014] use QR codes for an easy way of personal device synchronization. Instead of entering a relatively long file sharing key into a smartphone, a user can simply scan a QR code to establish the device synchronization with the smartphone. In that scenario QR codes are able to simplify the interaction with these software systems and to create a better experience for the user.

There are several services, which combine QR codes with file sharing.

In contrast to that, Kangee [Nobach, 2010] as well as TagMyDoc [Knova, 2015] are not focused on device synchronization, but they deal with the modification and exchange of files in the context of multiple users. TagMyDoc offers a version control system for documents, which can be edited by a specified group of people, and QR codes can be printed on a document to identify the latest version of a document. With Kangee a user can easily create a personal file exchange server and QR codes are generated for simple access to the files with a smartphone.

A similar idea is used in FabQR. QR codes refer to project files, which were uploaded to the FabQR server. As for the above mentioned software projects, the QR codes are supposed to create a better user experience by simplifying the process and by adding interactivity to it.

2.4 Interactive Fabrication

Different types of fabrication methods can be used for the creation of an object.

Willis et al. [2011] explain that there are different approaches for the fabrication of a physical object. Originally the creator of an object had to work on it physically with different tools. Nowadays the digital fabrication is widely spread and it changes the workflow of such an object creation process completely. The schemes for such objects are designed with a graphical user interface and the files are sent to a machine, which creates the physical object. The concept of interactive fabrication suggests to combine some aspects of these two different methods to enhance the overall user experience. These different approaches are shown in figure 2.3.

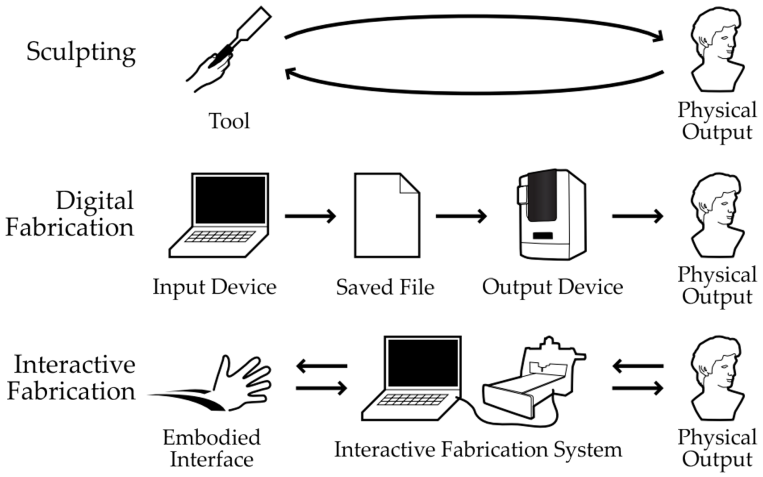


Figure 2.3: Approaches for the fabrication of an object
Source: Based on [Willis et al., 2011]

Constructable is a system for interactive fabrication with laser cutters, which is presented by Mueller et al. [2012]. With a laser pointer users can sketch the paths for the laser cutter directly on the material. A camera observes this procedure and optimized paths for the actual laser cutting are computed.

Since the QR codes in FabQR identify project files, the QR codes can be used together with a laser cutter software and a camera to create an interactive fabrication process.

2.5 Related Hardware and Software

There are different types of related hardware and software, which are presented here.

2.5.1 Raspberry Pi

The Raspberry Pi is very flexible and suitable for many different projects.

The Raspberry Pi [Raspberry Pi Foundation, 2008] is a small computer in the size of a credit-card, which features a low price, high portability and a large variety regarding connectivity and programmability. It is comparable to embedded systems and the different versions of the Raspberry Pi define the hardware specifications clearly. The Raspberry Pi was designed in such a way that it can be used in many different situations and projects. The operating system Raspbian [Raspberry Pi Foundation, 2012] is an operating system, which is specifically developed for the Raspberry Pi platform. Raspbian is based on the operating system Debian [Software in the Public Interest, Inc., 1993]. These operating systems use software packages to install new services and programs easily on the computer.

Because of these reasons a Raspberry Pi 2 with the Raspbian operating system is used for the development of the FabQR system. In the context of FabQR the well-known software packages Apache HTTP Server [Apache Software Foundation, 1995] and PHP [PHP Group, 1995] are used to create the web service of the FabQR system.

2.5.2 openFrameworks

openFrameworks [openFrameworks Community, 2004] is a software project, which combines several software projects to create a powerful toolkit for the development of new applications. It allows to use OpenGL on the graphics processing unit. It is written in C++ and supports a variety of computations in subject areas such as graphics and computer vision. openFrameworks is well-suited for FabQR because some graphics operations need to be performed.

2.5.3 OpenMAX

OpenMAX [Khronos Group, 2004] is a specification for accelerated multimedia processing. The Raspberry Pi 2 supports this specification and in FabQR hardware accelerated processing is useful because it helps to create a better user experience with a high software performance.

2.5.4 PHP QR Code

The software project PHP QR Code [Dzienia, 2010] is able to generate QR codes for arbitrary input texts. This project is a part of the implementation of FabQR.

2.5.5 ZXing

ZXing is a library, which is often used to read QR codes.

ZXing [ZXing authors, 2007] is a library for barcode and matrix code detection and decoding in images. It supports several programming languages and can read rotated and distorted QR codes. In the context of FabQR the processing of multiple QR codes in a single image is an important feature. ZXing 3.2.1 is used in FabQR.

2.5.6 PNG libraries

PNG (Portable Network Graphics) images are used in FabQR. LodePNG [Lode Vandevenne, 2005] and PngEncoder [ObjectPlanet, Inc., 1998] are used to encode and decode images in the PNG file format quickly.

2.5.7 PHPMailer

In FabQR the software project PHPMailer [Matzelle et al., 2001] is used to send emails.

2.5.8 VisiCut and VisiCam

VisiCut is a user-friendly program for laser cutting.

The laser cutter software project VisiCut [Oster, 2011] simplifies the task of laser cutting a lot. This project is developed in the programming language Java, which is the reason why it is mostly platform independent. One of the design goals of VisiCut is a very high usability. Users interact with a graphical user interface to load files for laser cutting and to specify positions, sizes and rotations for the different shapes. A variety of file formats such as SVG, EPS and DXF can be imported in VisiCut and it defines a customized file format, which is called VisiCut PLF (Portable Laser Format). Furthermore, VisiCut provides a selection of materials with thickness values and users can cut as well as engrave materials with VisiCut. Different types of laser cutters are supported by VisiCut because it includes a library, which is able to transform general and abstract laser cutting commands into specific instructions for several laser cutters.

In the figures 2.4 and 2.5 screenshots of VisiCut are shown. A user can simply modify the position of an object by moving the object in the preview window with the mouse or precise values for the position of an object can be entered.

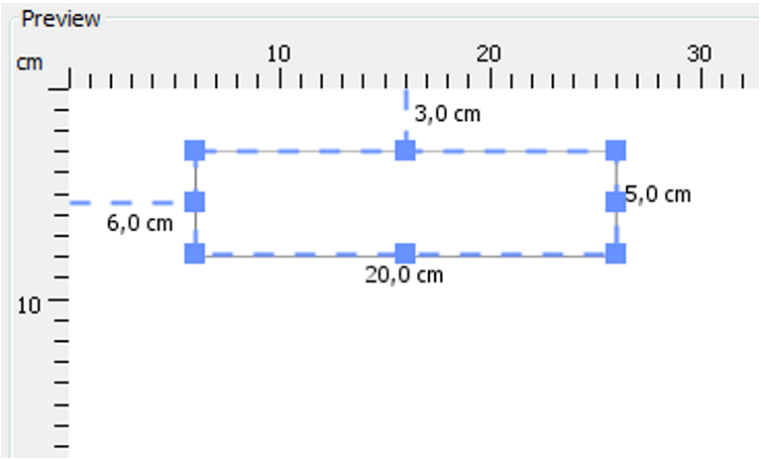


Figure 2.4: Preview window in VisiCut
Source: VisiCut [Oster, 2011]

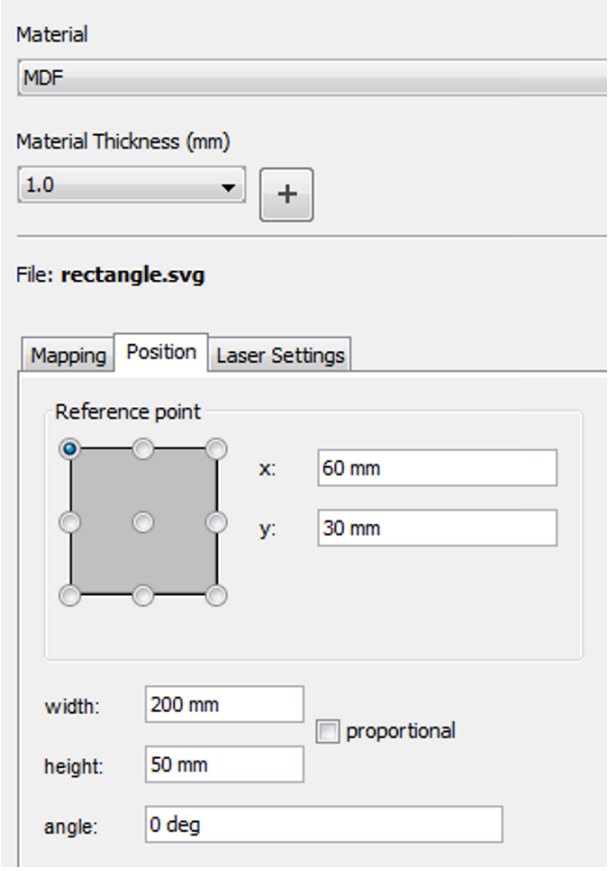


Figure 2.5: Material choice and positioning in VisiCut
Source: VisiCut [Oster, 2011]

A camera image for VisiCut can be provided by VisiCam.

VisiCam [Oster, 2013] is a software project, which extends and interacts with VisiCut. VisiCam basically adds the possibility to see a camera image of the laser cutter in VisiCut. A camera is mounted right above the laser cutter and provides an image of the cutting area inside the laser cutter. The camera can be accessed over the local area network. A perspective correction, which is based on the detection of specific markers, ensures that the camera image depicts the cutting area of the laser cutter without any camera perspective related distortions.

Since VisiCut and VisiCam are user-friendly and already include a camera, a FabQR integration for VisiCut is developed. Users can work with the laser cutter software by physically interacting with the QR codes of FabQR.

2.6 Sharing of Project Documentations and Knowledge

Sharing project documentations and knowledge in the FabLab community is still a problem.

As described by Määttä and Troxler [2011], the international network of FabLabs is also supposed to serve as learning environment for all participants of the network. FabLabs are based upon the concept of open source. This includes free and open access for everyone to specialized machines for the digital fabrication. Another part of this concept is that knowledge regarding the areas of design and fabrication should be exchanged between all users of FabLabs. According to the authors, it is comparatively simple to share such knowledge locally regarding time and space. But there are still a lot of difficulties in the sharing of knowledge for longer periods of time or across larger distances between the participants. Current systems or platforms for sharing such knowledge do not fit well to the demands of FabLabs. Especially with a growing number of FabLabs this issue becomes more important.

Wolf et al. [2014] mostly agree with the statements of Määttä and Troxler [2011] and in order to support this they conducted a study, in which 16 active participants of the FabLab network were interviewed. In this study the participants had to answer several questions regarding their behavior in the context of digital fabrication and sharing of knowledge and project documentations. Different types of FabLab users were included in this study such as FabLab managers, PhD students or designers. Most interviewees perceive FabLabs as locations of creativity and learning and see beneficial aspects of open access to tools, machines and knowledge. The study shows that there is a positive opinion about the sharing of knowledge and project documentations because it helps to prevent mistakes and it can improve own projects of the users. Some participants of the study also mentioned that it is a hard task to create a documentation of a project in such a way that other users are able to understand the project in all its details. The creation of a documentation is seen as an additional step after the fabrication of a project and not as a part of the fabrication process itself. Documenting a project is described as boring and time-consuming work.

The idea of FabQR was created to deal with these issues, which is the reason why these aspects are considered in the development of FabQR.

2.6.1 FabML

FabLab project data can be versatile and a flexible scheme for the exchange of data should be used.

Troxler and Zijp [2013] mention the technical details of data exchange between several participants of the FabLab network. Each FabLab is organized and structured differently, which is the reason why the exchange of data between a large variety of FabLabs is difficult. In order to deal with this issue, the concept of FabML recommends to define a meta language for a more detailed description of FabLab projects. There are several examples for metadata in this context such as the name of the author of a FabLab project, needed machines for the project and the materials, that are used in the fabrication of the project. As a common meta language the concept of FabML helps to simplify the exchange of data between different participating FabLabs.

In FabQR such metadata is collected and provided for data exchange as well. In order to be very flexible for various systems and FabLabs, the data is stored in a format, which is compatible to the well-known and standardized XML (Extensible Markup Language). There are a lot of systems, which are able to deal with input data in that format.

2.6.2 Comparison of Platforms

Several online platforms, which are currently used for sharing knowledge and project documentations in the FabLab community, are compared with each other in table 2.1. The most notable disadvantages of these systems are explained in the next paragraphs. As a contrast to these systems FabQR is included in the table as well.

There are several problems with big platforms for sharing FabLab project data.

Often such platforms are administered and operated by commercial companies. As soon as FabLabs start to use such platforms regularly for the sharing of knowledge and project documentations those FabLabs are in fact dependent on the platform and the company. Especially for commercial companies such dependencies are risky because the companies might suddenly shut down the platform, stop the development or change their platform to a commercial product because of economical reasons. There are platforms, which enable some additional features of their system for paid pro accounts. The companies have full control of the project data and knowledge and all these points do not fit well to the concept of open source in the FabLab community [Määttä and Troxler, 2011][Hemig, 2013][Troxler and Zijp, 2013]. The first column of table 2.1 indicates whether the platform has any kind of commercial background.

Many times the software of such platforms is not distributed as open source. Therefore, there are no opportunities to customize such platforms in any way, which is the reason why the platforms usually do not match to the requirements of the FabLabs [Määttä and Troxler, 2011]. A user study showed that FabLab users would prefer to share their projects and knowledge on a platform, which is provided by the FabLab itself [Hemig, 2013]. This aspect is shown in the second column of table 2.1.

An API (Application Programming Interface) adds flexibility to the platforms because a variety of systems can interact with the platforms by using such a provided API. This is the reason why a powerful API is a quality characteristic of these platforms and in table 2.1 this is indicated by the last column.

Platform	Non-commercial	Open Source	Official API
Thingiverse[1]	✗	✗	✓
Instructables[2]	✗	✗	✗
YouMagine[3]	✗	✗	✓
FabQR	✓	✓	✓

Table 2.1: Comparison of different platforms

All big platforms for sharing FabLab project data have a commercial background.

The first three platforms have a commercial background. Thingiverse is operated by the company MakerBot® Industries, LLC[4], which creates and sells 3D printers. Instructables offers paid pro accounts, which enable additional features in the platform. YouMagine is connected to the company Ultimaker B.V.[5], which sells 3D printers as well. Currently these platforms provide basic features for the sharing of knowledge and project documentations for free, but this might change quickly in the future because of the commercial interests. FabQR does not have any kind of commercial background at all.

None of the currently used platforms for the sharing of knowledge and projects publishes the software for the platform as open source. FabQR is distributed as open source and the software system can be customized.

Thingiverse and YouMagine provide an official API. In FabQR an API is used for flexibility as well.

Based on these different types of related work there are already some requirements for FabQR, which are explained in more detail in chapter 3 ”Own Work”.

[1] http://www.thingiverse.com/
[2] http://www.instructables.com/
[3] https://www.youmagine.com/
[4] http://www.makerbot.com/
[5] https://ultimaker.com/

Chapter 3

Own Work

In this chapter the own work for the FabQR software system is described. The implementation is supposed to fulfill some requirements, which are defined in the first part of this chapter. Afterwards a technical system overview is given, that shows how the different software projects interact with each other and how data is exchanged. The new program visicamRPiGPU is explained as an extension to VisiCam [Oster, 2013]. Of course some modifications to VisiCam [Oster, 2013] and VisiCut [Oster, 2011] need to be implemented to support the concept of the FabQR software system. As a preparation for the FabQR software system, several QR code measurements are taken to find appropriate parameters for the generation of well readable QR codes. Finally, the details of the implementation of the new FabQR system are described. All in all these explanations and descriptions are the documentation of the FabQR software system as well.

3.1 Requirements

The FabQR system is supposed to fulfill some requirements.

The implementation of the FabQR software system is supposed to fulfill the following requirements. Some of these characteristics and properties are already described in the different types of related work for the FabQR system. In the evaluation of the system it is checked, whether these requirements are actually met.

- **Requirement R1: Usability**
  A high usability is an essential requirement for each software system. Since the FabQR software system is integrated into an already existing software system and environment, it needs to be easy to use and learn for those users, who are already used to the current existing software system.
- **Requirement R2: High performance**
  The FabQR system needs to have a high performance in all parts of the system. Users usually do not want to use a software system, which needs a very long time to process user inputs. Especially for the interactive parts of the system, users expect to receive feedback of the program immediately.
- **Requirement R3: Incorporation of QR codes**
  QR codes are incorporated in the workflow of the software system of FabQR to simplify the identification and exchange of project data.
- **Requirement R4: Anonymous service usage**
  Users do not want to register with another website. The system must be usable without any manual authentication or registration of users.
- **Requirement R5: Flexibility: Data representation**
  All the data of the FabQR software system is stored in a data format, which is compatible to XML for flexibility purposes. XML data can be easily read by many different types of software implementations and programs.

