NTNU – Trondheim
Norwegian University of
Science and Technology

Android object recognition framework

Mats-Gøran Karlsen

Master of Science in Computer Science
Submission date: July 2012
Supervisor: Alf Inge Wang, IDI
Co-supervisor: Tor Ivar Eikaas, Cyberlab

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Cyberlab, a small company developing simulators and games for education and training, is investigating new and innovative game concepts for mobile applications.

Incorporating parts from the real world into games can facilitate development of games that better immerse the player. Object recognition is one way of doing this and Cyberlab is interested in using this technology in their portfolio. The aim is to develop an object recognition framework for smartphones suitable for usage in real-world conditions.

The goal of this project is to continue the development of an existing Android object recognition framework to improve its performance, flexibility and usability.

Assignment given: 29. January 2012.
Supervisor: Alf Inge Wang, IDI.
External supervisor: Tor Ivar Eikaas, Cyberlab.

Abstract

This thesis is a continuation of the author's specialization project where the ultimate goal is to build an object recognition framework suitable for mobile devices in real world environments, where control over parameters such as illumination, distance, noise and availability of consistent network architectures are limited. Based on shortcomings related to object recognition performance and architectural issues the author's goal was to increase the flexibility, usability and performance of the framework.

Literature was reviewed on frameworks in order to discover useful techniques for development and documentation. Together with a re-introduction to the implemented recognition scheme an evaluation of the original framework artefact was performed with regards to the goals of this thesis. The results from the evaluation aided in finding an approach that balanced trade-offs between flexibility, usability, correctness and performance. By using proven framework development and documentation tactics from the literature study the author created a new iteration of the framework, improving upon the previous solution. The result is a stand alone artefact containing a hierarchy of software packages which divide functionality and offer customization using a combination of inheritance and components. The introduction of components hides domain knowledge and allows for easier reuse.

In order to improve recognition performance and framework flexibility the author added external server support for image information extraction as well as support for the usage of different feature detectors and descriptor extractors. Because of time constraints the author did not test these new feature detectors and descriptor extractors suitability or performance. This testing can now be performed by the customer.

In order to ensure proper correctness a lower bound on the image resolution is set at 600x600 pixels. Using properly built models correct recognition in about 90% of the cases is achievable. The added support for server side information extraction improves the object recognition performance by 42% in ideal conditions using the lower bound images. This improvement is still not enough to meet the performance criteria and combined with other issues results in the framework falling short of being ready to build production environment applications.

Sammendrag

Denne oppgaven er en videreføring av forfatterens spesialiseringsprosjekt, hvor det endelige målet er å bygge et objektgjenkjennelsesrammeverk egnet for mobile enheter i reelle miljøer hvor kontroll over parametre som belysning, avstand, støy og internettilgang er begrenset. På grunn av svakheter knyttet til objektetgjenkjennelsens ytelse og arkitektur er forfatterens mål å øke fleksibiliteten, brukervennligheten og ytelsen til rammeverket.

Det ble utført en litteraturstudie av rammeverk for å finne nyttige teknikker for utvikling og dokumentasjon. Sammen med en reintroduksjon av den opprinnelige objektgjenkjenningen ble det gjennomført en evaluering av det opprinnelige rammeverket. Resultatene fra evalueringen hjalp til med å finne en tilnærming som balanserte fleksibilitet, brukervennlighet, korrekthet og ytelse. Ved å bruke velprøvde rammeverksutviklings- og dokumentasjonsteknikker fra litteraturstudien har forfatteren generert en ny versjon av rammeverket. Resultatet er en frittstående artefakt som inneholder et hierarki av programvarepakker som separerer funksjonalitet og tilbyr skreddersøm ved bruk av arv og komponenter. Innføringen av komponenter skjuler domenekunnskap og åpner for enklere gjenbruk.

For å forbedre ytelsen på objektgjenkjennelsen og rammeverkets fleksibilitet har forfatteren lagt til støtte for informasjonsuthenting fra bilder ved hjelp av en ekstern server, samt støtte for bruk av andre detektorer og deskriptorer. På grunn av tidsbegrensninger har forfatteren kun testet serverløsningen. Testing av ulike detektorer og deskriptorer kan nå utføres av kunde.

For å sikre tilstrekkelig korrekthet er en nedre grense for oppløsning satt til 600x600 piksler. Ved bruk av gode modeller vil korrekt gjenkjennelse kunne oppnås i ca 90% av tilfellene. Den tillagte støtten for ekstern server vil kunne forbedre ytelsen med opp til 42% ved ideelle forhold og bruk av minimumsoppløsning. Denne forbedringen er likevel ikke nok til å nå kvalitetsmålene. Kombinert med andre problemer resulterer dette i at rammeverket i sin nåværende form ikke kan benyttes direkte til kommersielle formål.

Preface

This thesis is the result of the subject *TDT4900 - Computer and Information Science, Master Thesis* at the Norwegian University of Science and Technology (NTNU). The author is a game technology Master's graduate student at at the Department of Computer and Information Science(IDI) which is a subsection of the Faculty of Information Technology, Mathematics an Electrical Engineering (IME).

The author would like to thank Alf Inge Wang for valuable input into the technical aspects of this document. Thanks to the good folks at Cyberlab, Tor Ivar Eikaas and Frank Jacobsen, for their valuable input and continued support for this project.

Most importantly my greatest gratitude goes to you Eva and Careca, my two soulmates. Without your continued support and encouragements coming this far would not have been possible.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Project Background

1.1 Motivation

This project is an extension of the author's previous project in the course *TDT4501 - Specialization project*. The goal of that project was to start development of an object recognition framework for Android. The results from that project showed that further improvements had to be done both with respect to improving performance of the recognition as well the architecture of the framework. The author's involvement with the framework ends with this thesis and new developers will be in charge of its continued evolution. Therefore the improvements of the framework must prioritize usability, to increase the potential for usage, as well as knowledge transfer from author the new developers.

The customer, Cyberlab, does not currently employ experts in the domain of object recognition and can be seen as novices in regards to framework usage. With time however they will become experts and possibly start using the framework in ways the current developer can not foresee. One important question is therefore how the framework architecture can allow for such unforeseen usage. Documentation must be of such quality that it supports reuse and minimize architectural drift over time.

Improvement of performance, here recognition speed, can not be performed in isolation. The correctness of recognition must be preserved. Therefore it is important to identify the context of the framework, i.e dictate its scope of usage. The author has to make sure that the framework design fits the domain usage and fulfils application domain requirements in order to avoid expensive re-engineering.

1.2 Project Goal

The goal of this project is to continue development of the framework artefact delivered by the author in the course *TDT4501 - Specialization project* towards production environment quality. The evaluation of the results from the previous project

form the the initial input into methods for improving the flexibility, usability and performance of the new framework iteration. The author will perform an in depth study of frameworks in order to find systematic approaches and best practices for implementation and documentation in order to increase the chance of framework reuse.

1.3 Project Context

Video games have become a multi billion dollar industry and by the end of 2007 it surpassed the music industry in revenue[4]. The mainstream breakthrough of video games together with the introduction of the mobile phone and its ever increasing functionality, gaming on these devices is gaining interest[5], also by developers of serious games like Cyberlab[6].

Cyberlab has for some time now looked into ways to enhance the learning experience by melting virtual with real world elements. The type of applications that use images to achieve this inter world melting need automated methods to extract and combine information, object recognition. There are two ways of using this kind of technology; bringing virtual objects/information into the real world, augmented reality[7], or bringing the object into the virtual world. The challenges concerning object recognition are many since there is no single solution to solve all problems. Highly domain specific knowledge is therefore required to be able to utilize this technology fully. For small developers, such as Cyberlab that focus elsewhere, gaining competence in this field may not be viable. In order for small companies to be competitive, focus must be on creating content not gaining domain knowledge outside of their field of interest[8]. Frameworks help in this aspect; hiding most of the implementation specifics and speeds up the development process.

1.4 Stakeholders

This is a list of the stakeholders and their concerns regarding the project.

1.4.1 Author

- Mats-Gøran Karlsen

The research and artefacts produced will give the author a better understanding of best practices involved in building frameworks and the challenges involved in using object recognition in real world conditions on mobile devices with limited processing power.

1.4.2 Supervisor

- Alf Inge Wang

The main concern for the supervisor is the academic value of the thesis. Documentation must be of such nature that further research within the topic is possible.

1.4.3 External Supervisor

- Tor Ivar Eikaas

In relation to this thesis the external supervisor is the project sponsor, also known as the customer or the client. The main concern is suitability, usability and usefulness of the resulting software artefact in regards to the domain of interest.

Chapter 2

Research

2.1 Research Question

Continued software development effort requires that high level goals of the software artefact are clearly stated. This is important in order to avoid architectural drift and define precise functional and non-functional requirements which serve as input and evaluation of the quality in the development effort. This realization leads the author to the following research question:

How to evolve the object recognition framework towards production quality?

This question requires that there are clearly stated high level goals which the framework tries to solve so an educated decision based on the evaluation of the existing solution can be made on whether development effort should be spent on improving existing functionality or adding new features.

In order to answer the research question and hence identify where focus should be when modifying the framework, the author has identified a list of sub-questions divided into three categories:

- Framework:

  Q1: What is the difference between a framework and other reuse techniques such as libraries?

  Q2: Which software qualities should be prioritized?

  Q3: What are the greatest risks involved in developing frameworks in regards to usability, reusability and flexibility?

  Q4: Are there common evolutionary patterns in frameworks which help evaluate their maturity?

  Q5: Are there any commonly used design or development techniques used in framework development?

Q6: How should the framework be documented?

Q7: What separates a good software framework from a bad one, and how can the author analyse the quality of the existing solution?

- Object recognition:

Q8: How to define correctness in the context of object recognition?

Q9: How to define performance in the context of object recognition?

Q10: How is the object recognition performed in the existing framework solution?

Q11: How can performance be improved while still maintaining correctness?

Q12: Are there scenarios where the quality of object recognition can change?

Q13: Are there possible design solutions to improve performance?

Q14: Are there possible algorithmic solutions to improve performance?

2.2 Research Methodology

In scientific research it is important to use tools and models that enable possibilities to evaluate the quality of the work done. Software development is difficult for several reasons. Basili[9] identifies three main reasons for this when it comes to software and software engineering; inherent complexity, lack of well defined primitives or components of the discipline, and that software is developed instead of produced. He further identifies three available research paradigms related to software development:

- **Engineering method:** In order to evaluate the research questions a system is developed based on requirements and tested if the artefact produced solves these in an adequate way. Possible changes to the hypothesis are made and the artefact is changed accordingly in an iterative process to better suit the requirements.
- **Empirical method:** Here statistical approaches are used to evaluate hypotheses, where collection of data is used as verification. This method is suited for evaluating the resulting artefact against existing solutions.
- **Mathematical method:** This method is based on the analytical paradigm and often the formal way of performing experiments. A formal hypothesis is developed. Data is collected through various experiments and results from these are compared to empirical observations.

In this thesis the author will use the engineering method as the main research method. The existing framework solution will be evaluated and a new iteration will be created solving the identified issues.

In order to perform a thorough evaluation a literature study into the topics frameworks and object recognition is conducted. The research is guided by a series of research questions that aim at helping answer the main research question in section 2.1. The results of this study is published in respectively chapter 5 (Q1-Q7), chapter 6 (Q8-Q9) and chapter 7.

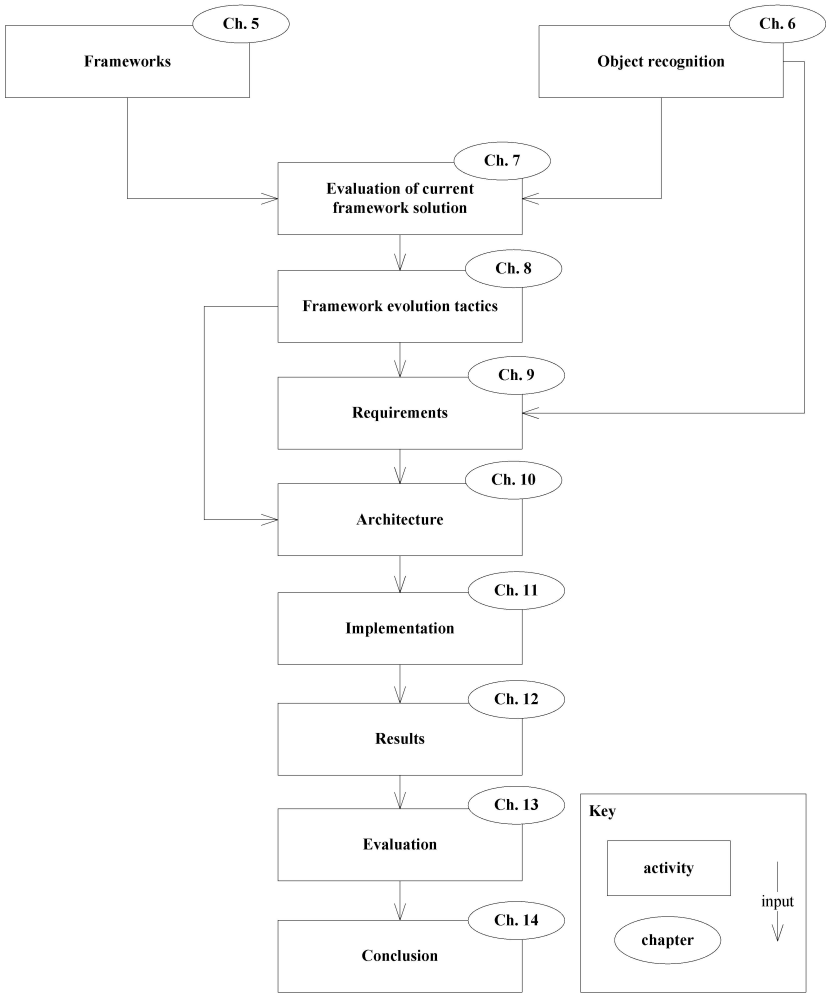


Figure 2.1: Research approach

The results from the prestudy will inform the author on the necessary steps needed to improve the framework in the direction stated by the goals of this thesis which is provided in chapter 8. Chapter 9 elicits the framework and example application requirements based on the evolution tactics and object recognition prestudy. Chapter 10 presents the architectural choices as a result of the requirements and the tactics chosen. Chapter 11 presents the example applications developed and

chapter 12 show the results of the quality of the object recognition and requirements involved in creating models of good quality. Chapter 12 also present the results of applying the performance enhancement tactic. Chapter 13 evaluates the success of the applied tactics. Chapter 14 sums up the results and concludes on the success of the project and points out future work.

Chapter 3

Previous Work

This thesis is a continuation of the authors previous work. A comprehensive explanation of the implemented object recognition scheme is given in chapter 6 and a presentation and evaluation of the resulting framework artefact is presented in chapter 7.

Chapter 4

Framework platforms

This chapter introduce distilled information on the Android development platform and the tools involved in development. The chapter ends with an introduction of the OpenCV library, which large parts of the framework rely on to perform basic object recognition and image manipulation tasks.

4.1 Android

Android is an operating system for mobile devices and is developed by the open handset alliance led by Google. Version 1.0 was released in May 2007 and has since then been under rapid development. The latest version of Ice Cream Sandwich (4.0.4) was released on March 28, 2012 [10]. The operating system was first intended for smartphones but the popularity of tablet devices has resulted in two different flavours of Android. Major versions starting with 2 is intended for smartphones and releases starting with 3 is intended for tablets or devices using larger screens. The reason for this distinguishing is mainly requirements related to different screen sizes. These differences are removed starting with version 4.

The system has widespread support with the backing from over 84 hardware, software and telecommunication companies. This is possible because the system relies on open standards. The software is released under the Apache License[10], allowing usage of the software for any purpose, redistribution, modification and distribution. The permissive nature of the license does not require modified versions to be distributed under the same license.

The system is based on the Linux-kernel. Middleware, libraries (such as Java-compatible libraries) and APIs are implemented in C and runs on an Apache Harmony based application framework. Applications are written in the Java programming language using a subset of the Java 6 SE API[11]. Swing, AWT and applet classes are replaced with custom libraries for graphics and mobile development. Source code is compiled, using just-in-time compilation, into Androids own Dalvik

executable and is run by the Dalvik virtual machine sitting on top of the operating system. Each application is a different user and runs on its own virtual machine (VM), inside an isolated process in the operating system. The security principle used by the operating system is *least privilege*[12]; each application is given access to the components it needs and no more.

4.1.1 Development target devices

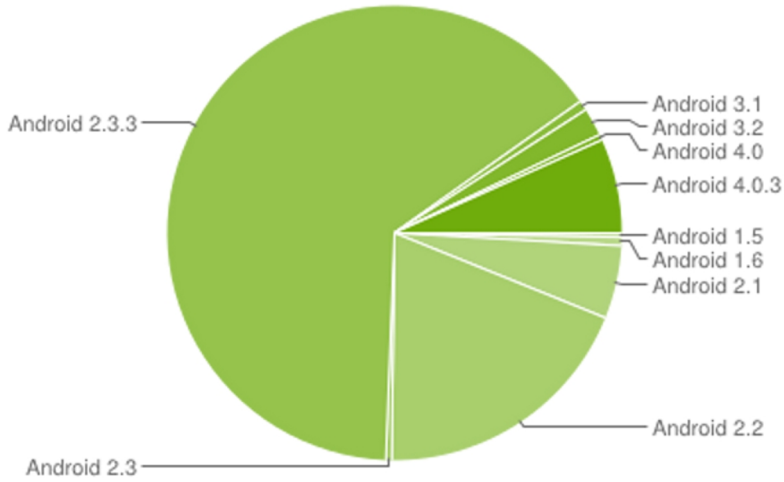


Figure 4.1: Distribution of Android devices as of June 2012[2]

[2] Developing software is a trade-off between using newer APIs, which are more mature and exhibit newer features, and availability of devices able to run it. As figure 4.1 shows, the majority of devices today use Android version 2.2 and 2.3.3. Newer versions of the operating system are backwards compatible and are able to run applications developed for earlier targets[13].

4.1.2 Tools

Development for Android is done using the Android SDK (Software Development Kit) which delivers all the necessary functionality to build applications such as Debugging, Profiling and Device access. This can be used as a standalone tool giving the developer the freedom of choice when it comes to third-party tools such as IDEs (Integrated Development Environment). Despite this, Eclipse is the recommended development environment and Android offers a ADT (Android Development Tools) plug-in for this tool. Installation instructions for these tools are given at the official Android development site[14].

Written Android development textbooks are starting to show up but the rapid development of the operating system renders most of them to quickly be outdated.

The official development pages[14] offer vast technical documentation which is precise, comprehensive and free. Lots of useful examples can be found here as well. The Android SDK also offers documentation through optional downloads.

4.1.3 Development devices

The SDK tools offers virtual devices through AVD (Android Virtual Device) which can be used to test and debug applications[15]. Each virtual device can be configured to emulate a vast amount of different handset configurations, enabling extensive testing. In order to do real world testing most Android systems offer the possibility of putting the device into debug mode. This allows the developer to connect the device to the computer through USB and install and test the application in real world conditions. This option offers the same possibilities as the virtual device[16].

4.2 OpenCV

Object detection is a huge field and requires extensive knowledge about different algorithms. In order to minimize the amount of critical code the framework relies on OpenCV for most of the image detection sub-tasks, image manipulation and matrix algebra.

OpenCV, initially developed by Intel, is an open-source computer vision library for extracting information from images[17]. It contains over 500 performance optimized functions. The majority is implemented in C/C++ and the objects themselves are mostly self contained. The aim of the library is to make computer vision accessible to programmers and users in the area of real-time human-computer interaction and mobile robotics. Users can both use the library for learning and because of its performance.

Areas addressed by the library are:

- object/human/face segmentation
- detection
- recognition
- tracking
- camera calibration
- stereo vision
- 2D/3D shape reconstruction

The library also provides data structures and compatible matrix algebra packages to support algorithms in the areas listed above, as well as important routines such as segmentation, feature detection and recognition.

4.2.1 Android and OpenCV

In 2010 the GSoC 2010 project ported the library to Android where. The library is compatible with Android 2.2, but versions greater than 2.3.1 is recommended[18]. The library can be obtained from the projects web pages[19] together with guides for installation. As of writing there are no official Android specific documentation except for installation instructions and sample code released as part of the library and short comments inside the library. The author is positive that this will change in the near future because of Googles active involvement in the project[20]. Until then, developers using the library must expect using a priori knowledge as well as trial and error. Official C/C++ documentation[21] is a good place to get an overview over available functions as well as looking at some sample applications to get a feel for the flow of control. The author recommends *OpenCV 2 Computer Vision Application Programming Cookbook*[22] for thorough explanations about usage of basic data structures in relation to common computer vision tasks. The examples in this book is also implemented in C/C++.

The official pages offers some best practices in order to ensure stability and performance[23]:

- The library communicates with the Android application framework through JNI (Java Native Interface) calls. These are quite expensive and should be avoided when possible especially in loops.
- Initialize data structures at the start of the application an reuse them later. The library extensively use pointers to avoid moving data back and forth.
- If possible, release data structures to the Android garbage collector to minimize application memory usage.

Part II

Prestudy

Chapter 5

Framework

Although frameworks, an object-oriented reuse technique, have been successfully used for some time a consensus on its definition in literature has not yet been reached. This may be an indication of the difficulty in grasping abstract concepts, which is one one of the key characteristics with reusable software. For people unfamiliar with frameworks this has led to confusion about what a framework is and how it differs from other reuse techniques. The following sections tries to clarify this in the context of this thesis, how to design, implement and document frameworks in order to support the goals of the thesis. The chapter informally answers research questions Q1-Q7 in section 2.1

5.1 Definitions

5.1.1 Framework

According to Fayad et.al[24] there are some reoccurring framework definitions found in literature. One is *"A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and design all or part of a system that is represented by a set of abstract classes and the way they interact"*. A second is *"A framework is the skeleton of an application that can be customized by an application developer"*. These definitions explain the structure and the intended purpose and they are not conflicting.

Johnson and Foote[25] define a framework as a partially complete application that can be customized to produce applications targeted for particular business units and application domains. The framework represents a reusable design of a system or subsystem that decomposes into a set of interacting objects. Johnson[26] summarizes the definition of a framework into an equation:

$$Framework = Components + Patterns \tag{5.1}$$

The framework consists of a library of reusable components that interacts in a predefined way described by the design patterns.

Aguiar and David[27] defines frameworks as reusable, semi-defined applications that can can be modified to produce custom applications.

Common to all of these techniques is the consensus that frameworks are a kind of design reuse documented in object-oriented code; a particular program is decomposed into objects and the communication pattern between these dictates the architecture of applications using the framework. There are examples of frameworks using non object-oriented languages[28].

Since the framework under development in this thesis is highly domain specific, the author will adopt the interpretation of Hautamäki[29]: *"A framework captures the programming expertise necessary to solve problems in a particular problem domain; it hides parts of the design that are common to all applications in that domain, and makes explicit the pieces that needs to be customized."*

5.1.2 Flexibility

Software flexibility is *"the ability of software to change easily in response to different user and system requirements"*[30]. In this thesis flexibility means to which degree the framework is customizable and relates to the range of possible application configurations the framework can support. Examples are application context, detector switching, support for external resources and dynamic application configuration.

5.1.3 Usability

This thesis will define usability as how easy it is for the framework user to accomplish a desired task and the kind of user support it provides[31]. The framework should try to:

- Minimize the learning curve.
- Minimize the impact of errors.
- Provide the necessary tools to support efficient use.
- Provide enough flexibility to support the needs of the users.
- Increase the confidence and satisfaction of using it.

5.2 Comparison to other reuse techniques

The most important parts of a framework are:

- The way that it is divided into its components.

- Reuse of the internal interfaces of a system and the way its functions are divided amongst the components.
- Secondary: Frameworks reuse implementation.

Frameworks share characteristics with general reuse techniques, in particular design reuse and object-oriented component reuse[24, 26, 29]. The following sections shortly introduce these two techniques and how frameworks differ along with some other key aspects of frameworks..

Early object-oriented reuse techniques relied heavily on usage of component libraries, implemented software that only require configuration. Their biggest advantage are their ease of use and they are considered the ideal reuse method since no knowledge about their inner workings are required. This is also their biggest disadvantage. Slight differences in requirements renders pre-existing components unusable and new ones must be added. Frameworks use component libraries as reusable code offering them a reusable context in which error handling, data exchange and interfaces are standardized. Frameworks also ease the the development of new components by providing specifications and templates for implementation. Because of the close relationship to component libraries, frameworks are often mistaken for being libraries. There is no strict line between a white-box framework and a simple class hierarchy and the latter has the possibility of becoming the first. In the simplest form a white-box framework is a program skeleton where the subclasses are additions to the skeleton. In most cases frameworks can be differentiated from libraries if there are dependencies among its components or if programmers complain about its complexity.

A component represents code reuse and a textbook represents design reuse. Since design reuse can be applied in more contexts it is applied earlier in the development process and therefore can have larger impact on the project. The problem is that this reuse is informal, require special tools and there are no standard design notation. Frameworks are similar to design reuse except it is expressed using a programming language and thus do not require special tools other than the standard development tool kit. The framework is represented by a set of classes (usually abstract), one for each kind of object. The interaction patterns between the objects define the architecture of the framework and is just as much part of the framework as the classes.

In traditional reuse the developer selects components from a library by writing a main program that call the components when necessary. This means that the developer decides when to call the components and she is responsible for the overall structure and flow of control in the program. In frameworks this is normally the opposite and it's called inversion of control or the Hollywood principle[32]. The main program is reused and the developer decides what components are plugged into it and makes new ones if necessary. The framework is then responsible for the overall structure and flow of control in the program.

Frameworks reuse analysis: The framework describes the kinds of objects that are important and provides a vocabulary for talking about a problem. Framework

experts will see problems in terms of the framework and will naturally divide it into the same components. This makes it easier to understand each others design because of the common vocabulary. This method of creating applications is opposite of standard software architecture approaches where the architecture is elicited from functional and non-functional requirements[33].

Analysis, design and code reuse are all important. In the long run analysis and design reuse provides the biggest pay off.

5.3 Classification

Fayad et al.[24] argues that although frameworks mostly are independent of domain they can be classified by scope and extensibility. Subsection 5.3.1 and 5.3.2 presents these two concepts.

5.3.1 Scope

Scope refers to where in the software/business chain the framework is being introduced. The scope is subdivided into the business and the application domain. The business domain is related to problem space (e.g. a banking application) whilst the application domain is oriented towards software design and addresses architectural issues. The three domains listed by Fayad et al.[24] are:

- **System infrastructure frameworks:** Used within a software organization and aims to simplify the development of portable and efficient system infrastructures (e.g. operating system, user interfaces).
- **Middleware integration frameworks:** Mainly used to integrate distributed applications and components into the software architecture allowing for modularization, reuse and seamless integration into distributed environments.
- **Enterprise application frameworks:** Address broad application domains and are cornerstones of enterprises business activities. Compared to the other two framework domains, enterprise frameworks are expensive to develop and/or purchase, but can provide a substantial return on investment since they support end user artefacts directly.

5.3.2 Extensibility

Regardless of scope, frameworks can also be classified based on the techniques used to extend them to achieve flexibility. This classification has two extrema:

- **White-box frameworks:** These rely heavily on inheritance and dynamic binding. Classes inherit functionality from their base classes using the "IsA"

relationship where the new objects are of the same type as their parents. Custom functionality is achieved by overriding predefined hook methods using patterns like template methods. The class name stems from the fact that the internals of the parent is visible to its children. The advantages of using inheritance is its straightforward use and the ease of modifying an existing implementation. This is also a downside since the developers need intimate knowledge of the internal structure of the framework. Another downside is that the objects lack run-time modifiability and this limits flexibility and ultimately reusability. Inheritance breaks encapsulation but this can be avoided by only using abstract base classes since these usually provide little to no implementation. White-box frameworks are widely used and they tend to produce systems that are tightly coupled to the frameworks inheritance hierarchies.

- **Black-box frameworks:** Functionality is added by composing objects using references and thus defines a "HasA" relationship. This is called object composition and new components are introduced into the framework by following standard interfaces defined by the framework. The components are integrated into the framework using patterns like strategy[34] and functor[35]. This mechanism is more flexible than inheritance because compositions can change dynamically at run-time. Objects involved are required to have well-defined interfaces so the user only need to understand the external interface. Other theoretical advantages are smaller classes and class hierarchies compared to white-box frameworks, where all functionality is available by using different combinations of components. In practice though the functionality is never rich enough and new components need to constantly be added. Black-box frameworks are generally structured using object composition and delegation rather than inheritance. The result is that black-box frameworks generally are easier to use and extend than white-box frameworks. On the flip side developing a black-box framework is harder since developers must define interfaces and hooks that anticipate a wider range of potential use-cases.

These two extrema represents the endpoints in an extensibility continuum where frameworks often contain behaviour from both. These gray-box frameworks are designed to avoid the disadvantages introduced by the two former. A good gray-box framework has enough flexibility and extensibility whilst still hiding unnecessary information from the application developer.

Looking at evolutional aspects of frameworks Johnson and Foote[25] notice that frameworks tend to evolve from white-box into black-box frameworks. The reason for this is that the design of the framework gradually becomes clearer and leads to components of higher functionality.

5.4 Strength and weakness

Failing to address the challenges in the following list[24], greatly increase the risks of failure when developing and using frameworks:

- **Development effort:** Creating high quality reusable frameworks that balances flexibility and usability for complex problem domains is difficult. Documenting the software process and design principles associated with developing them is key in order for them to be usable.
- **Learning curve:** Learning to use frameworks requires a lot of effort. It can take months to become productive and hands-on mentoring is often required in addition to providing documentation. If this extra cost cannot be amortized over several projects, a framework might never be cost effective. Another challenge is evaluating the suitability of the framework with the problem domain early in the project since this may not be apparent until after the learning curve has flattened.
- **Integratability:** Applications increasingly use several frameworks at once. If a framework is developed for isolated use, problems integrating it with other frameworks may cause unexpected problems ranging from documentation issues to architectural mismatches. Inversion of control is an essential feature of frameworks and integrating the event loops from two frameworks who are not designed to interoperate with each other is hard.
- **Maintainability:** Application requirements change frequently and so must frameworks. Framework evolution also result in change in applications relying on them. Maintenance includes both modification and adaptation and may involve both the functional and the non-functional level. Maintenance may take different forms such as adding/removing functionality or generalization. Maintenance requires deep knowledge of the inner workings of the framework and often application developers must rely on framework developers for this.
- **Validation and defect removal:** Well designed modular frameworks can localize the impact of software bugs. Despite this validating and debugging applications using frameworks can be difficult. There are several reasons for this.

    1. Well-designed frameworks typically create abstractions of application specific details using sub-classing, object composition or template modification. Although these techniques offer flexibility and extensibility, it greatly complicate their instances.
    2. Distinguishing bugs in the framework from bugs in the application are sometimes difficult for reasons such as requirement mismatch, overly coupled design or an incorrect implementation. Customizing components in a framework greatly increase the possible sources for errors.

3. Inversion of control and lack of explicit control flow. Single stepping through runtime behaviour with a debugger causes jumps back and forth between the framework code and the application code.
4. Developers may not understand the framework code, or simply do not have access to the code.

- **Efficiency:** Extensibility is enhanced by several levels of indirection. Examples are dynamic binding of objects by sub-classing and customizing interfaces. Generality and flexibility reduce efficiency in ways such as increased storage use due to lack of concrete data types, performance degradation due to additional overhead of invoking dynamically bound methods or lack of flexibility such as inabilities of placing objects in shared memory.

- **Lack of standards:** There are still no widely accepted standards for designing, implementing, documenting and adapting frameworks. Often vendors use industry standards in order to sell proprietary software. Standards are no guarantee that the software is suitable for usage in a domain.

In addition to the challenges mentioned in the list over, there are additional tradeoffs that must be considered when frameworks are involved. Subsection 5.4.1 and 5.4.2 summarize challenges and benefits mentioned in [24, 26, 29, 36].

5.4.1 Challenges

Reuse is valuable - but its not free. This section collects challenges mentioned in different articles and which must be considered before and while developing and using frameworks:

- Companies who want to take advantage of them must pay their price.
  - Frameworks share similar costs as other reuse techniques, they all require domain analysis and engineering, training of developers and inefficiencies are often introduced causing a big expense before benefits can be realized. Cost-benefit analysis should be performed before using them.
  - Benefits can only be gained over time and require up front investments.
  - Because of the iterative nature of development and difficulty related to implementation, frameworks are hard to create on schedule. Framework design can never be on the critical path of an important project.

- Frameworks are harder to learn than other reuse techniques:
  - Because of their customizability, reuse of high-level design and inter class dependencies, frameworks have complex interfaces and often a set of classes must be learned together.