- **Requirement R6: Flexibility: API**
  In order to provide flexibility for the client systems of a FabQR software system, an official API is implemented to provide access to the FabQR software system for a variety of client systems.
- **Requirement R7: System customization**
  Every part of the system needs to be customizable. This does not only include the possibility to setup and configure the system easily, but also involves the opportunity to change nearly every part of the software system.
- **Requirement R8: Raspberry Pi 2**
  The FabQR software system is supposed to run on a Raspberry Pi 2 to make use of those advantages, which were already mentioned earlier for the Raspberry Pi 2 platform.

3.2 System Overview

Multiple programs need to work together for FabQR.

The FabQR software system consists of multiple separate software projects, which need to work together to accomplish all tasks and data needs to be exchanged between the different programs for this purpose. For all parts of the implementation further details are explained in the rest of this chapter.

Here a rough overview of the implementation of the FabQR software system is presented. There are different tasks, which need to be fulfilled because the whole system would not work correctly otherwise.

First of all, the system needs to get an image of the camera, which is connected to the Raspberry Pi 2. A marker detection and perspective correction are applied to the image to provide a perfectly aligned image.

The image must be accessible in the local area network.

Other computers in the local network need to be able to access this image. The Raspberry Pi 2 provides this image on a web service, which can be accessed from the local network.

The image is scanned for QR codes and it is checked, whether any QR code contains a valid URL (Uniform Resource Locator) to a laser cutter scheme. All valid laser cutter schemes are downloaded and shown in a preview.

After the object has been laser cut a project documentation can be created and uploaded to the web service, which stores and displays it on a website. The website generates QR codes for these new uploaded and published projects as well.

In figure 3.1 an overview of the current original system is provided. In contrast to that, the modified overview of the system with the FabQR software system is shown in figure 3.2.

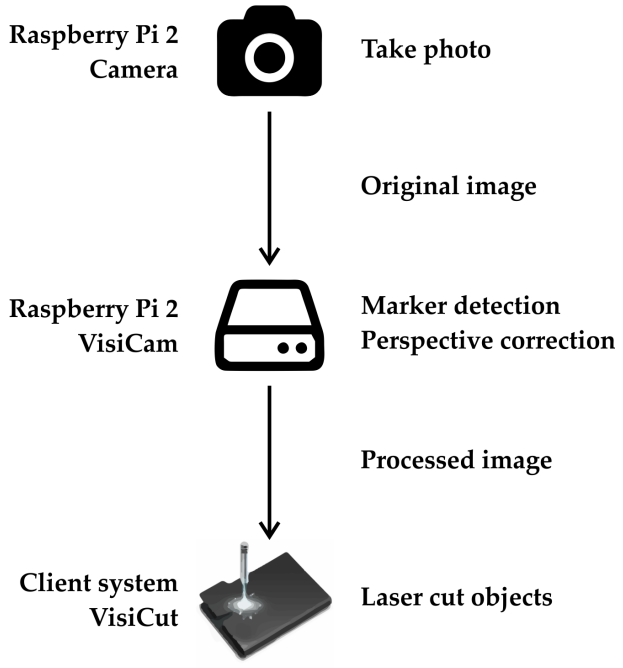


Figure 3.1: Original system overview
Includes icons from Font Awesome [Gandy, 2012], VisiCut [Oster, 2011]

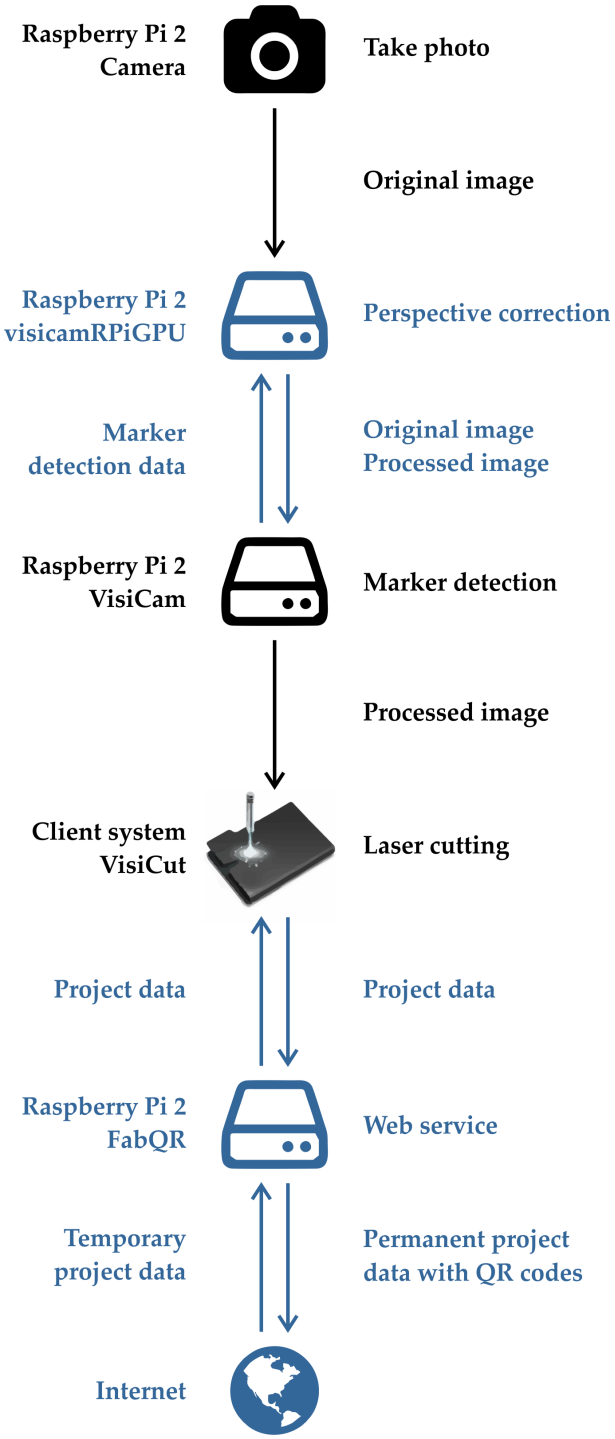


Figure 3.2: FabQR system overview
Includes icons from Font Awesome [Gandy, 2012], VisiCut [Oster, 2011]

3.3 visicamRPiGPU

The main task of the new program visicamRPiGPU is to provide hardware accelerated graphics operations for VisiCam [Oster, 2013] by using the GPU (Graphics Processing Unit) of the Raspberry Pi 2. Since the FabQR system is supposed to process the user input with the QR codes immediately, a very high performance is needed for this purpose.

CPUs are more versatile than GPUs, but they are usually slower. GPUs are specialized for fast graphics operations.

Currently these graphics operations are performed by VisiCam on the CPU (Central Processing Unit) of the Raspberry Pi 2. Although the CPU of the Raspberry Pi 2 provides some computational power, it is still way too slow to process such graphics operations immediately. For a single image with a resolution of 1920 x 1080 pixels this solution needs about 8 seconds of processing time, which is completely useless in the context of the FabQR software system. This is the reason why visicamRPiGPU is needed for the concept of FabQR.

In this chapter it is explained how the graphics operations work and why they are needed at all. The perspective correction is provided by OpenGL computations on the GPU of the Raspberry Pi 2 and the software project openFrameworks [openFrameworks Community, 2004] is used to get the possibility to perform computations on the GPU.

The Raspberry Pi 2 has support for the hardware acceleration specification OpenMAX [Khronos Group, 2004]. With OpenMAX the camera, which is connected to the Raspberry Pi 2, is controlled and the images are processed.

A documentation of the different parameters and configuration options for visicamRPiGPU is given as well.

visicamRPiGPU[a]

[a] https://github.com/FroChr123/visicamRPiGPU

3.3.1 OpenGL Perspective Correction

In VisiCam the perspective correction feature is implemented because it is very difficult to position the camera above the laser cutter perfectly. There are always rotations and slightly incorrect positions for the camera. This causes unpleasant distortions in the images, which are captured by the camera. Of course such distortions are unwanted because the camera is supposed to capture images, which are perfectly aligned to the edges of the laser cutter and show the flat surface of the laser cutting area.

In order to solve this issue a marker detection and a perspective correction are used. For the marker detection four markers are placed perfectly in the corners of the laser cutting area. Based on the positions of these four markers in the original and unmodified camera image a so-called homography matrix is computed in VisiCam.

The homography matrix describes mathematically how the pixels of the original image need to be modified to move the four markers into the corners of the image. In the processing of the image the homography matrix is applied to the original image to get rid of all rotations and distortions in the original image.

VisiCam and visicamRPiGPU need to provide data for each other.

For visicamRPiGPU the homography matrix values are provided by VisiCam. The Raspberry Pi 2 supports the specification of OpenGL ES 2.0 (Open Graphics Library for Embedded Systems), which is used to apply a high-performance matrix multiplication of the homography matrix with the original image data on the GPU.

The procedure for the marker detection in VisiCam provides a 3 x 3 homography matrix as result. In OpenGL all computations are based on 4 x 4 matrices with a different order of the values. This is the reason why the input homography matrix needs to be transposed and a new empty row and a new empty column need to be inserted before the homography matrix multiplication can be processed in OpenGL.

The following scheme shows how the input matrix needs to be modified to get the correct resulting matrix for the OpenGL computations:

$$\text{Input} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad \Rightarrow \quad \text{Result} = \begin{pmatrix} a & d & 0 & g \\ b & e & 0 & h \\ 0 & 0 & 0 & 0 \\ c & f & 0 & i \end{pmatrix}$$

In figure 3.3 a visual example for the marker detection and the perspective correction is given.

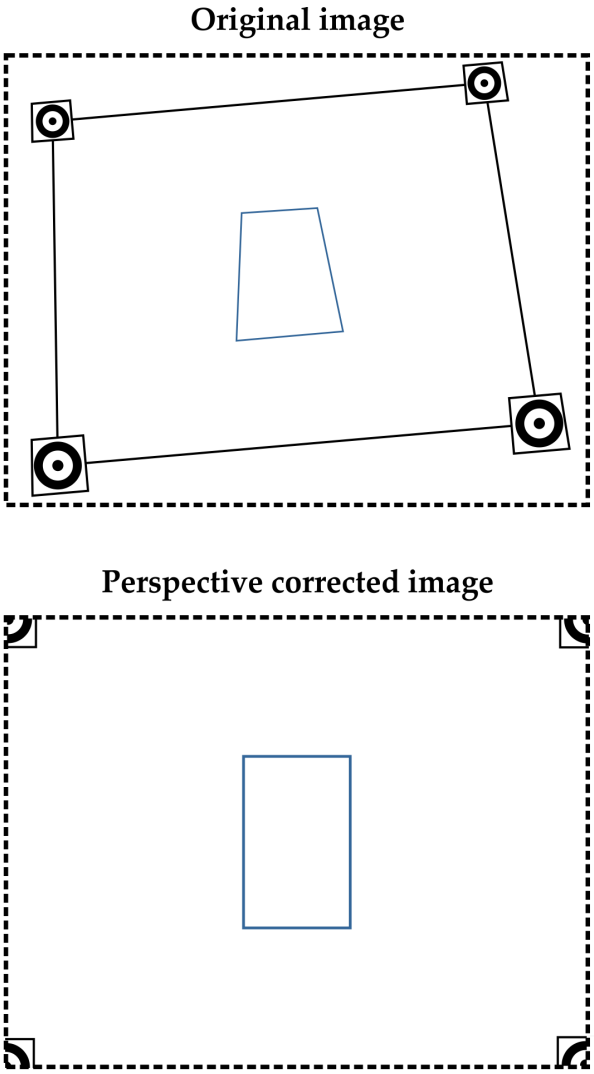


Figure 3.3: Perspective correction

3.3.2 OpenMAX Modules

In visicamRPiGPU OpenMAX [Khronos Group, 2004] is used to handle the camera controls and it provides hardware accelerated image processing. In the implementation and source code OpenMAX is often abbreviated as OMX. Although the OpenMAX specification is very detailed and consists of multiple documents with several hundred pages each, it is difficult to actually use the OpenMAX hardware acceleration in a new program on the Raspberry Pi 2.

The implementation of OpenMAX on the Raspberry Pi 2 does not correspond well with the OpenMAX specification.

The main reason for this circumstance is the fact that the actual implementation of the OpenMAX specification on the Raspberry Pi 2 is incomplete, inconsistent and it does not follow all specifications of OpenMAX. Additionally, there are vendor specific modifications in the implementation, which are not specified at all. Especially for these vendor specific modifications it needs to be guessed how the implementation actually works because there are no descriptive documents at all.

This issue is noticeable for the different image color formats, which can be used in OpenMAX. In the specification of OpenMAX a lot of different color formats are listed, but in the implementation of OpenMAX on the Raspberry Pi 2 only some of them are actually supported and correctly implemented.

The best color format for visicamRPiGPU would be RGB (Red, Green, Blue) with 8 bits for each color. This color format is specified in OpenMAX as `OMX_COLOR_Format24bitRGB888`, but as soon as this color format is used, the OpenMAX implementation returns an error because the implementation of OpenMAX on the Raspberry Pi 2 does not support that color format.

Instead, the vendor specific and unspecified color format `OMX_COLOR_Format32bitABGR8888` has to be used to support 8 bits for each color. Although the name of this color format mentions the order ABGR (Alpha, Blue, Green, Red), it actually behaves like RGBA (Red, Green, Blue, Alpha), which is a completely inconsistent behavior of this OpenMAX implementation. Therefore the image data is copied from the GPU in the RGBA color format with 8 bits for each color and it is directly passed on to OpenMAX. In this color format the alpha channel, which is basically used to describe the opacity, is not useful for visicamRPiGPU at all and it wastes resources of the system because there is no opacity in these images.

Different components are used to process data in OpenMAX.

OpenMAX uses so-called components to process data. Each component can have multiple input and output ports for data and every component has a state such as loaded, paused or executing.

Basically, there are two methods of moving data between components in OpenMAX. With tunneling a connection between an output port of one component and an input port of another component is created. As soon as there is new processed data available at the output port of the component it is immediately passed on to the input port of the next component.

In contrast to that, the data must be manually moved between OpenMAX components for non-tunneled component communication. In this case OpenMAX uses callback functions to provide information about the components for the other parts of the program. For example, a callback function is called as soon as a component has finished processing its input data.

All components of OpenMAX are identified by implementation specific component names. In order to utilize a component it has to be initialized by using the corresponding component name. After the initialization all ports of the component need to be disabled. Now component specific commands can be used to set parameters for the component. In the last step the tunnels, ports and states are configured correctly.

In the implementation of visicamRPiGPU all these abstract concepts of OpenMAX are needed. An overview of the OpenMAX components and the component communication in visicamRPiGPU is presented in figure 3.4.

visicamRPiGPU uses many OpenMAX features.

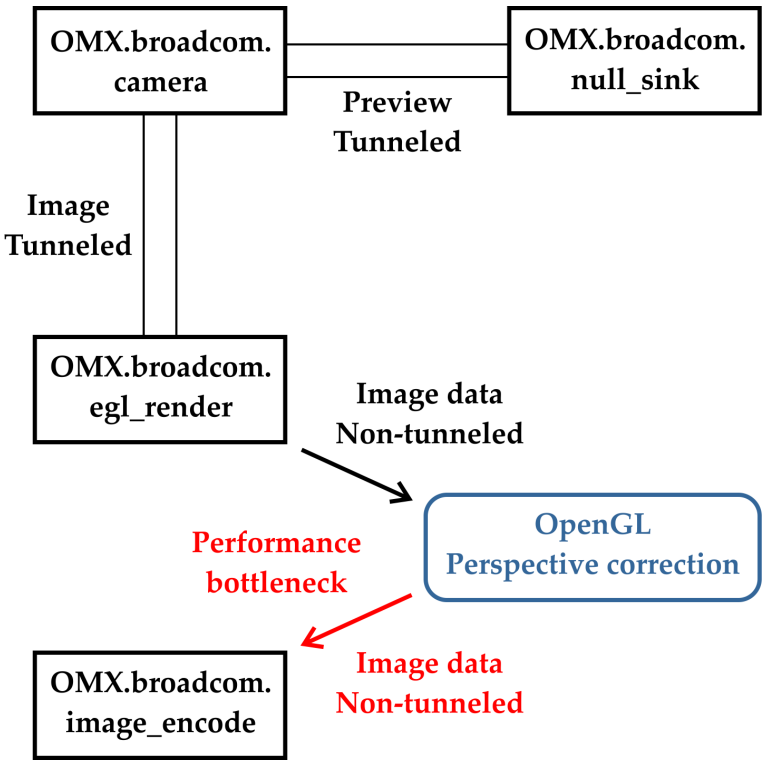


Figure 3.4: Data processing in visicamRPiGPU

This diagram for the data processing in visicamRPiGPU shows some additional aspects of visicamRPiGPU.