  - Before being able to use a framework users need to spend a lot of effort on understanding its underlying architecture, its design principles and how to customize it.
  - The fact that frameworks embed design reuse in code makes them specific to a particular programming language. Programming languages are good at describing the static interface of an object, but not its dynamic interface. This means that it is hard for developers to learn the collaborative patterns of a framework by reading it.

- Frameworks require better documentation and longer training than other systems. In addition to this challenge, documentation is difficult due to the frameworks abstractness.
  - Despite providing excellent documentation a framework must be used before it can be understood.
  - Mentoring from framework experts may be necessary.

- They are hard to develop, requiring more expertise and time than developing normal applications. This is the reason why they are not used more widely.

- White-box frameworks can be difficult to use because they require application developers to write a substantial amount of code.

- Black-box frameworks can be limiting.

5.4.2 Benefits

This section summarize benefits related to framework development and use collected from different articles:

- Frameworks are powerful since they can be used for just about any kind of application and a good framework can reduce the development by an order of magnitude.

- Frameworks embed design reuse in code which makes them easier for programmers to learn and apply than other forms of design reuse documentation.

- They don't need any special tools other than the normal programming environment.

- It is possible to combine frameworks and domain specific languages making the framework capable of embedding domain knowledge. This allows developers to save time and money during development.

- Frameworks form a basis for the domain specific language and aids experts in communicating using a common language. Framework experts also tend to map a framework onto problems equally.

- Frameworks help coordinate people working on the same project because the problem domain is divided into objects with well defined interfaces and specific responsibilities.
- Uniformity reduces the cost of maintenance since programmers can move from one application to the next without having to learn a new design.
- Application developers can focus on their particular problem domain.
- Frameworks allow rapid prototyping.
- Modularity: Frameworks encapsulate volatile implementation behind stable interfaces. This improves software quality since the impact of design and implementation changes are localized.
- Re-usability: Frameworks offer generic components and leverage and store experience and the knowledge of domain experts. This avoids recreation and revalidation of common solutions to reoccurring problems which improves the programmers productivity while enhancing quality, performance, reliability and interoperability.
- Reuse enables open systems. Developers can use components from different vendors, which means reuse of interface design.
- Extensibility: Frameworks offer hook methods to extend its stable interfaces. These methods decouple stable interfaces and behaviour, qualities that are essential to ensure flexibility.
- Inversion of control: Canonical processing steps are customized by event handler objects which are invoked via the framework's reacting dispatching mechanism. Preregistered object hook methods perform the specific processing on the events.

5.5 Development

Developing frameworks are significantly more difficult than developing applications and although there exists several techniques for developing object-oriented software they cannot be directly used when creating frameworks. The limited available formal techniques, two are presented in section 5.5.4, are mostly theoretical. This is because many experts believe that frameworks cannot be a result of systematic design[24], but rather a result of evolution in a bottom up fashion. The development is informal and performed through an iterative process which requires both domain and design expertise. Reasons for the iterative development cycle are:

- **Domain analysis:** Explaining the domain is difficult and mistakes done are discovered as the system is built, causing iteration.

- **The only way to learn what changes is by experience:** The framework explicitly define the parts of the architecture that are likely to change. Components allow for this variation and are easy to change, changes to interfaces and shared invariants are hard.

- **Abstractions depend on the original examples:** Each example introduced makes the framework more generalized and reusable. Because of the complexity related to building frameworks there are limitations on how many examples can be used as templates. New examples with other requirements causes change in the framework.

One of the greatest challenges is to balance ease of use with theory of the problem domain while still providing enough features in order to be useful. In practice this involves finding the reusable design and the hot spots which makes the framework flexible. Gamma et al.[37] notice that design patterns can be useful in the framework development process and they argue that frameworks using design patterns are far more likely to achieve high levels of design and code reuse than ones that don't use them. The problem is that in order to use patterns one must know them.

A white paper published by Taligent inc.[38] note that from a users perspective an easy to use framework is one that performs useful operations with no extra effort on the user. The framework should work with little to no client code and doing small changes to the default behaviour should create new sophisticated solutions. If users don't understand the framework they will build their own solution. This view is supported by Booch[39] who has observed that a framework will not be used if the cost of using it is higher than the developers perceived cost of writing an application without it. This requires a balance between simplicity in interfaces and the frameworks flexibility. A completely flexible framework can be difficult to learn and difficult for developers to support. The size of the framework shouldn't be too big either and developers should break down larger frameworks into smaller, more focused ones. These smaller frameworks should be designed to interoperate which improve flexibility and re-usability.

In order to be successful a framework should be:

- **Complete:** The framework supports needed features and provide default implementations where possible.

- **Flexible:** The abstractions are applicable in different contexts.

- **Extensible:** Users can easily modify and add functionality by using hooks provided by the framework.

- **Understandable:** By following standard design, coding guidelines and documentation the user interaction with the framework is clear.

5.5.1 Hooks and Templates

An abstract class provides part of the implementation to its subclasses. The methods that allow for customization are divided based on how they allow this[24]:

- **Template methods** define the skeleton of an algorithm by leaving some steps unimplemented. Subclasses must provide the missing functionality. Each step is defined as a separate method that can be redefined by a subclass, without changing changing the overall structure.
- **Hook methods** provide a default implementation which can be overridden by the subclasses. This enables extensibility by allowing applications to extend stable interfaces. Hook methods systematically decouple the stable interface and behaviours of an application domain from the variations required by instantiations of an application in a particular context.

The specific methods are characterized according to subjective perspectives. A method can either be a hook method or a template method - or both, depending on the contents and the context.

A class that has a template method is called a template class and a class with a hook method is a hook class. The notions of the hook- template classes are similar to the methods naming them.

The idea behind the template/hook metaphor is that the template methods in the template classes invoke hook methods in the hook classes. A hook class can in principle contain several hook methods , which are invoked in turn by the template method.

5.5.2 Hot Spots

Fayad et al.[40]. Successful frameworks must identify hot spots, places in the framework which can be customized. Binding time characterizes the point of time at which an alternative is selected and bound to the hot spot. This can be done beforehand by the application developer or at run-time by the user. Each hot spot allows the user or developer to plug in an application domain specific class or classes when building a custom application from the framework. The class that is to be plugged in must be compatible, thus not every class fit into these spots. There are two interfaces involved, the services that a class must provide(interface called from the framework) and the ones served (interface called from the custom class) by the framework to the class. The services the class must provide is represented by the commonalities between the different possible classes and hence known beforehand. The ones provided by the framework are often less clearly described. In black-box frameworks this is not a problem since the framework supplies pre-fabricated-to-fit classes. The developer selects one of them, possibly modifying them by supplying some parameters and plugs them into the framework. In a white-box framework an alternative must be developed as required. This is difficult if the interface to

be called is missing due to the required knowledge about the frameworks inner workings. The result is that the developer must learn the internals.

Examples of hotspots are unimplemented methods in an abstract class or concrete methods that are overridden in order to change the default behaviour.

5.5.3 Contracts and Protocols

In object-oriented reuse, a contract is a specification of the obligations and collaborations that must be followed by participating objects jointly working together to achieve a goal[41]. The contract specifies a set of participating objects and their individual obligations. This involves the type constraints in their method signatures, the semantics of their interface methods and constraints on the behaviour as well as dependencies between objects. The contract further specify the preconditions required before contract establishment as well as the invariant that must be maintained by the involved parties.

A Protocol is a specification of an object such as the type of messages that is possible to send it[25]. The type, which is a protocol by itself, of the arguments in the message is also important. The interface between objects is defined by the protocol, and objects that have identical interfaces are said to be interchangeable. A set of classes that define the same protocol are said to be plug compatible. From these, complex objects can be created by performing object composition, a programming style known as building tool kits.

Standard protocols are powerful because of polymorphism. Languages not supporting polymorphism use methods with similar but different names. Each language have combinations of their own unique protocols as well as ones that are used in other languages. Experienced programmers rely heavily on protocols and new programmers find it much easier to read and create new programs once the these have been learned. Standard protocols ensure that new components are compatible with old ones.

Although contracts and protocols are important for specifying collaborative obligations and interfaces, they also play an important role in communication between programmers. The shared vocabulary is reusable and aids in learning new classes.

5.5.4 Process

According to Johnson[42] the typical way of developing frameworks is to build an application in a particular problem domain and divide it into reusable and non-reusable parts. A second application is developed using as much code as possible from the first application. The developers notice that the framework obtained from the first application is not very usable. This is fixed. Then a third application is developed reusing as much software as possible. Again the developer notice that the framework is not completely reusable. This is fixed and the iteration cycle continues.

Based on the observed pattern, Johnson suggest a way to develop frameworks using two base applications functioning as templates. The two templates should be similar and illustrate features that need to be supported by the framework. Common abstractions in the two applications are found and developers figure a way to decompose them into standard components. The development team should include people who have already developed applications in the domain. After this is done the project team is divided into a framework group and two application groups. The framework group elicits and adds functionality needed by considering different applications and provide documentation and training . The application groups reuse as much of the code as possible and provide feedback to the framework group. The groups should also consider applications that will break the framework in order to define the limits of the framework.

The number of programmers involved making a framework should be 2-4[38]. If there are more than 4, the framework should be further split into smaller frameworks. The reason for this is that the communication pattern between the participants become less effective as the team size increase and degrades significantly when there are more than 4 people. More than one programmer is recommended unless the single programmer both is an experienced framework developer and a domain expert. Splitting a framework into smaller pieces and mapping them onto interconnected teams presents challenges related to communication.

According to the white paper presented by Taligent Inc.[38] the framework development effort is divided into four tasks.

- **Identify and characterize the problem domain**
  - Process outline: The problem domain is analysed and frameworks are identified by the similarities in sub-problems solved by different applications.
  - Examine existing solutions.
  - Identify which parts of the problem the framework will solve: The problems are modelled and abstractions are found.
  - Identify key abstractions: Past experiences are used to identify abstractions and the framework design is initiated. If the developers are novices in the domain or haven't developed applications for it, existing applications should be examined and writing a few applications should be considered. These applications can be used to identify the abstractions.
  - Get input from clients and refine solutions.
- **Define the architecture and design**
  - Design interaction patterns between users and the framework: Developers determine classes and member functions that users will interact with directly. Interaction must be as simple as possible, documentation must provide requirements and constraints in the framework in order to prevent errors and wrong usage. The framework developers should

strive to reduce necessary client code by providing concrete implementations, minimize the number of classes that must be derived, minimize the number of functions that must be overridden and use illustrative class and function names.

- Provide tools: Tools provide a mechanism to reduce the learning curve and help manage the framework. These can be big applications with a high level of functionality or simple executables. The tools can be documentation tools such as hyperlinked documentation or editors that modify the framework.
- Apply recurring design patterns.
- Get input from clients and refine solutions.

- **Implement the framework**
  - Implement the core classes.
  - Test the framework.
  - Ask clients to test the framework.
  - Iterate to refine design and add features: The design process is iterative and beginning with the initial design users determine how the framework can be improved. Developers should try and refine the framework by adding more default behaviour and extend the ways users interact with the frameworks data structures. New features must be implemented, tested and verified. In this step developers reanalyse the problem domain refine design by testing, feedback and using their own knowledge.

- **Deploy the framework**
  - Provide documentation. Documentation and code comments are an especially important part of frameworks. If users don't understand the framework they won't use it. The documentation must clarify which classes that can be used directly, which classes that must be instantiated and which classes that must be overridden. Users are interested in solving problems and usually the architectural aspects are less important. A variety of documentation is best and which types are normally dependent on the stakeholders.
    - At a minimum sample programs should be provided. Examples are concrete and exemplifies the flow of control. During development test application must be developed and these can form the foundation for the samples provided. The applications should demonstrate how to use the framework in different contexts.
    - Diagrams of the framework architecture.
    - Descriptions of the framework.
    - Descriptions on how to use the framework.
  - Establish a process for distribution: Developers must plan how the finished framework will be distributed and supported.

  - Provide technical support for the users: The benefits of the framework can only be reached if users have additional support where answers provided by the standard documentation is enough.
  - Maintain and update the framework. In the early stages of the life cycle most maintenance is related to bug fixing and providing support for extra features. Later the framework must be updated in order to adapt to changing requirements. The impact of these on users must be minimized. Constantly changing frameworks will break existing applications and makes the framework difficult to use. It's better to add new classes than to change existing class hierarchies. The same is true for methods.

Although most research involve iterative framework development there are some systematic approaches. Roberts and Johnson[36] propose using a pattern language as the main driver behind the development process. Each pattern consists of:

- Name.
- Context.
- Problem description.
- Forces affecting the pattern.
- Solution and rationale.
- Implementation instructions.
- Examples and related patterns.

The patterns are related to one another and they are overlapping. The order in which they are applied can vary. Here the author shortly summarize the patterns. The link between the patterns are showed by numbers in parenthesis.

1. **Three examples:** The developer has decided to build a framework and the problem is how one should start developing it. The major force behind this pattern is the notion that people generalize from concrete examples and that having a framework, even if it only marginally help, ease development. The solution to the problem is that three applications should be developed. The argument for this is that frameworks cannot simply be created by sitting down and thinking about the problem, very few people have the insight to come up with proper abstractions. Domain experts won't understand how to codify the abstractions in their heads and programmers won't understand the domain well enough to derive the abstractions. Most of the time the abstractions won't become apparent until after the framework has been reused. The implementation can be done in two ways. Firstly one can simply build three applications, making sure that code and people are carried over from one project to the next. People tend to build applications without trying to

reuse code until they suddenly realize they should build a framework. Since they already have developed the three applications they often do a good job at building the framework the first time. Secondly one can prototype several applications without creating industry strength version of them. People using the framework have to refactor it but the result is a lot closer than using a single application as a starting point. The development of the applications only require standard development techniques. The developer should strive to create a system that is flexible and extensible. The initial version of the framework will probably be a white-box framework(2).

2. **White-box framework:** The second application is under development and the problems is choosing between inheritance or polymorphic composition as a reuse technique. Inheritance results in strong coupling between components, but lets the developer modify them which involves programming. Polymorphic composition requires the developer to know what is going to change. The solution is to use inheritance by generalizing from the classes in the individual applications using patterns like template methods and factory methods to increase the amount of reusable code from the superclass. The reason for this decision is that inheritance is a straight forward way of allowing users to change code in object-oriented environments and new classes are simple to create from the existing ones. The semantics of the inheritance is not important, just the ability to reuse code and this is enough to start using the framework, which allows the developer to see what is likely to change. Later the framework can be converted to a black-box framework by encapsulating code and parametrizing it. At this point the developer should have the proper insight to point out which parts that that are likely to change and which parts that are not. The implementation relies on programming-by-difference. Each time a class similar to one used before is developed a new subclass is created and their common code is stored in the superclass. After a couple of subclasses are created the developer will realize which parts are stable and these are then extracted into an abstract class. The same is true for methods where similarities are collected into one method and moved into an abstract class. As the development progresses component libraries(3) are created. Black-box frameworks(4) are similar but use a different context.

3. **Component library:**The developer is creating the secondary and subsequent examples based on the white-box framework(2). The problem is how one avoids writing similar objects for different framework instances. The forces behind this are that using the framework requires unnecessary effort since components are easier to use and upfront it is difficult to decide which components users will need. The solution is to start with a simple library containing obvious objects and add new as required. The reason for this solution is that once new objects are added some will be problem-specific and never be reused and will eventually be removed from the framework. These will however provide insight into the type of code users must write. The ones that are reused will form a basis for major abstractions within the problem

domain and should be converted into objects in the framework. These objects are the concrete subclasses of abstract classes. The component library can be created by accumulating all of the concrete classes created for the various applications. In the beginning all classes are put in the component library. Classes that are being consequently reused by applications should be included in the component library. Over time a great amount of the components will be refactored into smaller ones and disappear. As components are added to the library the developer will start to see recurring code that is shared among the components. This is the time to start looking for hot spots(4), places in the code that change from application to application.

4. **Hot Spots:** The developer is adding components to the library and discovers that code is being rewritten and wants to eliminate common code. The force behind the solution is to collect changes in a few locations making them easier to track down and change. This can be done by separating mutable code from immutable code and encapsulating the immutable code within objects where possible. Reusing objects are easier that reusing methods. When this code is encapsulated the variation can be captured by composing objects rather than creating subclasses and writing methods. This will simplify the the reuse process and show users where the framework developer expect the framework to change. Good names makes the understanding of control flow less important for understanding the framework. The implementation relies heavily on what changes and refer to design patterns such as the ones presented by Gamma et al.[43] to find solutions to these challenges. The creation of hot spots often result in finer grained objects(6) causing the framework to become more black-box(7).

5. **Pluggable objects:** The developer is adding to the component library and most of the subclasses differ in trivial ways and she wants to avoid having to add these kind of subclasses. Adding new classes increase the complexity of the system and using methods with complex parameters are difficult to understand and use. The solution is to design adaptable subclasses that can be parametrized with messages to send, index to access, blocks to evaluate or other characteristics that distinguishes a trivial subclass from another. Creating new classes that implement these small differences is overkill. Instead one should add parameters to the instance creation protocol which aids in reuse of the original class without resorting to programming. If the difference between objects are constants, symbols or class references the developer could create instance variables by passing a reference to the objects constructor. If the variation is a small piece of code block this can be passed to the constructor and stored as an instance variable. Pluggable objects are one way of encapsulating hot spots in the framework and the parameters can be supplied by a builder(8).

6. **Fine-grained objects:** The developer is refactoring the component library in order to improve reusability and is uncertain when to stop dividing objects. More objects in the system obfuscates understandability but simply

choosing objects that implement the required functionality is easier to use since no programming is required. The solution is to break objects into smaller ones until dividing the object would result in objects without individual meaning in the problem domain. Frameworks will be used by domain experts and the framework will be supported by tools creating the compositions automatically. It is therefore more important to avoid programming. Implementing this solution involves finding classes in the component library that encapsulate multiple behaviours and are candidates for further decomposition, replacing the original with the decomposed objects that recreates the behaviour. Code duplication and the need to create additional subclasses for each new application is avoided. As the objects become more fine grained the framework becomes more black-box(7).

7. **Black-box framework:** The developer is creating pluggable objects by encapsulating hot spots and creating fine-grained objects. The problem is choosing between inheritance and polymorphic composition as a reuse technique. The same forces apply here as in the white-box framework(2). The solution is using inheritance as an organizational tool of the component library but using composition of components when creating applications. The inheritance will create a taxonomy of the different parts of the library easing browsing. Composition allows for maximum flexibility. When in doubt use composition. The reason for this is that using components are a lot easier than inheritance since the user doesn't need to have intricate knowledge about the framework. The reason for using inheritance as an organizational tool is that people like to organize things into hierarchies. The implementation is performed by converting inheritance relationships into component relationships. Common code is pulled out and encapsulated in new components. The previous patterns will provide techniques for locating and creating new component classes. Black-box frameworks can be supported by builders(8). These can browse the framework and compose objects graphically.

8. **Visual builder:** The developer has a black-box framework and objects can be created by simply connecting objects of existing classes. The scripts connecting these classes are similar where only the objects involved being different and the developer wishes to simplify the creation of the scripts. The compositions of the objects are convoluted and difficult to understand, building tools are expensive and the users are rarely programmers. The solution is to make a graphical program that lets the user specify the objects that the application will consist of. After the user has combined the objects the application is automatically created. The reason for this is that the code is a script which is simple to create automatically. The graphical interface is user friendly and uses tools that are familiar to the user. The implementation differs and can be an application entirely consisting of dialogue boxes and browsers, but usually graphs are used to represent the complex relationships. The builder is a visual programming language and requires the developer to create language tools(9).

9. **Language tools:** The developer has added a builder(8) that creates complex composite objects and wonders how to inspect and debug these objects. The tools available are inadequate for dealing with specialized compositions and building good tools is expensive. The solution includes building specialized inspect and debugging tools. The implementation involves finding the portions of the framework that are difficult to inspect and debug. Usually this involves portions where patterns such as wrappers and strategies are used to compose objects. The tools should allow the user to evade portions of the composition that are unimportant.

Another systematic theoretical method of creating frameworks is presented by Koskimies and Mössenböck[44]. The goal of this method is to avoid early commitment to an application specific architecture and classes. This allows for usage of general design patterns in the early stages of development.

The design is performed in two phases. Firstly problem generalization is done by using a single example problem which is generalized into its most general form aided by answering the following questions:

- *Which concepts of the problem domain exist in variants and should be treated uniformly?*
- *Is it possible to find a concrete concept that can be generalized into a more abstract one. thus making the framework reusable in situations for which it was not originally intended?*

Secondly the framework design is considered in reverse order, where implementation starts at the most general level from which new and more refined frameworks are obtained. The result is a hierarchy of more refined frameworks where design experience and domain knowledge is used to obtain the hot spots in each framework. For each framework the following questions are asked:

- *Which parts of the system might change?*
- *Where might a user want to hook custom code into the framework?*

The last step is to implement the original example using the resulting framework, demonstrating that the framework is applicable in this representative case.

5.5.5 Implementation

Frameworks take advantage of all three distinguishing characteristics of object-oriented programming languages[24]:

- **Data abstraction:** Represents an interface behind which implementations can change.

- **Polymorphism:** Enables the ability for a single variable or procedure parameter to take values of several types. This enables mixing and matching of components, lets an object change its collaborators at runtime and makes it possible to develop generic objects that can work with a wide range of components.
- **Inheritance:** New components are easy to make since only the differing parts needs to be implemented.

Standard interfaces make it possible to mix and match components and to build a wide variety of systems from a small number of components. New components that meet these interfaces will fit into the framework. The component designers automatically reuse the design of the framework since they must adapt to the interface standard. As mentioned in section 5.5.3 protocols define interfaces and it is important to standardize them. Johnson and Foote[25] have developed a set of rules for helping find standard protocols. The following rules help minimize the number of different names and maximize the number of names shared by the classes, making the interfaces easier to learn:

1. **Recursion introduction:** If one class communicates with a number of other classes, the interface to each of these should be the same. An operation implemented by performing a similar operation on the components of the receiver should have the same name so readers of the program note their connection. The result is that a method for a message sends the same message to other objects. If the other methods are in the same class the method is recursive even if there are no real recursion, hence the name recursion introduction. This rule can help decide the class in which an operation should be a method.
2. **Eliminate case analysis:** Checking the class of an object is almost always a mistake. Class checks should be replaced with messages sent to the object whose class is being checked. Methods must be created in the class candidates so they can respond to the message. Case analysis of the values of variables is usually a bad idea also and it would be better to have a separate class for each kind of variable. Eliminating case in accessing instance variables is difficult but just as important.
3. **Reduce the number of arguments:** Messages with many arguments are hard to read. With the exception of the constructor, a message with half a dozen arguments should be redefined. A message with less arguments is more likely to be similar to some other message thereby increasing the possibility of giving them the same name. The number of arguments can be reduced by breaking the message into several smaller ones or by creating a new class that represents a group of arguments. Often there will be several kinds of messages that pass the same objects. This set of objects can be replaced by a new object.

4. **Reduce the size of methods:** Well designed methods are usually small and classes only consisting of small methods are easier to subclass. Large methods probably need to be broken into pieces. Often a method is split when a subclass is made. Most of the inherited method is correct but one part needs to be changed. Instead of rewriting the whole method it is split into pieces and the one piece that has changed is redefined. This makes the superclass even easier to subclass.

The important classes in the framework are usually abstract and the framework often contains a component library that contains concrete subclasses of these. Each object in the framework is described by an abstract class which is to objects what frameworks are to programs, a design[29]. The abstract classes are places where common code reside and they provide a means of creating standard interfaces that subclasses must follow and thereby offer flexibility to the software that use them. Abstract classes can have three types of methods:

- Abstract methods which are empty interface skeletons that the subclasses must provide implementations for.
- Template methods (ref. section 5.5.1) which are abstract algorithms implemented by abstract operations. These are partially implemented.
- Base methods, hook methods (ref. section 5.5.1), that are fully implemented.

Creating abstract classes are difficult, but one possibility is to generalize from a set of concrete classes[42]. Create an empty superclass and move common code and variables from subclasses into to the new superclass. Rename functions to give the classes the same interface. The code in the subclasses are usually similar but not the same. Abstract out the differences and move the rest to the superclass. The result is that subclasses contain more methods, but these are smaller. Look for commonalities and represent each idea once. Breaking large functions into smaller ones will help reduce bugs related to reusability. This holds true for classes as well, and large classes should be broken into smaller classes or components. Multiple inheritance can be simulated by dividing objects into smaller pieces which in many cases improves the design of the system. It is important to use descriptive names for classes, functions and variables in order to increase readability and it helps users notice connections.

The ability to perceive generalization is often related to experience in a domain and in general programming. A sign that abstractions have been found is that code size decreases. Similar to the the rules for finding protocols, Johnson and Foote[25] propose a set of rules for finding abstract classes:

1. **Class hierarchies should be deep and narrow:** Well developed class hierarchies should be several layers deep. Shallow hierarchies are evidence that change is needed but not how. The obvious way is to create new superclasses from a set of common classes. The classes are likely to provide

different methods for a specific message, but it is often possible to split a method and implement parts of it in both the superclass and the subclass.

2. **The top of the hierarchy should be abstract:** Inheritance for the sake of generalization or code sharing is a good indicator for the need of a new subclass. The methods and variables inherited from the superclass can be moved into its own abstract class. The abstract class becomes the new parent class of both the original superclass and the new subclass.

3. **Minimize access to variables:** The main difference between abstract and concrete classes is the presence of data representation. A class can be made more abstract by eliminating the dependence of the data representation. One way to do this is to access all variables by sending messages. Data representation can be changed by redefining the accessing messages.

4. **Subclasses should be specializations:** Inheritance can be used in different ways, but the ideal is specialization. The elements of the subclass is seen as elements of the superclass. Usually the subclass will not redefine any of the inherited methods but rather add new ones. A special case is creating concrete classes from abstract ones which is different from subclassing concrete classes. The abstract class works as a template forcing subclasses to implement certain methods. The abstract class may also implement some methods in an overly general fashion, so subclasses must redefine them. If a superclass can be switched with a subclass in all situations, the subclass is a specialization. There are cases where subclasses are not specializations and these occur in cases where the superclass is poorly designed. It might take a great deal of work to determine a proper design. In projects with strict deadlines the trade-off may be that the developer decides to use this poorly designed superclass and generates subclasses that might be more general than their parent.

Some programming languages, such as Java, separate interfaces from classes[24]. In these languages it is possible to describe an entire framework in terms of interfaces. These systems however can only specify the static aspects of an interface and lack the collaborative model of object interaction which are important in frameworks. The consequence is that Java frameworks tend to have both an interface and an abstract class defined for a component.

5.6 Documentation

A good framework relies on good designs and implementations, but this is far from enough. If the developers fail to convey their knowledge to maintainers and users the framework is useless. This implies that in addition to commenting code and using descriptive names accurate and comprehensible documentation is crucial to the success of the framework. The lack of good documentation is a major reason why many frameworks fail[45].

In order to fulfil their purpose, framework documentation must convey[46]:

- **The purpose of the framework:** A framework is a reusable design for solutions to problems in a particular domain. The problem domain itself must be described. This allows user to filter out frameworks that are inappropriate for their problem domain.
- **How to use the framework:** The documentation should show how to build applications. Most users want to know as little as possible about the framework and therefore they are not interested in the description of the design.
- **The detailed design of the framework:** The design includes the different classes in the framework as well as how instances of these collaborate. Interconnecting objects without understanding how they work is possible, but understanding the details unleash all the benefits associated with using frameworks (ref. section 5.4.2).

5.6.1 Stakeholders

One of the difficulties in documenting software is the fact that it must address different audiences that require different kinds of information. Butler and Dénommé[47] identify four types of stakeholders that framework documentation must address:

- **Regular re-user:** Customize the framework in ways expected by the developers. The documentation must address each expected customization.
- **Advanced re-user:** Customize the framework in unexpected ways by combining the hot spots(ref. section 5.5.2) in original ways. The documentation must convey the principles and constraints of the hot spots and how these collaborate.
- **Framework developer:** Evolves or generalizes the framework in order to extend its range of applicability and flexibility. In addition to the other issues of documenting the framework, architectural information must be conveyed by using similar techniques as in standard software development.
- **Developer of another framework:** Wants to learn the principles behind the flexibility of the framework. The design patterns must be documented.

5.6.2 Knowledge Presentation

Documentation related to software artefacts are divided into two categories based on the intended purpose of the documentation[27][48]:

Prescriptive
How to use the framework. The goal of this type is to provide knowledge on common issues related to the domain of the software and how they are solved

Descriptive
How is the framework built, i.e information related to the software architecture design. The goal is to transfer knowledge from the developers to advanced users seeking to either analyse, maintain, modify or test the software.

5.6.3 Documentation practices

Finding a formal way of documenting frameworks has proven to be difficult. Studies have been performed to evaluate the usefulness of different techniques, but these are mostly pragmatic and case sensitive. Some of the techniques that has proven useful in different contexts are shortly described here. The list is based on the short survey of methods in [47]:

- **Example application:** The framework is documented through source code for a working example application. Often this is the only documentation available to users. In order for this to be useful there should be several graded example applications illustrating the different hot spots (ref. section 5.5.2) in the framework. This technique is often viewed as minimal documentation and it is often used together with other documentation such as cookbooks.
- **Recipe:** Typical examples of reuse is presented as recipes. These are informal and often presented in natural language, supported by pictures and source code. A recipe often follows a certain structure with sections such as purpose, individual steps, cross references to other recipes and source code. Recipes are used by cookbooks.
- **Cookbooks:** A cookbook is a collection of recipes often supported by a guide to the contents either in form table of contents or by using the first recipe as an overview for the cookbook.
- **Pattern Language:** Informal pattern languages can be used for documenting frameworks using natural language. These must not be mistaken for design patterns such as those presented by Gamma et al. [43]. The patterns are based on the cookbook approach, where recipes are standardized by creating them using an Alexandrian form[49], and renamed to patterns. Each pattern consists of a problem description, detailed discussion of ways to solve the problem and a solution summary. Their organization follows a spiral approach, where the first pattern is an overview of the framework concepts and an introduction to the other patterns. The most frequently used patterns are presented early. Patterns including details related to collaboration and architecture are delayed as long as possible.

- **Interface Contract:** Refer to section 5.5.3.

- **Design Pattern:** Design patterns present solutions to common design problems. They describe the problem and its context, the solution and the consequences of applying it. Problems might be illustrated by examples. The solution is presented by describing objects and classes participating in the design, as well as the contracts (ref. section 5.5.3) that must be followed. A pattern may be supported by collaboration diagrams and concrete example solutions. An important part of the pattern description is analysis of benefits and trade-offs related to applying the pattern. In software documentation design patterns are effective at conveying micro-architectures[50], the details of interfaces and implementations of components, their composition and how they interact.

- **Motif:** Motifs are created using a specific template containing the name and intent, reuse description, steps involved in customization and cross references between motifs, design patterns and contracts. Design patterns presents the architecture and the contracts(ref. section 5.5.3) provide details about collaborations, dependencies and obligations between the participating objects relevant to the motif.

- **Framework Overview:** The context of the framework is the first step helping users reuse frameworks. Here the domain jargon is defined and the scope of the framework is presented such as what it solves and which parts are flexible. Example applications and an overview of the rest of the documentation can be presented here. The framework overview is often the first part in a cookbook.

- **Reference Manual:** An object-oriented reference manual describes each class. Normally the description involves presenting the purpose, responsibility, the role of its data members and information about its methods. Method descriptions involve presenting its functionality, its pre and post conditions and affected data members. In framework documentation this can be information about what role a class or method play in providing flexibility for hot spots.

- **Hooks:** Hooks provide solutions to well known problems[51]. They explain how and where a design can be changed, the requirements which must be fulfilled in doing this as well as constraints that must be followed. Together these explain the effects imposed on the system by the hook. A hook description usually consists of a name, the problem solved by the hook, method of adaptation, parts of the framework affected by the hook, other hooks required in order to use this hook, the components that participate in the hook, constraints and comments. Hooks can be organized using hot spots, and a hot spot tends to have several hooks within it.

5.6.4 A process

Although there exists a set of useful documentation practices (ref. section 5.6.3) there is a lack of well defined processes for documenting frameworks. Some of the reasons for this is a lack of standards, cost related to the time-consuming effort of creating documentation, the need for reusable documentation and the need to satisfy the different stakeholders (ref. section 5.6.1) among others.

Aguiar and David[27] propose a process driven by a set of patterns. The patterns are based on the Alexandrian form[49] and the result of applying a pattern is a documentation artefact. The goals of the pattern language are:

- Help developers document frameworks systematically.
- Increase awareness of typical problems faced when documenting frameworks.
- Expose trade-offs between cost, quality, detail and complexity.
- Provide practical guidelines on how to balance the trade-offs by finding the best combination of documents, activities and tools to the specific context at hand.

Figure 5.1, taken from the original article, gives an overview of the patterns involved and how they are interconnected.

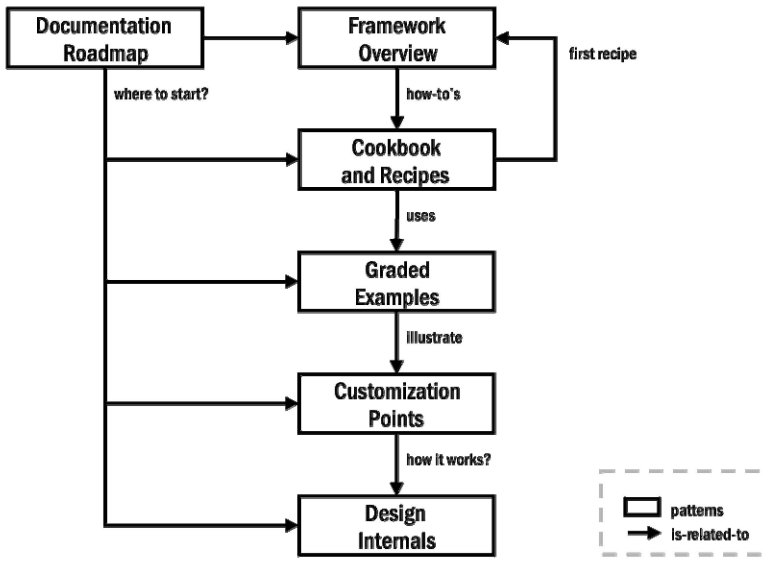


Figure 5.1: Documentation patterns and their relationships

As figure 5.1 shows, the pattern language describes a path commonly followed when documenting a framework. The relationship between the patterns form a documentation skeleton which is organized similarly to the patterns in the pattern language (ref. section 5.6.3). It also follow the recommendations given by Johnson (refer to the introduction of 5.6.3).

Here a short summary of each pattern in figure 5.1 is given:

- **Documentation roadmap:** In order to satisfy all stakeholders (ref. section 5.6.1), the documentation that comes with frameworks contain a lot of information presented and organized in different ways and at different levels of abstraction. The developers want to present this information in a user-friendly way where readers quickly can find the information they need. The solution is affected by the stakeholders and the way they reuse the framework. The problem can be addressed by providing a roadmap for the rest of the documentation, which reveal its organization and how different pieces of information fit together. This helps readers quickly find the entry points and the information they are looking for. The roadmap help navigation in both directions, enabling the reader to notice connections. In order to be effective the roadmap should be task-oriented and provide:
  - Topics organized by audience, kind of task and order of use.
  - Emphasize the main entry points and subordinate the secondary ones.
  - Links between the topics and the roadmap. The importance of each of these are dependent on the type of framework, the type of audience being addressed, the kinds of reuse explicitly supported in the documentation and which tasks are deemed most important.
- **Framework overview:** In order to be effective the documentation needs to convey information on the purpose of the framework, how to use it and how it is designed and implemented. In addition to this the framework must customize the presentation based on the type of stakeholder it wishes to address. The developers must provide a quick, but precise summary of all this information to all stakeholders so they can early on evaluate if the framework fit their needs. The kinds of information presented is affected by differing requirements related to completeness of information and the intended stakeholders. It is important that the information is easy to understand. This can be achieved by describing the problem domain and the range of problems the framework was designed to solve. Additionally a common vocabulary for the framework should presented. This can be done by defining a basic vocabulary supported by a rich set of concrete examples. In combination these describe the range of problems the framework covers and which parts that are flexible. The framework overview should refer and link to graded examples, documentation roadmap and cookbook and recipes.
- **Cookbook and recipes:** In order to achieve effective reuse, explanatory how-tos are important. Documentation needs to be effective in guiding users

to information that helps them learn using the framework. How the documentation should do this is affected by the respective stakeholder (ref. section 5.6.1), the correct balance between prescriptive and descriptive information(ref section 5.6.2), the most commonly used tasks in the framework, requirements for completeness, cost-effectiveness and ease of usage. The article authors propose creating one recipe for each framework customization organized in a cookbook. The cookbook and recipes can be switched for other approaches such as design patterns (For an explanation of recipes, cookbooks and design patterns refer to section 5.6.3). The most important part of the documentation is the kind of information conveyed. Mostly this should be prescriptive information (ref. section 5.6.2) that instructs users how to customize the hot spots(ref. section 5.5.2) in the framework. In addition architectural constructs and design details should be explained briefly when necessary. The rationale for this is that the best documentation for novice users are the ones that provides detailed instructions on using a specific feature without describing the theory behind it in detail. This implies that large parts of the framework must be created for this kind of use. The understanding of the theory behind the frameworks design and architecture grows with usage. Since recipes focus on how to use the framework it is useful to enhance their usefulness by referring to related customizable points and source code.

- **Graded examples:** Information conveying the potential areas of usage are of great importance to new users. These users are task oriented and the developers must convey graded and concrete information that helps potential users evaluate the appropriateness of the framework and help new users create artefacts from the framework with minimal effort. Therefore documentation must be task-oriented, convey information appropriate for the different stakeholders (ref. section5.6.1) and be cost-effective at the same time. The pattern propose providing a small but representative set of graded training examples to illustrate the problems that the framework solve as well as its features. Each example should illustrate a single new way of customizing the framework and the complexity of the examples should differ. The complete set of examples should cover the entire framework. Usage of hypertext links in source code connecting the example with useful information such as related examples as well as executable code are valuable for the overall understanding of the framework. Concrete examples are a perfect complement to documentation that more abstractly conveys the same kind of information. They also help the understanding of the flow of control between the object instances. By providing graded examples the information is separated based on the stakeholders framework expertise, helping both novices and experts.

- **Customization points:** The users need information on how to customize the framework and the documentation must be organized in such a way that it clearly convey which parts of the framework are customizable and how the customization is performed. How the information should be conveyed depends on the customization task imposed by the framework, the correct

balance of prescriptive and descriptive information, the kinds of stakeholders involved, the need for complete information and easy to use documentation. In order to provide this the documentation should include a list of all the hot spots and for each one in detail describe the hooks(ref. section 5.6.3) and the hot-spot subsystem (inheritance or object composition) that implements the flexibility. The list of customization points should be organized by different criteria in order to increase documentation usability. The lists helps users evaluate the suitability of the framework with regard to their problem domain and can do so with more confidence.

- **Design internals:** Advanced usage of the framework may not be well enough documented and therefore information regarding the design the framework must be available. As with the other patterns it is important to effectively convey this information. The documentation must inform users of the underlying principles and the framework's basic architecture, balancing the descriptive and prescriptive information while minimizing the design information complexity. This can be done by providing detailed information about the design internals of the framework, especially the parts involving hot spots. The information can be presented as design patterns (ref. section 5.6.3) or source code among others. Documenting architectures is difficult and time consuming.

Chapter 6

Object Recognition

The wide range of possible applications for object recognition has resulted in an area where extensive research has been made. The main problem is that of the generic vision problem:

"Given a sequence of images, for each pixel determine whether it belongs to some particular object or other spatial construct, localize all objects in space, detect and localize all events in time, determine the identity of all the objects and events in the sequence, determine relationships among objects and relate all objects and events to the available world knowledge."[52]

The following sections give a definition of object recognition and the concerns that must be accounted for in order to use this technology in real world applications. The end of this chapter briefly introduces the method originally used in the framework. The chapter informally answers research questions Q8-Q10 in section 2.1

6.1 Definitions

6.1.1 Object Recognition

Object recognition is classified as a subtask of computer vision, a field that covers core technology for automated image analysis enabling computers to see. The goal of this subtask is to identify one or several pre-defined learned objects or object classes in one or more images[53].

6.1.2 Correctness

In order to get measurable quantities of how well the object recognition works it's important to elicit requirements and constraints. Recognition in the framework is considered correct when it fulfils the requirements in the following list set forth by Treiber and Treiber[54], where the values are domain specific:

- **Accuracy:** The object position must meet a certain accuracy. This might be error bounds that must not be exceeded.

- **Recognition reliability:** A constraint on the error rate, i.e. the number of false positives, must be met.

- **Invariance:** The method used for recognition must be insensitive to some kind of variance. The choice of algorithm should strive to maximise inter-class variance while minimizing intra-class variance. The introduction of variance may be a part of the image acquisition process as well as or derived directly from the objects themselves. The application determines what kinds of variance the recognition scheme has to cope with:

  - Illumination: In grey scale images illumination strength, angle and colour affects image intensity. Regardless of illumination the object should be recognized.
  - Scale: The area of pixels in the image belonging to the object varies. Often this is a result of the distance between the object and the image sensor. The algorithm used should compensate for this variability.
  - Rotation: The rotation of the object is often not known. The system should be able to determine this.
  - Background clutter: Images may show more than just the object. Recognition should be unaffected by this.
  - Partial occlusion: In some cases the whole object might not be visible in the image. Recognition should to some extent be able to handle this.
  - Viewpoint change: Images are often 2D representations of objects located in 3D space. 2D appearance in images strongly depend on the position of the image sensor relative to the object. Invariance to this property is desirable but this is impossible for arbitrary shapes. Partial invariance is possible.

6.1.3 Performance

Performance can in the context of object recognition be interpreted as a measure of how well the recognition performs in terms of recognition(i.e. number of false positives), how fast the algorithm is expressed as asymptotic notation or expressed as **how much time has passed between the recognition is started until it finishes measured in seconds.** The author will use the last of these possible interpretations. The performance is used in pragmatic terms where the measure is device dependent. The author will use the Google Nexus S as a benchmark(ref. appendix A).

6.2 Taxonomy

In the general case object recognition, as the extension of the generic vision problem, has been proven computationally intractable by Tsotsos[55, 56] and Parodi et al.[57]. They observe however that depending on the size of the input, bounded visual search or small task guidance can turn an exponential time complexity vision problem into a linear one. This property has led to significant effort into developing approximation algorithms that solves specific problems, but not optimally. The different solutions can roughly be divided into two categories based on how information is extracted from the image[58, 54]:

- **Global, appearance-based methods:** Algorithms that aims at recognizing the complete object. These methods usually learn from sets of images where the object is present. Common global features are extracted and statistical classification techniques are applied.
- **Local, feature-based methods:** Focus here is on keypoints, areas in the image that have certain unique characteristics. Extracting these usually implies three steps:

  1. Extract and describe local features using models and test images.
  2. Select image features that match the model.
  3. Elect the best subset of keypoints (called inliers) that present the highest correlation with the model.

  The set of keypoints from the election process represent unique properties linked with the object in the frame, from which recognition can be performed.

Treiber et al.[59] argue that although global recognition methods generally are easier to implement and yield better performances than local ones, they are not well suited for real world applications. The global methods suffer from lack of discrimination and can only classify clearly distinct objects. Further strict quality requirements related to image segmentation and removal of clutter (e.g. background) make these methods unsuitable in cases where control over the environment is limited. Local recognition models on the other hand generally perform better in uncontrollable environments. The reason for this is their invariance to illumination changes and background clutter as well as being able to cope with intra class variance. This comes at a price though, higher computational complexity.

6.3 Conceptual Models For Local Object Recognition

The overall process of performing automated object recognition involves several steps which can be divided into subtasks. These subtasks often cross into other

areas of computer vision and are therefore applicable in fields outside local object recognition[53]. The following list gives a short overview of the subtasks involved for performing local object recognition.

- **Pre-processing:** Before the image can be used there may be certain requirements needed before the computer vision method can be applied. Examples are re-sampling or noise reduction.
- **Feature extraction:** Different feature extraction algorithms are used to obtain information from the image. These can be global features such as lines, edge ridges or more complex ones such as texture, motion or local features such as corners, blobs and points.
- **Detection/Segmentation:** Here decisions are made if points or regions in the image contain information that is relevant for further data extraction. Typical examples are selection of interest points or segmentation of image regions that contain a specific feature of interest. This step is used to remove irrelevant data and make the dataset smaller.
- **Model generation:** Models are created using techniques specific to the recognition scheme. These can be based on statistical data, complex 3D models or simple ones where the model is created based on information extracted from a single image. The models are stored in a database and used to determine if a specific object is present in an image.
- **High-level processing:** The segments are used to extract a model which forms the basis for the specific task performed by the application. Examples of tasks are verification that data in the model fit specific assumptions, classifying the model as being a specific object, the model being an object that fit into a predefined category or comparing different views of the same object.
- **Decision making:** Here the final application decision is made. Decisions can be if the model(s) pass criteria set by the application or if a specific object is recognized.

These subtasks can be divided into three processes: Information extraction, model creation and matching. Conceptual models of how these processes relate to the subtasks are given in figure 6.1 - 6.3. The models depicts a pipeline architecture[60], a sequential process where each subtask is considered a dedicated unit that solves part of the problem, commonly used in image processing[61, 62].

Figure 6.1: A model of image information extraction

Figure 6.1 shows how an image, $f(x,y)$, first is preprocessed before features in the image are being extracted. These features are either being enhanced by adding context information or removed by filtering. The result from these steps create a model of the unknown object, U, which can be used as a model in itself, used to match against a database of known objects or used further in construction of a model. A possible scenario of further model construction is shown in figure 6.2, where several unknown objects, U_i, are used to create one object model using a model generator (High-level processing). The framework in this thesis uses a variant of this technique.

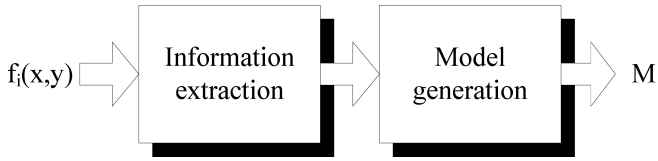


Figure 6.2: A possible paradigm for creating object models using several 2-dimensional images

A possible model for the matching process is shown in figure 6.3. The matching (High-level processing) is preceded by the information extraction process (figure 6.1). The Unknown object, U, and pre-existing database models, M_i, are fed into the matcher. The final result, $r_i(U,M)$, contains the decision made in the last step where an answer to the original query is given.

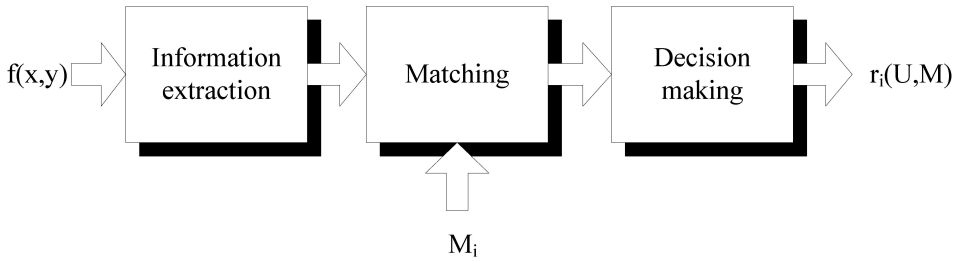


Figure 6.3: A model for performing recognition using an image with a collection of object models

6.4 The Current Framework Solution

The object recognition currently implemented in the framework (from the specialization project[63]) is based on a solution developed specifically for devices with limited computational power developed by Revaud et al.[64]. Information extraction is done by using local recognition methods and models are created by performing linear combination of two views of the object.

6.4.1 Information extraction

The recognition method presented by Revaud et al. does not pose any limitation on technique used for information extraction. However the implementation only supports the SURF[65] keypoint detector and description extractor, the same method used by the original authors. The main reason for this choice is SURFs invariance to skew, anisotropic scaling and partially perspective effects invariability.

SURF keypoint detector

Keypoint detection, also known as feature extraction, is in SURF based on convolving the input image with a Hessian matrix in order to find points of interest, also known as blobs, or in the case of extraction; keypoints . Convolution is a common used technique in image processing. When performed in the spatial domain the process is equivalent to moving a 180° rotated filter (mask. here: Hessian matrix), $w(x,y)$, over the image, $f(x,y)$, and calculating the sum of the products at each location, resulting in a new processed image[66]:

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{6.1}$$

At a given point $\mathbf{x}$=(x,y) in the image I, the hessian matrix H($\mathbf{x}$,σ) in the point $\mathbf{x}$ using scale σ is defined as:

$$H(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix} \tag{6.2}$$

where $L_{xx}(\mathbf{x},\sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image I at point $\mathbf{x}$. This is similar for $L_{xy}(\mathbf{x},\sigma)$ and $L_{yy}(\mathbf{x},\sigma)$. The result of convolving the image with the mask is that regions exhibiting sudden variations in pixel intensity when moving to one of their neighbouring pixels are highlighted by the mask indicating object edges[67], keypoints. Figure 6.4 shows that the different gaussian masks highlight different features. The combination of these in (6.2) creates a filter that highlights horizontal, vertical and diagonal pixel changes, edges.

SURF exploits the determinant of (6.2) to determine the scale of each keypoint.

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{6.3}$$

D_{xx}, D_{yy} and D_{xy} in (6.3) are the approximations of the Gaussian second order derivatives depicted in figure 6.4. The value in front of D_{xy} is a weight used to simplify the calculation. The magnitude of the determinant gives the level of change at that specific point. If (6.3) is zero the property of the determinant dictates that either two rows or two columns are equal[68]. This means that there are no edges. Big or small values indicate sharp edges. Determining the scale of these changes are done by applying masks of different sizes to the image. For comparison reasons the filter responses are normalized with respect to their mask size.

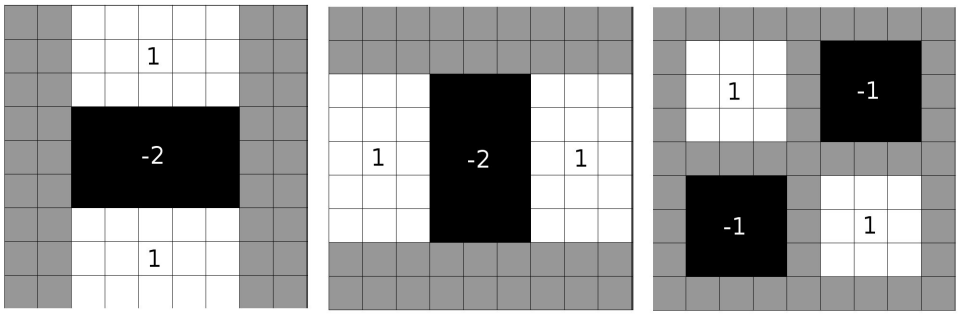


Figure 6.4: Approximate Gaussian second order derivatives. Left to right: $\frac{\partial^2}{\partial y^2}g(\sigma)$, $\frac{\partial^2}{\partial x^2}g(\sigma)$, $\frac{\partial^2}{\partial x \partial y}g(\sigma)$. Grey regions equals zero.

SURF descriptor extractor

The SURF descriptor extraction process consists of 2 steps:

1. Fixing a reproducible orientation based on information around the blob.
2. Construct a square region aligned to the selected orientation and extract the SURF descriptor.

Step 1 is achieved by convolving the area around the keypoint with Haar-wavelet masks(ref. figure 6.5) in a radius proportional to the scale at which the keypoint was detected. Once the wavelet responses have been calculated these are represented as vectors in a coordinate system spanned by horizontal responses along one axis and vertical responses along the other which is aligned with the two axis in the image. The dominant orientation is found by summing the responses within a sliding orientation window covering an angle of $\frac{\pi}{3}$. This ends in a new set of vectors and the longest of these represent the orientation.

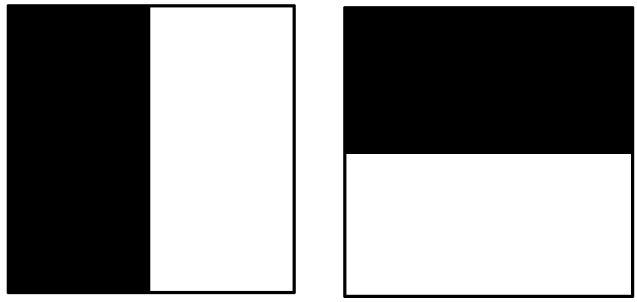


Figure 6.5: Haar wavelet types used for SURF. These respond to vertical and horizontal image features.

Step 2 is performed by the construction of a square region around the keypoint. This region is proportional to the scale at which the keypoint was detected and oriented in the direction found in step 1. Figure 6.6 shows that further subdivision is performed into regions of size 4x4 which again is split into 5x5. In each 4x4 region a vector, $\mathbf{v}$, is formed by performing Haar-wavelet filtering at regular intervals. In the figure, one of these regions are greyed out and callout A indicates 5 of these points. For each region and filter direction, the cumulative sum of the pixel values is calculated. The vector, $\mathbf{v}$, contains these sums and their absolute values:

$$\mathbf{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{6.4}$$

After performing this operation on all sub-regions a descriptor vector of size 64 can be extracted for each keypoint.

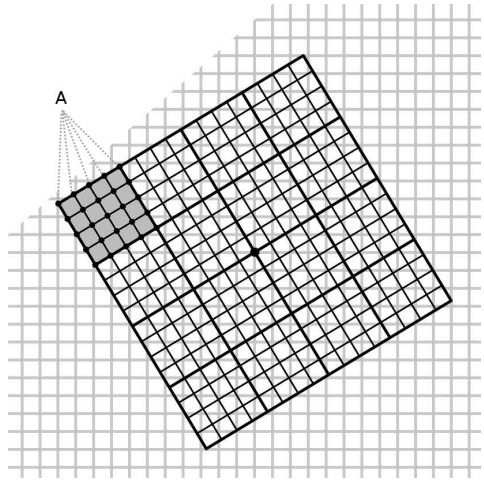


Figure 6.6: SURF descriptor extraction for one keypoint (pixels on grid intersections). Grey background grid shows the pixel orientation in the image. The black grid is rotated in the direction of the vectors in step 1 of the extraction process.

6.4.2 Model creation

Figure 6.7, taken directly from the original article[64], depicts that the overall model construction process is divided into four steps.

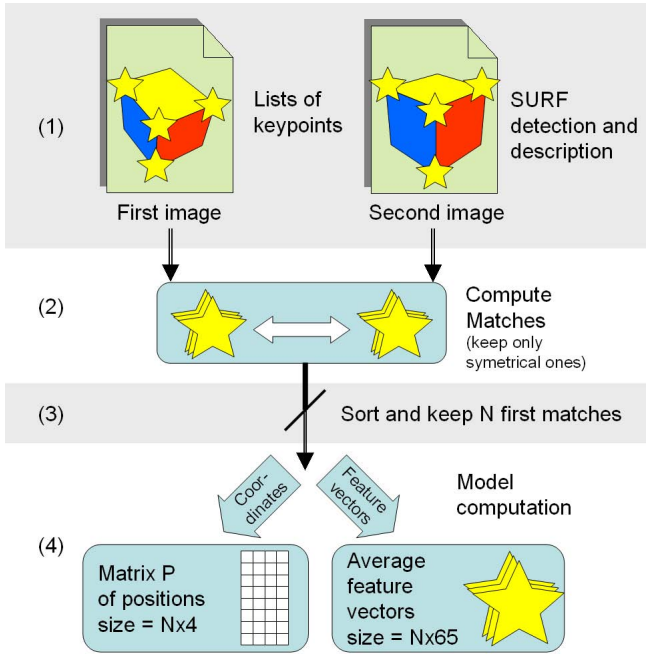


Figure 6.7: Summary of model construction

The idea behind this scheme is that an object can under different transformations be expressed as a linear combination of a small number of views. This makes it possible to obtain real 3D recognition and at the same time avoid the cost of creating a 3D mesh. This saves time both during model construction and recognition.

The first step is information extraction and is performed as described in section 7.2.1 for two images showing the object in different poses. The second step is matching analogous to the method described in section 6.4.3 with the addition of performing symmetric matching on the keypoints:

$$symmetricmatch(\hat{v}, V_B) = \begin{cases} match(match(\hat{v}, V_B), V_A) : \{v_A, v_B\} \\ otherwise : null \end{cases} \tag{6.5}$$

The third step is reducing the size of the model. Reduction is performed by sorting the matches using the following quality criterion where k and k' is a matching pair of keypoints:

$$score = k.strength \cdot k.scale \cdot k'.strength \cdot k'.scale \tag{6.6}$$

The scale dictates how far away the keypoint can be detected and the strength improves robustness to illumination changes. Optional filtering to remove outliers (spurious matches) can be performed before creating a subset containing the N best matches which is used for further model construction.

The final step is the creation of the model. The model consists of two parts; an averaged descriptor vector and a position matrix. The rationale for only keeping an averaged descriptor vector is the assumption that the matched and sorted descriptors returned from the previous step are nearly identical for both images. For each matching pair(k, k') the average of a descriptor is defined as:

$$M.V_i = \frac{k.V_i^1 + k'.V_i^2}{2}, \forall i \in [1, 64] \tag{6.7}$$

where 64 is the number of features used by SURF for each keypoint.

The model position matrix is created using the minimal model presented by Ullman and Basri[69]. The model states that every possible view of an object can be obtained from two of its views provided that the image is considered translucent, i.e. no occlusion.

O is a rigid object, an ordered collection of 3D points. P_1 is an image of O and P_2 is the image of O following a rotation R. R, U and the identity matrix, I, are 3x3 row vectors:

$$R = [r_1, r_2, r_3]^T \tag{6.8}$$

$$I = [e_1, e_2, e_3]^T \tag{6.9}$$

$$U = [u_1, u_2, u_3]^T \tag{6.10}$$

Any given 3D point $\rho = (x_1, y_1)$ of O is in P_1 expressed as:

$$\begin{aligned} x_1 &= e_1 \cdot \rho \\ y_1 &= e_2 \cdot \rho \end{aligned} \tag{6.11}$$

and in P_2:

$$\begin{aligned} x_2 &= r_1 \cdot \rho \\ y_2 &= r_2 \cdot \rho \end{aligned} \tag{6.12}$$

A third view, P_3, of the object is obtained by applying (6.10) to O. The point ρ in this new view is:

$$\begin{aligned} \hat{x} &= u_1 \cdot \rho \\ \hat{y} &= u_2 \cdot \rho \end{aligned} \tag{6.13}$$

Assuming that e_1, e_2 and r_1 span $\Re^3$ and a_1, a_2, a_3, b_1, b_2 and b_3 are scalars:

$$\begin{aligned} u_1 &= a_1 \cdot e_1 + a_2 \cdot e_2 + a_3 \cdot r_1 \\ u_2 &= b_1 \cdot e_1 + b_2 \cdot e_2 + b_3 \cdot r_1 \end{aligned} \tag{6.14}$$

$$\Downarrow$$

$$\begin{aligned} \hat{x} &= u_1 \cdot \rho = (a_1 \cdot e_1 + a_2 \cdot e_2 + a_3 \cdot r_1)\rho = a_1 \cdot x_1 + a_2 \cdot y_1 + a_3 \cdot x_2 \\ \hat{y} &= u_2 \cdot \rho = (b_1 \cdot e_1 + b_2 \cdot e_2 + b_3 \cdot r_1)\rho = b_1 \cdot x_1 + b_2 \cdot y_1 + b_3 \cdot x_2 \end{aligned} \tag{6.15}$$

Equations (6.15) hold for every point ρ of O. The scalar coefficients are the same for a given 3D position.

Let X_1, X_2 and $\hat{X}$ respectively be the vector of all the x coordinates of the feature points from P_1, P_2 and P_3. Y_1 is the vector from all y coordinates in P_1 and $\hat{Y}$ is all the y values from P_3:

$$\begin{aligned} X_1 &= [x_{11}, x_{12} \cdots x_{1N}] \\ X_2 &= [x_{21}, x_{22} \cdots x_{2N}] \\ Y_1 &= [y_{11}, y_{12} \cdots y_{1N}] \end{aligned} \tag{6.16}$$

$$\begin{aligned} \hat{X} &= a_1 \cdot X_1 + a_2 \cdot Y_1 + a_3 \cdot X_2 \\ \hat{Y} &= b_1 \cdot X_1 + b_2 \cdot Y_1 + b_3 \cdot X_2 \end{aligned} \tag{6.17}$$

Within the N-dimensional space spanned by (6.16), $\hat{X}$ and $\hat{Y}$ in equation (6.17) exists in a three-dimensional subspace spanned by X_1, X_2 and Y_1. Theoretically for any image taken of the object O, $\hat{X}$ and $\hat{Y}$ reside within this subspace and thereby any 2D view of the object O can be found using (6.17).

The 3 vectors in (6.16) are clustered in a new position matrix called P:

$$P = [X_1, Y_1, X_2] \tag{6.18}$$

In order for P, the model position matrix, to be used for translation, the model coordinates must be transformed into homogeneous coordinates[70]. This is done by adding a fourth unit vector T:

$$T = [1_1, 1_2 ... 1_N]^T \tag{6.19}$$

as a fourth row to (6.18):

$$P = [X_1, Y_1, X_2, T] \tag{6.20}$$

The original authors of the article further discuss methods for optimization of recognition speed by orthogonalizing P before storing it. This does not yield correct results, the matrix must be orthonormalized.

The memory cost of the model is small with a total of 68N floating point values:

- 65N values for SURF descriptors
- 3N values for positions matrix (T vector in (6.20) is dispensable)

The model creation is bound by the keypoints matching which is $O(M^2)$, where M is the average number of keypoints per image.

The model only takes orthogonal projection over a plane into account but seems to perform well despite this limitation.

6.4.3 Matching

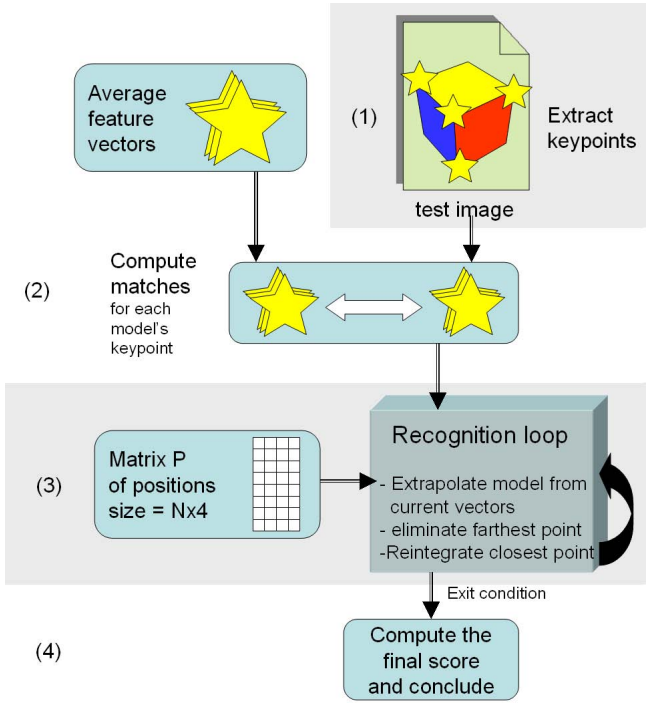


Figure 6.8: Summary of recognition process

Figure 6.8 is taken directly from the original article[64]. The recognition is done in 4 main steps.

The first step is information extraction and is performed as described in section 7.2.1. Next a match of the keypoints in the model with the image keypoints is

attempted using the descriptor matching presented in the following subsection. The euclidean distance (6.31) is calculated NxM times, where N is the number of model keypoints and M the number of image keypoints. For each model keypoint an ordered list, L_i, containing all matching image keypoints below a given threshold, T_{match}, is kept. The ordering is such that the top ranked keypoint in each list initially feed the recogition loop in step 3.

To extrapolate the object, the matcher uses a full allignment scheme presented by Ullman and Basri[69]. This enables the recognizer to handle object occlusion (here: unmatched model keypoints). Revaud et al. proposes that the position matrix, P, in (6.20) should be orthogonalized, this requirement is incorrect. In order for the optimalization scheme to work the matrix must be orthonormalized which can be proven by substitution in equation (6.25). Using this the quality of the matching is the correlation between a vector (descriptor from a keypoint), V, and the subspace spanned by P. The projection of V onto P yields a scalar product. The orthogonal projection gives the minimal distance toward a subspace[71] and hence this method can retrieve both coefficients, A and B. This enables the extrapolation of the model from the test image using (6.17):

$$\begin{aligned} A = P^T X \Rightarrow \hat{X} = PA \\ B = P^T Y \Rightarrow \hat{Y} = PB \end{aligned} \tag{6.21}$$

In order to handle occlusion, i.e. some rows in P and V is 0, a matrix P' is constructed using P and U (a NxN identity matrix deprived of rows corresponding to the missing keypoints):

$$P' = U \cdot P = [P'_1, P'_2, P'_2, P'_4] \tag{6.22}$$

A and B are found when equations (6.23) are minimized:

$$\begin{aligned} U \cdot \hat{X} - U \cdot X = P' \cdot A - U \cdot X \\ U \cdot \hat{Y} - U \cdot Y = P' \cdot B - U \cdot Y \end{aligned} \tag{6.23}$$

$$\Downarrow$$

$$\begin{aligned} A = [(UP)^T (UP)]^{-1} P^T (UX) \\ B = [(UP)^T (UP)]^{-1} P^T (UY) \end{aligned} \tag{6.24}$$

If P is orthonormalized the following optimization is possible:

$$(UP)^T (UP) = I_4 - [(I_n - U)P]^T [(I_n - U)P] \tag{6.25}$$

When $k < \frac{N}{2}$, where k is the number of missing keypoints, the right hand side of (6.25) is prefered since $O(k) < O(N-k)$.