It is important to notice that the preview port of the camera component is enabled. The camera itself needs the data of the preview port for automatic image capturing adjustments such as brightness and color corrections. Since the actual data of the preview port is not needed at all, a tunneled communication is used to send it to the component `OMX.broadcom.null_sink`, which just ignores all incoming input data.

The component `OMX.broadcom.egl_render` renders the input image data directly on the GPU of the Raspberry Pi 2. Because of this circumstance the image is directly available for further processing in OpenGL.

In visicamRPiGPU the CPU is still the performance bottleneck.

Although visicamRPiGPU uses GPU computations and the hardware acceleration implementation of the OpenMAX specification on the Raspberry Pi 2 for huge performance improvements, there is still a performance bottleneck in the program. After the image was processed on the GPU it needs to be passed on to the input port of the component `OMX.broadcom.image_encode`. Unfortunately the CPU of the Raspberry Pi 2 is involved in this procedure because the data basically needs to be copied from the GPU memory to the input memory buffer of the OpenMAX component.

Since the CPU of the Raspberry Pi 2 is not very fast, this procedure is comparatively slow. In time measurements this step needs roughly 50 ms for an image resolution of 1280 x 720 pixels and 100 ms are required for a resolution of 1920 x 1072 pixels. In comparison to the original timings of roughly 8 seconds and performance of VisiCam [Oster, 2013] these timings of visicamRPiGPU are still a huge improvement and these values are useful for the FabQR software system.

The component `OMX.broadcom.image_encode` is finally used to convert the raw image data into a compressed JPEG image file. This compression is needed because the raw image data could not be transferred quickly over the network otherwise. The Raspberry Pi 2 only supports a maximum data rate of 100 Mbit/s for the network connection.

3.3.3 visicamRPiGPU Arguments

visicamRPiGPU needs seven input arguments to start correctly. These arguments are explained in the following list.

- **Argument 1: Width in pixel**
  The width argument must be in the range of 640 to 1920 because of camera limitations. For the OpenMAX implementation this value must be a multiple of 32. For the resolution an aspect ratio of 16:9 and a width of 1280 pixels are recommended.
- **Argument 2: Height in pixel**
  The height argument must be in the range of 480 to 1080 because of camera limitations. For the OpenMAX implementation this value must be a multiple of 16. For the resolution an aspect ratio of 16:9 and a height of 720 pixels are recommended.
- **Argument 3: Refresh time in seconds**
  This argument must be a value greater than zero. It defines the refresh time interval. On each refresh an original image is saved instead of a processed image and the values of the input homography matrix are updated. The validity of the process ID of the starting process is checked as well.
- **Argument 4: Process ID of starting process**
  If this argument is not equal to zero and there is no process running with this process ID on the system, visicamRPiGPU will terminate itself. This is used to stop visicamRPiGPU correctly as soon as VisiCam [Oster, 2013] is stopped.
- **Argument 5: File path to homography matrix input**
  This argument specifies the file path to the homography matrix input. It is recommended to use an absolute file path.

- **Argument 6: File path to processed image output**
  This argument specifies the file path to the processed image output. It is recommended to use an absolute file path. Since this file is written very frequently, this file path should refer to a file, which is stored in the memory of the Raspberry Pi 2. Usually this shared memory is already configured for the path `/run/shm/` on the Raspberry Pi 2.
- **Argument 7: File path to original image output**
  This argument specifies the file path to the original image output. It is recommended to use an absolute file path.

3.3.4 GPU Memory Split

The GPU memory needs to be configured for visicamRPiGPU.

For visicamRPiGPU some memory of the GPU is required. The highest possible resolution of 1920 x 1072 pixels requires approximately 140 MB of GPU memory. For the recommended setting of 1280 x 720 pixels a GPU memory value of 128 MB is sufficient. To set this value on a Raspberry Pi 2 architecture with the Raspbian [Raspberry Pi Foundation, 2012] operating system the command `raspi-config` can be utilized as root user.

3.3.5 File Locking

All files, which are read and written by visicamRPiGPU, are used by other processes as well. To avoid the situation that any file is read and written at the same time by multiple processes, file locking mechanisms are applied. Without file locking such situations are likely to lead to critical and hidden errors in the different programs. The programs always use blocking wait to access the files correctly.

The most important thing about file locking in visicamRPiGPU and VisiCam [Oster, 2013] is the circumstance that the file locks need to be recognized correctly by C++ and Java. The function `flock` is not able to fulfill this requirement. Therefore, the function `lockf` needs to be used because it is correctly recognized by the default Java file locking mechanisms.

3.3.6 Signal Handler

In Unix operating systems signals are used to establish a very simple type of inter-process communication. For example, there are signals to pause, continue or stop the execution of a program. In openFrameworks [openFrameworks Community, 2004] a default signal handler is used, which tries to cleanup everything correctly before the execution of the application is actually stopped.

The default signal handler of openFrameworks does not work correctly for visicamRPiGPU.

For visicamRPiGPU this cleanup routine does not always work correctly and the program freezes. In this case it can only be terminated with the signal `SIGKILL`. To solve this issue the default signal handler is overwritten in visicamRPiGPU and all signals, which ask the program to stop, are simply replaced by a `SIGKILL` signal. With the `SIGKILL` signal the correct termination of visicamRPiGPU is ensured in this context.

3.3.7 Settings Header File

There is a special settings header file in visicamRPiGPU, which is called `visicamRPiGPU-settings.h`. In this file several settings for visicamRPiGPU can be configured. These settings are basically the parameters, which are sent to the OpenMAX components `OMX.broadcom.camera` and `OMX.broadcom.image_encode`. For the camera component there are settings such as brightness, contrast and exposure compensation. For bright environments a negative value for the exposure compensation setting is usually useful. For the created JPEG files the quality can be configured in this file as well.

3.4 VisiCam Modifications

The main task of VisiCam [Oster, 2013] is to provide perspective corrected images of the laser cutter area for other computers in the local area network. As already explained earlier, the original and unmodified version of VisiCam performs the marker detection and the perspective correction on the CPU of the Raspberry Pi 2, which is the reason why the whole procedure takes a lot of time.

VisiCam provides a web service for the processed image and configuration of the system.

VisiCam runs as a service on the Raspberry Pi 2 and it provides a web service on a configurable port. A log file for VisiCam is created as well. With the web service not only the processed image can be accessed from the local area network, but it also provides a website for the configuration of VisiCam. With this website several options for VisiCam such as the resolution, file paths and marker detection settings can be configured.

For each marker an area in the original image needs to be specified, in which the marker detection procedure tries to find the corresponding markers. With these areas it is also defined, which marker belongs to which corner of the image.

Since the native perspective correction of VisiCam is way too slow for the FabQR software system, the perspective correction needs to be outsourced from VisiCam. For this purpose visicamRPiGPU needs to be integrated into VisiCam and both programs need to work together to solve this task.

The modifications of VisiCam are not only limited to the integration of visicamRPiGPU, but there are also other improvements for VisiCam, which are introduced with the modifications of VisiCam.

VisiCam fork[a]

[a] https://github.com/FroChr123/VisiCam

3.4.1 visicamRPiGPU Integration

VisiCam is modified in such a way that is has support for visicamRPiGPU. In the combination of these two software projects VisiCam has control over the visicamRPiGPU application. This means that VisiCam starts and stops visicamRPiGPU as it is needed. In VisiCam all command line arguments for visicamRPiGPU are known.

VisiCam uses Unix specific commands to start and stop visicamRPiGPU. In general this circumstance is no problem for the platform independence of VisiCam at all because visicamRPiGPU is anyway developed for the Raspberry Pi 2 with the Raspbian [Raspberry Pi Foundation, 2012] operating system.

VisiCam has control over the visicamRPiGPU process.

As soon as the configuration of VisiCam is changed it starts, stops or restarts visicamRPiGPU if needed. In the context of the new inactivity timer feature for the camera visicamRPiGPU needs to be stopped and started as well. If no images are requested for a configurable period of time, visicamRPiGPU will be stopped and this will turn off the camera as well. It is started again as soon as images are requested again. VisiCam needs to ensure that only one instance of visicamRPiGPU is active because only one process of the system is allowed to access the camera at the same time.

This inactivity timer is probably a better setting for the hardware than the other settings, which are currently in use for that purpose. In the first setting the camera is permanently turned on. The other setting turns the camera on and off for each single image.

VisiCam and visicamRPiGPU need to exchange data to work correctly. Again, file locking mechanisms are used to ensure correct file accesses because otherwise both processes might try to access a file at the same time, which would cause arbitrary errors in the programs.

Basically, visicamRPiGPU provides the original and unmodified camera image for the marker detection in VisiCam. VisiCam computes the homography matrix for the perspective correction and stores this data in a file for visicamRPiGPU. With this data visicamRPiGPU can compute the processed image, which is provided to other computers in the local area network by VisiCam.

3.4.2 Other VisiCam Improvements

Similar to visicamRPiGPU, a configurable refresh timer is implemented in VisiCam. Each refresh causes VisiCam to run the marker detection and to store the new homography matrix into a configurable file. In the original and unmodified version of VisiCam the marker detection and the matrix computation are performed for each single image. This is not needed at all and wastes system resources because the position of the markers is unlikely to change a lot.

Memory leaks cause the system to continuously lose available memory space and this leads to system failures.

Additionally, a so-called memory leak is solved in the new modifications of VisiCam. Each process on a system can reserve memory space for computations. A memory leak means that a process reserves memory space but this memory space is never released for other processes of the system again, although the reserved memory space is no longer needed by that process at all. Because of this circumstance critical system processes can run into trouble and this causes system failures. Each run of the marker detection algorithm caused such a memory leak in VisiCam, which is now solved in the new modifications.

It might happen that several computers access the processed image of the VisiCam web service at the same time. For this case some parts of the implementation of VisiCam are now assigned with the Java keyword `synchronized`, which means that these parts of the implementation are not allowed to be executed in parallel.

New bash scripts are introduced for an easier maintenance of VisiCam. The setup documentation is modified to speed up the compiling of VisiCam dependencies.

3.5 VisiCut Modifications

VisiCut [Oster, 2011] is used to simplify the laser cutting. The laser cutters and materials are defined in VisiCut and it provides a GUI (Graphical User Interface) for the users, who are able to prepare their laser cutting jobs easily with VisiCut.

With the new modifications of VisiCut it is supposed to support the FabQR software system. Several new files and Java classes need to be integrated into VisiCut for this purpose. New dialogs and GUIs need to be implemented for the FabQR integration into VisiCut as well. The different parts of the integration of the FabQR software system into VisiCut are explained on the next few pages.

The modifications also introduce the support for a projector setup. For this setup a special hardware configuration is needed.

Since VisiCut is a huge software project, which consists of many different source files and Java classes, there are many possibilities for other miscellaneous improvements of the software project. Such improvements are also included in these new modifications for VisiCut.

VisiCut requires Java version 7 with the modifications.

For some of the new modifications Java version 7 is needed because some of the new external libraries have this as a requirement. This is not a big issue at all, since Java version 7 is already comparatively old and Java version 6, which is required by VisiCut itself, is already quite outdated. In addition to that, using newer Java versions can be beneficial because of noticeable performance improvements as well.

VisiCut fork[a]

[a] https://github.com/FroChr123/VisiCut

3.5.1 Automatic VisiCam Images

Currently users need to click a button in the GUI of VisiCut in order to manually fetch a new processed camera image from VisiCam. For the integration of the FabQR software system into VisiCut it is important that VisiCut fetches these images automatically from the modified VisiCam version.

This feature is provided by the new Java class `RefreshCameraThread`. There is a new configuration option, which determines how often this class tries to fetch a new camera image. With this option the refresh rate of the camera can be adjusted. Low values for this setting such as 50 milliseconds lead to a video-like preview, which is needed by the FabQR software system.

3.5.2 Webcam Support

A webcam can be used as an alternative to the VisiCam setup to provide camera images for the detection of QR codes. For this purpose new GUI elements are introduced, which open a dialog with the current webcam image. This image is continuously updated by the class `TakePhotoThread`, which already exists in VisiCut.

Unfortunately, the unmodified version of this class is relatively inflexible. It defaults to a resolution of 176 x 144 pixels and it uses a fixed refresh rate of 100 ms. With these settings there is a notable delay for the camera images and the images have a very low quality.

The FabQR integration needs high refresh rates and images with a good quality.

Because of these reasons this class is modified. The refresh rate is now configurable and it defaults to the value of 20 ms. With this setting the delay of the camera image is of course reduced. Additionally, the camera can now be configured to use one of the resolutions, which are always available in Java: 176 x 144 pixels, 320 x 240 pixels or 640 x 480 pixels. For the FabQR integration the highest resolution of 640 x 480 pixels is used to provide high quality images for the detection of QR codes.

3.5.3 QR Code Detection

With the modifications the library ZXing [ZXing authors, 2007] is included in VisiCut for the detection of QR codes in VisiCam images or webcam pictures. The new Java classes `RefreshQRCodesTask` and `QRWebcamScanDialog` initiate the detection and processing of QR codes in the images of VisiCam or the webcam. If the QR code detection is enabled in the preferences, it will be handled by the new Java class `QRCodeScanner`.

Internally the `QRCodeScanner` class controls a thread for the detection of QR codes with the ZXing library. This class extends the default Java `Observable` class, which is well-known in the Observer pattern in software engineering. This pattern is used to report the resulting QR code data to other classes, which implement the `Observer` interface.

ZXing offers basic support for the detection of distorted and rotated QR codes. Although it claims to be able to detect multiple QR codes at once in an image with the `QRCodeMultiReader` class, this class is not used for this purpose in the integration of the FabQR software system into VisiCut. The main reason for this is the circumstance that the results for the detection of multiple QR codes in a single image with this class are relatively bad and it is very unlikely to detect only two QR codes in a single image with it correctly.

The class for reading multiple QR codes with ZXing is in practice unable to detect multiple QR codes in an image reliably.

Additionally, the ZXing library is primarily developed for the use case of scanning only a single QR code once. Therefore, ZXing is optimized for performance and not for quality of the QR code detection.

In contrast to that, the detection of the QR codes for FabQR is supposed to have a high quality and performance at the same time. The QR codes need to be detected correctly in each run of the QR code detection algorithm. Therefore, a custom and advanced algorithm for the detection of multiple QR codes in an image is developed, which works as depicted in figures 3.5 and 3.6.

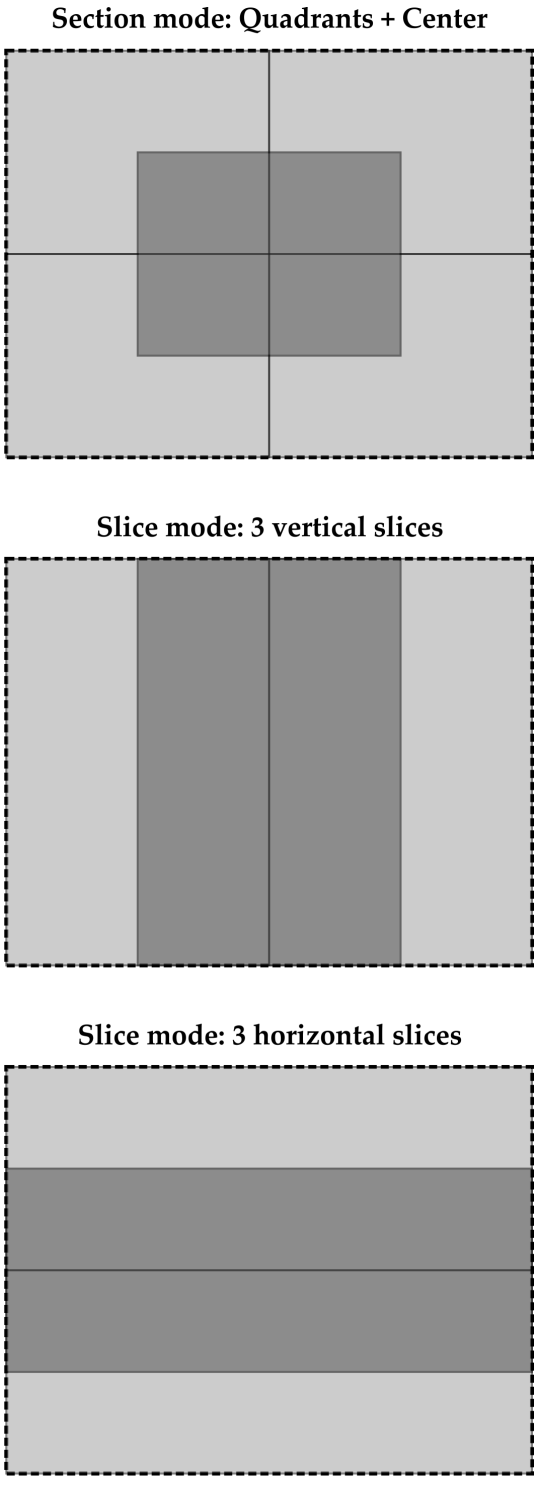


Figure 3.5: Detection for multiple QR codes

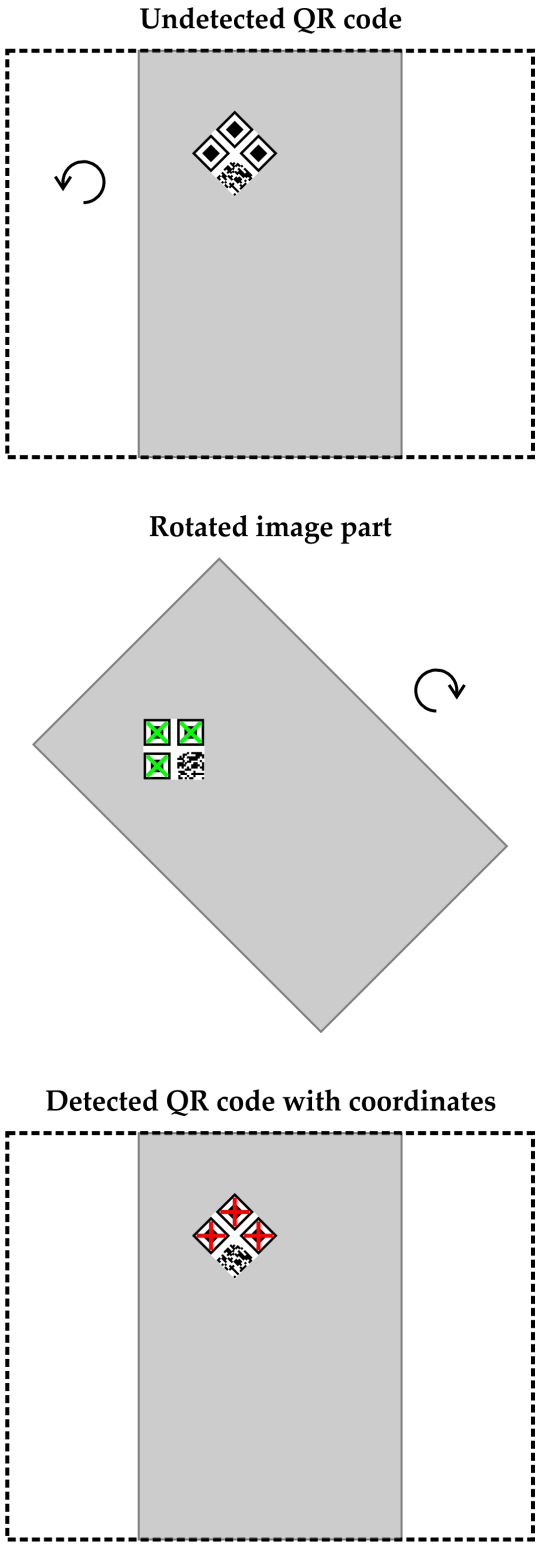


Figure 3.6: Detection for rotated QR codes

The detection of single QR codes is used multiple times to imitate the detection of multiple QR codes.