The result of this scheme is a memory usage of 4N and an alignment scheme bound by $O(N)$.

After extracting the model the detection loop estimates the distance between the matched keypoint and the estimated model keypoint:

$$Dist = (X - \hat{X})^2 + (Y - Y^2)^2 - S \tag{6.26}$$

Dist is an N rows vector and the distance is corrected by S, a vector containing the scales of the currently image keypoints used:

$$S = [scale_1, \cdot, scale_N]^T \tag{6.27}$$

The correction done by S tries to account for the imprecision of its location and thereby remove outliers (false matches).

The image keypoint K_i' associated with the model keypoint K_i, where i = argmax(Dist), is disconnected from K_i. This results in a missing match and the point will be interpolated in the next iteration.

In order to recover a hypotethical matching for each lonely keypoint, K_i, select the image keypoint, K_i', in L_i which presents the smallest euclidean distance, d, to its predicted position if $d < \sqrt{distModel}$.

The loop exits if one of the following conditions are met:

- $distModel = max_i(Dist) < T_{dist}$
- $remainingMatches < T_{remaining}$
- number of iterations > 8N

Matching is based on the following score:

$$score = \frac{T_{dist}}{distModel} \cdot \frac{nbRemaining}{T_{remaining}} \tag{6.28}$$

$distModel$ and $nbRemaining$ (remaining number of matched keypoints) are exit parameters of the loop, T_{dist} and $T_{remaining}$ are predefined parameters.

Perfect matching yields an infinite score and bad matching gives scores toward 0.

Table 6.1 shows all the changeable matcher parameters.

Parameter	Where	Effects
Ratio	Model creation	Number of matched keypoints between the 2 object images, gives model creation filter more keypoints to chose from, number of spurious matches
T_{dist}	Recognition	Upper threshold for the distance between the worst aligned extrapolated model keypoint and image keypoint, acceptance of outliers, matching score
$T_{remaining}$	Recognition	Lower threshold for remaining matched keypoints after alignment, acceptance of outliers, matching score
T_{match}	Recognition	Match threshold between model keypoint and image keypoint

Table 6.1: Changeable parameters for mobile object recognizer

Descriptor matching

The descriptor matching is performed in the same manner as described by Bay et. al[65] and is shortly explained here for completeness.

Given two images, A and B, their set of descriptors are given by:

$$V_A = \{v_A : \text{descriptor of interest point in image A}\} \tag{6.29}$$
$$V_B = \{v_B : \text{descriptor of interest point in image B}\} \tag{6.30}$$

Matching between two keypoints is done by calculating the Euclidean distance[72], d, between their corresponding descriptor vectors, $\mathbf{v_1}$ and $\mathbf{v_2}$:

$$d(\mathbf{v_1}, \mathbf{v_2}) = \sqrt{(v_{11} - v_{21})^2 + (v_{12} - v_{22})^2 \cdots (v_{1N} - v_{2N})^2} \tag{6.31}$$

Equation (6.31) describes the length of the connecting N-dimensional line segment between the two points. The set of spatial distances between a keypoint in A and all keypoints in B is then given by:

$$D = \{\{d, v_B\} : \text{the euclidian distance, } d, \text{ between } \hat{v} \in V_A \text{ and all } v_B \in V_B \text{ ordered by } d\} \tag{6.32}$$

The nearest neighbour ratio matching strategy is applied. A match between two keypoints is found when the best match is within a given ratio, r, of the second best match:

$$match(\hat{v}, V_B) = \begin{cases} min(D) \leqslant r \cdot min(D - min(D)) : v_B \\ otherwise : null \end{cases} \tag{6.33}$$

Chapter 7

Evaluation Of The Current Framework Solution

This chapter evaluates the current framework solution and propose different tactics for improving the framework architecture and object recognition performance. The chapter informally answers research questions Q8-Q10 in section 2.1

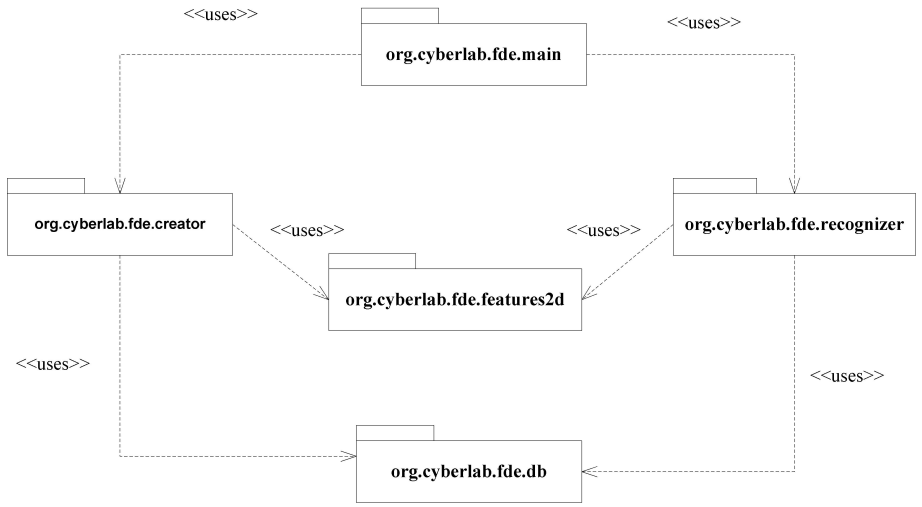


Figure 7.1: Package diagram of existing solution

The framework is currently distributed as an entity containing two prototype applications, a model builder and an object recognizer. These are meant to show the framework usage. Figure 7.1 shows the package organization:

main
The entry point of the prototype application. This package consists of one activity that enables the user to choose further activities.

creator
Implements the model creation prototype.

recognizer
Implements the object recognition prototype.

db
The implemented database handler used by *creator* and *recognizer*.

features2d
The developed framework code. A class diagram of the package contents is shown in figure 7.2 followed by a description of each class.

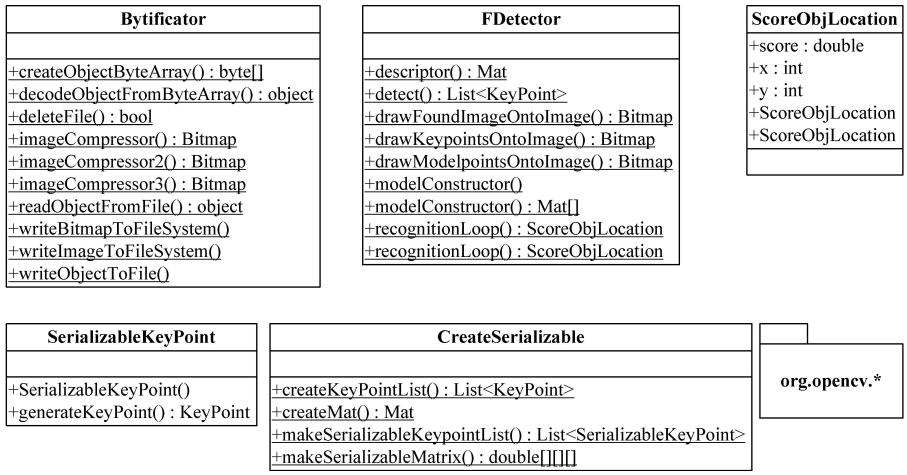


Figure 7.2: Overview of framework classes

Figure 7.2 shows the classes contained within the *features2d* package:

Bytificator
Provides methods for image compression (resizing) and file operations necessary to store data to the file system.

FDetector
Provides all necessary functionality for the object recognition and model construction as well as translation and display of the model onto the image.

CreateSerializable
Provides functionality for conversion between non-serializable data structures used by OpenCV and serializable data structures.

ScoreObjLocation
Provides functionality for storage of object score and location in the image where the object is detected. The latter only provide crude functionality and should be enhanced.

SerializableKeyPoint
Provides a serializable data structure which can be used for storage.

OpenCV
Is considered a part of the framework since this delivers important functionality and data structures used.

All classes in the framework except *ScoreObjLocation* and *SerializableKeyPoint* are static classes. The reason for this is performance. The application does not need to use unnecessary memory references to an instantiated class. Static classes also simplifies parallelization since they are stateless.

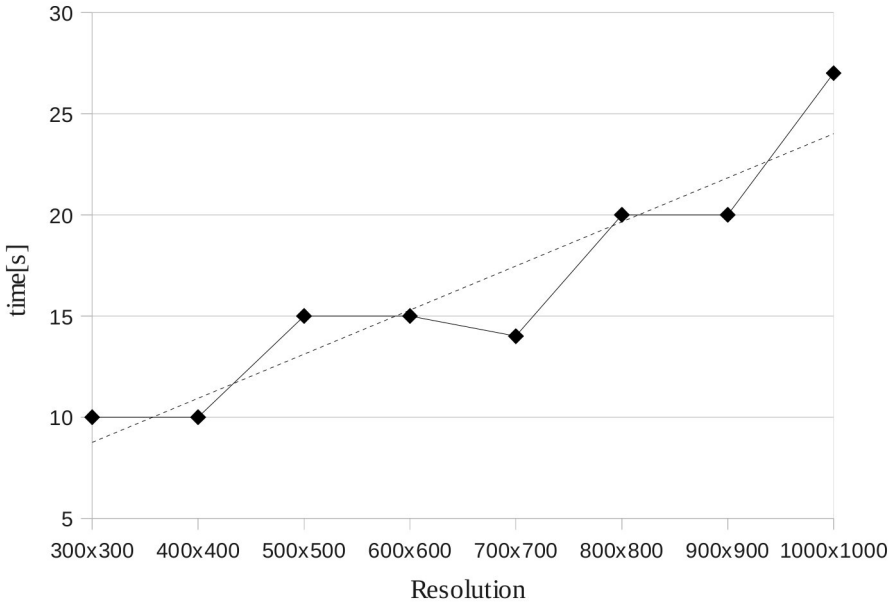


Figure 7.3: Recognition performance on a test object at different resolutions using a database with 25 models. (Based on isolated test. The complexity of the image influence performance and there are cases where recognition is twice as fast for each resolution.)

The original report[63] concludes that the correctness of the recognition is acceptable if the image input resolution is at 600x600 pixels but the speed of the recognition is too slow. At a resolution of 400x400 pixels the recognition of objects

in a small database containing 25 objects takes 5-10 seconds. The performance requirements are 1-2 seconds. The slowest parts of the recognition are related to feature detection and descriptor extraction.

Simple performance tests, using one test object, show that the speed of recognition correlates with the size of the image used for recognition (ref. figure 7.3). Feature extraction on larger images result in an increased number of detected keypoints, causing a ripple effect which degrades recognition performance.

The image size also influence the correctness of the recognition as figure 7.4 shows. Increasing the resolution from 400x400 pixels to 600x600 pixels results in the average number of correct matches going from 64% to 92%.

In some cases the matcher yields results where several objects in the database are matched against the test image. The object recognition scheme does not yet support recognition of several objects in one image. In cases where there is supposed to be only one object in the image it is possible to exploit this knowledge by iteratively decreasing the value of the T_{match} parameter until none or only one object is matched. This parameter (ref. table 6.1) controls the radius at which two descriptors are said to be matching. This tightens the bound on matching between object model keypoints and the test image keypoints. Originally the object matcher performs work linearly with the size of the database, adding this extension cause indeterministic matching performance.

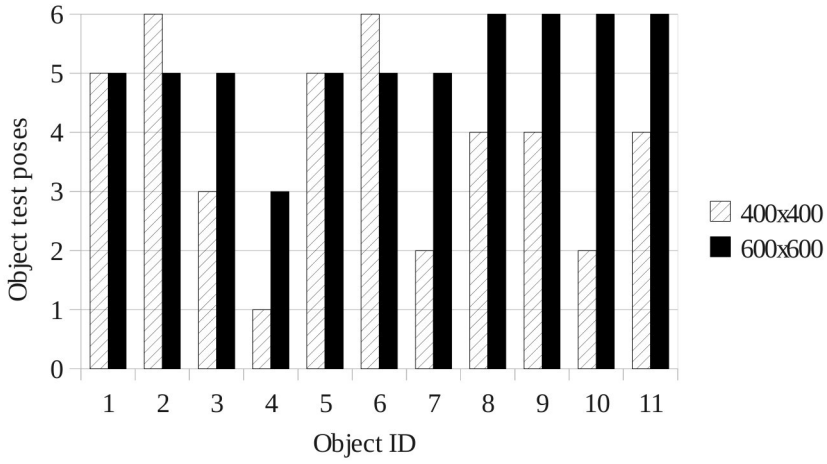


Figure 7.4: Recognition correctness on a test object at different resolutions using a database with 25 models. Each object was captured at the same distance. Recognition were performed from 3 different positions; to the left of the object, centered in front, and to the right of the object. For each position, recognition was performed twice based on how the smartphone was rotated; the same way as the images used for model construction and arbitrary rotation . Recognition were only performed once for each position i.e. there were no retries if recognition failed.

The choice of using the SQLite3 database provided by the operating system combined with storage of data in the file system proved to be one of the most complex parts of the implementation. The database was originally designed to hold several images with descriptors and keypoints plus support for several recognition schemes. The author first tried to store binary data into the database but this proved so ineffective for storage and retrieval that it hampered performance severely. Synchronization between files and database, an over-engineered database as well as the ability to edit objects resulted in method calls with many parameters and the object edit method in *ImagesDbAdapter*(fig. 7.2) is quite complex. Activity diagrams had to be used in order to get the implementation right. The author concluded that the database is over-engineered and could be simplified quite a lot. The choice to stick with the original design was based on the fact that the only purpose of the database was to provide storage for testing of the object recognition. A new storage solution must be designed for production use.

7.1 Architecture

This section analyze the framework based on the research in chapter 5 and 6.

The current state of the framework fails in both hiding common design knowledge from the user and explicitly stating which pieces that needs to be customized. As an example the model creation and recognition process, in figure 6.7 and figure 6.8, is implemented as part of the prototype application with support from the framework. These parts could be extracted from the applications and offered as hot spots in the framework through abstract activities or tasks.

The extensibility in the framework heavily relies on inheritance and is a sign of a young and immature framework. The current state is closer to a library where the user picks implemented functionality and calls these from her own main loop herself. This means that there is no inversion of controls and no default behaviour that simplifies the framework usage. The framework could be made more black-box by wrapping the OpenCV Mat and KeyPoints in self sustaining objects that represent the recognition schemes supported in the framework. Further by using generalization, common interfaces for these objects can be standardized allowing for flexibility where new recognition schemes can be added and mixed with existing ones. Customizable methods can offer default as well as custom framework behaviour. This allows for development of contracts and protocols that must be followed, and once learned should ease development of applications. This also allows for separation of functionality into a package and class tree structure which ease framework browsing and simplifies maintenance. The current framework only consists of a flat structure with all classes in it. The combination of these enhancements results in usage of all three reuse techniques in object-oriented software: data abstraction, polymorphism and inheritance.

The current framework does not take novice users into account because of the domain knowledge exposed due to the integrated nature of the prototypes. The

framework should be completely separated from the applications and distributed as a separate artefact (eases maintenance), using the applications as documentation. Further, the documentation process should take advantage of the knowledge in section 5.6 eliciting the purpose of the framework, how to use it and the detailed design. This ensures that concerns from all stakeholders are being taken into account; transferring framework developer knowledge through both descriptive and prescriptive methods. The documentation process should be performed systematically by applying relevant patterns in a similar fashion as presented in section 5.6.4, resulting in inter-connected documentation which enhances reusability.

The development of the framework followed the same pattern as the one proposed by Johnson (ref. section 5.5.4), where example applications forms the basis for the framework.

7.2 Object Recognition

This section elicits possibilities in enhancing the performance of the object recognition and matching.

7.2.1 Information extraction

Here the author presents possible solutions for extending and performing image information extraction in order to improve performance while still maintaining recognition correctness. The possible solutions are divided into two categories where the author looks into possible algorithmic solutions and framework design solutions.

Algorithmic options

The recognition scheme in section 6.4 can be customized by changing the information extraction solution. Because the framework heavily relies on the OpenCV library, the number of possible algorithms for information extraction is limited by the capabilities of these. Implementing new solutions requires too much effort due to challenges related to compatibility and maintenance issues with both the OpenCV library and the framework.

The available options are divided into feature detectors and descriptor extractors. By combining compatible detectors and extractors, new information extraction solutions can be constructed. Invariance to illumination changes and background clutter are properties shared by all local information extractors (ref. section 6.2). Invariance to partial occlusion and viewpoint changes are concerns mostly handled by the individual recognition scheme. These facts reduce the correctness concerns to scale and rotation invariance. Their performance evaluation is compared against SURF.

Keypoint detector	Correctness		Performance
	Scale invariant	Rotation invariant	
FAST[73]	No (Yes with pyramid adapter).	No.	Faster.
STAR[74]	Yes.	Yes.	Faster.
SIFT[65]	Yes.	Yes.	Slower.
ORB[73]	No. (Yes with pyramid adapter)	Yes.	Faster.
MSER[75]	Yes. Improves with pyramid adapter.	Yes.	Faster.
GFTT[76]	No.(Yes with pyramid adapter)	Yes.	Faster.
HARRIS[75, 77]	No.(Yes with pyramid adapter)	Yes.	Faster

Table 7.1: Available feature detectors in the OpenCV library. In addition to the ones listed a few adapters over detectors exist: GridAdapted, PyramidAdapted, DynamicAdapted. The performance column compares their generally considered speed against SURF.

All the different keypoint detectors in table 7.1, except for FAS, are suitable for information extraction in uncontrollable environments. How well they perform as part of the recognition scheme in the framework is largely dependent upon how repeatable their keypoint extraction is as well as the amount of keypoints extracted. Low repeatability and few keypoints would suggest challenges in both matching and model creation.

Descriptor extractor	Correctness		Performance
	Scale invariant	Rotation invariant	
SIFT[65]	Yes.	Yes.	Slower.
ORB[73, 78]	No	Yes.	Faster.
BRIEF[79]	No	No.	Faster

Table 7.2: Available descriptor extractors in the OpenCV library. Each of these can be wrapped in an opponent adapter (opponent color space instead of RGB). The performance column compares their generally considered speed against SURF.

As table 7.2 shows there are currently no alternative extractors currently available in OpenCV that can replace the SURF descriptor extractor. Calonder et. al[79]

point out that rotation invariance is less important because of the widespread use of orientation sensors in todays mobile devices. These can be used to control the object orientation in the images, allowing the usage of much faster extractors such as U-SURF (orientation dependent SURF descriptor extractor). Additionally the authors of BRIEF point out that a scale invariant version is planned in the future. The addition to scale invariance in BRIEF should also benefit ORB since the latter use a modified version of the BRIEF descriptor extractor (Sterable BRIEF)[78].

Design options

The author has identified the following architecture design options for improving the information extraction performance:

- Utilize available resources in multicore devices:
  - Segment the image into smaller regions and perform information extraction on the segments in parallel. Challenges related to this solution is how to solve feature detection on the segment borders.
  - Implement parallel SURF[65].
- Utilize the GPU through Renderscript[80] or OpenCL. Renderscript is officially supported by Android. OpenCL is available through some third party libraries such as the one developed by ZiiLabs [81]. The disadvantage is that both options only support a subset of the available Android GPUs (GPGPU development : General-purpose computing on graphics processing units).
- Use external server(s) to perform whole or part-whole information extraction. The challenges related to this solution is server availability, secure transmission of data, bandwidth requirements and error management because of unreliable network access[82].

7.2.2 Matching

The author considers identification and extension of new recognition schemes outside the scope of this thesis and will therefore only look into framework design possibilities.

Design options

The author has identified the following architecture design options for improving the matching performance:

- Utilize available resources in multicore devices:

  - Parallel matching is possible by dividing the model database into subsets and run matching on each subset. The challenges are how to handle matches in several subsets. The solution to is case dependent.
  - Utilize the GPU using Renderscript or OpenCL. The problems with these solutions are elicited in section 7.2.1.

- Use external server(s) to perform matching. The challenges here are similar to those presented in section 7.2.1.

Part III

Own Contribution

Chapter 8

Framework Evolution Tactics

The sections in this chapter present the author's tactics and trade-offs for achieving the goals of improving flexibility, usability and performance.

8.1 Development

The development will follow the same process as the author's previous project[63] which ended in the first iteration the framework. This follows the process and patterns presented in section 5.5.4. The author will develop new example applications that differ slightly from the original prototypes while reusing as much code as possible. This should lead to new abstractions.

The new goals are rooted in the prestudy of frameworks (ref. chapter 5) and the evaluation of the current framework solution (ref. chapter 7):

- Completely separate the framework from the example applications by moving it into its own Android library project.
- The examples will provide prescriptive documentation for framework usage (ref. section 5.6.2).
- Move toward components: Hide domain knowledge (here: OpenCV data structures) by wrapping them in framework objects and offer methods for creating them. This makes the framework more black-box and easier to use (ref. section 5.5.3).
- Use patterns such as Factory, Templates and Singleton to utilize well known patterns. These aid in describing framework behaviour and should ease usage, increase code reuse and increase the chance of framework success (ref. section 5.5.3)

- Use abstract classes and interfaces to establish contracts and protocols (ref. section 5.5.3).
- Provide abstract and concrete activities and tasks to help expose the framework hooks, templates and hot spots and provide default framework behaviour(ref. section 5.5.1 and section 5.5.2).
- Separate concerns: Move common functionality into packages to ease maintenance (debugging and adding new functionality) and increase flexibility (ref. section 5.5.5).
- Use the implementation tactics in section 5.5.5 such as data abstraction, polymorphism and inheritance to introduce a framework hierarchy which ease maintenance and usage.

8.1.1 Object recognition

This section presents the tactics and trade-offs for improving the object recognition correctness, flexibility, usability and performance.

Correctness

To achieve the correctness goals in section 6.1.2 the author puts a lower constraint on 600x600 for input image resolution. This should result in correct recognition in about 90% of the cases if the model is satisfactory and the image capture device is at arms length from the object. The length from the object is considered satisfactory for the customers application domain (mainly exhibitions).

Flexibility vs. Performance

In order to improve the object recognition flexibility the author chooses the following tactics:

- Modify the recognition scheme to allow for usage of all feature detectors and descriptor extractors in OpenCV. Not all are suitable (ref. section 7.2.1). The correctness and performance of these compared to SURF will not be tested in this thesis in favour of starting development of external server support for information extraction. The addition of this feature allows for increased flexibility.
- Add support for external information extraction. This is a trade-off between performance and flexibility. The author wants to use external support to increase recognition performance. The highest performance boost should be expected when the whole recognition scheme is performed on an external server because of its higher computational capabilities and less data transfer

(only the results need to be returned instead of a data structure). The reason for only adding information extraction is that the author wishes to improve flexibility. Information extraction is only seen as a step towards full external recognition. By gradually adding steps the application developer can be offered a series of alternative ways of adding external support. I.e. performing full recognition externally or doing any other variant where each step in figure 6.1-6.3 potentially can be performed on either the device or on the server.

Usability vs. Performance

- Simplify the model storage solution. The OpenCV Android library does not support model storage directly. By using the Java Native Interface the author can access the OpenCV C++ operations that allow for data storage in human readable files. Accessing these will greatly improve storage performance over the existing solution since the current framework converts between OpenCV and Java data structures. Using the OpenCV solution ensures cross platform compatibility.

8.2 Trade-offs

This section describe the rest (ref. section 8.1.1) of the trade-offs that needs to be considered.

8.2.1 Flexibility vs. Usability

One of the major concerns in enhancing the framework flexibility is that it affects usability. The author will try to avoid negative effects by wrapping domain knowledge into components thus allowing for flexibility while enhancing usability. This however affects framework maintainers negatively since debugging and black-box components put higher demands on their programming and domain knowledge skills. This is considered secondary to framework usage.

8.2.2 Usability vs. Performance vs. Correctness

Increasing performance of the recognition scheme using an external server can complicate application development and deployment. The framework will add support for automatic fallback. Applications can use default implemented behaviour that allow applications to run in degraded mode; if the network or server fails the application will further compress the image and perform all operations on the device. This will favour performance and flexibility over correctness. The application will still be usable if it suddenly is isolated. The degraded recognition correctness will put stricter demands on the object capture process (Findings show (ref. section

12.3) show that lower resolution affects the invariance to rotation and the allowed distance to the object).

8.3 Documentation

The documentation is crucial for the success of the framework. The author planned on using the systematic process presented in section 5.6.4. This is not possible however with the limited time available. Therefore the author decided to provide all the documentation necessary through this document, the example applications and the code documentation (Javadoc and Doxygen).

The author recommends that the framework maintainer separates this document from the framework by extracting the framework documentation information and combine it with the code examples and code documentation from the applications by applying the systematic process in section 5.6.4.

Chapter 9

Requirements

This chapter present the requirements for the framework and the example applications. Some of the sections are taken from the author's previous report[63] and they are included since requirement specifications are an essential part for software artefacts. These requirements act as guidelines for application design, architecture and implementation.

9.1 Framework

This section elicits the functional and non-functional requirements for the framework.

9.1.1 Functional Requirements

Table 9.1 shows the functional requirements together with a unique *ID*, a short *description* of the requirement as well as a *priority* of either high (H), medium (M) or low (L). The ID is used for easy referral to each requirement. The description explains the requirement itself. The priority indicates the relative importance of the requirement.

ID	Description	Priority
ORR1	Support for recognition of 2D objects	H
ORR2	Support for recognition of 3D objects	H
ORR3	Identify recognized object	H
ORR4	Identify location of recognized object in image	M
ORR5	Identify pose of detected object in image	L
ORR6	Enable retrieval of object information	L
ORR7	Object recognition is invariant to rotation	H

Table 9.1: Functional requirements for object recognition framework

ID	Description	Priority
ORR8	Object recognition is invariant to scale	H
ORR9	Object recognition is invariant to illumination	H
ORR10	Object recognition is invariant to noise	M
ORR11	Object recognition is invariant to occlusion	L
ORR12	Model data structure is suitable for persistent storage	H
ORR13	Build object model	H
ORR14	Object matcher	H
ORR15	Expose modifiable parameters	M
ORR16	Indicate quality of built model	H
ORR17	Support for information extraction via a server	H
ORR18	Support stand alone object recognition	H

Table 9.1: Functional requirements for object recognition framework cont.

9.1.2 Non-functional Requirements

Table 9.2 shows the non-functional requirements for the framework. The table follows the structure as the one in section 9.1.1. Refer to that section for the table explanation.

ID	Description	Priority
ONR1	Hide domain specific knowledge	H
ONR2	Perform object recognition within 1-2s	H
ONR3	Code provides development documentation	H
ONR4	Provide example application showing object recognition	H
ONR5	Provide example application showing model construction	H
ONR7	Support Android 2.3.1 and newer	H
ONR8	Size of object model as small as possible	M

Table 9.2: Non-functional requirements for framework

9.2 Example Applications

This section elicits the functional and non-functional requirements for the example applications.

9.2.1 Functional Requirements

The functional requirements are divided into two parts.

- There are a set of functional requirements directly linked to documenting the framework functionality. These are listed in table 9.3.

- The customer wants proof that the framework is suitable for their application domain. Therefore the author decided to develop the examples as a quiz and a quiz administration application. Both take ideas from the original framework prototypes. In order to identify the quiz functional requirements a different approach was applied; a set of user stories supported by use cases.

Framework functionality demonstration

Table 9.3 shows the functional requirements linked directly to demonstrating the framework functionality. The table has the same structure as the one in section 9.1.1. Refer to that section for the table explanation.

ID	Description	Priority
PR1	Test object model construction	H
PR2	Test 2D object recognition	H
PR3	Test 3D object recognition	H
PR4	Identify recognized object	H
PR5	Indicate quality of generated object model	H
PR6	Provide persistent storage for object models	H
PR7	Enable object query on stored object models	H
PR8	Indicate location and pose of identified objects	L
PR9	Expose recognizer settings in order to optimize framework	H
PR10	Test external information extraction	H

Table 9.3: Functional requirements for example applications

User Stories

The author decided to make user stories the backbone for the example applications. The user stories were developed in collaboration with the customer to ensure that their needs were met. The idea is that you write short notes about what the different users would like to do. The user stories state what kind of goals a user has for the application - what functionality the user wants to have - thereby indicating the required development effort. They differ from typical requirement specifications in that they focus on user needs and not how the system is implemented. Each user story is given a priority based on the context of the application.

The two applications have two different distinct actors. These are listed in table 9.4.

Actor ID	Description	Example of actions
A1.Admin	Administers models in database	Device touch- graphical interface, can build object models using images from device camera, can delete models, can add models.
A2.Visitor	Plays a quiz game driven by object recognition	Device touch- graphical interface, answer questions by clicking on correct answers, answer questions by taking a picture of objects with built in camera, instantiate categories by photographing objects.

Table 9.4: Framework example application system users.

The user stories are written: As a 'user' I want to 'goal' so that 'reason' and are given a priority. Each user story is given a unique ID which is used for easy referral to each story. The description explains the requirement itself. Since the example applications mostly focus on showing off framework functionality, such as recognition and model creation, the stories involving these actions are prioritized.

Quiz administration application

U1. Administrator installs application.
As an administrator I want to know how to install the application so I can use my own device when interacting with the system.
Priority: Low

U2. Administrator starts application for the first time.
As an administrator I quickly want to know how to use the application so I can start modifying the database.
Priority: High

U3. Administrator adds model to database.
As an administrator I want to add a new object to the database so that I can modify/edit answers to assignments.
Priority: High

U4. Administrator removes model from database.
As an administrator I want to remove a model from the database to avoid storing outdated references to objects.
Priority: Medium

U5. Administrator browses database models.
As an administrator I want an overview of all the models in the database so that I can easily find models of interest.
Priority: High

U6. Administrator browses object recognition questions.
As an administrator I want an overview of all the questions that involve object recognition so that I can efficiently edit the ones I am interested in.
Priority: High

U7. Administrator links model with question.
As an administrator I want to link a model with a question so that I can change the answer.
Priority: High

Quiz application

U8. Visitor installs application.
As a visitor I want to know how to install the application so I can play the quiz on my own device.
Priority: Low

U9. Visitor starts application for the first time.
As a visitor I quickly want to know how to play the quiz so I can concentrate on answering questions.
Priority: Medium

U10. Visitor must initiate a new quiz category by finding a category object.
As a visitor I must find the correct object so I can apply my knowledge on a specific subject.
Priority: High

U11. Visitor answers object recognition questions.
As a visitor I must find the correct object to the quiz question so I can gain new knowledge about it.
Priority: High

U12. Visitor answers quiz questions.
As a visitor I want to answer a quiz so I can apply my knowledge on a subject.
Priority: Medium

9.2.2 Use Cases

From the user stories the author made use cases that show how the example applications will work.

Quiz administration application

Use Case Name	**U1.Administrator installs application.**
Participating actors	A1.Admin Download support system
Flow of events	1. Administrator activates support for installation from unknown sources. 2. Administrator downloads application. 3. Administrator installs application.
Extensions	2a. Administrator initiates download using 2D QR-code. 2b. Administrator enters download URL into device web browser.
Entry conditions	Administrator wants to interact with system using her own device.
Exit conditions	Administrator successfully downloads application and starts to use it. Administrator can't install application and needs support.
Quality requirements	Download and installation takes reasonable time to finish.

Table 9.5: Administrator installs application.

Use Case Name	**U2.Administrator starts application for the first time.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator starts application. 2. Application greets administrator and present short instructions on usage and how to photograph objects for model construction. 3. Administrator clicks OK after reading the instructions. 4. Application informs the administrator on how to find the instructions for later reading.
Extensions	None
Entry conditions	Administrator has successfully installed the administrator application.
Exit conditions	Administrator understands how to use the application and knows how to construct object models.
Quality requirements	None

Table 9.6: Administrator starts application for the first time.

Use Case Name	**U3.Administrator adds model to database.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator enters application. 2. Administrator selects add object. 3. Application presents the administrator with the image acquisition interface. 4. Administrator chooses image1 containing pose A of object. 5. Administrator chooses image2 containing pose B of object. 6. Administrator enters object data. 7. Administrator selects create object model. 8. Administrator determines if object model is satisfactory.
Extensions	3a. Administrator opens instructions on how to photograph new objects. 8a. Model is unsatisfactory. 8a1. Administrator selects cancel. Return to 3. 8b. Model is satisfactory. 8b1. Administrator selects OK.
Entry conditions	Administrator wants to add a new object in database.
Exit conditions	Application confirms model creation. Application publishes model in list.
Quality requirements	Model creation process gives feedback on progress. Indication of model quality.

Table 9.7: Administrator adds model to database.

Use Case Name	**U4.Administrator removes model from database.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator browses models (U5). 2. Administrator selects model. 3. Administrator selects delete.
Extensions	None
Entry conditions	Administrator wants to delete a model from the database.
Exit conditions	Application confirms deletion of object model.
Quality requirements	Application prompts for deletion verification.

Table 9.8: Administrator removes model from database.

Use Case Name	**U5.Administrator browses database models.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator enters application. 2. Administrator selects browse models.
Extensions	None
Entry conditions	Administrator wants to get an overview of the models in the database.
Exit conditions	Application presents list of models.
Quality requirements	None.

Table 9.9: Administrator browses database models.

Use Case Name	**U6.Administrator browses object recognition questions.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator enters application. 2. Administrator selects browse assignments.
Extensions	None
Entry conditions	Administrator wants to get an overview of the assignments with object recognition answers in the database.
Exit conditions	Application presents a list of the object recognition questions.
Quality requirements	None.

Table 9.10: Administrator browses quiz object recognition assignments.

Use Case Name	**U7.Administrator links model with assignment.**
Participating actors	A1.Admin Administrator application
Flow of events	1. Administrator enters browse assignment (U6). 2. Administrator selects a specific assignment in the list. 3. Application presents assignment modification interface. 4. Administrator selects modify answer. 5. Application presents browse models list. 6. Administrator selects specific model. 7. Application returns to assignment modification interface. 8. Administrator selects save. 9. Application presents confirmation dialog. 10. Administrator confirms. 11. Application confirms administrator action. 12. Application returns to assignments list.
Extensions	3a. Administrator cancels operation. 3a1. Application returns to assignments list. 7a. Administrator has selected wrong model. Return to 4. 10a. Administrator selects cancel. 10a1. Application confirms action. Return to 3.
Entry conditions	Assignment exists in database. Model exists in database.
Exit conditions	Application confirms assignment modifications. Application modifies assignment.
Quality requirements	Application confirms administrator actions.

Table 9.11: Administrator links model with assignment.

Quiz application

Use Case Name	**U8.Visitor installs application.**
Participating actors	A2.Visitor. Download support system.
Flow of events	1. Visitor activates support for installation from unknown sources. 2. Visitor downloads application. 3. Visitor installs application.
Extensions	2a. Visitor initiates download using 2D QR-code. 2b. Visitor enters download URL into the devices web browser.
Entry conditions	Visitor wants to use the application using her own device.
Exit conditions	Visitor successfully downloads application and starts to use it. Visitor can't install application and needs support.
Quality requirements	Download and installation takes reasonable time to finish.

Table 9.12: Visitor installs application.

Use Case Name	**U9.Visitor starts application for the first time.**
Participating actors	A2.Visitor. Quiz application.
Flow of events	1. Visitor starts application. 2. Application greets user and presents short instructions. 3. Visitor clicks OK after reading the instructions. 4. Application informs the visitor how to find the instructions for later reading.
Extensions	None.
Entry conditions	Visitor has successfully installed application.
Exit conditions	Visitor understands how to use the application and how the application ties into the visitor center presentations.
Quality requirements	None.

Table 9.13: Visitor starts application for the first time.

Use Case Name	**U10.Visitor must initiate a new quiz category by finding a category object.**
Participating actors	A2.Visitor. Quiz application.
Flow of events	1. Application starts camera. 2. Visitor photographs object. 3. Application performs object detection
Extensions	4a. Application recognizes the object and initiates the quiz category. 4b. The object is not recognized. Return to 1.
Entry conditions	Visitor has successfully started application.
Exit conditions	Visitor understands assignment and starts searching for a solution. Visitor doesn't understand assignment and can't give an answer. Application is in assignment mode.
Quality requirements	None.

Table 9.14: Visitor must initiate a new quiz category by finding a category object.

Use Case Name	**U11.Visitor answers object recognition questions.**
Participating actors	A2.Visitor. Quiz application.
Flow of events	1. Visitor clicks on image acquisition button. 2. Application presents the visitor with the image acquisition interface. 3. Visitor takes picture of hher solution. 4. Application performs recognition on the object. 5. Application gives feedback on the Visitors answer.
Extensions	5a. Visitor gives the wrong answer to the assignment. Return to 2. 5b. Visitors gives the correct answer. The application moves on to next question.
Entry conditions	Application has given the visitor an assignment.
Exit conditions	Application is in quiz mode (U12).
Quality requirements	Application gives visitor clear feedback on solution.

Table 9.15: Visitor answers object recognition questions.

Use Case Name	**U12.Visitor answers quiz questions.**
Participating actors	A2.Visitor. Quiz application.
Flow of events	1. Application asks the visitor a question and gives the visitor 4 choices. 2. Visitor clicks on 1 of the choices. 3. Application gives feedback on the answer. 4. If quiz is not completed return to 1.
Extensions	3a. Visitor gives the correct answer. 3a1. Application informs the visitor that she has answered correct. 3b. Visitor gives the wrong answer. 3b1. Application informs the visitor that she gave a wrong answer. Return to 2.
Entry conditions	Visitor has solved assignment.
Exit conditions	Visitor has completed the quiz.
Quality requirements	Application gives visitor clear feedback on each answer.

Table 9.16: Visitor answers quiz questions.

9.2.3 Non-functional requirements

Table 9.17 shows the non-functional requirements for the example applications. The table has the same structure as the one in section 9.1.1. Refer to that section for the table explanation.

ID	Description	Priority
PNR1	Run on Android 2.3.1 and newer	H
PNR2	Perform object recognition within 1-2s	H
PNR3	Stand alone application	H
PNR4	Separate concerns	H
PNR5	Long running task shall run in its own thread	H
PNR6	Avoid ANR dialog	H
PNR7	Code is commented	M

Table 9.17: Non-functional requirements for example applications.

Chapter 10

Architecture

This chapter presents the architecture and the rationale for the choices made for both the framework and the examples. In order to evaluate all functional and non-functional requirements of the framework the architecture of example applications is presented. Functionality from chapter 9 is mapped onto the framework by indicating the fulfilled requirement in parenthesis.The authors choices of architectural views are based on the stakeholders presented in section 1.4 and 5.6.1.

10.1 Framework Overview

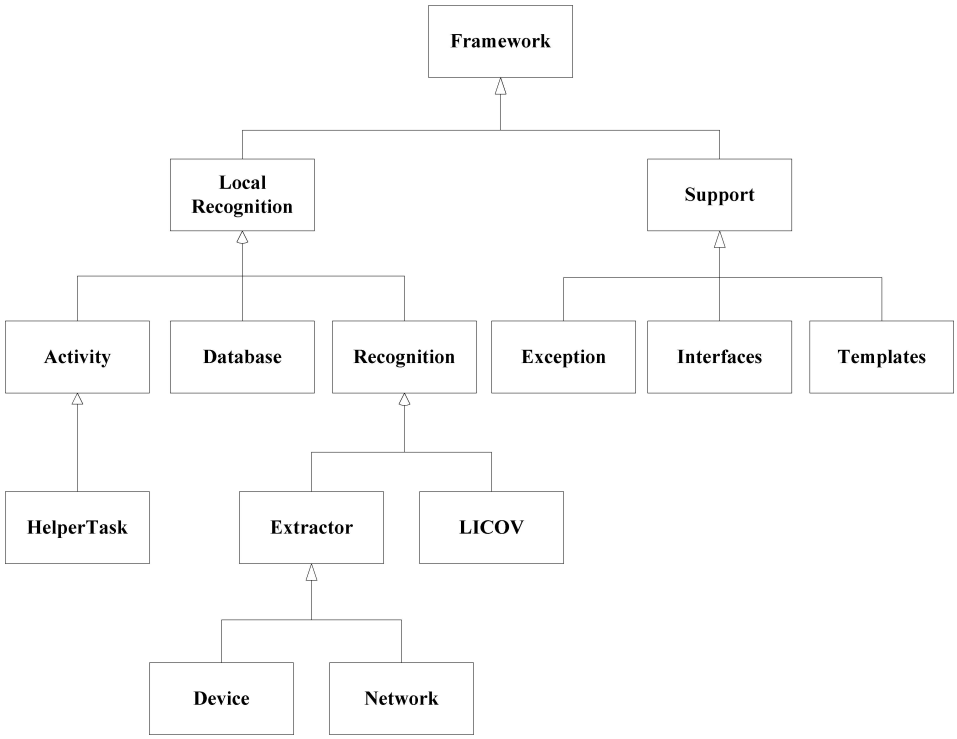


Figure 10.1: Framework hierarchy

Figure 10.1 shows the framework package hierarchy. The reason for creating a hierarchy where each package encloses a set of functionality is to improve modifiability, code reuse, ease maintenance and to ease framework browsing. The structure follows the rules in section 5.5.5 where the tree is deep and narrow and the abstractness decreases with tree depth.

The framework is divided into two parts, support and recognition. The support package contains helper classes and methods not directly linked to the object recognition itself. The framework currently only has support for local recognition. If, in the future, the framework owner wishes to add global recognition, the local recognition node should be moved into a new parent named recognition. A new sub tree should be added under recognition named global recognition where all functionality to global recognition is put. Common functionality shared by the local- and global recognition is put in the parent.

10.1.1 Package Diagram

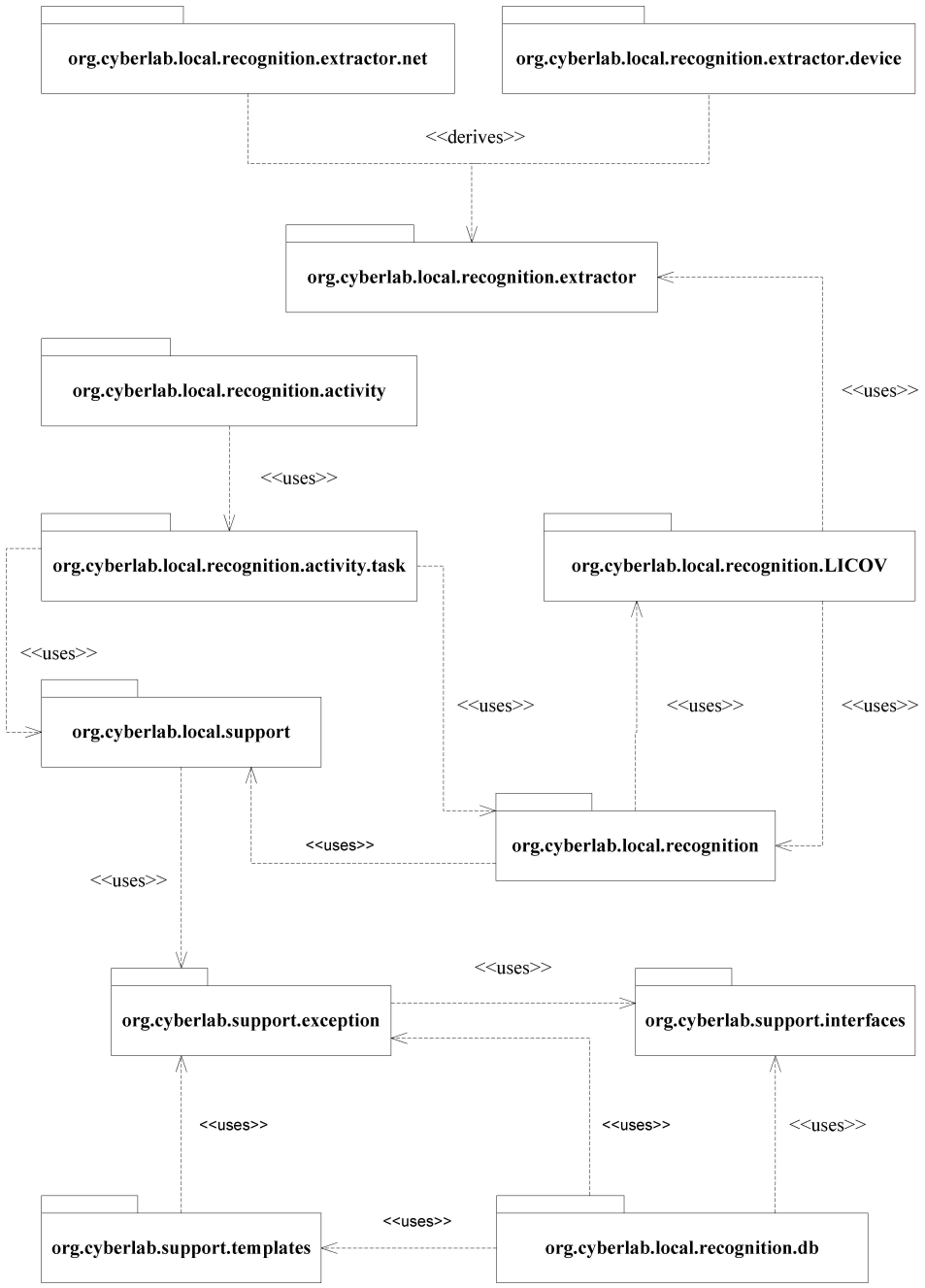


Figure 10.2: Framework package diagram

Figure 10.2 shows the packages in the framework and how they relate to each other.

recognition

Wraps all object recognition functionality in the framework. This package contains common code shared by the sub packages.

recognition.LICOV
: This is the currently only implemented recognition scheme in the framework, which is explained in section 6.4. This recognition scheme has been named LICOV by the author and is an acronym for **LI**near **CO**mbination of **V**iews.

recognition.extractor
: The framework supports local (performed on device) and external (performed on an external server) information extraction. This package wraps this functionality and contains an abstract class that implements common code and define interfaces that must be followed by the extractors.

  recognition.extractor.net
  : This package wraps all functionality related to the external information extraction.

  recognition.extractor.device
  : All functionality related to local information extraction is implemented here.

recognition.activity
: This package offers abstract activities and tasks that can be extended by the framework user and is meant to ease application implementation by explicitly exposing hooks and hot spots in the framework.

  recognition.activity.task
  : This package contains abstract and concrete tasks as well as a task manager. The concrete tasks implement default framework behaviour for information extraction, model creation and object detection.

recognition.db
: The framework supports storage of data in SQLite databases. This package offers a default database handler, an abstract database adapter and a concrete database adapter.

support

All support functionality (not directly connected to the object recognition) used by the recognition package is put into its own package. This functionality is also valuable to the framework user.

support.exception
: Contains a helper exception class which allow for more accurate and effective debugging.

support.templates
: Offer classes, implemented with the template pattern, for copying data between different sources.

support.interfaces
: Contains the interface specification for the exception class in *support.exception* and an interface specification for classes planning to use the *support.templates*.

10.1.2 Class Diagrams

Recognition

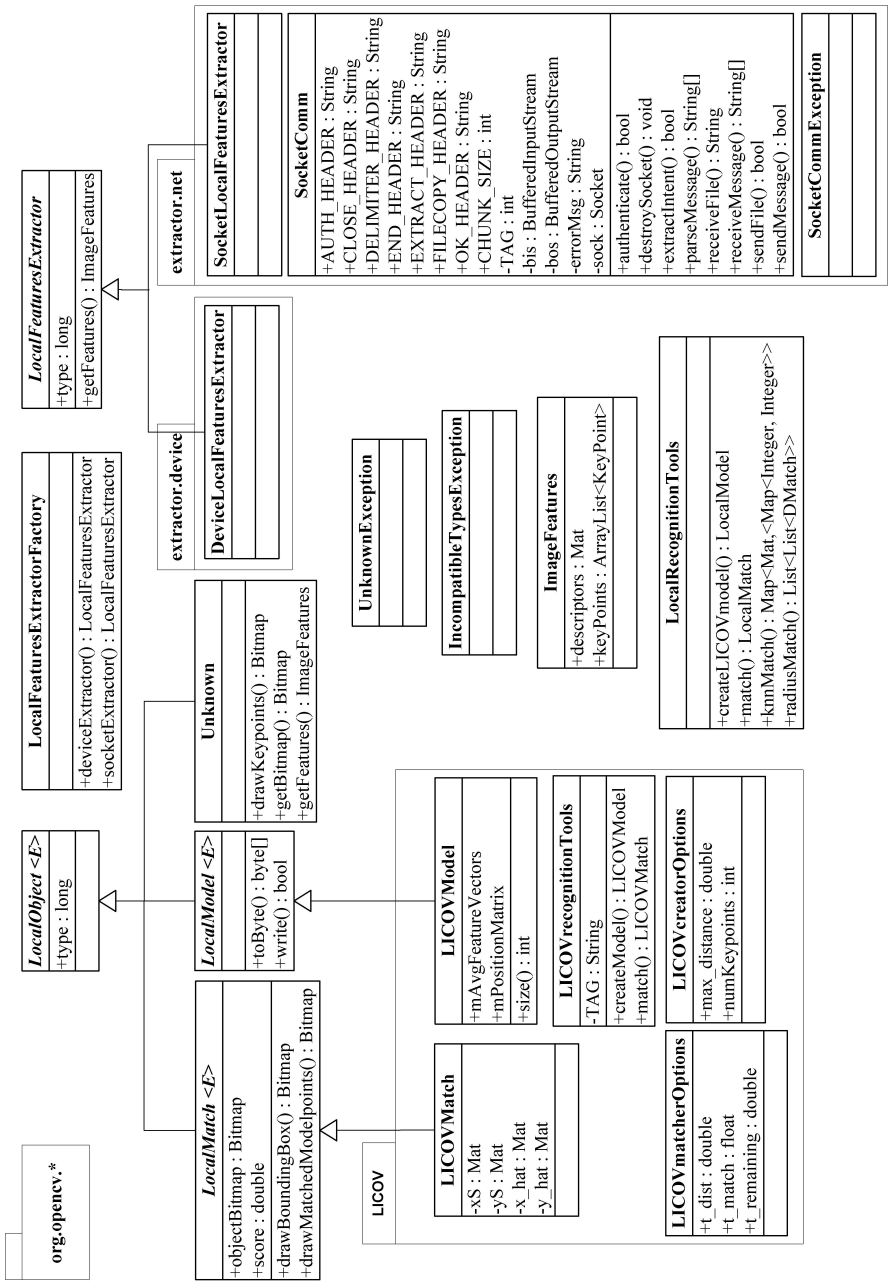


Figure 10.3: Overview of the framework recognition package

Figure 10.3 shows all classes in the recognition package. The classes rely heavily on the OpenCV library and this is therefore considered part of this package.

The package contains abstract and concrete objects that wraps OpenCV data structures, thereby hiding domain knowledge from the user (ONR1), inside three types of objects *LocalMatch*, *LocalModel* and *Unknown*. All of these extend *LocalObject* which contains one variable; type. Objects created using different feature detectors and descriptor extractors will be of a different type. This allows for type checking (i.e. checking for compatible objects). The objects form a hierarchy where the concrete objects form the leafs:

Unknown
: Contains the information extracted from an image. This is either an unknown entity that needs to be recognized or an object that can be used as a basis to create models. The Unknown wraps *ImageFeatures*, which contains the features and descriptors extracted from the image as well as the image itself. The reason to keep the image is because of the dependency between the compressed image and the keypoint positions. The original image can be compressed more than once, because of the fallback functionality in *UnknownTask*. Therefore it is convenient to explicitly connect the *ImageFeature* with the bitmap. This allows for easy implementation of a function called *drawKeypoints* in Unknown, returning a new bitmap with the keypoints drawn onto it.

LICOVModel
: This class wraps the recognition model (ref. section 6.4.2) into its own self sustainable object. This object must implement the methods defined in *LocalModel*. This allows for a common way of handling all models in the framework. The methods are meant to force all framework models to support self serialization. This simplifies model storage greatly as the sequence diagram in figure 10.15 shows (ORR12).

LICOVMatch
: Match results are also wrapped inside objects. Figure 10.3 shows that match results for the LICOV scheme returns four *Mat* objects. Wrapping these simplifies handling of the match results while hiding specific knowledge on how the recognition (described in section 6.4.3) works. All match objects in the framework must implement the methods defined in *LocalMatch* thus allowing for a common way of handling all match objects in the framework. Each match object contains a reference to the bitmap used for recognition. This simplifies the drawing of the matched model keypoints onto the image as well as drawing a bounding box enclosing the keypoints. These methods are used to show where the object is in the image.

Wrapping the OpenCV data structures in components allow for methods creating them. *LICOVrecognitionTools* implement all methods necessary for creating *LICOVmatch* (ORR14) and *LICOVmodel* objects (ORR13). These offer default

methods that use the SURF feature detector and descriptor extractor as well as a set of default parameters. Customization is offered by parametrization where all OpenCV feature detectors and descriptor extractors are supported. Creator and matcher parameters can be changed by modifying the default values in *LICOVcreatorOptions* and *LICOVmatcherOptions* (ORR15).

The *LocalRecognitionTools* use all the methods in *LICOVrecognitionTools* and implements some specific helper methods used by the latter. This class is meant as a collector for all object creation tools in the framework. Framework users only need to use this class. This makes the framework easier to use since all supported recognition schemes are collected in one class.

ImageFeatures are created using one of the *LocalFeaturesExtractor*s. The extractors can be created by accessing the *LocalFeaturesExtractorFactory*. The factory creates default or custom *DeviceLocalFeaturesExtractor* (ORR18) or *SocketLocalFeaturesExtractor*(ORR18) . All OpenCV feature detectors and descriptor extractors are supported.

SocketLocalFeaturesExtractor relies on *SocketComm* for communication with the external information extraction server. For more information on the communication protocols refer to section 10.2.6.

Activity

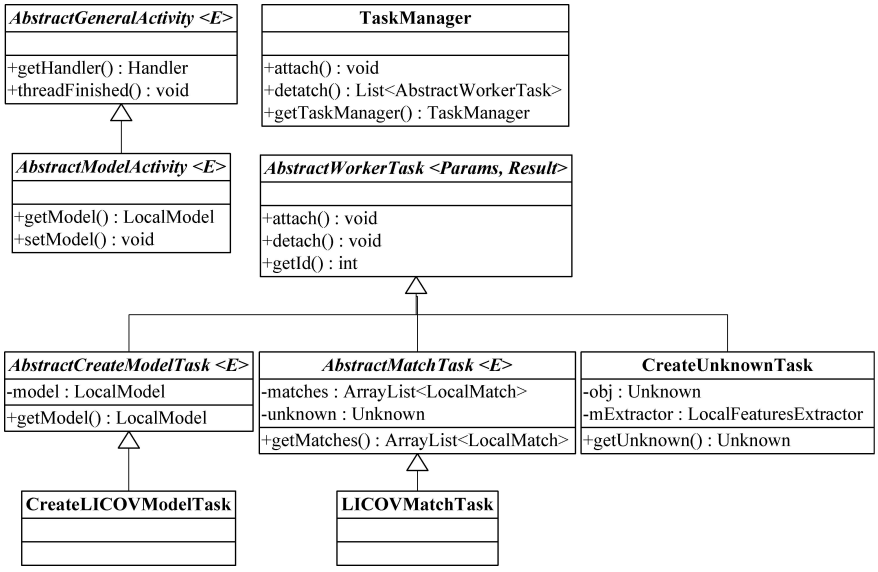


Figure 10.4: Overview of the framework activity package

Figure 10.5 shows all classes contained within the activity package.

The package contains two abstract activities that are meant to ease the implementation of activities involved in creating models, *AbstractModelActivity*, and

matching objects, *AbstractGeneralActivity*. These contain methods for starting the camera and accessing the built in image library in the device. Two abstract methods *cameraActivityResult* and *libraryActivityResult* are exposed where the users can implement their own camera and library result handlers.

The abstract activities are supported by abstract and concrete tasks. The tasks form a hierarchy that is meant to match the object hierarchy in the recognition package. All tasks subclass *AbstractWorkerTask* which allows for usage of the *TaskManager*. The *TaskManager* is a support operation for activities that wishes to obtain their reference to their tasks if they are restarted. This can happen when they run in the background, such as when the camera activity is in the foreground. In order to free up memory, the Android operating system can stop background activities[83] and the activities lose all their object references. This reference can be stored by calling detach on the *TaskManager* for each task before the activity is killed. The reference can be obtained again by calling attach on restart. This is possible because *TaskManager* is a singleton object.

Support

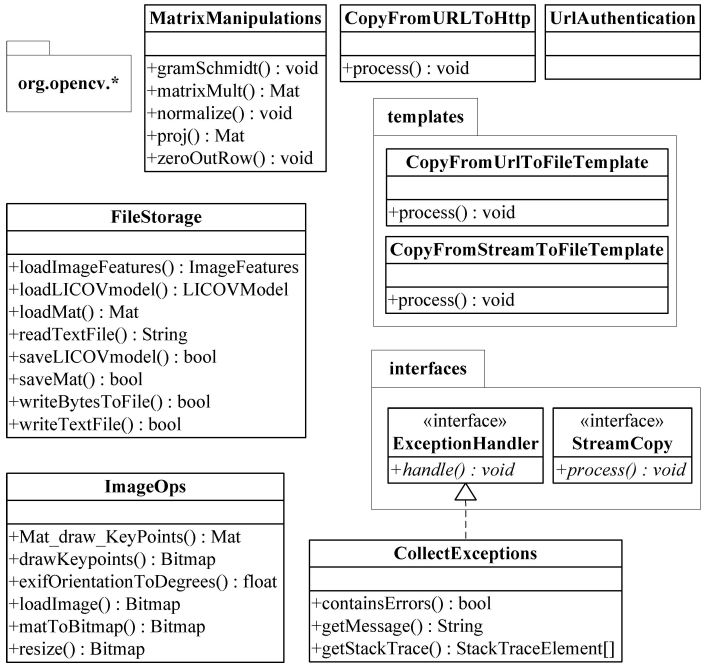


Figure 10.5: Overview of the framework support package

Figure 10.5 shows all classes contained within the support package. The classes rely heavily on the OpenCV library and therefore this is considered part of the package.

MatrixManipulation offer matrix operations on *Mat* objects that are not available in OpenCV. All methods are static to improve performance.

ImageOps offer methods for converting Mat objects to bitmaps, resizing images, drawing keypoints and loading images from the file system into memory. All methods are static to improve performance.

CopyFromURLToHttp, *CopyFromUrlToFileTemplate* and *CopyFromStreamToFileTemplate* are methods for copying binary data between different sources. The first is not implemented as a template because the methods for copying to HTTP sinks are dependent on the script that receives the data. The *StreamCopy* interface must be implemented by classes that wishes to use the copy templates.

UrlAuthentication is a helper class for feeding web server login credentials to the built in Authenticator in Android.

CollectExceptions is a helper class for collecting a train of exceptions in order to correctly represent the exception stack where the "causing" exception is on top of the stack. This should allow for more accurate and effective debugging.

FileStorage offer methods for saving and loading OpenCV data structures and framework object models to the file system. This class uses the Java Native Interface to directly access the OpenCV C++ file storage operations. This allows for a fast, easy and consistent way of storing fully cross platform compatible OpenCV objects in XML or YAML file format. This is a huge framework feature improvement. Earlier, all OpenCV *Mat* and KeyPoint data structures had to be converted into Java serializable data structures before being saved to binary files. Saving and loading was a slow and painful process since the primitives in the *Mat* structure change with the type of descriptor extractor used for creating them. This causes complex type checking when converting between the Java structures and OpenCV structures. All methods are static to improve performance.

DB (ORR3, ORR6)

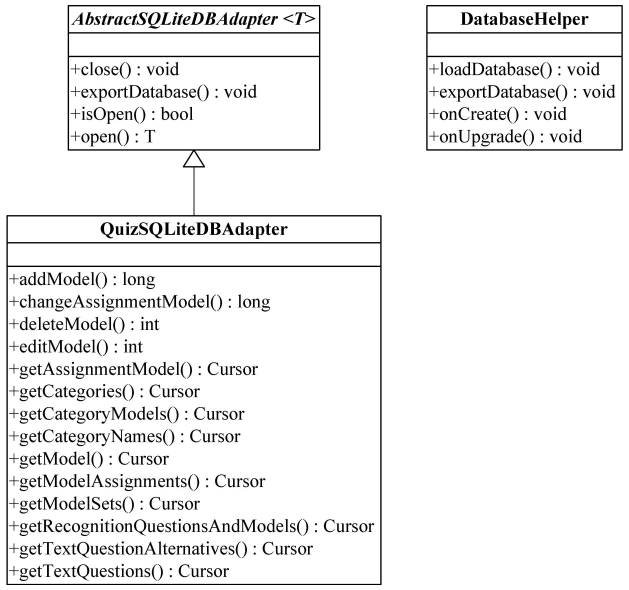


Figure 10.6: Overview of the framework db package

Figure 10.6 shows all classes contained within the db package.

DatabaseHelper is a concrete class for creating the database from a text file which contains legal SQLite syntax for initializing a database or copying an already existing database into the application. The *DatabaseHelper* also has an export function.

AbstractSQLiteDBAdapter uses the *DatabaseHelper* for importing, loading and exporting databases as well as for opening and closing the database. This is common functionality shared by applications.

QuizSQLiteDBAdapter is a concrete database implementation used by the example applications. The rationale for placing this class in the framework is based on the component library approach presented in section 5.5.4. All code shared by applications should be collected in a common component library. Over time this should help the framework developers see new abstractions. The hope is that some code from this class will help in that respect and the parts that are common between this class and new *SQLiteDBAdapter*s can be extracted into new components. Then the parts that are application specific can be moved out of the framework.

10.2 Example Applications

The purpose of this section is to show the usage of the framework. How the sequence of calls are made to perform model creation and recognition (ONR4,

ONR5). The rest of the functional and non-functional requirements are mapped onto the example applications and the framework.

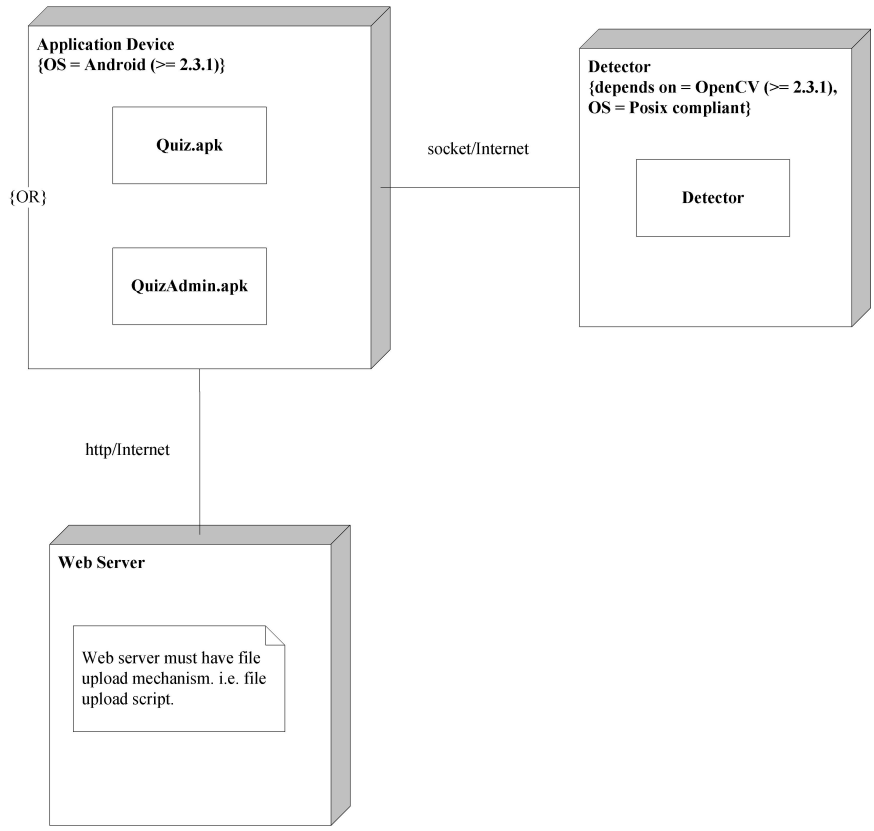


Figure 10.7: System deployment. Framework requires Android version >= 2.3.1 (ONR7, PNR1).

Figure 10.7 shows how both example applications are deployed. The applications use external server support for information extraction (PR10). The application and the *Detector* communicates using a custom communication protocol. For more information about this refer to section 10.2.6.