The advanced algorithm for the detection of multiple QR codes in a single image is based on the fact that the QR code detection for a single QR code in an image works well with the ZXing [ZXing authors, 2007] library. The idea is to split the complete input image into multiple image parts and each part is checked separately for a single QR code. The results of all single QR code detection runs for the different image parts are combined, which finally leads to a detection of multiple QR codes in a single image.

It is important to mention that the splitting schemes of the input image need to be chosen carefully. In the implementation the section mode and the slice mode are used for this purpose. Examples for these modes are depicted in figure 3.5. In the section mode the image is split evenly into rectangles in the horizontal and vertical dimension. In contrast to that, the slice mode creates rectangles, which have the same length as the input image for one dimension and the image is only split in the other dimension. Therefore, the slice mode can be applied horizontally and vertically.

In figure 3.5 some highlighted darker areas are included in the examples. These areas show overlapping areas for the detection of the QR codes. Overlaps need to be included to minimize the probability that a QR code at an arbitrary position is always split into multiple parts by the used splitting schemes. For the first example this means that a QR code, which is placed exactly in the center of the image, could not be recognized by only analyzing the quadrants of the image. Since the overlapping rectangle in the center of the image is analyzed as well, a QR code in the center of the image could be detected by analyzing this rectangle.

The ZXing library is only able to detect QR codes with rotation values around multiples of 90 degree well. This implies that it has issues with the detection of rotated QR codes, which have rotations near to 45, 135, 225 or 315 degree. The advanced and custom algorithm for the QR code detection includes a solution for this problem as well. The basic idea of this solution is shown in figure 3.6.

Graphics operations in Java do not have a high performance.

For the solution of this issue with these specific rotated QR codes three vertical slices of the full image are created. These slices are rotated by 45 degree in counter-clockwise direction. In figure 3.6 it is depicted that this step causes these rotated QR codes to have a new rotation value near to a multiple of 90 degree. Because of this step the ZXing library is able to detect such QR codes in the rotated image part. After that the coordinates of the rotated QR codes are mapped back to the full image. Only three parts are used in this part of the algorithm because the rotation of the image parts takes much time in Java. There is also an option, which enables and disables this feature.

In order to implement the interactive placing of laser cutter schemes with QR codes the coordinates and the rotations of the QR codes in the image need to be computed. The ZXing library returns the coordinates of the finder patterns in the corners of each detected QR code. The coordinates are relative to the top left corner of the image part, which is analyzed by the ZXing library. The center coordinate and the rotation of the QR code are computed from these three values.

Of course the coordinates need to relate to the dimensions of the full input image for the QR code detection algorithm. Multiple coordinate systems are involved in these computations. The global coordinate system has its origin in the top left corner of the full image and the vertical axis is inverted. Each image part has a normalized local coordinate system, which has the same properties as the global coordinate system with a changed origin of the coordinate system. For the rotated image parts there is an additional rotated local coordinate system.

It is complicated to map the coordinates of the detected QR codes from the rotated local coordinate systems back to the normalized local coordinate system because the coordinates need to be correctly rotated by 45 degree in clockwise direction. The mapping of the coordinates from the normalized local coordinate systems back to the global coordinate system is simple because the global coordinates of the top left corner of each image part are known. This mapping just requires two simple additions.

Detecting the same QR code multiple times needs to be avoided.

Additionally, it needs to be ensured that the same physical QR code is not detected multiple times in the different image parts. Before a new QR code is added to the result set of all detected QR codes it is checked, whether a QR code with roughly the same global center coordinates is already stored in the result set. This is the reason why the final result set of all detected QR codes does never contain the same physical QR code multiple times.

The laser cutter files of the URLs, which are saved in the QR codes, are downloaded and imported in VisiCut. For the webcam image all laser cutter schemes are imported with the default settings and they appear in the top left corner of the preview in VisiCut. The positions and rotations of the detected QR codes are mainly used for the VisiCam camera image. The center coordinates and rotations of these schemes automatically correspond to the center coordinates and rotations of the associated QR codes. It is ensured that the laser cutter schemes fit into the preview correctly. Additionally, the modifications implement the correct rotation around a common center position for multiple objects in PLF files.

As soon as new QR codes are detected in the VisiCam images, the editing mode is changed to QR code editing. This means that some other parts of the GUI are disabled for this period of time because this would cause some issues with the internal data structures of VisiCut otherwise. The outlines of these new objects, which are loaded because of QR codes, are highlighted with thick green lines in the preview. In each iteration of the responsible implementation the temporary new objects are removed and the VisiCam image is analyzed for QR codes again and this might lead to the insertion of new objects again. This approach guarantees that the objects are always loaded correctly. The rotation angles of new loaded objects always snap to the nearest multiple of 15 degree for an easier rotation of the objects with the QR codes.

There are special data structures to store additional data for an object, which is imported with a QR code.

Once the QR code editing mode is enabled a new button is shown, which stores the current positions of the new objects and loads the correct laser cutting mappings. Again, it needs to be ensured that the same QR codes are not imported multiple times here. For this purpose a similar concept as before is used, which involves the checking of already existing data and stored coordinates. The new objects, which have been imported with QR codes, can now be used as any other regular object in VisiCut. The QR code editing mode is disabled and it will be enabled again, if QR codes are detected at other positions.

This user interaction with the FabQR integration in VisiCut is presented in figure 3.7. In this example the laser cutting schemes of two QR codes are imported in VisiCut with the corresponding center positions and rotations.

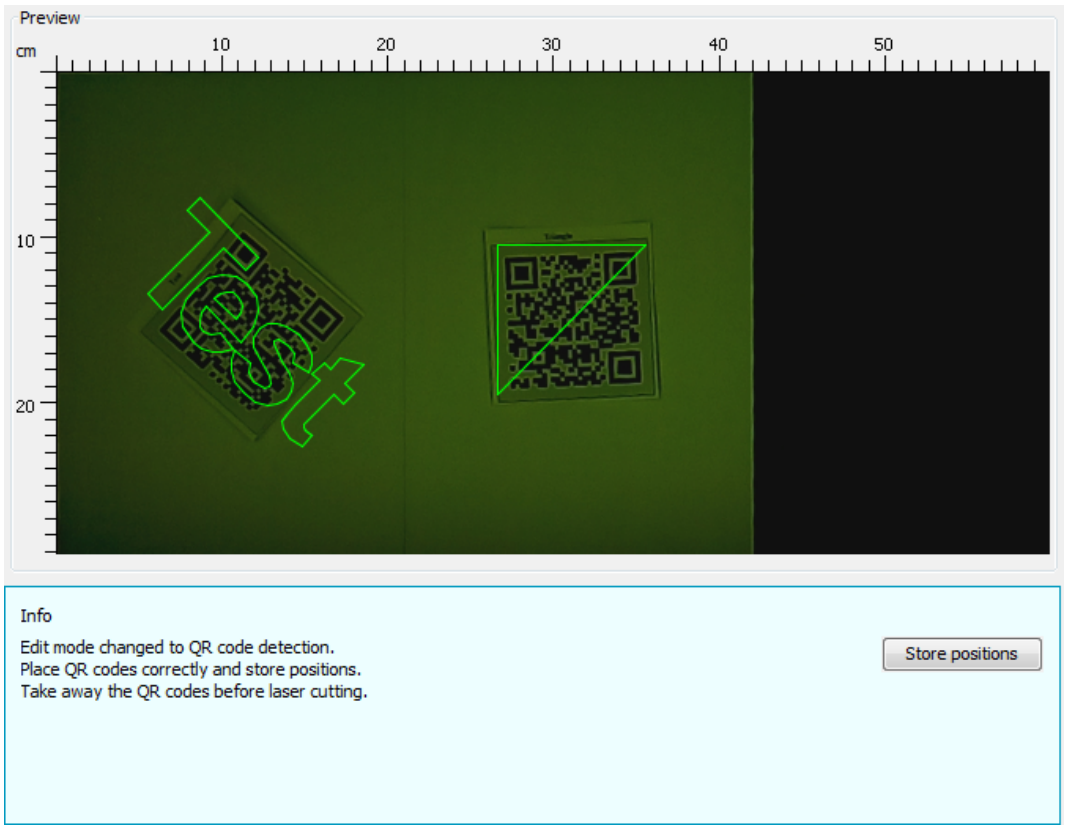


Figure 3.7: Preview for QR code imported objects

3.5.4 File Management

With the modifications the management of files in VisiCut is changed as well. Temporary files are needed by VisiCut for various purposes. For some operating systems the deletion of these temporary files does not always work correctly. Therefore, the handling of temporary files is changed. With the modifications the names of all temporary files start with the prefix "VisiCutTmp_", which is the reason why these files can be identified easily. VisiCut tries to delete these files now on every start.

HTTP redirects are supported by the downloader, which is needed for the FabQR web service.

The new Java class `CachedFileDownloader` is responsible for downloading the files, which are associated with the QR codes. The `CachedFileDownloader` is able to deal with HTTP (Hypertext Transfer Protocol) redirects correctly and it can check for valid maximum file sizes and correct file extensions. The maximum file size is set to 10 MB and the file types PLF and SVG are currently allowed. As the name already suggests, the class uses a cache for the URLs and the downloaded files. This avoids unnecessary downloads of the same file. Currently 25 URLs and related files are stored in the cache. If the `CachedFileDownloader` detects that it is supposed to download a file from the private area of the configured FabQR web service, it will also add authentication details to the HTTP request for the download.

3.5.5 Projector Support

In early stages of the FabQR concept it was planned to create a setup with a projector, which is mounted right above the laser cutter. With the projector a preview of the laser cutter schemes can be projected right onto the material inside the laser cutter. Unfortunately, the hardware setup for this is relatively complicated because the projector needs to deal with the ambient light and it needs to be positioned precisely. Because of these reasons such a real projector setup is currently untested.

Nevertheless, the software component for the projector support is theoretically functional. The URL of the web service, the resolution of the preview image and the refresh rate are configurable. The web service needs to process the PNG data, which is sent within this upload. Authentication details are supported for this file upload as well.

The new Java class `PreviewImageExport` is used to create the PNG image of the current preview in VisiCut. A black background is used and the outlines of the laser cutting schemes are highlighted with thick red lines. The PngEncoder [ObjectPlanet, Inc., 1998] library is used to create the corresponding PNG image quickly. This export of the VisiCut preview is needed for the upload of the project documentation as well.

The PngEncoder library can create PNG files quicker than the native Java functions.

Since the aspect ratios of the configured projector resolution and the laser cutter dimensions do not necessarily need to match, the preview of VisiCut is aligned at the top left corner of the exported image and it is evenly scaled up as much as possible within the configured projector resolutions. This means that there might be an intentional blank space in the exported preview image at either the right edge of the image or at the bottom of it. An example for this is provided in figure 3.8. For reasons of presentation, the intentional blank space at the bottom of the image is highlighted with a green line and some text. Everything above the green line represents the preview in VisiCut.

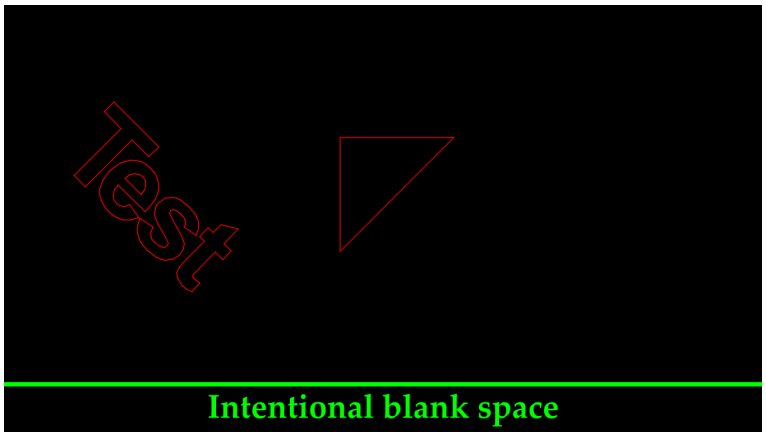


Figure 3.8: Preview image export in VisiCut

3.5.6 FabQR Upload

If the FabQR options are configured correctly in VisiCut and the user starts a laser cutting job by pressing the ”Execute” button, there will be a dialog, which asks the user to publish this project on the websites of the FabLab. The laser cutting job is actually sent as soon as the user has answered the dialog. This dialog is depicted in figure 3.9.

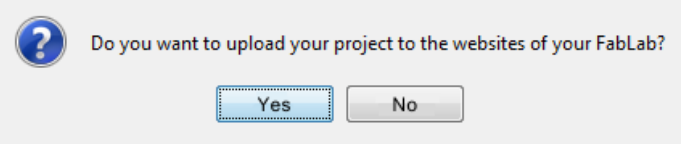


Figure 3.9: FabQR upload confirmation dialog

Users are asked to publish FabLab projects.

If the user chooses ”No” in this dialog, the laser cutter job is just sent as usual and nothing else happens. But if the user chooses ”Yes”, the laser cutting job is sent to the laser cutter and a new dialog for the upload to the configured FabQR web service appears.

In the FabQR upload dialog the user can insert information about the project. At the top of the dialog there is a status message, which shows the current status of the upload. Depending on the chosen license for the publication of the project it is optional to enter the name and the email address. In the bottom left corner of the dialog the user has the possibility to select VisiCam or the webcam to take a photo of the finished laser cut object. Of course a name for the project needs to be specified, which needs to have at least three characters. A license needs to be chosen as well and there are short descriptions for the different licenses. Currently a set of Creative Commons [2001] licenses is available for this selection. Other tools and machines, which are involved in the creation of the project can be specified as well. A short description of the project is required for the project documentation. At the bottom of the dialog the user can close the dialog or start the upload of the project documentation.

Additionally, some other data is automatically collected and sent to the FabQR web service by VisiCut. The configured name of the FabLab, the name of the laser cutter model and the material settings are included as well. The public accessible URLs of laser cutter schemes, which have been imported with QR codes, are inserted into the project documentations as references as well. An exported preview image is also created. Of course this FabQR web service client needs to correspond to the definitions of the APIs of the FabQR web service.

Some data is automatically included in the created project documentations.

The upload dialog in VisiCut for the FabQR software system is presented in figure 3.10. An exemplary project documentation is created with this dialog.

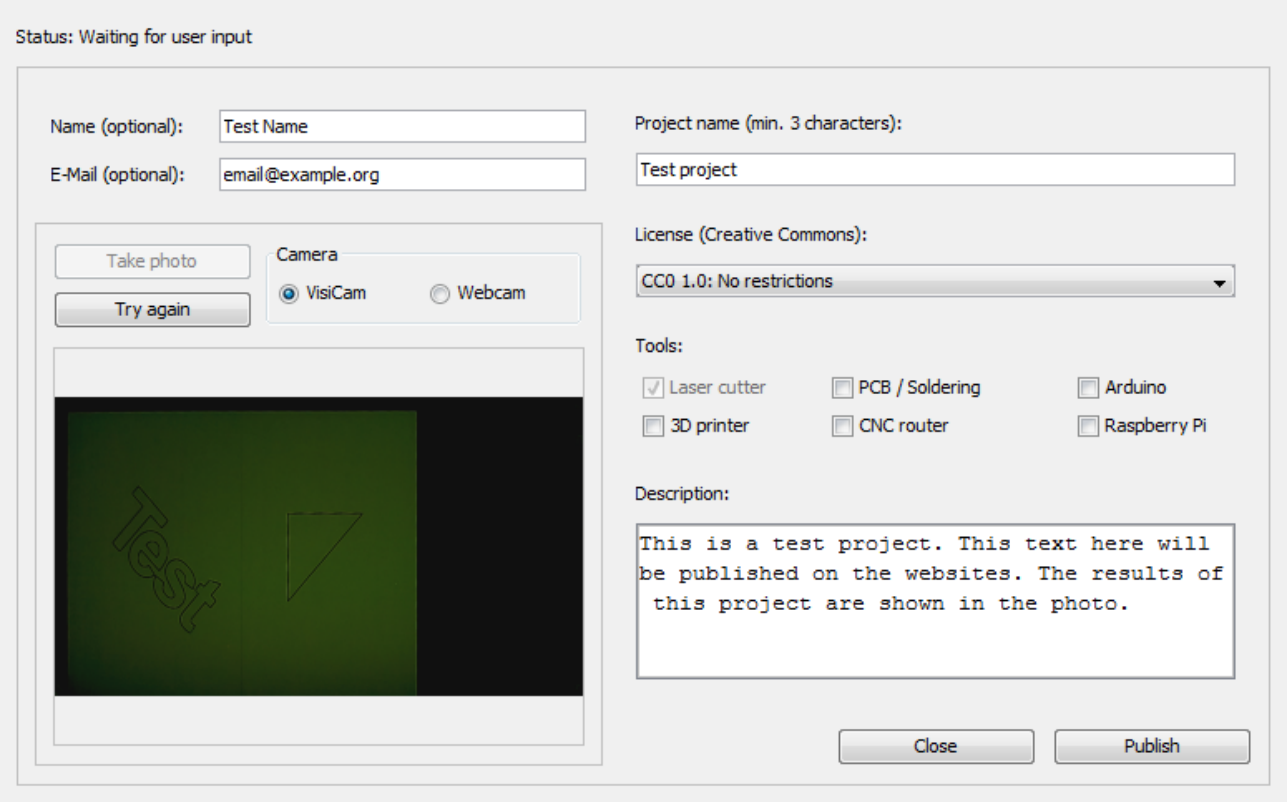


Figure 3.10: FabQR upload dialog

3.5.7 GUI Changes

Several new configuration options are added with the FabQR integration.

With the modifications for the integration of the FabQR software system into VisiCut several changes in the GUI need to be introduced. Mainly these changes include new options in the different configuration dialogs. Since there are also some passwords for the FabQR software system, a warning is shown as soon as passwords are exported within the settings of VisiCut. Figure 3.11 shows the configuration dialog for the FabQR web service.

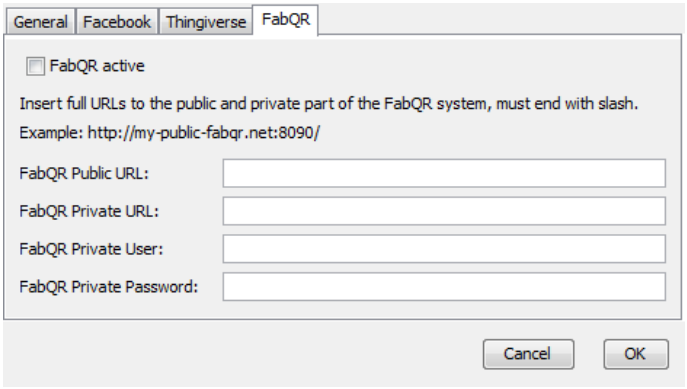


Figure 3.11: FabQR configuration dialog

Just before FabQR was integrated into VisiCut, other contributors of VisiCut changed the behavior of the GUI in the main view of VisiCut for smaller resolutions and displays. This feature is optimized with the modifications for the FabQR integration. Furthermore, the new Java class `IconLoader` is introduced to handle the access to icons, which are related to additional extensions of VisiCut.

3.5.8 Concurrent List Access

The class `PlfFile` is one of the main data structures in VisiCut. Basically it is a wrapper for a simple `LinkedList` in Java. This is a huge problem for concurrent accesses by different threads of VisiCut. Therefore, most accesses to this list use the keyword `synchronized` now and a copy of the list is provided for most read-only accesses.

3.6 QR Code Readability

For the FabQR software system it is important that the QR codes have a high readability in the camera images. The use cases for the QR codes in FabQR are not only limited to printed QR codes, but users can also use their smartphones to show the QR codes to the cameras. In this context there are severe issues with the detection of QR codes from smartphone displays.

Figure 3.12 shows that the main problem for the detection of QR codes from smartphone displays is related to the brightness of the displays and the intensity of the emitted light. Usual cameras with default settings are not able to handle the emitted light of the smartphone displays well, which is even intensified by dark backgrounds as shown in the picture.

QR codes on smartphone displays are more difficult to detect.

In this image the camera reacts that sensitively to the emitted light of the smartphone display that it displays the light as a bright white cloudy spot around the light source. This effect is clearly visible in that figure and it causes a lot of issues in the detection of QR codes from smartphone displays because the quality of the QR codes is in general reduced and some distortions are added to it.

Figure 3.12: QR code on a smartphone display

In figure 3.13 the quality loss of QR codes in camera images and the related distortions are shown.

Bright smartphone displays are likely to add distortions to displayed QR codes.

There are noticeable differences between the original QR code and the QR code in the camera image. Because of the oversensitivity to the emitted light of the smartphone display all black areas in the QR code are modified and wiped out from the QR code in the camera image. The black areas are in general smaller than they are supposed to be and the edges are less clear. Additionally, all corners are rounded off.

The readability of QR codes does not only depend on the quality of the QR codes, but it is also dependent on the physical size of the QR codes and the distance to the cameras as well.

Because of these effects several measurements of QR codes are taken. The results of these measurements are evaluated to find acceptable settings for the generation of QR codes in the web service of the FabQR software system. These settings are related to the physical size of the QR codes and the brightness of the background of the QR codes.

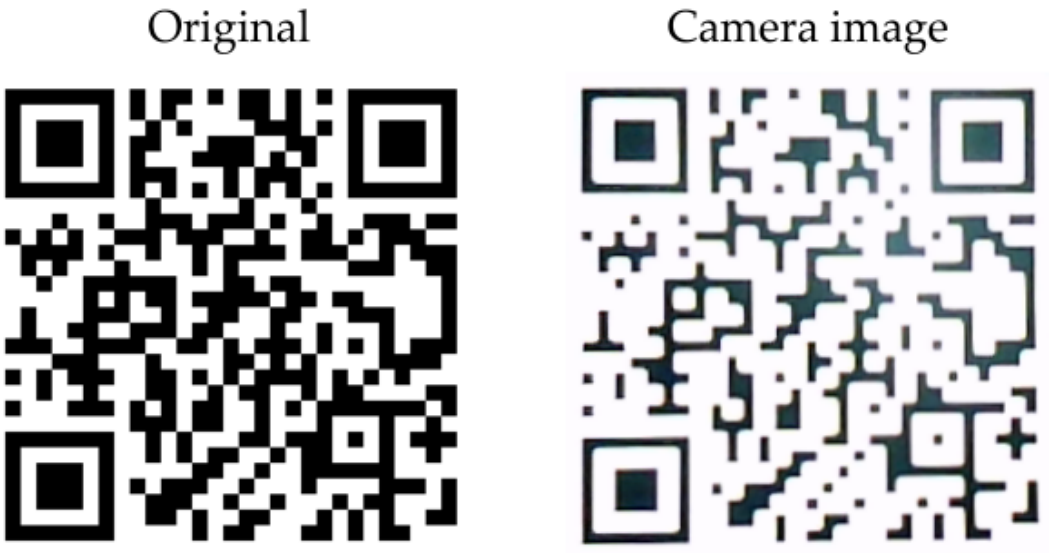


Figure 3.13: Loss of QR code quality in camera images

3.6.1 Test Arrangement

For the measurements of the QR codes a specific test arrangement is used. To some extent the measurement results are influenced by these settings, but since common hardware with default configurations is used for these measurements, it is expectable that these results are similar to the results of most other settings.

All QR codes for the measurements have the same content.

The QR codes for the measurements always contain 33 alphanumeric characters and they are created with the highest error correction level. In the brightness measurements the QR codes always have a physical size of 5.5 cm side length. The side length measurements always use printed QR codes with a brightness value of 100%.

For the software the ZXing [ZXing authors, 2007] library in version 3.2.1 is used. The webcam is used with default settings and a resolution of 640 x 480 pixels. All measurements are taken in front of a white background.

In the measurements the maximum readable distance is measured. This value represents how likely it is that the measured QR code is detected correctly and because of that it is related to the readability of the QR code.

In order to measure this maximum readable distance value the QR code is roughly placed in the center of the camera image at a large distance and it is slowly moved towards the webcam. As soon as the software detects the QR code in the camera image the current distance of the QR code to the webcam is measured. Each setting is measured twice to compute average values.

In figure 3.14 a scheme of the test arrangement is presented. Figure 3.15 shows the image of the webcam in this test arrangement.

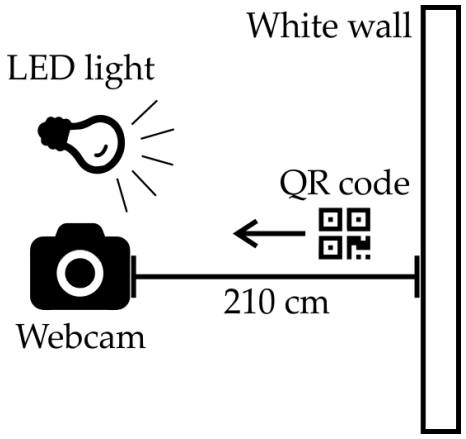


Figure 3.14: Scheme of the test arrangement
Includes icons from Font Awesome [Gandy, 2012]

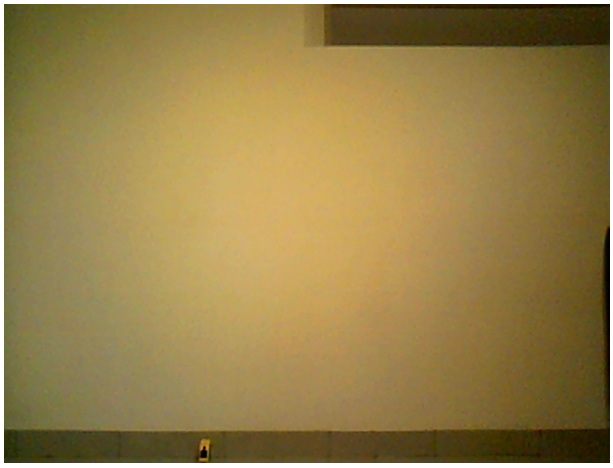


Figure 3.15: Webcam image of the test arrangement

3.6.2 Measurement Results

The measurement results are shown in figures 3.16 and 3.17. A value of 0.0 cm for the maximum readable distance indicates that this QR code is not readable at all. Based on these results a brightness of 90% and a side length of 7.0 cm are chosen for the generated QR codes of the FabQR web service. With the blank space of the frame around the generated QR codes they have a total side length of 8.3 cm.

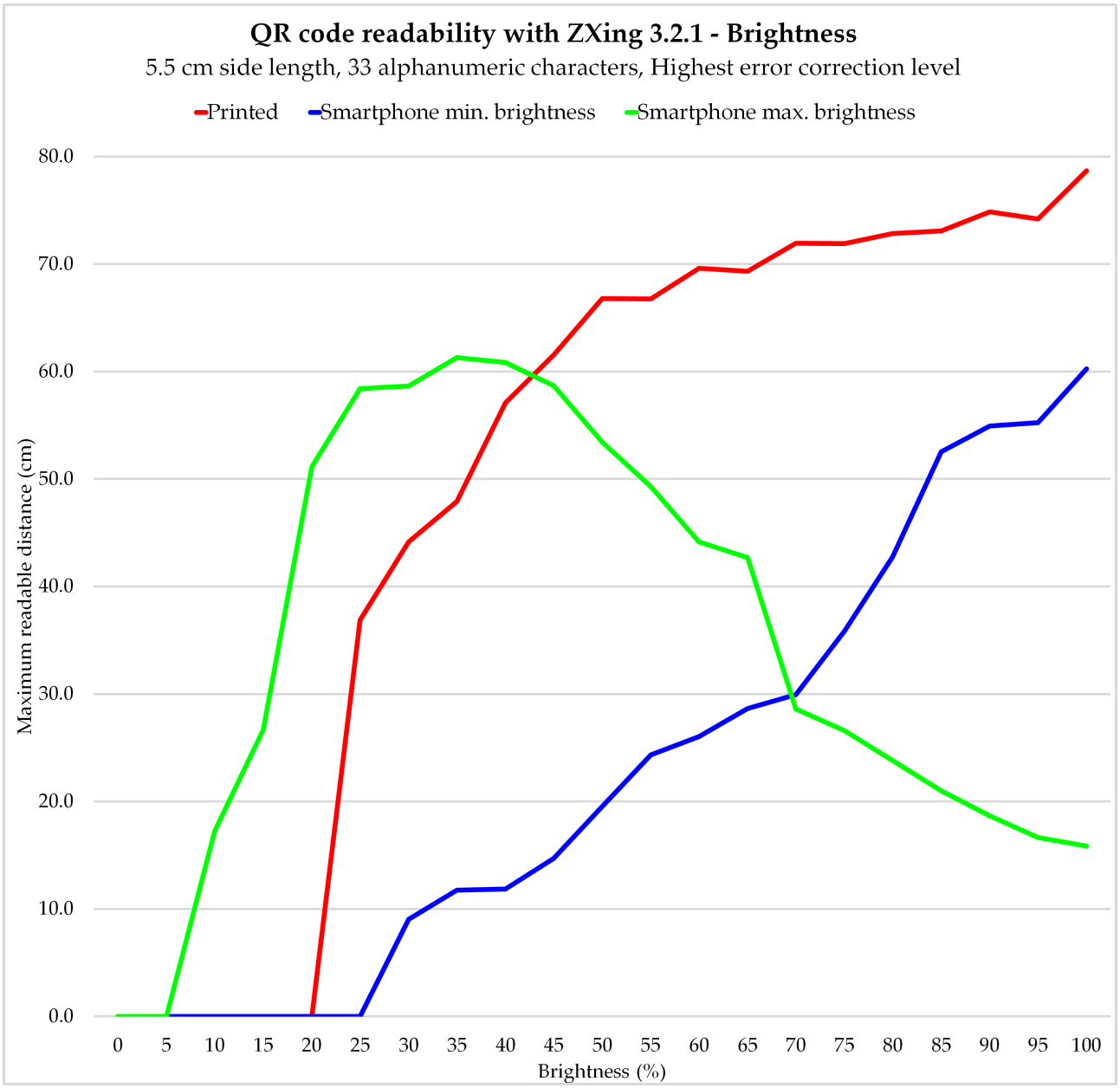


Figure 3.16: QR code readability: Brightness

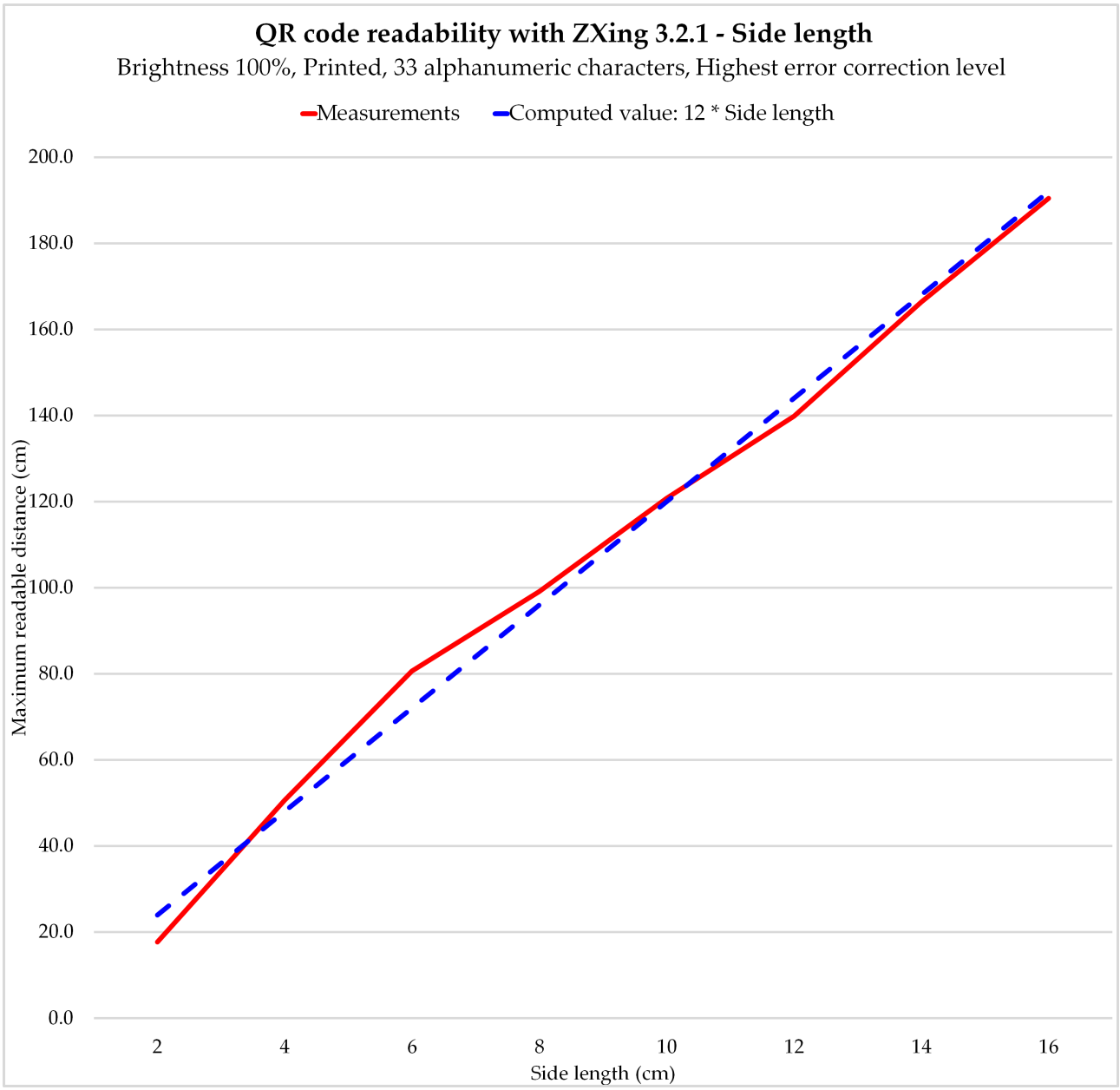


Figure 3.17: QR code readability: Side length

3.7 FabQR Web Service

The FabQR web service is a very important piece in the FabQR software system. It includes several functions and features to handle the sharing of knowledge, project documentations and project files on the website of the FabQR system.