The applications share information by copying a SQLite database to and from a web server. The framework does not currently support sharing of data across devices. This functionality is added to the examples because this aids in showing the framework in real world contexts. The implementation is extremely simple and will not be discussed further in the main part of this thesis since it is outside the scope of the current framework. For more information on how the solution is implemented refer to appendix E.

10.2.1 Recognition Functionality

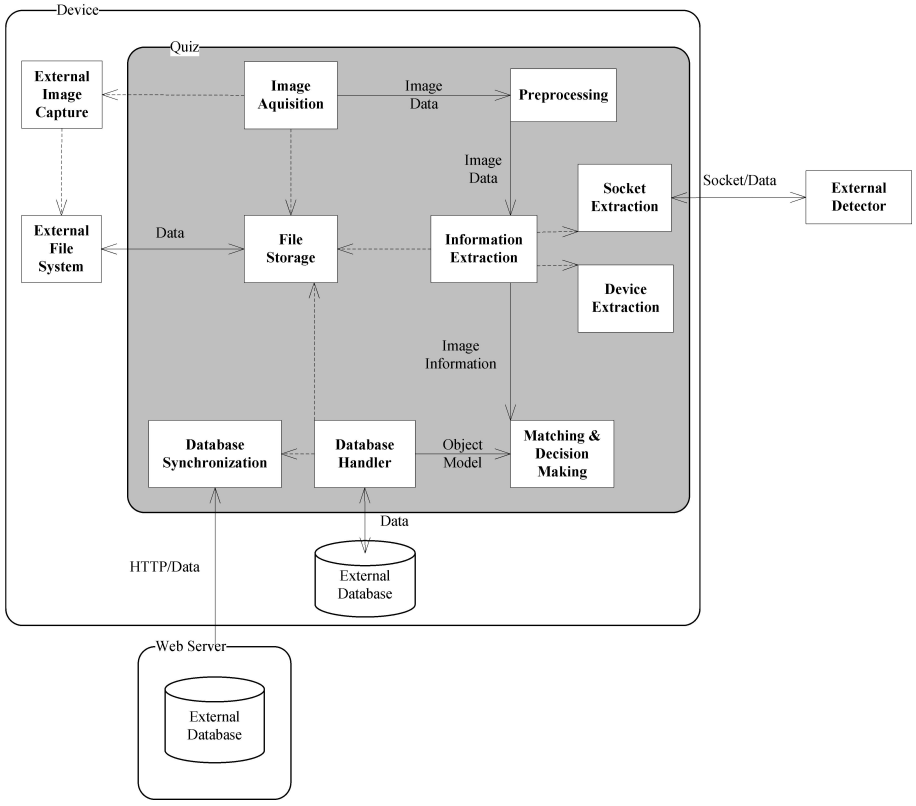


Figure 10.8: Object recognition

This section presents the logical view of the recognition functionality in the framework in context of the Quiz application(PR2, PR3, ONR4 ,PNR3). Figure 10.8 shows an informal view of this process(PNR4). The application fetches the shared database from an external resource and loads it onto the device. If this fails the application falls back to a backup stored on the device (a local preinstalled database or an earlier downloaded database if the application has been run before). The application uses functionality offered by the Android operating system for image capture. The image is stored on the file system and a reference is sent back to the application. The process follows the pipeline architecture presented in figure 6.1 and figure 6.3. The image is compressed and loaded into memory. By default the external detector is used with an image resolution of 600x600. If this extractor somehow fails the applications falls back to the device extractor but resizes the image to 300x300. After extraction the data is sent to a matcher. The extracted image data is matched against existing database models. The matcher results are analysed and a decision on the existence of a model in the image is performed.

Collaboration

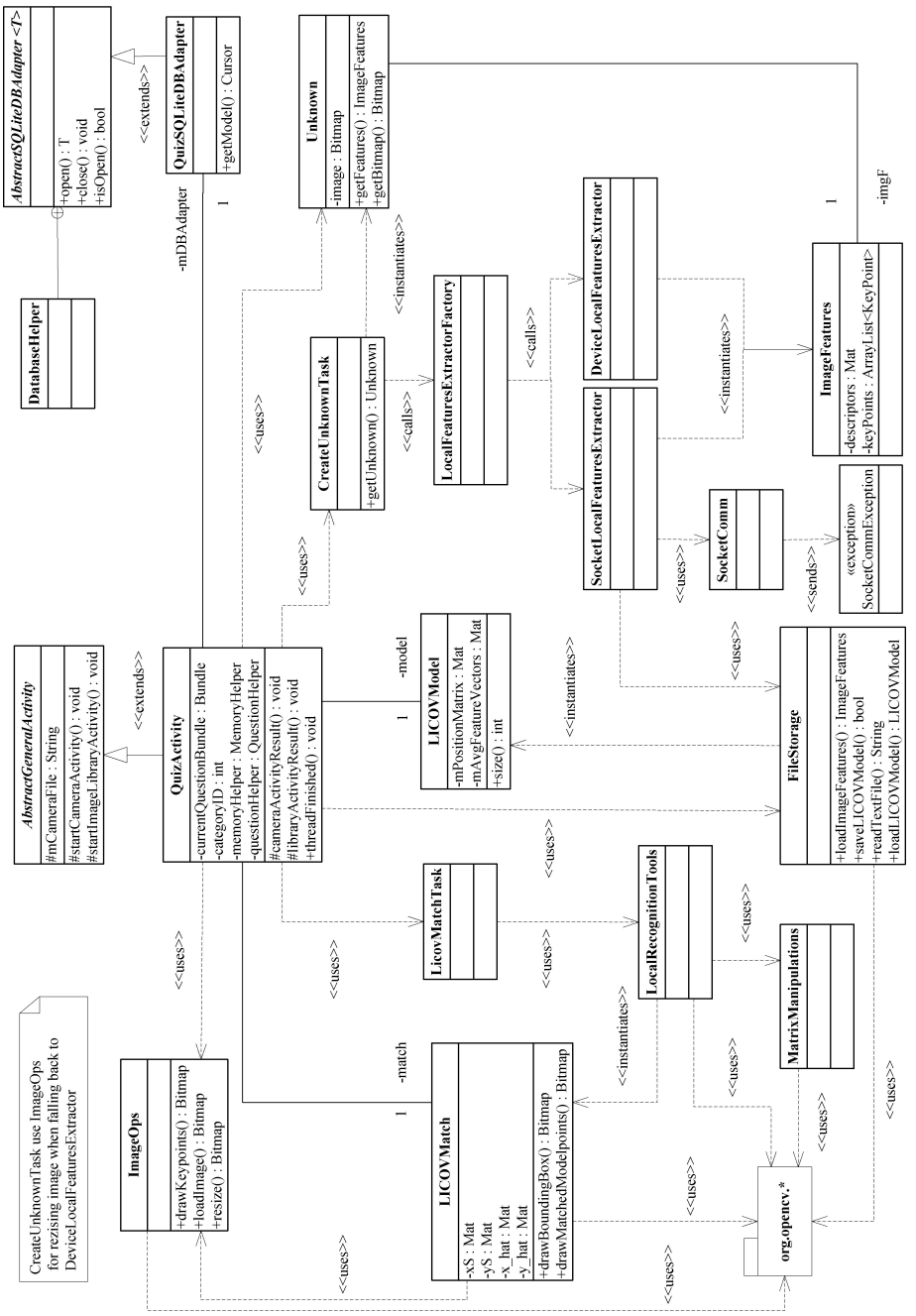


Figure 10.9: Simplified object recognition class collaboration diagram

Figure 10.9 shows a simplified class collaboration diagram of the object recognition activity. QuizActivity is the only class that is part of the application, the rest are framework classes. The diagram shows how all domain knowledge is hidden inside objects and there are no direct communication between the application and OpenCV. The regular framework user only needs to know the steps involved in creating the Unknown object and matching it against the models in the database. This framework share the property which is common among frameworks, inter class dependency (ref. section 5.1.3). Novice framework users must learn a set of classes at once, which can be difficult.

Flow of Control

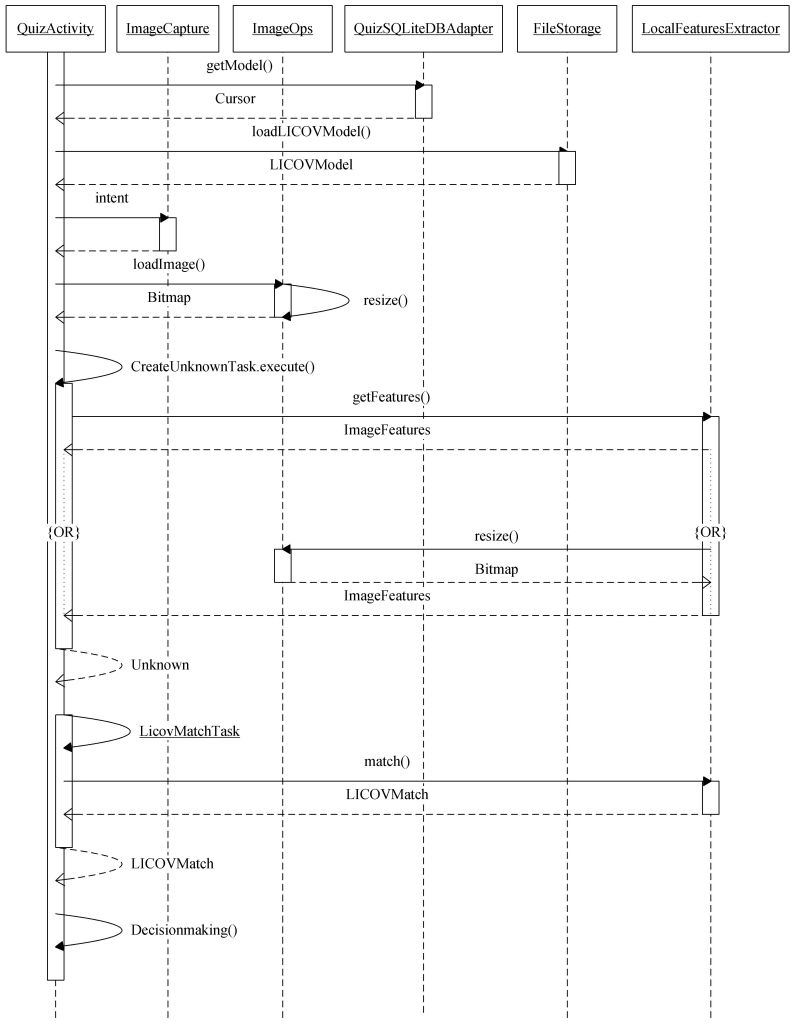


Figure 10.10: Simplified sequence diagram for object recognition

The sequence of calls necessary to perform the recognition is shown in figure 10.10. This explains the flow of control between the classes in the framework and between the framework and the application. The diagram also shows how the main processing is handed over from the main user interface thread to the worker threads thus avoiding a locked user interface(PNR5,PNR6).

10.2.2 Model Creation Functionality

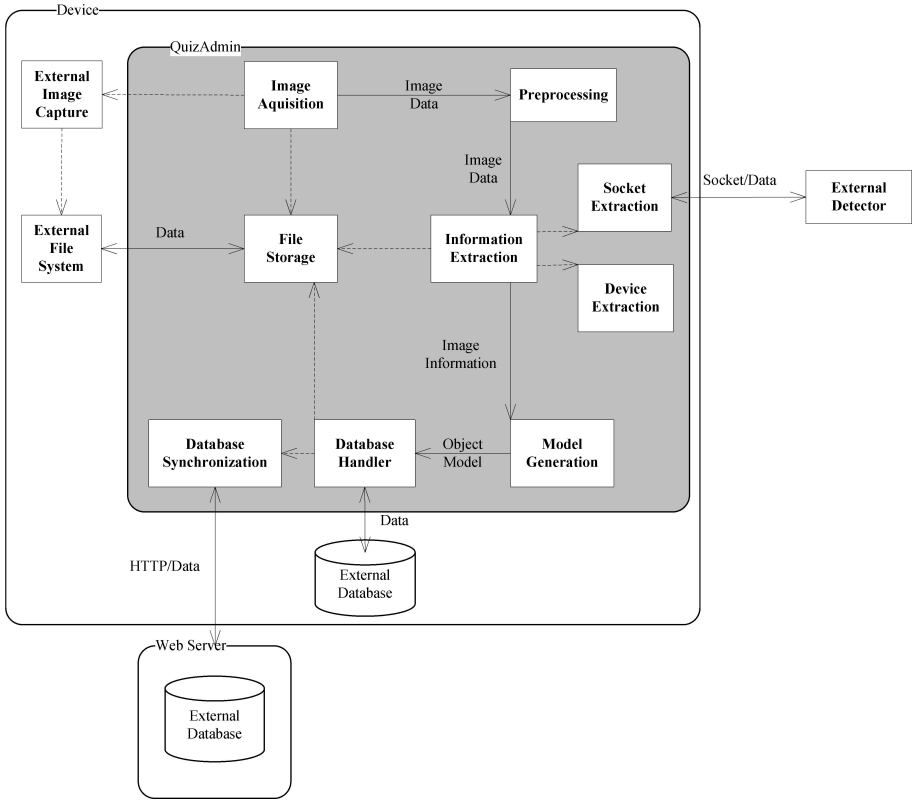


Figure 10.11: Model creation

This section presents the logical view of the model creation(PR1,PR2,PR3, ONR5) functionality in the framework in context of the QuizAdmin application. Figure 10.11 shows an informal view of this process(PNR4). The similarity between this figure and figure 10.8 proves design reuse. The application grabs the external database from the web server and when the database modifications are finished the application copies the database back to the web server. The QuizAdmin application does not have a preinstalled database. The rationale for this is that no one will use an outdated database. The application fetches images using the built in camera activity in Android. The process follows the pipeline architecture presented in

figure 6.1 and figure 6.2. The image is resized and loaded into memory. By default the the external detector is used with an image resolution of 600x600. If this extractor somehow fails the applications falls back to the device extractor but resamples the image to 300x300. After information extraction the data is sent to a model generator that builds the model. The model is serialized and stored in the database.

Collaboration

Figure 10.11 shows a simplified class collaboration diagram of the model creation activity. Model creation is the most complex and most difficult activity to learn in the framework. This is because there are lots of tasks and models to keep track of. This is worsened by the fact that activities are restarted when they are in the background. The *CreateLicovModelActivity* uses a helper class *CustomImageView* to keep track of the connections between the *ImageView* on the screen, the bitmap and the *Unknown* object. Aside from these classes the domain knowledge involved in creating the model is wrapped within objects and the application has no direct communication with the OpenCV library. This collaboration diagram is similar to the model recognition diagram in figure 10.9 and proves code reuse and design reuse. Dependencies between classes is common for frameworks and are one of the reasons for them being hard to learn (ref. section 5.1.3). Novice users must expect to learn several classes at once.

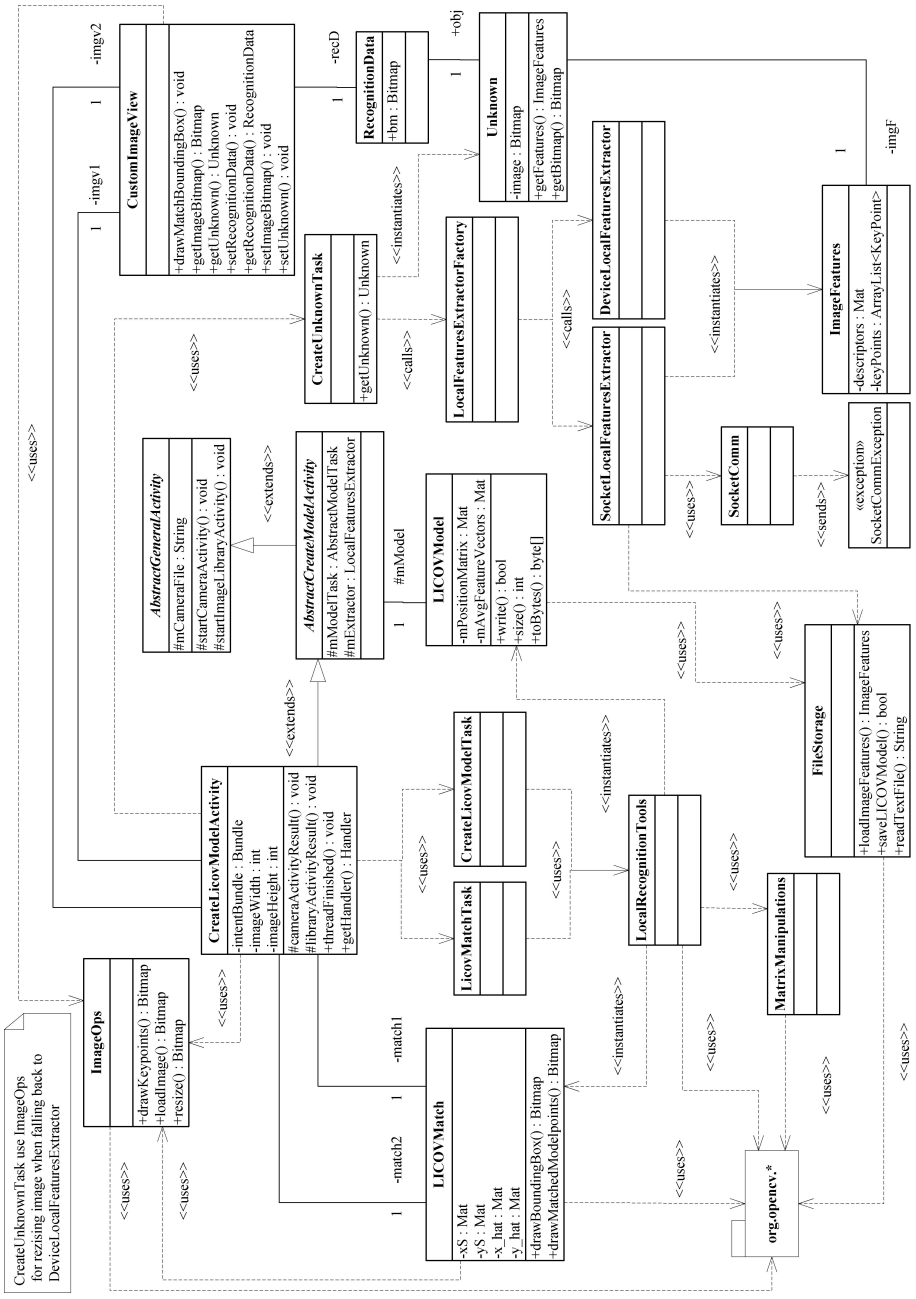


Figure 10.12: Simplified model creation class collaboration diagram

Flow of Control

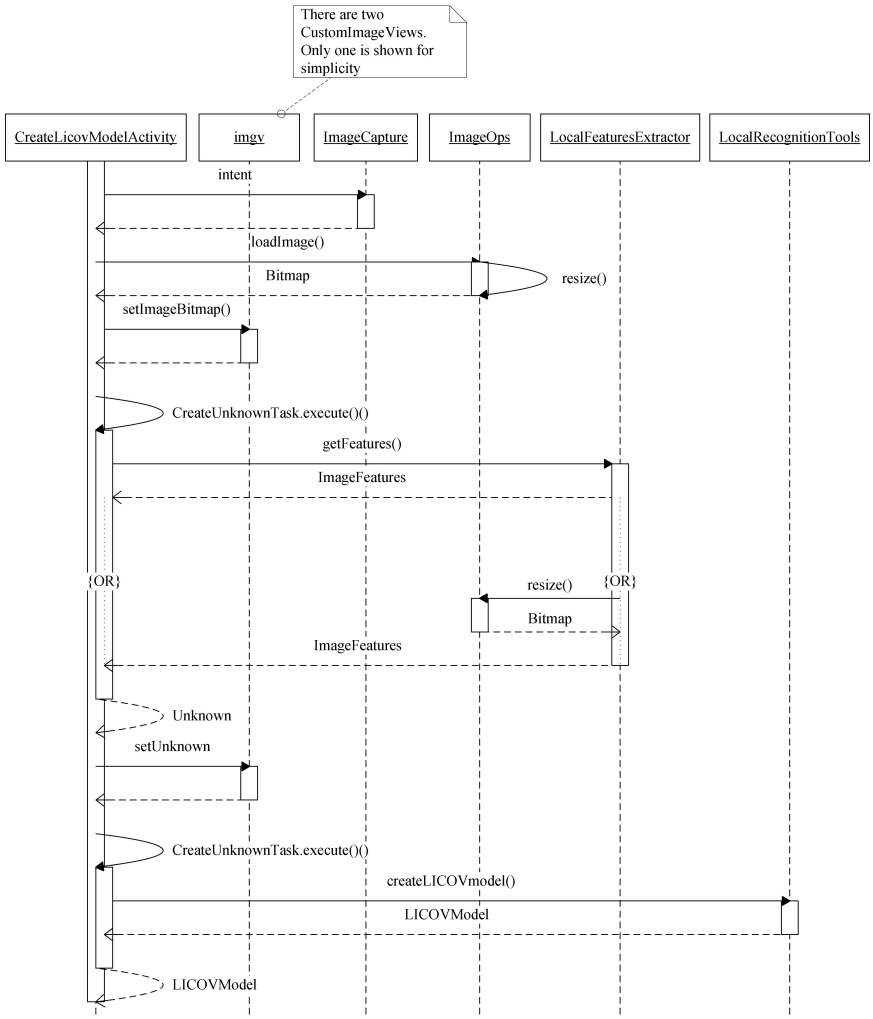


Figure 10.13: Simplified sequence diagram for model creation.

A simplified process view of the model creation is shown in figure 10.13. This helps communicate the flow of control between the classes in the framework and between the framework and the application. The information extraction and model creation is performed on separate threads freeing up the main thread to do perform user interaction(PNR5,PNR6).

10.2.3 Model Storage

This section explains how applications can save models by using built in functionality in the framework (PR6,PR7).

Database Design

The database functionality is shown in the entity-relationship diagram in figure 10.14. The database is constructed to support the quiz. The interesting part in the context of the framework is how simple model storage is. The entity *Model* is the only part involved in the storage of the model. The framework serializes the object into a human readable file, i.e. XML, and stores this file as a blob in the database together with metadata that identifies the object.

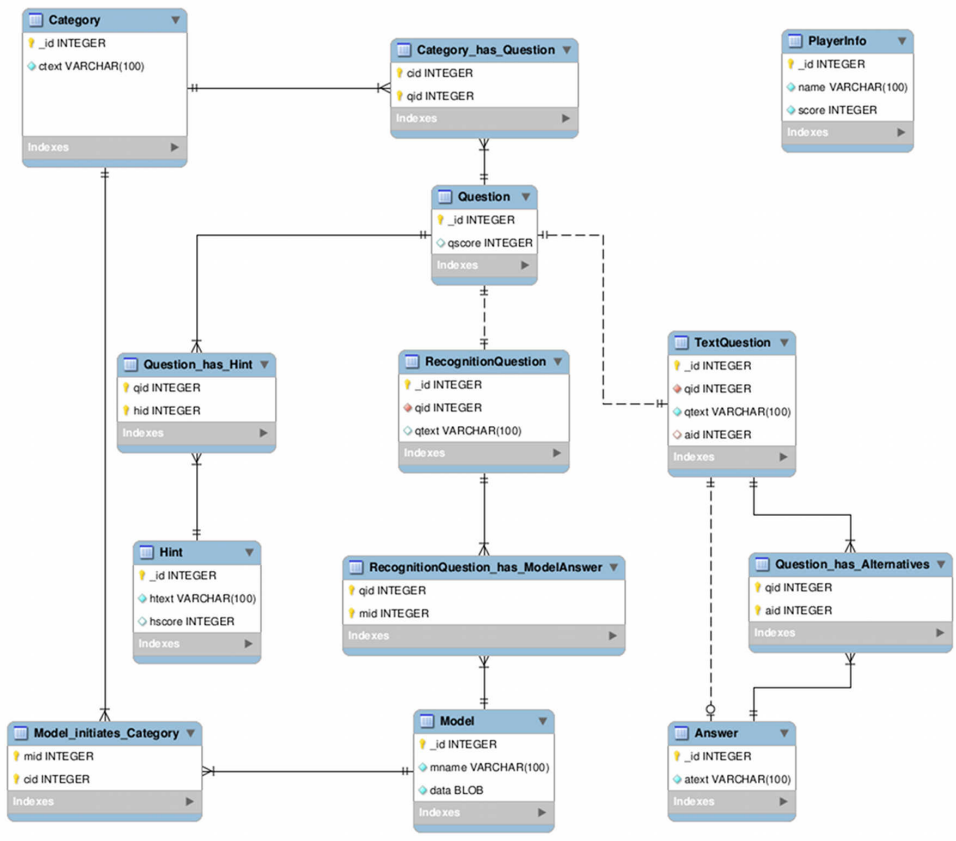


Figure 10.14: Entity relationship diagram of database

Flow of Control

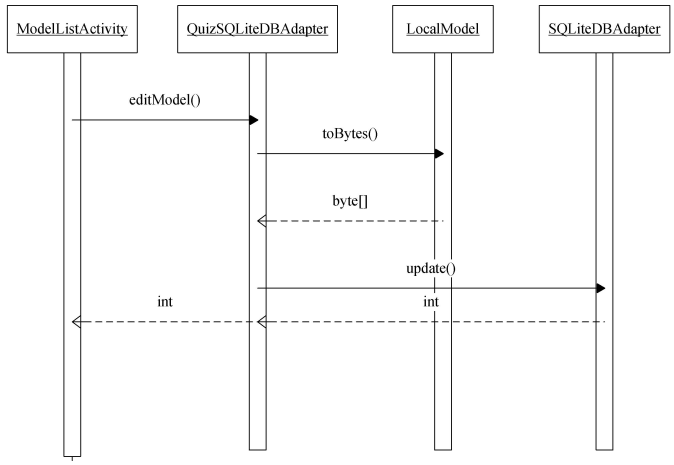


Figure 10.15: Simplified sequence diagram for model storage

Thanks to the simplified way of serializing data the process of storing objects has become a trivial exercise (ref. figure 10.15). The user calls the *QuizSQLiteDBAdapter* with a reference to the object. *QuizSQLiteDBAdapter* asks the model to serialize itself and receives a byte array representation of the human readable file. The bytes are stored as a blob in the database.

10.2.4 External Detector

This section presents the external information extractor(ORR17), shipped as part of the framework, and how this communicates with the Android framework.

10.2.5 Collaboration

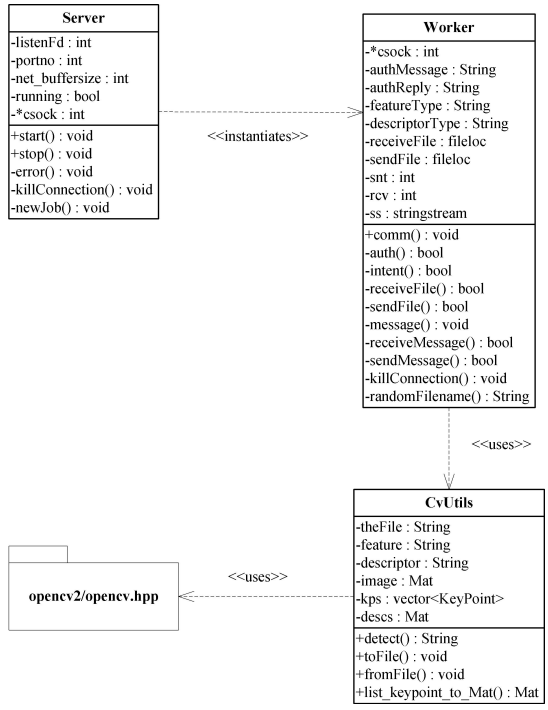


Figure 10.16: Simplified detector class collaboration diagram

Figure 10.16 shows a simplified class collaboration diagram of the detector. The *Server* listens for connections and instantiate *Worker* threads when new connections are established. The *Worker* thread class implements all of the communication protocols such as authentication, intent clarification and data transfer. Currently only information extraction is supported. Extraction is performed by instantiating *CvUtils* that use OpenCV functionality to perform keypoint detection and descriptor extraction. The data is serialized into OpenCV XML or YAML and returned to the *Worker* which transfers the data back to the Android application.

10.2.6 Socket Communication

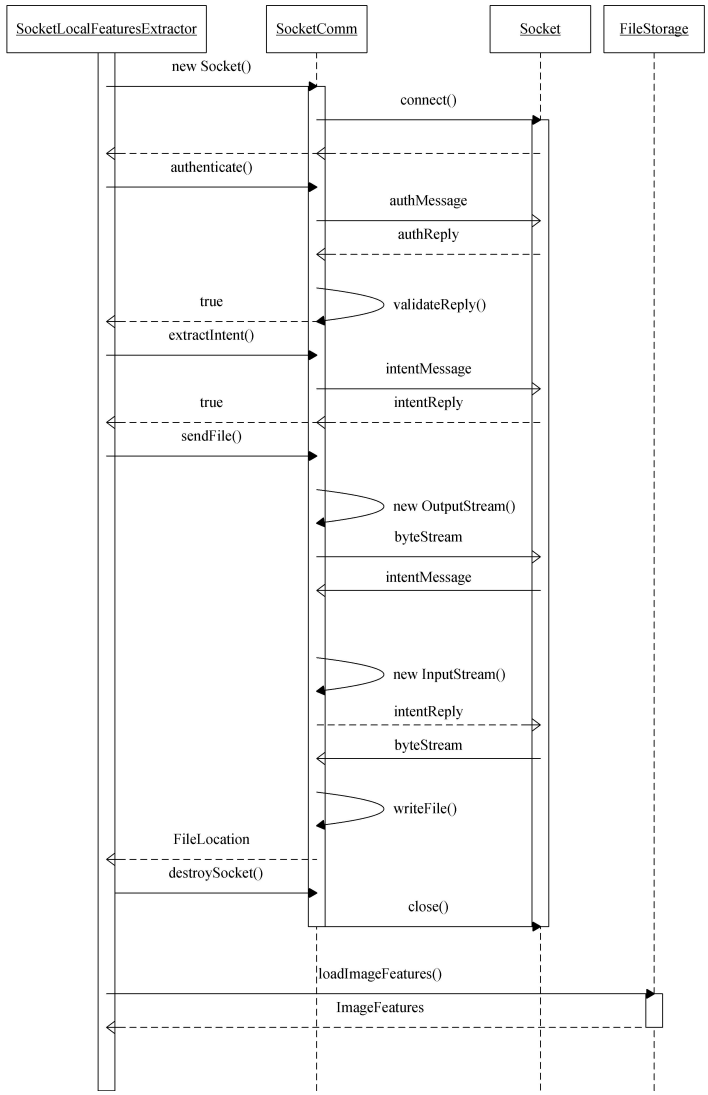


Figure 10.17: Simplified sequence diagram for the socket communication between an application and the external detector

Figure 10.17 shows how the Android framework applications communicate with the server. The client and server performs two way authentication using strings. After this succeeds the client sends an intent which clarifies what it wants the server to do. Currently the only intent supported is information extraction but the way of sending intents allow for protocol extensibility. After the intent has been clarified the client sends information about the file such as file name and size. After the

server replies the client sends data. The server performs its operations and return the file in the same manner as the client-to-server copy operation. When this is done both close their connections. For protocol specifics refer to the *SocketComm* (Javadoc) or the *Worker*(Doxygen) documentation.

Chapter 11

Implementation

This chapter presents the application user interfaces and maps the rest of the functionality from section 9.2.1 onto them. Challenges and problems related to the implementation are discussed.

11.1 Example Applications

This chapter presents the user interface in the two example applications. User story U1 and U8 are not seen here and they are part of the application installation process explained in appendix D. The mapped functionality is shown in parenthesis.

11.1.1 Quiz

After the quiz application is started the user is presented with one of the two views shown in figure 11.1. If the application is started for the first time the user is greeted with a welcome message (U9). The message is adapted to fit as framework documentation where highlights of the framework changes are listed along with application usage information. After closing the message the application enters the camera activity and the user is tasked with finding an object that initialized a quiz category. The quiz does not implement a scoring system and ends when all categories have been answered.

Since the application is prescriptive documentation for the framework, a debug menu is added. This can be accessed by cancelling the camera in the *"find category"* mode and pressing the options button. The user can then either manually choose to re-download the database (if she has set up application compatible support for this. For more information refer to appendix E) or choose to edit the application preferences (PR9). A view of these two options is shown in figure 11.2.