Basically, the web service is consists of two parts. The first part is public accessible over the Internet. In contrast to that, the private part can only be accessed with the correct combination of a username and a password. This combination is known to the FabQR integration in VisiCut.

The web service offers possibilities for anonymous uploads of project files to the FabQR system in the public accessible part of the implementation. These files are only stored temporarily and they will be deleted after a configurable period of time. The files are stored in the private section of the web service and can only be accessed from VisiCut in the FabLab. For each uploaded file the users receive a QR code from the website, which is used for the identification of the file. This procedure ensures that the anonymous file upload of the web service is not abused and the whole process is validated by the physical presence of the users in the FabLab.

In the private part of the FabQR web service FabLab projects can be published by the FabQR integration in VisiCut by using the API. For each of these documentations a new QR code is created as well and all related data for this project is published and it is public accessible over the Internet after this step.

All published FabLab projects are displayed on the website of the FabQR system. There are multiple pages, which show all published projects, and the newest projects are always shown at the top of the first page.

The structure of the FabQR website is similar to a blog.

FabQR[a]

[a] https://github.com/FroChr123/FabQR

The QR code settings are chosen according to the results of the QR code measurements.

All QR codes of the FabQR web service are generated with the same settings by PHP QR Code [Dzienia, 2010]. It needs to be mentioned that the caching option of this software project must not be disabled because it might generate wrong QR codes otherwise. The second lowest error correction level M (15%) is used for all QR codes because it has an acceptable balance between data capacity and redundancy. The QR code measurements show that it is beneficial to use a brightness of 90% and a side length of 7.0 cm for the QR code settings. Additionally, there is a frame around the generated QR codes, which is useful to avoid difficulties in the detection of QR codes as well. With this frame the QR codes have a side length of 8.3 cm.

With the software project PHPMailer [Matzelle et al., 2001] the QR codes of the website can be sent as email attachments. The users can enter their email address and the website sends the QR code to that email address. Additionally, there is an option to simply print a QR code directly in the web browser.

In addition to that, the administrator of the FabQR software system has access to a website, which can remove temporary file uploads or already published project documentations completely. This website is accessible in the private area of the FabQR web service and an additional password is needed for the identification of the administrator of the system.

These are all the different use cases for the users, which are currently supported by the implementation of the FabQR web service. Additional details of this implementation are documented and explained on the next few pages. Especially the API for publishing projects with the private section of the FabQR web service is described.

In figure 3.18 an example for the displayed project documentations on the website of the FabQR implementation is given.

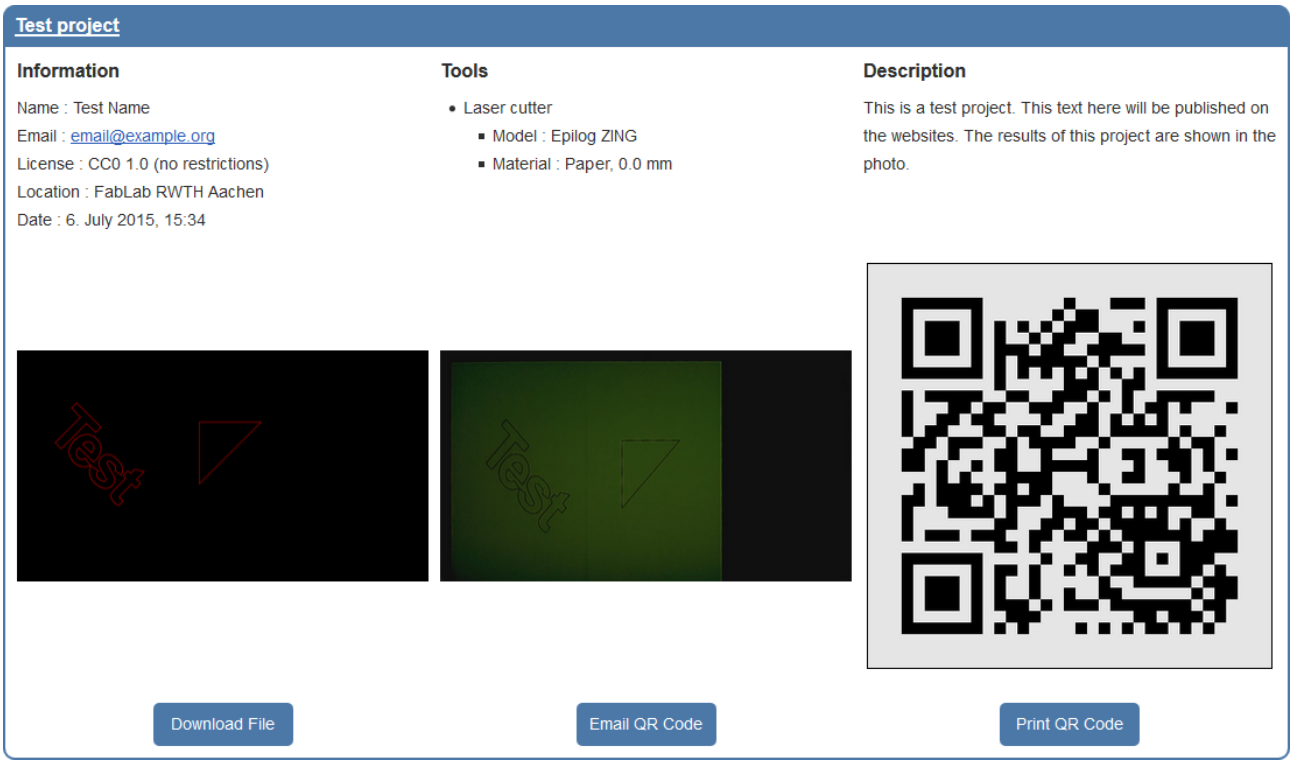


Figure 3.18: Project documentation on the FabQR website

3.7.1 Installation Scripts

For a simple installation of the FabQR web service a large bash installation script is used. This bash script is mainly designed for the Raspberry Pi 2 with the Raspbian [Raspberry Pi Foundation, 2012] operating system, but it can also work for other operating systems such as Debian [Software in the Public Interest, Inc., 1993]. The installation script works interactively and asks the user for several configuration values. It can be used multiple times to reconfigure the FabQR web service.

The installation script needs to be executed as root user because it applies severe changes to the system. For example, all required software packages are installed. A new system user with the name "fabqr" is created as well and the home directory of this user contains most files of the FabQR web service. The symbolic link `fabqr_data` in this directory should reference a location in the system, which has a lot of available storage. The project documentations and the uploaded files are saved at this location. For the Raspberry Pi 2 it is recommended to use a USB storage device for this purpose.

The FabQR web service is installed as a service in the system and it starts automatically with each system boot.

All the required files for the FabQR web service are downloaded and copied to the correct directories in the system. The required software packages are configured correctly as well. The installation script sets up a service script for the FabQR web service and there are bash scripts to start and stop the FabQR system easily. All these bash scripts are supposed to simplify the initial setup and the maintenance of the system.

3.7.2 APIs

Here the APIs of the FabQR web service are explained. There are two API files, which are supposed to be used by other programs. In general the PHP [PHP Group, 1995] API scripts of the FabQR web service use the `$_POST` and `$_FILES` variables to receive data. In the two API files different requirements for the received data are defined.

The API file `api_upload_file.php` in the public accessible directory of the FabQR web service can be used to perform temporary file uploads to the system. This API will process the uploaded file `$_FILES["inputFile"]`. Currently the allowed file extensions are configured as SVG and PLF. This API returns the URL to the QR code for this uploaded file.

In contrast to this API file, the file with the name `api_uploadproject.php` in the private section of the FabQR web service has to deal with uploaded project documentations and because of this the API definition is comparatively complicated. The list below explains all possible values for this API.

- **Mandatory:** `$_FILES["inputFile"]`
  This is the project file. Currently the allowed file extensions are configured as SVG and PLF. The integrated FabQR client in VisiCut always creates and uploads PLF files.
- **Mandatory:** `$_FILES["imageScheme"]`
  An image in the PNG file format is required here. This shows the abstract scheme of the project. VisiCut uses the exported preview image for this purpose.
- **Optional:** `$_FILES["imageReal"]`
  An optional image in the JPEG file format. JPEG is used instead of PNG because it usually has a smaller file size than PNG images for real photos. This is optional because some clients might not have the possibility to create camera images.
- **Mandatory:** `$_POST["name"]`
  This is the name of the author of the project. If the chosen license requires to include an attribution of the author, this information must not be an empty character string.
- **Mandatory:** `$_POST["email"]`
  This is the email of the author of the project. If the chosen license requires to include an attribution of the author, this information must not be an empty character string.
- **Mandatory:** `$_POST["licenseIndex"]`
  The license index acts as an identifier for the chosen license. It needs to correspond to one of the available licenses in the configuration file of the FabQR web service.

- **Mandatory:** `$_POST["projectName"]`
  The project name must be at least three characters long. This value is configurable.
- **Mandatory:** `$_POST["tools"]`
  This is a comma-seperated list of tools, which are involved in the creation of the project. The separator value for the list can be changed in the configuration. New or unknown tools are supported for this information as well. In VisiCut the laser cutter is always selected for this information.
- **Mandatory:** `$_POST["description"]`
  The description of the project must never be an empty character string.
- **Mandatory:** `$_POST["location"]`
  The location for the project always needs to be provided and an empty character string is not allowed.
- **Optional:** `$_POST["lasercutterName"]`
  The name of the laser cutter model can be used optionally. The FabQR integration in VisiCut determines and sends this information automatically.
- **Optional:** `$_POST["lasercutterMaterial"]`
  This is a character string, which describes the used material for the laser cutter. The name and the thickness of the material should be mentioned. The FabQR integration in VisiCut determines and sends this information automatically.
- **Optional:** `$_POST["references"]`
  The referenced URLs of the project are sent as comma-separated list. The separator value for the list can be changed in the configuration. The integrated FabQR client in VisiCut mentions other already published project files in this information, which are imported with QR codes into VisiCut and are used for the creation of the new uploaded project.

3.7.3 Project IDs

Each temporary file upload and published project is identified by a project ID.

In the FabQR web service public and private project IDs are used. The public project IDs are used for the identification of published projects. Private project IDs are used for temporary file uploads. Exactly seven alphanumeric characters are used in a project ID.

In total there are 36 possibilities for each position because a number (0 to 9) or an alphabetical character (a to z) can be chosen. This solution uses the available alphanumeric character set of the QR codes efficiently and it does not require case sensitive file paths in the operating system. This is useful because it allows the FabQR web service to run on other machines with other operating systems as well. Each project ID is generated randomly and in total 36^7 = 78,364,164,096 project IDs are available.

3.7.4 Data Representation

The received and processed data needs to be stored and represented in the FabQR web service. For this purpose basically two similar types of data representations are used: XML and XHTML (Extensible Hypertext Markup Language).

XML is often used for the representation of data and many different software projects support the processing of XML files. Because of these reasons XML is used for the index files of the public and private section of the FabQR web service. Both `projects.xml` files use the same structure and they store the most important data of all available projects in the respective area of the FabQR web service. In this structure the main node contains the URL and it describes its own area in the FabQR web service with the words ”public” or ”private”. For each project node the id, name, location and creation timestamp are stored.

XHTML combines the advantages of XML and HTML.

XHTML is a combination of XML and HTML (Hypertext Markup Language). HTML is the default markup language for the creation of websites. XHTML is very useful because it combines the advantages of XML with the features of HTML. This means that XHTML is able to define a website and it can be processed by common software for the processing of XML files. Because of these reasons XHTML is used to store and represent the data of the project documentations in the FabQR web service.

3.7.5 URL Structure

In the FabQR web service a special structure for the URLs is used and HTTP redirects are involved in this. This special structure of the URLs reduces the amount of characters, which need to be stored in the QR codes of the web service.

There are two types of markers in the URLs. These markers are currently defined in such a way that the marker `/d/` in the URL indicates a download of a published project file and the marker `/t/` refers to the download of a temporary file.

The first set of redirection rules is defined in the configuration files of the Apache HTTP Server [Apache Software Foundation, 1995]. For example, these redirection rules change the suffix of the requested URL `/d/1234567` for the exemplary project ID 1234567 to the new suffix `/redirect.php?marker=d&projectId=1234567`.

Such a `redirect.php` file exists for the public and the private part of the FabQR web service implementation. In these files the second stage of redirects is performed. Basically, the redirect implementation in these files is able to find the uploaded project file and the final redirect leads directly to this requested file.

3.7.6 Security

Since the FabQR web service is reachable over the Internet, several security issues and the abuse of the system need to be considered in the security of the system. The system must be able to detect unusual behavior of users or very large amounts of requests and it must react appropriately to it.

In the FabQR installation script the software package fail2ban [Jaquier, 2004] is installed. In general fail2ban is able to analyze different log files and as soon as threshold values are passed new rules for the software firewall of the system are created. Usually the thresholds are defined by a maximum limit for requests per time for a specific IP (Internet Protocol) address and with these thresholds unusual or abusive behavior is detected. If such behavior is detected, the corresponding IP address of that computer will be blocked in the software firewall for a specific amount of time to protect the FabQR web service.

For the FabQR web service five log files are protected with fail2ban rules: SSH (Secure Shell) logs, Apache access logs, Apache error logs, FabQR temporary file upload logs and FabQR email logs. The latter two log files are created by FabQR itself to avoid abusive behavior in the context of these features of the system. The fail2ban rules for these log files provide a basic security for the system.

SSH is often used to access the command line of other systems remotely.

In addition to that the other software packages, which are required for the FabQR software system, are configured for an increased security as well. Since the project files in the context of FabQR are usually smaller than 1 MB, the configured maximum upload size of 10 MB does not interfere with the proper functioning of the FabQR software system and it provides additional security. Furthermore, the software packages are configured in such a way that no version numbers of the installed software packages are sent over the Internet. This makes it more difficult to attack the system by using version specific security issues of the installed software packages and the security of the system is increased.

As already mentioned earlier, the private part of the FabQR web service is protected with a HTTP authentication. Since the authentication details need to be known for VisiCut in the FabLab, the username and the password need to be stored on this public accessible computer. This circumstance is not a big issue for the security at all because this protection is mainly supposed to protect against malicious requests from all over the Internet.

Additional security measures can be applied to increase the security of the system.

Although all these measures provide a basic security for the FabQR software system, it is of course not able to replace a full-fledged security concept. Several advanced security measures can be applied as well. For example, the configuration of SSH keys for remote command line logins and a decent network configuration can additionally improve the security of the system a lot.

3.7.7 Projector Support

In the FabQR web service the support for the projector images is included as well. The API file `api_pngdisplay.php` in the private section of the FabQR web service receives and processes the relevant input data for this purpose.

In this script the uploaded file `$_FILES["data"]` is checked for data and an image in the PNG file format is expected here. This file needs to be copied to a configurable file path, which is frequently accessed by a process to display the PNG image. Therefore, file locking mechanisms are used again to avoid read and write conflicts. For the file locking the function `flock` is sufficient in this context.

The program, which displays the PNG image, is implemented in the file `fabqr_framebuffer_png.cpp`. This program expects two arguments, of which the first one is a file path to the framebuffer and the second one describes the file path to the input PNG image. The library LodePNG [Lode Vandevenne, 2005] is used for the fast decoding of the PNG image. The PNG image is decoded in the RGBA color format with 8 bits for each channel.

The PNG data is directly written into the framebuffer of the system.

For the Raspberry Pi 2 with the Raspbian operating system the pixel data is directly written into the specified framebuffer of the system. With the default settings the framebuffer `/dev/fb0` refers to the HDMI (High-Definition Multimedia Interface) output of the Raspberry Pi 2. A projector can be connected to this output.

There are three main issues, which need to be solved to write raw image data correctly into the framebuffer of the system.

First of all, the framebuffer has a specific color format. For the Raspberry Pi 2 the framebuffer has a RGB565 color format, which means that 5 bits are spent for the red and blue channels and 6 bits are used for the green channel. The program needs to convert the pixel data from the decoding color format of the PNG image into this different color format of the framebuffer.

Other system processes modify the framebuffer as well. In order to display the PNG image correctly the automatic display standby feature and the blinking of the cursor in the command line are disabled by the installation script of the FabQR software system.

The raw pixel data is written into the framebuffer line by line. The width and the height of the framebuffer need to be known to display the image correctly. Although the width of the framebuffer is configured, the framebuffer might have some additional virtual space at the end of each line for different resolutions. This is the reason why a virtual and a real width value are used in this context. The virtual width value is always greater than or equal to the real width value. For each right end of a line the difference between these two values is filled up with empty black pixels to display the PNG image correctly.

Chapter 4

Evaluation

In the first part of this chapter the System Usability Scale is explained. Afterwards the results of the evaluation are presented.

4.1 System Usability Scale

The System Usability Scale quantifies the usability of a system.

For the evaluation of the FabQR software system the System Usability Scale [Brooke, 1996] is used. The System Usability Scale is a quick, simple, efficient and robust method for the measurement of the usability of a system.

Brooke [1996] mentions that the usability of a system is always strongly connected to the context, in which the system is actually used. Additionally, complex and full-fledged questionnaires with many items quickly lead to frustration for the participants, which is the reason why the System Usability Scale only consists of ten items. By using a high-contrast order of questionnaire items the System Usability Scale tries to prevent that the participants do not really think about their answers. Different aspects of the usability of a system are covered by the System Usability Scale. The final score is calculated from the answers of the participants and can range from 0 to 100 and a score of 100 denotes the best possible result.

4.2 Results

Here the results of the user study are presented and it is checked, whether the requirements of the software system are fulfilled.

4.2.1 User Study

Since the FabLab is used by a variety of people with different backgrounds and abilities, the final questionnaire includes some questions regarding the personal background and experience of each participant. This information can be used to find possible relations between the calculated System Usability Scale value and the personal information of each participant. The raw data of the user study and the unmodified questionnaire are shown in appendix C.

In total twelve people participated in the user study.

In table 4.1 the final results for the participants P1 to P12 of the user study are presented. The figures 4.1 and 4.2 show the questionnaire with the corresponding collected data.

Participant	System Usability Scale
P1	87.5
P2	62.5
P3	80.0
P4	27.5
P5	75.0
P6	72.5
P7	80.0
P8	82.5
P9	80.0
P10	72.5
P11	75.0
P12	55.0

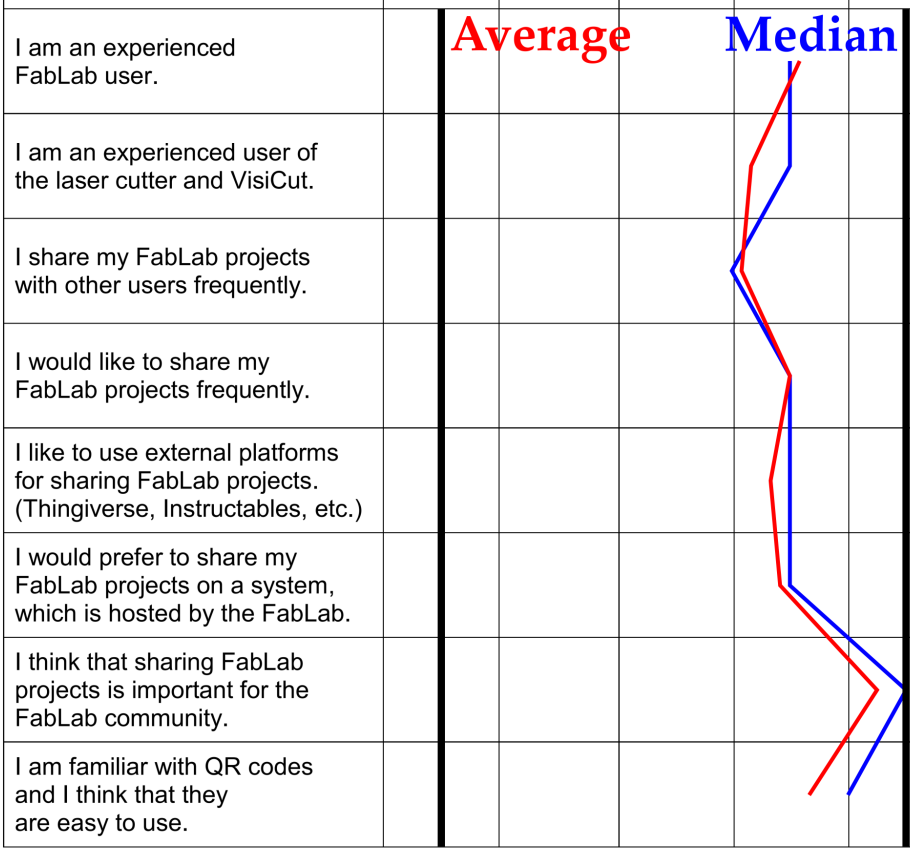
Table 4.1: System Usability Scale results

FabQR – Using QR Codes in the FabLab Workflow
User study (Page 1 / 2)

Personal information and opinion

Gender:	83.3%	Male	16.6%	Female
Occupation:	50.0%	Student	8.3%	Student assistant
(You can check	0.0%	FabLab admin	16.6%	FabLab staff
multiple answers)	0.0%	Research assistant	33.3%	Other

Question	*Strongly disagree*	*Disagree*	*Neutral*	*Agree*	*Strongly agree*
	--	-	o	+	++
I am an experienced FabLab user.					
I am an experienced user of the laser cutter and VisiCut.					
I share my FabLab projects with other users frequently.					
I would like to share my FabLab projects frequently.					
I like to use external platforms for sharing FabLab projects. (Thingiverse, Instructables, etc.)					
I would prefer to share my FabLab projects on a system, which is hosted by the FabLab.					
I think that sharing FabLab projects is important for the FabLab community.					
I am familiar with QR codes and I think that they are easy to use.					



Please turn the page.

Figure 4.1: User study page 1 with data

FabQR – Using QR Codes in the FabLab Workflow
User study (Page 2 / 2)

System Usability Scale
(© Digital Equipment Corporation, 1986, John Brooke)

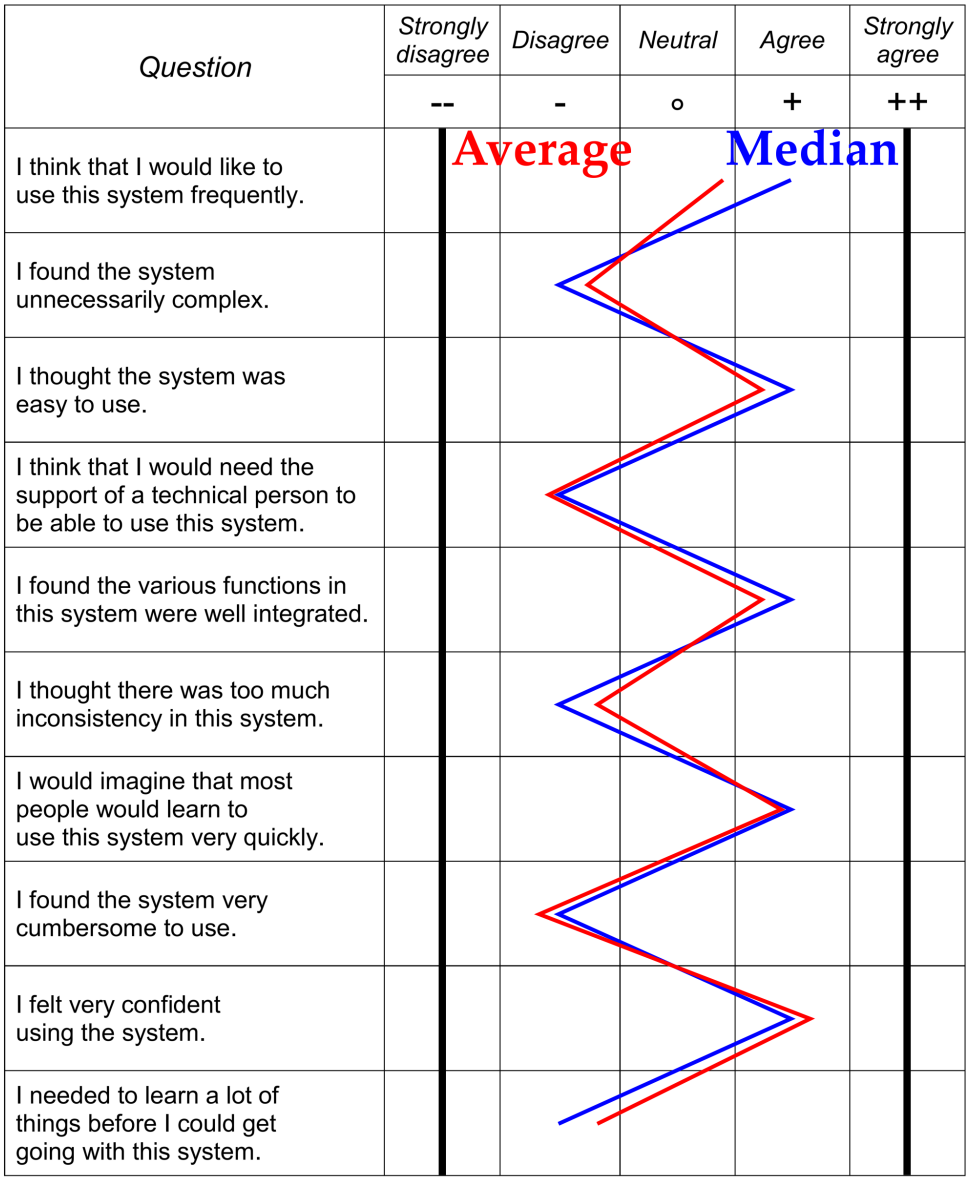


Figure 4.2: User study page 2 with data

With these values an average System Usability Scale of 70.83 is calculated. According to Bangor et al. [2009] the interpretation of this value is located a little below the adjective good and it is in the range of acceptable System Usability Scale results.

Participant P4 is an outlier in the datasets.

It is noticeable that the System Usability Scale rating of participant P4 is much lower than the ratings of all other participants of the user study. Without this result the average System Usability Scale would be 74.77, which is of course a better value in comparison to the original average value.

A possible explanation of this result can be found in the personal background of participant P4 in table C.1. In contrast to all other participants, participant P4 strongly disagrees with QR codes in general. Since the FabQR system is based on the usage of QR codes, this result might indicate that people without any prior experience with QR codes can have some trouble with the FabQR system.

Additionally, some of the FabLab visitors and user study participants were recognizably in a hurry and of course this circumstance can have some influences on the quality of the answers of the participants in general.

Participant P4 also mentioned that the user study should have some outliers in the datasets because people might think that the user study would have been faked otherwise. This statement was probably caused by a combination of the previously mentioned aspects. Because of the result of participant P4 and this statement the validity of the System Usability Scale answers of this participant is in general questionable.

In figure 4.1 the average values and the median values for the collected datasets of the first page of the questionnaire are shown. It is noticeable that most of the answers are located in the agree section for all questionnaire items on this page. Nearly all participants strongly agreed on the importance of sharing FabLab projects with other users of the FabLab community.

Visualizations of System Usability Scale datasets are often similar to zigzag patterns.

The average values and the median values for the gathered datasets of the second page of the questionnaire are depicted in figure 4.2. The visualization of these values looks like a zigzag pattern. This is caused by the structure and the order of the System Usability Scale questionnaire items and for a positive final score the zigzag pattern is supposed to look exactly like this. In the System Usability Scale each questionnaire item is followed by an opposing questionnaire item, which leads to such a zigzag scheme. The order of the spikes in that zigzag pattern determines if the result is negative or positive.

There is a huge difference between the average and the median value for the first questionnaire item of the System Usability Scale. This means that there are many different answers of the participants for this questionnaire item, which asks the participants, if they would like to use the FabQR system frequently.

4.2.2 Requirements

This is the part of the evaluation, which deals with the requirements of the FabQR software system. It needs to be verified that all mentioned requirements are fulfilled.

- **Requirement R1: Usability**
  The System Usability Scale can be used to quantify the usability of a system. With a final result of 70.83 in the user study it is shown that the system has an acceptable usability. Previously it was already mentioned that it might make sense to ignore the outlier in the collected datasets. Without the outlier the final System Usability Scale shows a final result of 74.77, which is even better.

- **Requirement R2: High performance**
  Since the Raspberry Pi 2 always has the same hardware specifications for the performance of the system, it can be expected that the results for the performance do not vary a lot for all software parts, which run on the Raspberry Pi 2. As already explained earlier, the response time for the processed camera images can range from several milliseconds to roughly 100 milliseconds, which mainly depends on the configured resolution. For the whole system this processing time is fast enough to feel interactive. In contrast to that, the performance of the QR code processing time in VisiCut [Oster, 2011] highly depends on the system specifications. The performance on the slowest system, which was used in the development of FabQR, was still acceptable.

- **Requirement R3: Incorporation of QR codes**
  The implementation of FabQR successfully introduces QR codes into the FabLab workflow. This requirement is fulfilled by the software system.

- **Requirement R4: Anonymous service usage**
  No login is required for the FabQR system. The users and the projects are authorized by physical presence in the FabLab.

- **Requirement R5: Flexibility: Data representation**
  In the FabQR system the project data is stored as XHTML data and the index files are saved as XML data. Since XHTML is fully compatible with the XML standard, the flexibility in the representation of data is given and this requirement is met by the FabQR software system.

- **Requirement R6: Flexibility: API**
  The FabQR system itself uses its API to deal with the different tasks. The system does not require a specific client and other software applications can easily interact with the provided API as well, which adds a lot of flexibility to the whole system.

- **Requirement R7: System customization**
  Since FabQR is distributed as open source software, nearly all parts of the software can be modified and customized. Additionally, installation scripts and configuration files are provided for a simple installation and reconfiguration of the system.

- **Requirement R8: Raspberry Pi 2**
  The FabQR implementation is able to run on a Raspberry Pi 2. The software project visicamRPiGPU even needs to run on a Raspberry Pi 2 to provide hardware accelerated computations for huge performance improvements.

Chapter 5

Summary and Future Work

In this chapter the results of the bachelor thesis are summarized. Additionally, an outlook for possible developments in the future is given.

5.1 Summary and Contributions

First of all, the problematic situation of FabLabs regarding the lack of sharing knowledge and project documentations is explained. This unpleasant circumstance is the reason why the concept of the FabQR software system is developed. With the properties and characteristics of the different types of related work the requirements for the FabQR implementation are defined and it is explained why the currently available platforms for sharing knowledge and project documentations in the FabLab community are often not very well suited and it is outlined why these systems have some disadvantages. A documentation for the implementation of the FabQR system is created as well. The positive results of the evaluation show that the FabQR system has the potential to solve these issues in the FabLab community by incorporating QR codes for the identification of project data into the workflow of FabLabs.

5.2 Future Work

There are several additional ideas and features, which could be implemented in the FabQR software system in the future.

5.2.1 Performance Improvements

In VisiCut [Oster, 2011] several graphics operations need to be performed for the FabQR system. Especially the detection of QR codes, which have a rotation of roughly 45, 135, 225 or 315 degree is currently comparatively inaccurate and has a low performance because the required graphics operations for the detection of these QR codes are implemented with native Java operations at the moment. These graphics operations could be performed with the graphics processing unit of the system for huge performance improvements.

In this context the patterns for the detection of multiple QR codes in an image could be optimized as well. The current solution for this just chooses different areas of the image with some overlap and it scans these areas for a single QR code. For this algorithm there are many possibilities to tune the quality of the detection and the performance.

5.2.2 Projector Setup

The FabQR system includes software support for a projector setup as well. In this setup a projector is mounted right above the laser cutter and the current preview of the laser cutter schemes can be projected right onto the material, which is supposed to be cut with the laser cutter. This setup has special requirements for the hardware and particularly the ambient light. In future developments such a projector setup could be created and evaluated.

5.2.3 Stand-alone FabQR Client

Currently the only client for FabQR is integrated in VisiCut [Oster, 2011]. A stand-alone FabQR client could be developed because the FabQR system supports a variety of clients due to its APIs, which can be easily used by other software systems. Currently the FabQR system is mainly developed for the use case of the laser cutter, but such a stand-alone client could support other machines and tools as well.