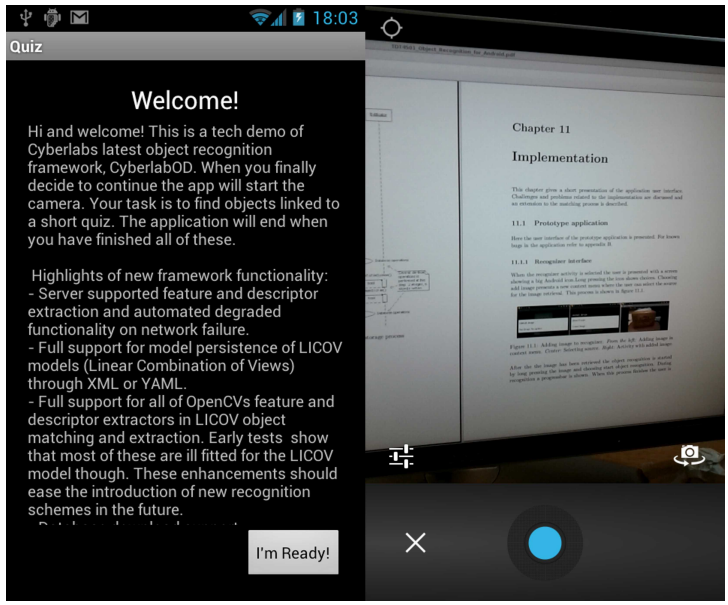


Figure 11.1: Initial state of the quiz application. *Left:* First run welcome screen. *Right:* Camera activity.

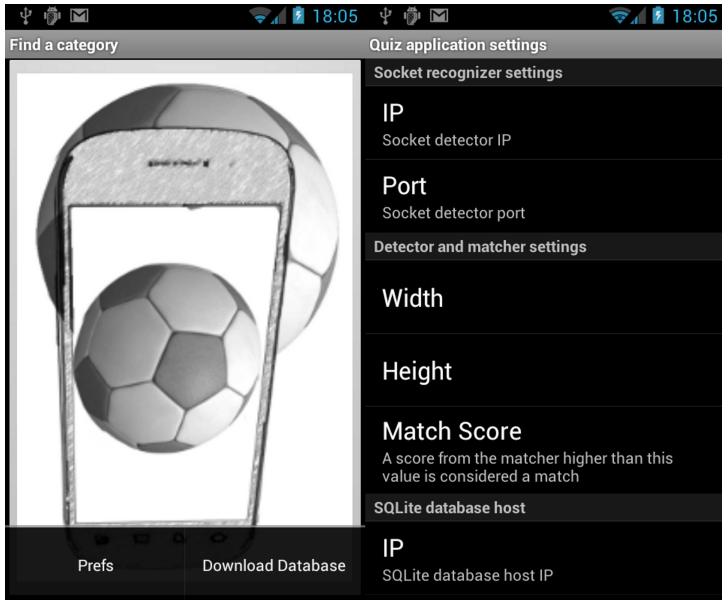


Figure 11.2: Cancelling the camera activity when the application is in *Find a category* mode allows access to the application settings. These are not meant to be accessible in production applications. *Left:* The options menu. *Right:* Application preferences menu.

After the user has found an object the object recognition process is started. If the object initializes a category the category quiz is started (PR4, U10). If the category has been previously answered, the application informs the user and asks her to find a different category. These three steps are shown in figure 11.3.

The application offers two different kinds of questions; text questions and object recognition questions. The first is shown in figure 11.4 where the system asks the user a question along with a series of answer alternatives (U12). The user continues to answer until she is correct. The system gives feedback on the answers.

A recognition question is shown in figure 11.5. The system asks a question and the task of the user is to find the object that answer this correctly (U11). The system moves on to the next question when the user provides the correct object(PR4).

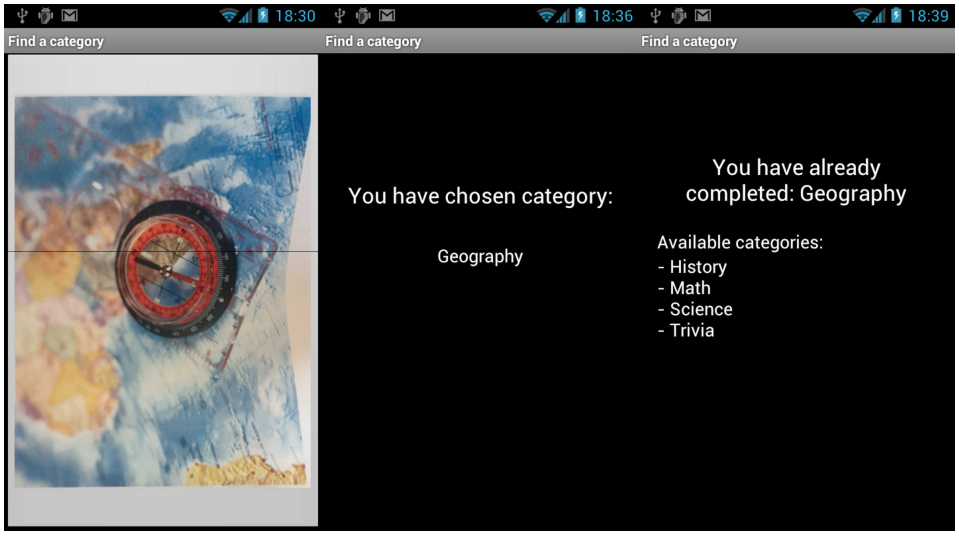


Figure 11.3: Quiz application performing object recognition on an image to initialize a quiz category. *Left:* The captured image is shown along with a scan bar indicating that detection is in progress. *Center:* The image to the left initializes geography questions. *Right:* It's not allowed to redo previously answered categories.

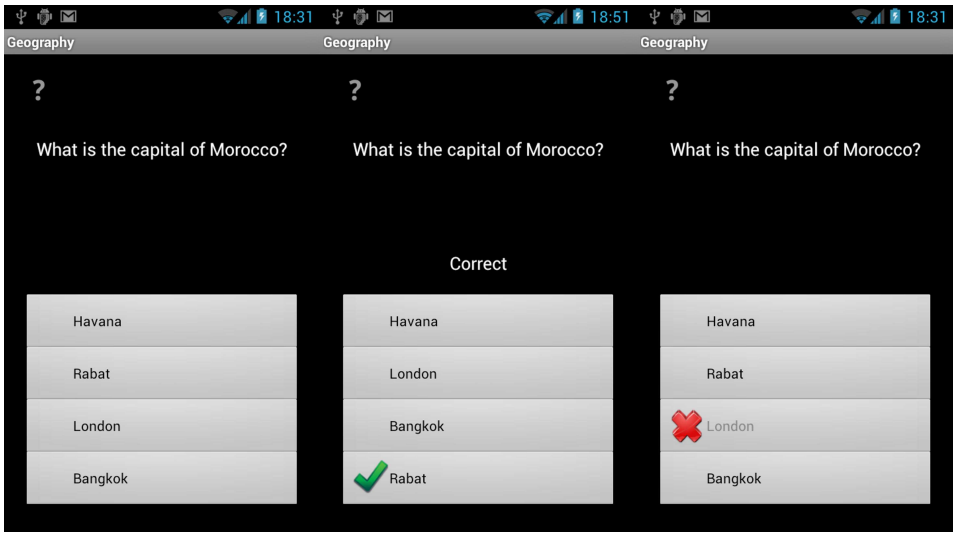


Figure 11.4: Quiz application text question. *Left:* A geography text question with available alternatives. *Center:* User has answered correctly. *Right:* User has answered incorrectly.

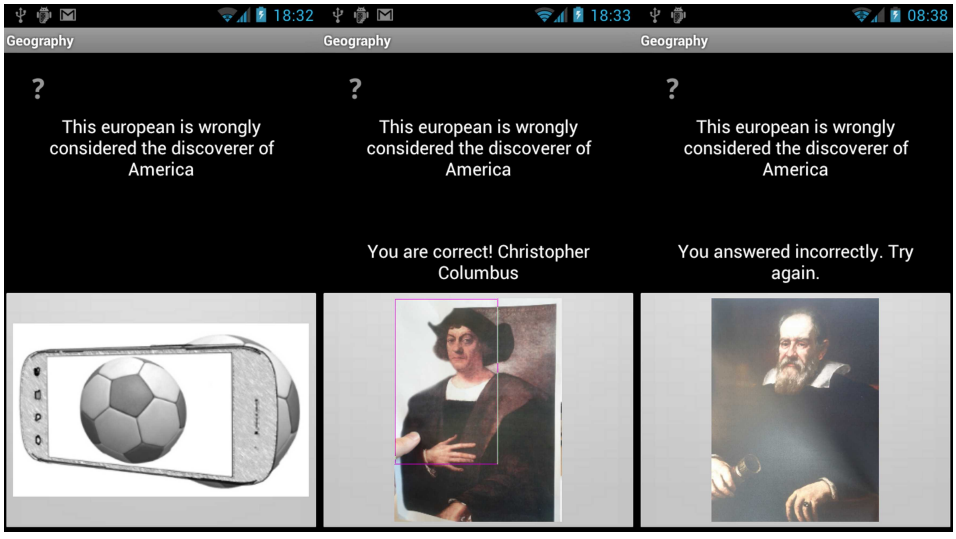


Figure 11.5: Quiz application object recognition question. *Left:* A geography question with an image button initializing the camera activity. *Center:* User has found the correct object and the application indicate the area of recognition (PR4, PR8). *Right:* User has not found the correct object.

11.1.2 QuizAdmin

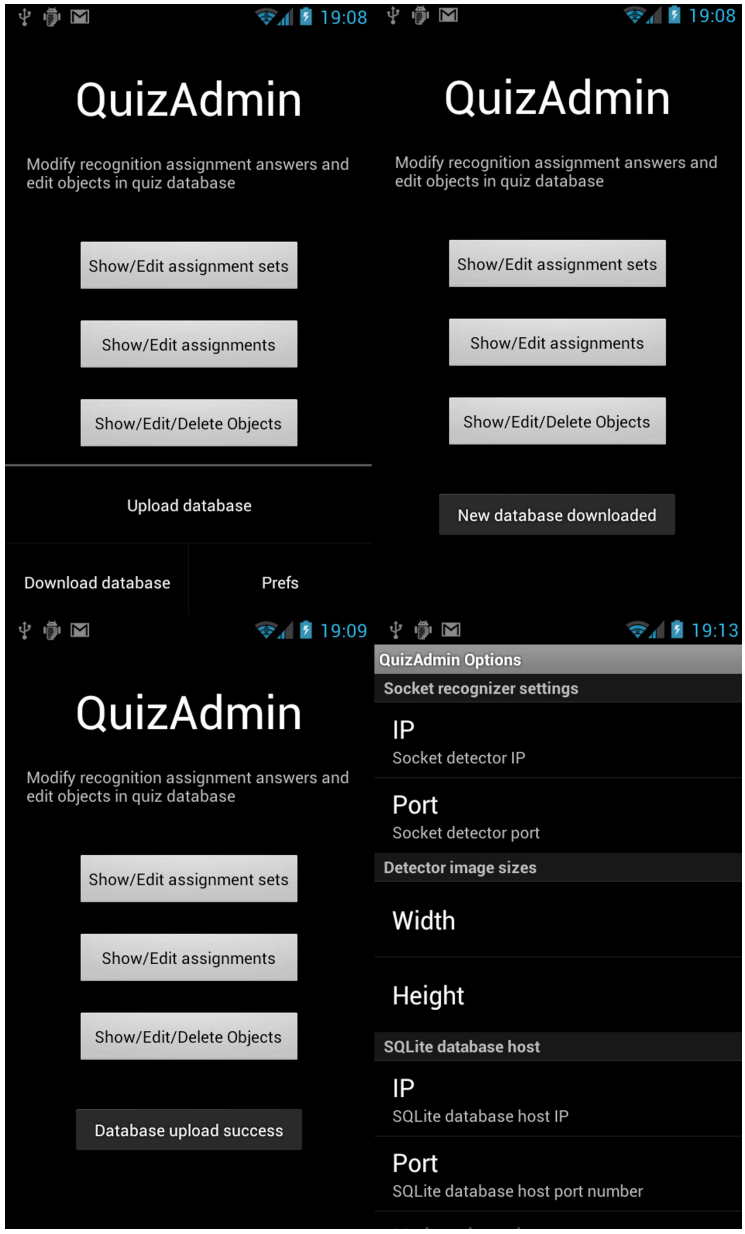


Figure 11.6: Initial state of the quiz admin application. *Top left:* The options menu is only accessible in this activity. *Top right:* The database has been downloaded successfully. *Lower left:* The database has successfully been uploaded. *Lower right:* Application preferences menu.

The quiz administration application only involve parts of the database that are related to the object recognition. The reason for this is to simplify the application so it better suits as framework documentation. The user is not allowed to add categories or questions, but she can edit the answer by either choosing an existing model in the database or adding a new model.

When the application is started the user is presented with the activity shown in figure 11.6. This activity has an options menu that allows for downloading and uploading of the database as well as an option for changing the application preferences such as changing the socket server settings and database synchronizer settings.

The user has three options for accessing and filtering information in the database. She can either choose to filter by category, questions or just view all the models in the database (U5,U6). The options are shown in figure 11.7. For example the user can select category and from there view all recognition questions in that category.

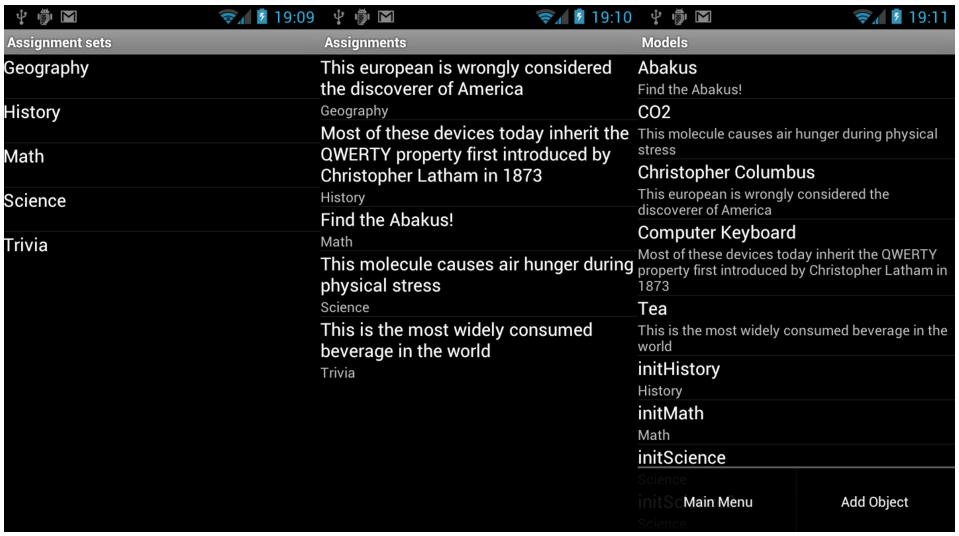


Figure 11.7: The application shows all categories that contain recognition questions in the database. *Left:* Categories with object recognition questions. Clicking a category shows assignments in the category. *Center:* All object recognition questions in the database. Clicking a question allows for editing of the answer. *Right:* All models in the database. Clicking an item in the list allows for deletion if the object is not connected to a question. New models can be added from the options menu.

The user can add new models by selecting add object from the options menu when she is in the database models view (U3). The first time she enters model creation the application presents a walkthrough giving tips for creating good models (U2). This help can be accessed at any time through the options menu in the model creation activity. (ref. figure 11.8).

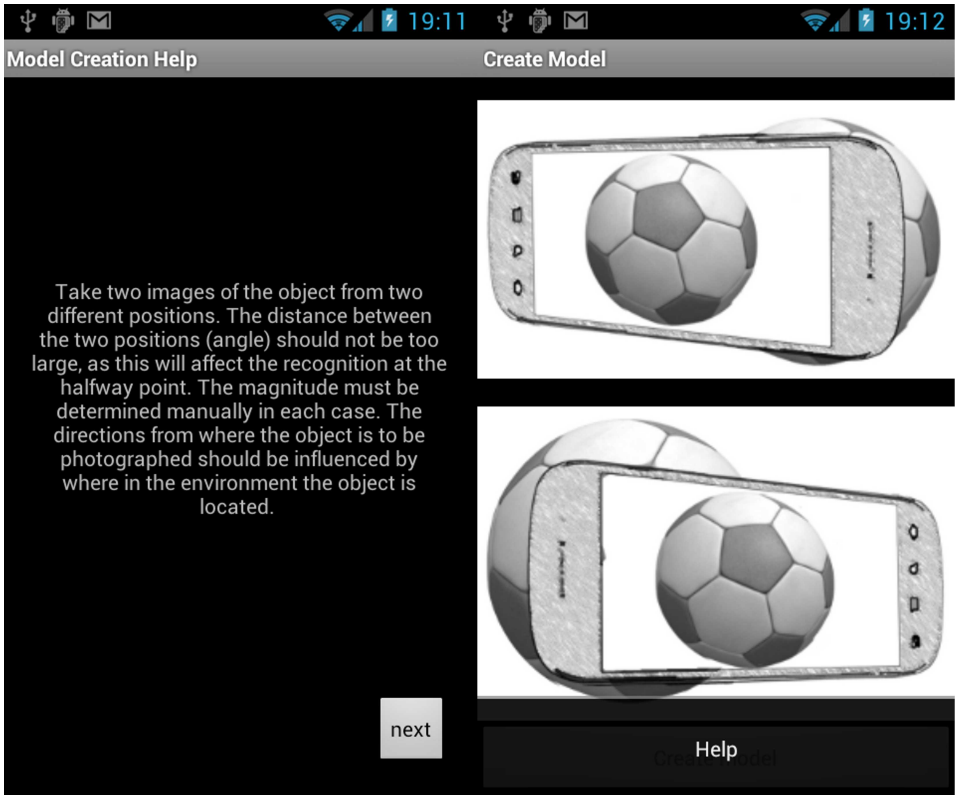


Figure 11.8: Model creation help is initialized automatically the first time the user tries to add a new model. *Left:* Help goes through a series of steps explaining the creation process. *Right:* Model creation help is accessible from the options menu.

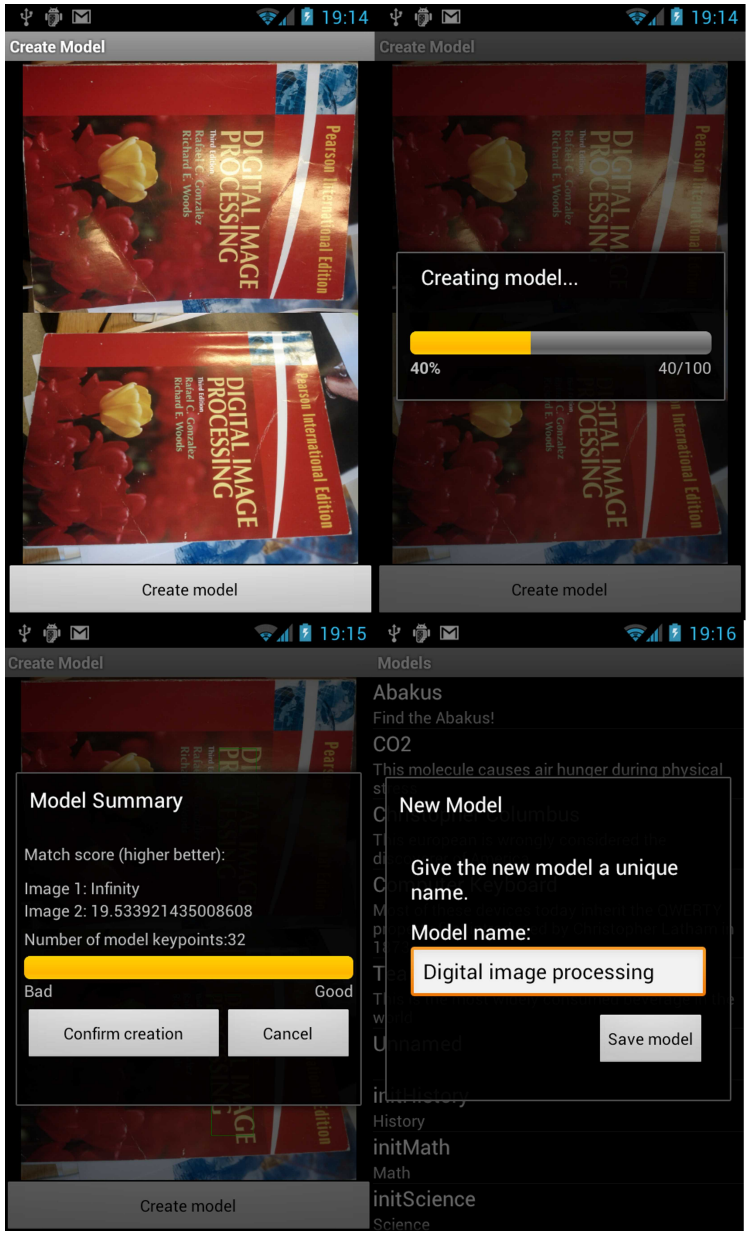


Figure 11.9: The steps involved in creating an object model. *Top left:* The user grabs two images of the object from two different poses. *Top right:* The application shows a progress bar during construction. *Lower left:* A model summary is shown after the construction indicating the quality of the model(PR5). The user confirms if the model is satisfactory. *Lower right:* Users adds metadata to the model before saving (PR4).

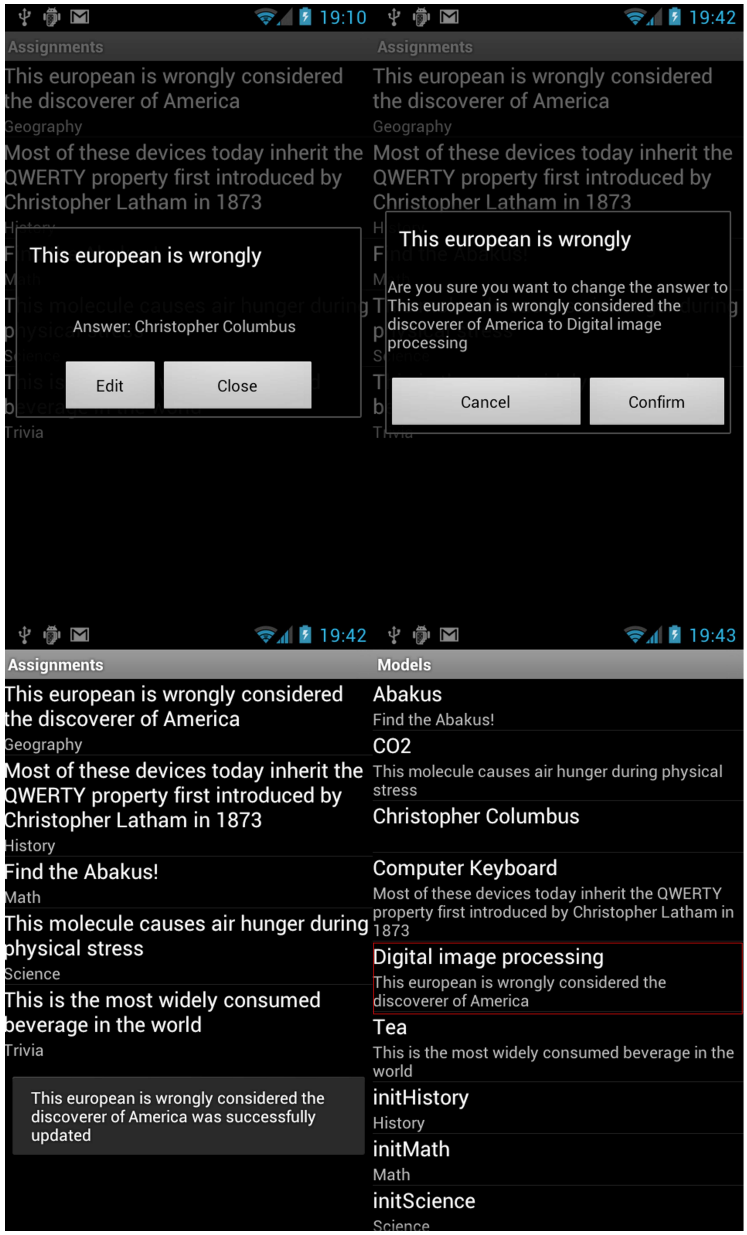


Figure 11.10: The edit recognition question answer process. *Top left:* The user selects a recognition question and choose edit. *Top right:* The users selects a model from the list and confirms the change. *Lower left:* The system confirms that the change has been done. *Lower right:* The new answer is now connected to the recognition question.

Figure 11.10 shows the process involved in editing a recognition question answer

(U7). This task is straight forward; the user select the recognition question, select edit and chooses the new answer from the model list.

The processes involved in creating the model is shown in figure 11.9. The user takes two images of the object from two poses and selects create model. After the model has been built the user is presented with a summary model quality (PR5). By selecting cancel the model is mapped onto the input images, shown by a bounding box around the matched keypoints(PR8). If the user is satisfied she recreates the model. If not she should change one or both of the input images.

Figure 11.11 shows the process of deleting a model (U4). The delete option is only available to the user if she selects *"Show/Edit/Delete Objects"* from the main menu. The system only allows deletion of models that are not connected to a recognition question. This is done in order to prevent breaking the database integrity.

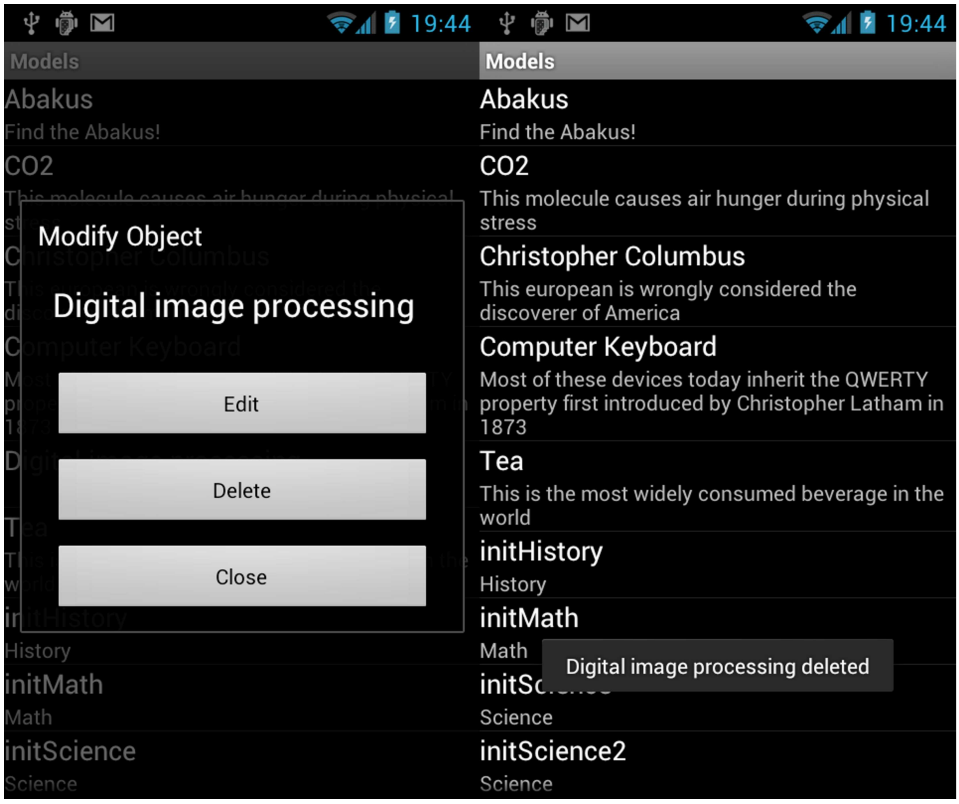


Figure 11.11: The user is allowed to delete models that are not connected to questions. *Left:* The delete option is shown if the user selects the *Show/Edit/Delete Objects* option in the main menu. *Right:* The model was successfully deleted.

11.2 Framework bugs

The example applications were developed and debugged on different Android virtual devices and the authors own Android device (ref. appendix A). The applications run fine on these. This is not true for other devices however, such as the HTC Desire HD and the Samsung Galaxy SII. The problem is related to the custom camera activities implemented by the device manufacturers, which causes the operating system to behave differently. The example applications are restarted spuriously while they are in the background. For one camera activity the background application can be restarted several times. This makes it hard to keep track of the state of the application. This behaviour is spurious on the HTC Desire HD and consistent on the Samsung Galaxy SII. Since the author does not have access to the latter for debugging, fixing it is difficult. This problem is critical and should be given the highest priority by the framework maintainer.

The bug is not in the parts of the applications that belong to the framework, but the solution should be moved into the framework since this will be a common problem for all applications in the domain.

All known bugs in the framework and the example applications are listed in appendix F along with ideas for solutions, the reason for them not being solved and the authors assessment on their repair difficulty.

Until the bugs are fixed the author recommends using a Samsung Galaxy Nexus S for application testing and framework development.

Chapter 12

Object Recognition Results

This chapter presents recommendations for initial settings of model construction and object recognition, model creation and settings, recognizer settings and the frameworks recognition correctness and performance. The two first sections, model building and correctness, are taken from the authors original report[63]. The recognition scheme used is the same, but the underlying architecture is changed. These are included to show the constraints and the suitability of the implemented recognition solution and provide documentation for the model construction. The last part documents the performance changes after applying the performance increase tactic.

The last of the unmapped functional requirements in table 9.1 are mapped to show that the recognition solution fulfils the requirements. The mapping is given in parenthesis.

12.1 Model building

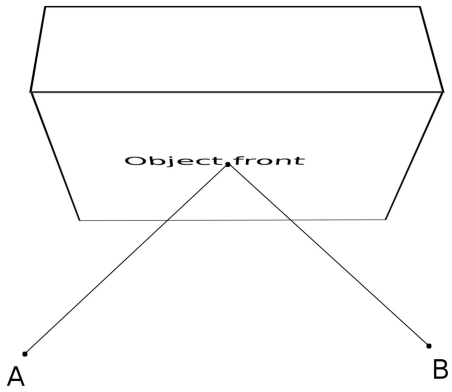


Figure 12.1: Model creation: Object photographed from two different directions A and B

Model construction must be influenced by where in the environment the object is to be placed. If the object is located at the same height as the average person and in such a way that the capture device is nearly perpendicular to a specific surface, figure 12.1 shows two good positions (A and B) from which the model can be constructed. This should give good recognition results from views between A and B. On the other hand a to wide an angle between these will cause trouble finding a representative set of keypoints. The author recommends making sure that the model consists of all of the 32 keypoints, (current model limit). A wide angle will also cause problems in recognizing the object from the center position (middle of A and B in figure 12.1). Figure 12.2 gives two examples of object poses which have provided good models.

These are not absolute rules and are based on the authors own trial and error. Tweaking to fit specific purposes must be expected.

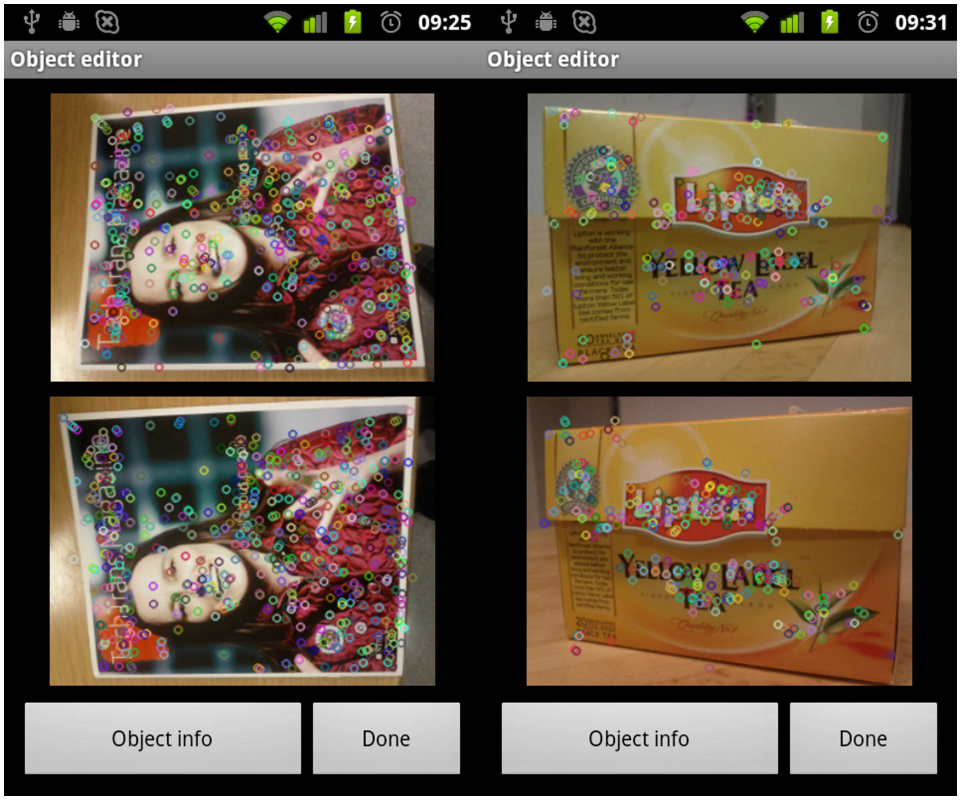


Figure 12.2: Examples of model bases that yield good recognition results.