5.2.4 FabQR Network

FabQR could be extended in such a way that multiple instances of the FabQR system can exchange data with each other. A network of multiple FabLabs could be established with that feature. Additionally, it adds redundancy to the system because the data of a project could be stored multiple times in the network. This feature can work similarly to peer-to-peer file sharing networks.

5.2.5 Website Features

Currently the website of the FabQR system provides a basic support for displaying, uploading and downloading project data. Additionally, QR codes can be sent as email attachments and they can be printed directly. This basic set of features can be extended by new features such as an improved style of the website or a search functionality for the projects.

5.2.6 Long-term Evaluation

Introducing a new system into already existing workflows and habits of users is always difficult because of various reasons. In the evaluation, which was presented previously, it was only checked, whether the system has an acceptable usability. Although the results look promising, this does not automatically imply that the FabQR software system will be used frequently by several FabLabs. Because of these reasons a long-term evaluation could be created to examine, if the system is actually used.

Appendix A

First System Draft and Concept

The final idea and concept of the FabQR software-system evolved in many discussions and conversations and changed a lot since the first system draft and concept, which is depicted in figure A.1.

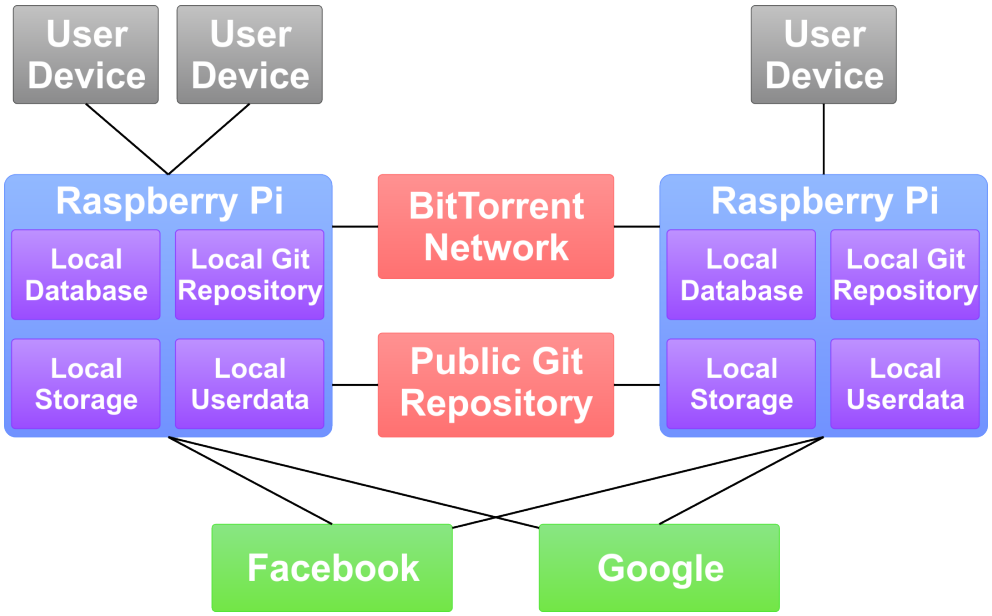


Figure A.1: First system draft and concept

Appendix B

QR Code Measurements

Several measurements of QR codes are taken to adjust the parameters of the generated QR codes correctly. The tables contain the raw measurement data. Each setting is measured twice and the distances refer to the maximum readable distance.

These datasets are measured with ZXing [ZXing authors, 2007] in version 3.2.1, 33 alphanumeric characters and the highest error correction level. A webcam with a resolution of 640 x 480 pixels is used and the QR codes are captured in front of a white background.

For the brightness measurements the QR codes always have a side length of 5.5 cm. For the side length measurements printed QR codes with a brightness of 100% are always used. A maximum readable distance value of 0.0 cm denotes that this QR code is not readable at all.

Brightness (%)	Distance 1 (cm)	Distance 2 (cm)
0	0.0	0.0
5	0.0	0.0
10	0.0	0.0
15	0.0	0.0
20	0.0	0.0
25	33.9	39.9
30	41.8	46.5
35	46.9	48.9
40	56.4	57.8
45	61.6	61.5
50	66.7	66.9
55	66.9	66.6
60	68.6	70.6
65	69.5	69.2
70	72.1	71.8
75	72.2	71.6
80	72.7	73.0
85	72.7	73.5
90	73.1	76.6
95	73.5	74.9
100	77.5	79.9

Table B.1: QR code measurements: Printed

Brightness (%)	Distance 1 (cm)	Distance 2 (cm)
0	0.0	0.0
5	0.0	0.0
10	0.0	0.0
15	0.0	0.0
20	0.0	0.0
25	0.0	0.0
30	9.2	8.9
35	11.8	11.7
40	11.9	11.8
45	14.8	14.6
50	19.1	20.0
55	24.8	23.9
60	26.2	25.9
65	29.1	28.2
70	29.0	30.9
75	37.6	34.1
80	44.4	41.1
85	52.9	52.2
90	53.7	56.2
95	56.4	54.1
100	61.1	59.4

Table B.2: QR code measurements: Smartphone min. brightness

Brightness (%)	Distance 1 (cm)	Distance 2 (cm)
0	0.0	0.0
5	0.0	0.0
10	15.8	18.7
15	23.7	29.6
20	51.0	51.3
25	58.8	58.0
30	58.4	58.9
35	60.9	61.7
40	61.5	60.2
45	58.4	59.0
50	52.0	54.9
55	51.2	47.4
60	43.6	44.7
65	43.1	42.3
70	28.1	29.1
75	27.2	26.0
80	23.2	24.4
85	21.7	20.3
90	19.1	18.2
95	16.1	17.2
100	15.6	16.1

Table B.3: QR code measurements: Smartphone max. brightness

Side length (cm)	Distance 1 (cm)	Distance 2 (cm)
2.0	17.6	17.8
4.0	49.5	51.8
6.0	78.8	82.4
8.0	97.0	101.4
10.0	120.4	121.2
12.0	142.8	136.8
14.0	166.2	166.6
16.0	189.6	191.2

Table B.4: QR code measurements: Side length

Appendix C

User Study

In the user study the participants were asked to fill out two pages with questions. The raw collected datasets and the corresponding questionnaire are shown on the next few pages.

In table C.1 the participants of the user study and their answers are depicted. The questionnaire items are numbered according to their order in the questionnaire, which means that Q1 refers to the first item whereas Q18 relates to the last item. The participants are identified by P1 to P12.

Table C.2 contains additional personal information of the participants.

The images C.1 and C.2 show the two pages of the questionnaire.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
Q1	++	+	++	+	+	++	-	++	+	○	○	++
Q2	-	○	++	+	++	+	--	++	+	○	○	++
Q3	++	-	+	○	++	+	○	○	○	-	+	++
Q4	++	+	+	○	++	+	+	○	++	-	+	++
Q5	+	++	○	+	++	++	++	-	++	-	-	+
Q6	○	++	+	○	○	+	++	++	+	-	+	++
Q7	++	++	++	+	++	++	++	++	++	+	+	++
Q8	++	+	+	--	++	++	++	○	++	+	++	+
Q9	+	--	+	--	+	+	-	++	++	○	+	+
Q10	-	○	-	++	-	--	-	-	-	-	--	○
Q11	++	○	+	○	+	--	+	++	++	+	+	○
Q12	--	-	--	++	-	-	--	--	-	-	-	-
Q13	+	+	+	-	+	++	+	+	○	+	+	○
Q14	-	-	○	+	--	++	--	--	-	-	-	○
Q15	++	+	+	○	+	+	+	+	++	+	+	-
Q16	--	-	--	-	-	--	--	-	-	-	○	○
Q17	+	+	+	-	+	++	++	++	++	+	+	+
Q18	--	-	--	+	○	--	-	+	○	-	-	○

Table C.1: User study data

Participant	Gender	Occupation
P1	Male	Student, Other
P2	Male	Student
P3	Male	FabLab staff
P4	Male	Other
P5	Male	Other
P6	Male	FabLab staff
P7	Female	Student
P8	Male	Student assistant
P9	Female	Student
P10	Male	Student
P11	Male	Other
P12	Male	Student

Table C.2: User study personal data

FabQR – Using QR Codes in the FabLab Workflow
User study (Page 1 / 2)

Personal information and opinion

Gender: ☐ Male ☐ Female

Occupation: (You can check multiple answers)
☐ Student ☐ Student assistant
☐ FabLab admin ☐ FabLab staff
☐ Research assistant ☐ Other

Question	*Strongly disagree*	*Disagree*	*Neutral*	*Agree*	*Strongly agree*
	--	-	o	+	++
I am an experienced FabLab user.					
I am an experienced user of the laser cutter and VisiCut.					
I share my FabLab projects with other users frequently.					
I would like to share my FabLab projects frequently.					
I like to use external platforms for sharing FabLab projects. (Thingiverse, Instructables, etc.)					
I would prefer to share my FabLab projects on a system, which is hosted by the FabLab.					
I think that sharing FabLab projects is important for the FabLab community.					
I am familiar with QR codes and I think that they are easy to use.					

Please turn the page.

Figure C.1: User study page 1 without data

FabQR – Using QR Codes in the FabLab Workflow
User study (Page 2 / 2)

System Usability Scale

Question	*Strongly disagree*	*Disagree*	*Neutral*	*Agree*	*Strongly agree*
	--	-	o	+	++
I think that I would like to use this system frequently.					
I found the system unnecessarily complex.					
I thought the system was easy to use.					
I think that I would need the support of a technical person to be able to use this system.					
I found the various functions in this system were well integrated.					
I thought there was too much inconsistency in this system.					
I would imagine that most people would learn to use this system very quickly.					
I found the system very cumbersome to use.					
I felt very confident using the system.					
I needed to learn a lot of things before I could get going with this system.					

Figure C.2: User study page 2 without data

Bibliography

Apache Software Foundation. Apache HTTP Server, 1995. Online, accessed: June 3rd, 2015. URL http://apache.org/.

Aaron Bangor, Philipp Kortum, and James Miller. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*, 4(3): 114–123, 2009.

Aram Bartholl. Dead Drops, 2010. Online, accessed: April 28th, 2015. URL https://deaddrops.com/.

BitTorrent, Inc. Sync, 2015. Online, accessed: May 1st, 2015. URL https://www.getsync.com/.

John Brooke. SUS - A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability Evaluation in Industry*, pages 189–194, London, United Kingdom, 1996. Taylor & Francis.

Center for Bits and Atoms, Massachusetts Institute of Technology. The Fab Charter, 2012. Online, accessed: July 3rd, 2015. URL http://fab.cba.mit.edu/about/charter/.

Creative Commons. Creative Commons, 2001. Online, accessed: July 9th, 2015. URL https://creativecommons.org/.

DENSO ADC. QR Code® Essentials, 2011.

Dominik Dzienia. PHP QR Code, 2010. Online, accessed: May 1st, 2015. URL http://phpqrcode.sourceforge.net/.

Fab Foundation. fablabs.io, 2014. Online, accessed: July 4th, 2015. URL https://www.fablabs.io/.

Dave Gandy. Font Awesome, 2012. Online, accessed: July 4th, 2015. URL http://www.fontawesome.io/.

Tim Hemig. FabCenter - Webapplication to support users and administrators of FabLabs with creating and sharing documentation, 2013. Diploma thesis, RWTH Aachen University, Aachen. Online, accessed: May 5th, 2015. URL https://hci.rwth-aachen.de/materials/publications/hemig2013a.pdf.

Jaquier, Cyril. fail2ban, 2004. Online, accessed: July 10th, 2015. URL http://www.fail2ban.org/.

Khronos Group. OpenMAX, 2004. Online, accessed: June 3rd, 2015. URL https://www.khronos.org/openmax/.

Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Lindsay Munroe, Sebastian Schrittwieser, Mayank Sinha, and Edgar R. Weippl. QR Code Security. In *Fourth International Workshop on Trustworthy Ubiquitous Computing (TwUC 2010)*, 2010.

Knova. TagMyDoc, 2015. Online, accessed: May 2nd, 2015. URL http://www.tagmydoc.com/.

LiveQoS. SuperBeam, 2014. Online, accessed: May 1st, 2015. URL http://superbe.am/.

Brent R. Matzelle, Andy Prevost, Jim Jagielski, and Marcus Bointon. PHPMailer, 2001. Online, accessed: June 28th, 2015. URL https://github.com/PHPMailer/PHPMailer.

Bakhtiar Mikhak, Christopher Lyon, Tim Gorton, Neil Gershenfeld, Caroline McEnnis, and Jason Taylor. Fab lab: An alternate model of ict for development. In *2nd International Conference on Open Collaborative Design for Sustainable Innovation*, 2002.

Stefanie Mueller, Pedro Lopes, and Patrick Baudisch. Interactive Construction: Interactive Fabrication of Functional Mechanical Devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Tech-*

nology, UIST '12, pages 599–606, New York, NY, USA, 2012. ACM.

Anu Määttä and Peter Troxler. Developing open & distributed tools for Fablab project documentation. In Sebastian Hellmann, Philipp Frischmuth, Sören Auer, and Daniel Dietrich, editors, *Proceedings of the 6th Open Knowledge Conference, OKCon 2011, Berlin, Germany, June 30 & July 1, 2011.*, volume 739 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

Leonhard Nobach. Kangee, 2010. Online, accessed: May 2nd, 2015. URL `http://getkangee.com/`.

ObjectPlanet, Inc. PngEncoder, 1998. Online, accessed: June 3rd, 2015. URL `http://objectplanet.com/pngencoder/`.

openFrameworks Community. openFrameworks, 2004. Online, accessed: June 3rd, 2015. URL `http://openframeworks.cc/`.

Thomas Oster. VisiCut, 2011. Bachelor thesis, RWTH Aachen University, Aachen. Online, accessed: May 3rd, 2015. URL `http://hci.rwth-aachen.de/visicut`.

Thomas Oster. VisiCam, 2013. Online, accessed: May 3rd, 2015. URL `https://github.com/t-oster/VisiCam`.

PHP Group. PHP, 1995. Online, accessed: June 3rd, 2015. URL `http://php.net/`.

Raspberry Pi Foundation. Raspberry Pi, 2008. Online, accessed: June 3rd, 2015. URL `https://www.raspberrypi.org/`.

Raspberry Pi Foundation. Raspbian, 2012. Online, accessed: June 3rd, 2015. URL `https://www.raspbian.org/`.

Software in the Public Interest, Inc. Debian, 1993. Online, accessed: June 3rd, 2015. URL `https://www.debian.org/`.

Peter Troxler and Harmen Zijp. A Next Step Towards FabML: A narrative for knowledge sharing use cases in

Fab Labs. *International Fab Lab Association, the 9th International Fab Lab Conference, Fab 9, Research Stream*, 2013.

Lode Vandevenne. LodePNG, 2005. Online, accessed: June 3rd, 2015. URL `http://lodev.org/lodepng/`.

Karl D.D. Willis, Cheng Xu, Kuan-Ju Wu, Golan Levin, and Mark D. Gross. Interactive Fabrication: New Interfaces for Digital Fabrication. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, pages 69–72, New York, NY, USA, 2011. ACM.

Patricia Wolf, Peter Troxler, Pierre-Yves Kocher, Julie Harboe, and Urs Gaudenz. Sharing is Sparing: Open Knowledge Sharing in Fab Labs. *Journal of Peer Production*, 5, 2014.

ZXing authors. ZXing, 2007. Online, accessed: May 1st, 2015. URL `https://github.com/zxing/`.

Index

ABGR 31
Apache HTTP Server 15, 68–69
API 21, 25, 64, 79

Color Format 31, 71
Concept 5–7
CPU 28
Creative Commons 52

Dead Drops 10
Debian 15, 63
Digital Fabrication 14

Evaluation 73–80

FabLab 1–3
fablabs.io 3
FabML 20
FabQR Web Service 26–28, 61–71
fail2ban 69
File Locking 36, 39, 70
Future Work 82–84

GPU 28
GPU Memory Split 36
GUI 41, 54

HDMI 71
HTML 68
HTTP 50, 70

Installation Scripts 63–64, 80
Instructables 22
Interactive Fabrication 14

LodePNG 16, 70

Marker Detection 26–29, 31, 38
Memory Leak 40

OMX ..... *see* OpenMAX
openFrameworks 15, 28, 37
OpenGL 29–31
OpenMAX 16, 31–34

PCB 2
Perspective Correction 26–31, 33
PHP 15, 61–71
PHP QR Code 16, 62
PHPMailer 16, 62
PLF 17
PNG 16, 70
PngEncoder 16, 51
Projector Support 50–51, 70–71, 82

QR Code 11–14, 24, 43–49, 79
QR Code: Error Correction Level 12, 62
QR Code: Measurements 57–60, 87–90
QR Code: Readability 55–60

Raspberry Pi 15, 25, 80
Raspbian 15, 63
Requirements 24–25, 78–80
RGB 31, 71
RGBA 31

Security 69–70
Signal Handler 37
SSH 69
Summary 81
synchronized Java Keyword 40, 54
System Usability Scale 73–78

Thingiverse 22

URL 26, 68
User Study 74–78, 91–95

VisiCam 18, 26–28, 38–40
visicamRPiGPU 26–37
VisiCut 17, 26–28, 41–54

XHTML 67, 79
XML 20, 24, 67, 79

YouMagine 22

ZXing 16, 43–47

Typeset July 12, 2015