The model has two tunable parameters: Number of keypoints and ratio(ref. table 6.1). These are set 32 and 0.7, which are the same values as in the original articles

[64, 65]. If needed these can be changed through *LICOVcreatorOptions* (ref. figure 10.3) in the recognition package (ORR15). Models are built using image sizes of 600x600.

12.2 Recognition settings

The default recognizer parameters are shown in table 12.1. These have been found through trial and error and are considered good starting points. The parameters are explained in section 6.4.3 and table 6.1. They can be accessed through the *LICOVmatcherOptions* (ref. figure 10.3) in the recognition package (ORR15).

Parameter	Value
T_{dist}	150
$T_{remaining}$	8
T_{match}	0.22

Table 12.1: Default recognizer parameters

12.3 Correctness

This section proves that the recognition scheme is suitable for real world environments, where factors such as light and object perspective varies, if the the image resolution is above above a certain threshold.

12.3.1 2D recognition (ORR1) and invariance to perspective, scale and rotation

The recognition scheme's invariance to perspective, scale(ORR8) and rotation(ORR7) were presented in figure 7.4. Appendix B show the data from the two tests.

Recognition at 600x600 resolution provides better results with correct recognition in 92% of the cases. 2% of the images are wrongly identified. At resolution 400x400 correct recognition is only achieved in 64% of the images. 33% are not recognized at all and 3% are wrongly identified. From table B.1 it is obvious that some images are more difficult to match and sensitivity to rotation is higher than in the 600x600 case.

12.3.2 3D recognition , rotation and recognition in cluttered images

Figure 12.3 shows that the recognition scheme is able to handle recognition of objects in a cluttered 3D environment(ORR2, ORR10). The models used are shown in figure 12.2.

Figure 12.3: 3D recognition of the models shown in figure 12.2. Recognition works on rotation and in cluttered scenes

12.3.3 Invariance to occlusion

The recognition scheme's ability to handle occlusion is based on where the model keypoints are located. As long as a good portion of the keypoints are not occluded the recognition scheme works(ORR11). The more keypoints, the better occlusion is supported at the expense of speed and memory (bigger models and more keypoints to match against). Figure 12.4 show that using 32 keypoints still yields acceptable results.

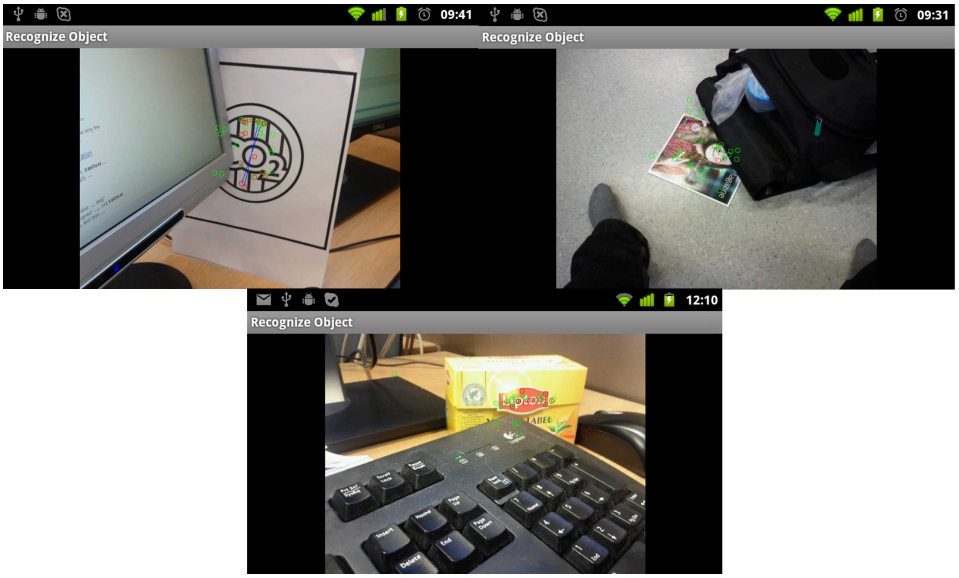


Figure 12.4: Recognition on occluded objects: A glyph (2D object. ref. figure G.7) , a magazine and a box (3D objects. ref. figure B.2)

12.3.4 Invariance to noise

Figure 12.5 show that the recognition scheme is able to handle images with noise (ORR10). The one on the left is taken in low illumination conditions (to introduce sensor noise) and the image to the right is blurred because of movement.

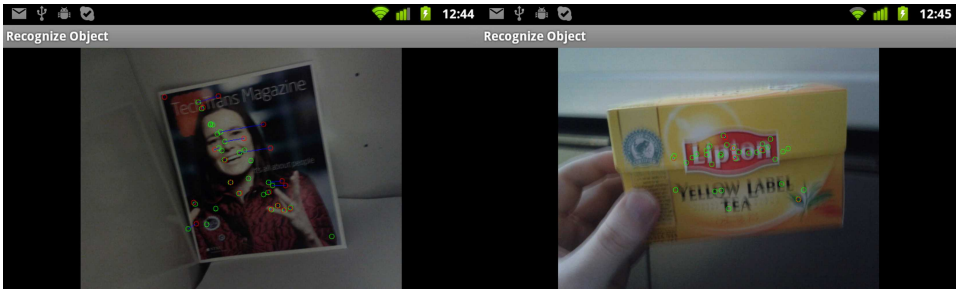


Figure 12.5: Recognition on grainy/blurry images of a magazine and a box. The model is created using views shown in figure 12.2

12.3.5 Invariance to illumination

Figure 12.6 shows the hallmark of using local keypoint features such as SURF. The recognition scheme handles illumination changes extremely well. The two models used are shown in figure 12.2 where the model images are taken under completely different illumination(ORR9).

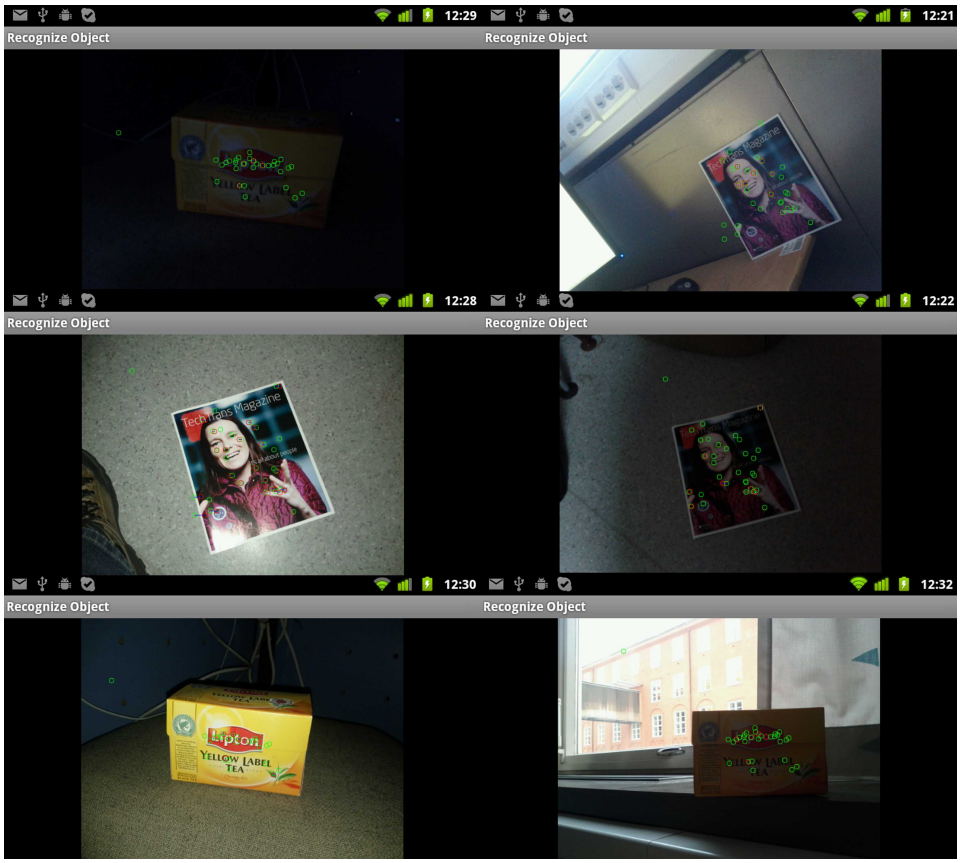


Figure 12.6: Recognition on a magazine and a box under different illumination conditions. The model is created using views shown in figure 12.2

12.4 Performance

This section discuss the success of the performance improvement tactic.

12.4.1 Recognition speed

The author chose to implement an external server for information extraction in order to improve the object recognition performance. A series of simple tests using the quiz example were performed to document the performance improvement . For a thorough explanation of the setup and the result details refer to appendix C. To ensure correctness a minimum resolution of 600x600 was used. Recognition without external server support took on average 21 seconds. By using external server support the average was reduced to 9 seconds, a 43% increase in performance. This number is optimistic since the device ran on a network with a short path to the server. In real world settings the number of devices connected to the server will also degrade performance.

The recognition performance is not adequate to fulfil the non-functional requirements ONR2 and PNR2.

Part IV

Evaluation

Chapter 13

Evaluation

In this chapter the author evaluate the research, development and documentation process. The author reflect on the success of the framework evolution and discuss the success of applying the chosen tactics to reach the thesis goals of improving flexibility, usability and performance.

13.1 Research

The previous specialization project formed the background for this thesis. The authors domain and application knowledge gained from that project and the evaluation from that project formed the basis for the continued development effort. Since the author initially did not know how to solve the remaining challenges a series of research questions were created. The main issue was in which direction the framework should be evolved. The research questions were divided into two categories, frameworks and object recognition. Since these questions do not have one simple answer, the presentation of the possible different solutions are meant as a presentation and discussion of the possible solutions.

Answers to research questions Q1-Q7 are presented informally through the literature study in chapter 5. The questions formed the basis for a systematic approach for performing an in depth study aiming at informing on proven successful techniques and best practices for developing and documenting frameworks. A proper definition of flexibility and usability in the context of framework as well as a definition of what a framework is allowed for deeper understanding of which directions the framework could be evolved. The challenges and benefits help in communicating which factors that needs to be considered by the customer if they ever plan to benefit from using this framework. The two main lessons to remember is that frameworks are not free and it is important that the customer commits the time and resources necessary if they ever wish to benefit from using the framework. The

other lesson is that the success of a framework is both dependent on good development practices and proper documentation that takes the framework stakeholders concerns into consideration.

Answers to research questions Q8-Q10 are answered in chapter 6 in the same informal manner as the ones answered in chapter 5. The importance of proper documentation and knowledge transfer were the main reason for including this chapter. The chapter explains the context of local object recognition and how the implemented solution for recognition and model creation works. This also allowed the author to correct previous mistakes and remove parts not necessary for understanding the solution. Correctness and performance can be ambiguous. The author presented definitions for these in context of the object recognition in this thesis.

The research on frameworks and object recognition allowed for a proper evaluation, rooted in scientific knowledge, of the artefact delivered from the specialization project. The newly gained knowledge allowed the author to answer the rest of the research questions, Q11-Q12 by giving possible solutions to the challenges presented in the specialization report as well as tactics for evolving the framework with respect to the goals of improving flexibility, usability and performance.

The knowledge gained from eliciting solutions for research questions Q1-Q12 allowed the author to answer the the main research question: *How to evolve the framework towards production quality?*. The answer to this question is given by the new artefact, its documentation and the rationale behind the changes.

13.2 Development

The author chose to continue the development using example applications as the basis for the framework development. This method of development follows the systematic approach presented by Roberts and Johnson which is presented in section 5.5.4. This tactic combined with applying the rules presented in chapter 5.5.5 has resulted in separation of the framework and the example applications. The framework is now distributed as an Android library project which can be reused by referencing it in the application projects.

The author think that this tactic has proven extremely useful. Reusing the code from the original prototypes has allowed for new abstractions and more important development of components. The usage of OpenCV over two semesters has helped the author understand an important component of the framework. The author better understand the steps necessary for manipulating structures and know the limitations of this component. One of these limitation is that the Android port has no support for persistence for the *KeyPoint* and *Mat* data structures. This functionality is available in the C++ implementation. The author therefore decided to use Java Native Interfaces to directly access these operations enabling a cross platform storage solution for the framework as well as object self serialization. Self serialization simplifies the application storage implementation (ref. section 10.2.3).

The continued use of the tactic has allowed the author to evaluate its qualities. Changing tactic at this stage of development would not have been as useful since the application domain already was fleshed out in the specialization project. Also valuable time must have been spent on learning a new development method.

The prestudy worked as a brainstorming exercise that allowed the author to elicit possible solutions for improving the object recognition performance. Research into framework development practices helped the author define series of goals for improving the framework architecture. The framework had a flat class structure containing all the functionality. The framework was shipped as part of an application. The framework code was intertwined with the example's. By applying tactics such as abstract activities the author managed to separate the framework from the examples. Using tactics for building components allowed for wrapping of domain knowledge and creating a framework that requires less domain knowledge to use. Applying the full set of tactics resulted in a framework which separate concerns into their own packages organized as a hierarchy. This increases flexibility and help future framework developers decide where to put new functionality and users can more easily find the functionality they need.

The cross platform storage solution allowed for easier implementation of an external information extractor. The author chose the external information extractor solution over the one using a different extractor. The rationale for this was that a lot of time have had to be used to research and test the suitability of different information extractors with the currently implemented recognition solution. This would have hampered the flexibility and usability goals. A middle ground was chosen where the author added support for all feature detectors and descriptor extractors in the framework, allowing for flexibility where the framework users themselves can perform the suitability tests themselves.

To improve flexibility the author chose to implement parts of the recognition scheme. The author chose to use socket connections for communication using C++ supported by OpenCV. The initial choice of using OpenCV proved to be a good choice together with implementing a cross platform storage solution. This allowed for easier development of an external information extractor where most of the required functionality already was provided through OpenCV. The choice of only implementing parts of the recognition on the external server was to ensure that not too much effort were committed before a proof of concept for the solution proved usable. The author had to validate that there were a significant enough performance gain in using an external solution, a lot of time is wasted on socket initialization and data communication, which compared to local operations are extremely slow. If this had failed the author would have fallen back to an alternative where new information extractors would have been tested. Implementing the object recognition in a series of steps further improve the frameworks flexibility. The author recommends continued stepwise implementation of the pipeline nodes in figure 6.1-6.3. Users can then create their own mix of externally supported recognition that better fit their application domain.

Currently the external server is not ready for production environments. The au-

thor prioritized implementing protocols and the recognition scheme. The server is dependent on POSIX systems and Eclipse for compilation. Nor is the data sent between the client and the server encrypted, exposing the authentication protocol which is sent in clear text.

The author provided hooks and default behaviour through abstract activities and abstract and concrete tasks. These proved extremely useful. Figure 10.8, 10.9, 10.11 and 10.12 show high design and code reuse. The default implementations allow application users to implement already working solutions.

The application examples contains a bug that renders them unusable on a certain devices. Time constraints and lack of access to devices that coherently expose the problem are the reasons for the bug not being fixed. The bug is not exposed by the authors own development device (ref. appendix A) and the Android virtual devices. This problem must be given the highest priority from the framework maintainer. The lesson learned is that development of Android applications involving external activities should be performed using a set of Android devices from different manufacturers. This does not ensure complete cross device compatibility, since the applications run fine on Samsung Nexus S devices but not on Samsung Galaxy SII devices, but using different devices should reduce the risk of developing device dependent applications. Catching problems such as this at an early stage reduces the problem solution complexity.

The author has proved that the framework is usable within the customers domain by providing examples that incorporate the object recognition in a knowledge quiz. All the requirements in chapter 9 are by the author considered satisfied except for performance. The lower bound on the computational complexity is provided by the correctness requirement where 600x600 is considered the lowest allowable resolution, which should allow for about 90% correct recognition at arms length and suitable models (ref. appendix B). Using external information extraction over local recognition on the development device, an increase in recognition performance of 42% is possible in ideal conditions. This reduces the average recognition time from 21 seconds to 9 seconds (ref. appendix C) in the quiz application. This is still far from the requirement of 1-2 seconds, but a decent improvement in recognition speed. In order to achieve the performance goal there are several choices which can be applied in isolation or in combination involving requirements for faster devices, relaxation of correctness requirements, relaxation of performance requirements, faster keypoint detector, faster descriptor extractor or perform the full recognition on an external server. Visual effects can also be applied to distract the user, making the recognition seem faster than it really is.

The author is optimistic that adding all object recognition to the external server will improve the object recognition performance even further. The reason for this is the availability of increased computational resources. Less data has to be transferred to and from the server. Currently the original bitmap has to be sent and the OpenCV data structures has to be returned. XML and YAML are not optimized for reducing the storage size since they use text to store primitives. Possibilities of reducing the size using compression should be looked into.

13.3 Documentation

Documentation is crucial for the framework success. Initially the user planned to follow the documentation process presented in section 5.6.4. This is not possible however because of the limited time available. Therefore this document along with the example applications and the code documentation (Javadoc and Doxygen) are the framework documentation. This is not ideal since the users must search through a lot of information that are not part of the framework. The author has however enabled hyperlinks in the table of contents, allowing users reading the pdf version to quickly jump to chapters of interest. Because of this the thesis contains extensive information of how the object recognition is implemented, the architectural choices and the rationales and trade-offs for creating them. Collecting as much information in one document reduces the risk of information being incomplete or lost.

The author recommends that the framework owner applies the process in section 5.6.4 and creates hyperlinked documentation, extracting the parts necessary from this document and combining it with the code snippets from the example applications and using the Javadoc and Doxygen html files as support. The framework document structure should be similar to the one in figure 5.1, where potential users quickly can identify the frameworks suitability with their needs. Following the systematic process ensures that the documentation will address the concerns of the stakeholders in section 1.4 and 5.6.1.

The example applications are dependent on external resources. The author realize that this is not optimal for framework users that plan on using these for learning purposes. The rationale for doing this was was a trade-off between the customers needs and the documentation needs of the framework. The applications show off the framework in a context suitable for the customer while providing information on framework usage. Simpler stand alone applications should be considered that are easier to instantiate. One way to solve this is to implement the original prototypes using the new framework. This would also help in proving the suitability of the framework.

Based on the requirements in section 5.6 the author considers the provided documentation complete:

- **The purpose of the framework:** The purpose of the framework is given in part I of this document.
- **How to use the framework:**
  - Installation instructions are given in appendix D and appendix E.
  - How to use the framework is documented by the example applications, code documentation (Javadoc and Doxygen) and chapter part III of this document.
- **The detailed design of the framework** is presented in part III using different views based on the needs of the identified stakeholders in section 1.4 and 5.6.1.

Documentation of the framework is not enough by itself. As described in section 5.4 often mentoring from developers and expert users are necessary. Therefore the author has offered to be available for a limited amount of time in order to help the customer and the framework maintainer until they gain the knowledge necessary to begin application development and continued development of the framework.

13.4 Overall thesis evaluation

The author believe that the comprehensive prestudy is necessary in order to gain enough domain knowledge to produce a satisfactory artefact. This combined with the vast amount of documentation necessary to document it properly is by the author considered to much work for one student alone, which the size of this document supports. Research in chapter 5.5.4 states that framework teams should be composed of 2-4 developers, unless the developer is a domain expert and experienced framework developer. Based on these facts and with the exception of the serious bug in the example applications the author is quite satisfied with the achieved result. The framework architecture is greatly improved over the old one and the tactics implemented for development and documentation has improved flexibility, usability and performance. The framework is not ready for commercial production quality but a lot closer.

Chapter 14

Conclusion

14.1 Conclusion

This master thesis is a continuation of the author's previous work from the *TDT4501 - Specialization project*. The documentation, artefact and evaluation from that project formed the initial input where the goal was to improve upon the existing solution with respect to flexibility, usability and performance. In order to find a solution to the main issue, *How to evolve the framework towards production quality?*, the author defined a series of sub-questions that by finding possible answers to these should provide enough knowledge to define the improvement approach. The answers to the questions were informally conducted by performing a literature prestudy into frameworks and object recognition along with formal definitions of flexibility, usability correctness and performance in context of this thesis. The results from the two previous chapters formed input to an evaluation of the initial framework artefact, which gave possible answers to the remaining research questions.

From the prestudy the author defined a set of goals and tactics for improving the flexibility, usability and performance of the framework. The end result is a framework that use proven successful techniques for implementation and documentation which should increase the chance for framework reuse. The framework has evolved from a completely white-box framework delivered as part of a prototype application to a stand alone grey-box framework with customizable components. The components hide object recognition domain knowledge from the users, allowing them to focus on application problems rather than object recognition problems. The provided examples are customized towards the customer and they show that the framework is suitable for their needs.

The framework offers increased flexibility by supporting external information extraction. This improves the object recognition performance by as much as 42% when images of 600x600 pixels are used. This image size forms the lower bound

for the computational complexity since lower resolutions affect correctness negatively. Resolutions of 600x600 ensure correct recognition in 90% of the cases when the device is at arms length from the object and the object model is of satisfactory quality. Flexibility and usability are further increased in the framework by implementing default behaviour that falls back from external to local information extraction if the connection with the server fails, allowing them to run in isolated environments. The correctness requirement is relaxed in order to preserve performance. The default fallback resolution is 300x300 and correctness is reduced to around 60%.

The bugs in the framework and example applications, lack of encryption between the framework and its external resources and the unsatisfied performance requirement result in a framework falling short of being ready for production environments. It is however a lot closer than its predecessor.

14.2 Future Work

In this section the author presents suggestions for future work which will guide the continued evolution of the framework. The suggestions are broken up into smaller sections suggesting different approaches for improvement

Evaluate the existing solution

The customer should evaluate the quality of the current framework solution. The results can be used as input to an new framework iteration. The evaluation fits with the development process used by the author. The evaluation works as a quality feedback loop that help improve the framework quality (ref. section 5.5.4).

The evaluation can be performed using different resources. The author has identified the following:

- Evaluation is performed by the customer
- The evaluation can be done as part of a software architecture course, such as NTNUs *TDT4240* which currently is taught by the thesis' supervisor. The framework can be used as a case study where groups of students use the framework to develop applications that may be helpful to the customer. They can give feedback on tactics (good and bad) used in the framework and point out weak parts of the documentation while learning from the author's mistakes. It is important to also provide this documentation for the framework being useful for them.
- Evaluation can be performed through a project specialization course such as TDT4501. The same course from which this project originated. A student evaluates the solution and point out weak spots that needs improvement. As part of this project proper documentation can be created. I.e. by using the technique discussed in section 13.3

Number of student developers

In order to ensure development of new and high quality iterations of the framework the development team should consist of more than one student. Domain experts should also be made available.

- The continued evolution should be done by more than one student. Most students are not experienced framework developers nor domain experts. 2-4 students should be satisfactory. (5.5.3). There should be at least one student familiar with image processing (object recognition expert) and one familiar with software architecture (architecture and documentation expert).
- There should be a supervisor with knowledge in the field of computer vision in addition to the the software architecture supervisor. Getting professional help aid novice users in producing higher quality artefacts. The institute, at which the author currently studies, has such expertise in Prof. Blake and Assoc. Prof. Hokland.

Information extraction

SURF is moved into non-free part of OpenCV starting with version 2.4.1. The commercial implications of this needs to be clarified.

Example applications

This section presents future work for improving to the example applications ranked in order of importance:

1. Fix the bug in the example applications. An explanation of these and possible solutions are presented in appendix F.
2. Add new simpler example applications that can run in isolation. (i.e re-implement the old prototypes in the previous version of the framework using the new framework iteration).

Detector

This section presents future work for improving to the detector ranked in order of importance:

1. Fix remaining bugs in detector. An explanation of this and possible solutions are presented in appendix F.
2. Add support for windows sockets.
3. Make detector independent of Eclipse.

4. Secure communication with the client.
5. Add support for matching.
6. Add support for model creation.
7. Add support for removal of outliers in both model creation and matching. This can be done using RANSAC which is provided by the OpenCV library. Building better models and matching should be more accurate since outliers will be removed. Where this can be implemented in the model construction. process is documented in section 6.4.2.
8. Add support for OpenCV 2.4.1
9. Standardize the communication and data transfer protocol, i.e HTTPS and XML. This reduces the amount of critical code in the framework thus easing maintenance and improving stability. Standardized protocols prolongs the life of a system.

Framework

This section presents future work for improving the framework ranked in order of importance:

1. Fix remaining bugs in framework. An explanation of this and possible solutions are presented in appendix F.
2. Use *AsyncTask* get method instead of the *get** methods in the tasks in *org.cyberlab.local.activity.task*. For more information about this refer to their Javadoc.
3. Secure communication with the server.
4. Add support for external matching
5. Add support for external model creation
6. Add support for removal of outliers in both model creation and matching. This can be done using RANSAC which is provided by the OpenCV library. Building better models and matching should be more accurate since outliers will be removed. Where this can be implemented in the model construction. process is documented in section 6.4.2.
7. Test and add new information extraction methods using other detectors and extractors.
8. Add support for synchronization of SQLite databases via external server.
9. Add support for data storage in external database, ex MySQL.

10. Add support for OpenCV 2.4.1.

11. Check if bug in OpenCV k-nearest neighbour (knn) match is fixed in the new version of OpenCV. The workaround in the framework (located in *LocalRecognitionTools* and *LICOVrecognitionTools*) should be removed. This reduces outdated code as well as amount of code in framework that needs to be maintained.

12. Standardize the communication and data transfer protocol, i.e HTTPS and XML. This reduces the amount of critical code in the framework thus easing maintenance and improve stability. Standardized protocols prolongs the life of a system.

13. Research and find other object recognition schemes suitable for mobile devices. The extensive research in the object recognition field results in new and innovative solutions.

Documentation

This section presents future work for improving to the framework documentation ranked in order of importance:

1. This is the most important part and will continue to be: Update documentation to reflect changes and to avoid drift. Fixing outdated documentation is time consuming. The developers involved in making changes must update the documentation to reflect their changes. Over time these may not longer be available and critical architectural information may be lost.

2. Generate stand alone hyperlinked documentation that minimizes the need for maintenance from this document, example applications and the code documentation. The process in section 5.6.4 is suitable for this purpose.

3. Create code snippet examples from the example applications.

4. Look into other possible ways of performing documentation that require less resources and decrease the necessity for maintenance.

References

[1] "Simple guide to installing android apk files." http://www.brighthub.com/mobile/google-android/articles/37151.aspx, 6 2012.

[2] Google, "Platform Versions." http://developer.android.com/about/dashboards/index.html, 6 2012.

[3] Droid Sector, "Samsung Nexus S." http://www.droidsector.com/devices/samsung-nexus-s, 6 2012.

[4] M. of Culture and C. Affairs, "Report no. 14 to the storting: Video games." http://www.regjeringen.no/en/dep/kkd/Documents/regpubl/stmeld/2007-2008/report-no-14-2007-2008-to-the-storting.html?id=518787, 6 2012.

[5] Wikipedia, "History of video games." http://en.wikipedia.org/wiki/History_of_video_games, 6 2012.

[6] C. O. AS, "About cyberlab." http://www.cyberlab.org/wp/wordpress/?page_id=2, 6 2012.

[7] Y. Marion, D. E. Holmstrøm, J. Eriksson, M. Opheim, K. Babington, K. B. Viktil, and J. ÌĄs Valero, "Augmented reality using android," tdt4290 customer driven project, Norwegian University of Science and Technology (NTNU), http://www.ntnu.edu/, 11 2010.

[8] M. Zyda, "From visual simulation to virtual reality to games," *Computer*, vol. 38, pp. 25 – 32, sept. 2005.

[9] V. Basili, "The experimental paradigm in software engineering," (Berlin, Germany), pp. 3 – 12, 1993//. experimental paradigm;software engineering;complexity;well defined primitives;research paradigms;experimentation;software research;software development;.

[10] Wikipedia, "Android (operating system)." http://en.wikipedia.org/wiki/Android_(operating_system), 6 2012.

[11] M. H. Goadrich and M. P. Rogers, "Smart smartphone development: Ios versus android," (Dallas, TX, United states), pp. 607 – 612, 2011. Android;Apps;Eclipse;IOS;Iphone;Java;Objective-C;Smartphones;Xcode;.

[12] Android, "Application fundamentals." `http://developer.android.com/guide/components/fundamentals.html`, 6 2012.

[13] Android, "Development Considerations." `http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#considerations`, 6 2012.

[14] Android, "Installing the sdk." `http://developer.android.com/sdk/installing/index.html`, 6 2012.

[15] Android, "Managing Virtual Devices." `http://developer.android.com/tools/devices/index.html`, 6 2012.

[16] Android, "Using hardware devices." `http://developer.android.com/tools/device.html`, 6 2012.

[17] G. Bradski, "The opencv library," *Dr.Dobb's Journal*, vol. 25, no. 11, pp. 120–120–125, 2000. 202684726; 62275148; Copyright Miller Freeman Inc. Nov 2000; Bradski, Gary; English; 18657; 120-125; San Mateo; IDRD; Programming languages; Data imaging; DDJOEB; Nov 2000; 1243684; Computer programming.

[18] O. project, "Opencv for android release notes." `http://code.opencv.org/projects/opencv/wiki/Android_Release_Notes`, 6 2012.

[19] O. project, "Android." `http://opencv.willowgarage.com/wiki/`, 6 2012.

[20] Google, "GSoC: Open Source Computer Vision Library (OpenCV)." `http://www.google-melange.com/gsoc/org/google/gsoc2012/opencv`, 6 2012.

[21] O. project, "Opencv v2.3 documentation." `http://opencv.itseez.com/`, 6 2012.

[22] R. LaganiÃÍre, *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, 2011.

[23] O. project, "Android best practices." `http://opencv.alekcac.webfactional.com/android/android-best-practices.html`, 6 2012.

[24] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, ch. Application Frameworks, pp. 3–22. Wiley Computer Publishing, 1999.

[25] R. Johnson and B. Foote, "Designing reusable classes," *Journal of object-oriented programming*, vol. 1, no. 2, pp. 22–35, 1988.

[26] R. E. Johnson, "Frameworks = (components + patterns)," *Commun. ACM*, vol. 40, pp. 39–42, October 1997.

[27] A. Aguiar and G. David, "Patterns for effectively documenting frameworks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6510, pp. 79 – 124, 2011.

[28] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, "Application frameworks," in *Building Application Frameworks*, ch. 1, pp. 3–27, Wiley Computer Publishing, 1999.

[29] J. Hautamäki, *A survey of frameworks*. University of Tampere, Department of Computer Science, 1997.

[30] *McGraw-Hill Dictionary of Scientific & Technical Terms*. The McGraw-Hill Companies, Inc., 6 ed., 2003.

[31] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ch. Understanding Quality Attributes, pp. 71–98. Addison-Wesley, 2007.

[32] J. Vlissides, "Protection, part i: The hollywood principle," *C++ Report*, vol. 8, no. 2, 1996.

[33] L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*, ch. Creating An Architecture, pp. 69–261. Addison-Wesley, 2007.

[34] E. Gamma, R. Helm, R. E. Johnson, and J. Vissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ch. Behavioral Patterns, pp. 221–351. Addison-Wesley Professional, 1995.

[35] Oracle, "Functor Pattern." `http://coherence.oracle.com/display/INCUBATOR/Functor+Pattern`, 2012 6.

[36] D. Roberts, R. Johnson, *et al.*, "Evolving frameworks: A pattern language for developing object-oriented frameworks," *Pattern languages of program design*, vol. 3, pp. 471–486, 1996.

[37] E. Gamma, R. Helm, R. E. Johnson, and J. Vissides, *Design Patterns: Elements of reusable object-oriented software*, ch. Introduction, pp. 1–29. Addison-Wesley Professional, 1995.

[38] T. Inc., "Buidling object-oriented frameworks." `http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf`, 6 2012.

[39] G. Booch, "Designing an application framework," *Dr Dobb's Journal-Software Tools for the Professional Programmer*, vol. 19, no. 2, pp. 24–35, 1994.

[40] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, ch. Framework Design by Systematic Generalization, pp. 353–377. Wiley Computer Publishing, 1999.

[41] R. Helm, I. M. Holland, and D. Gangopadhyay, "Contracts: specifying behavioral compositions in object-oriented systems," in *Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, OOPSLA/ECOOP '90, (New York, NY, USA), pp. 169–180, ACM, 1990.

[42] R. E. Johnson, "How To Develop Frameworks." `http://wiki.lassy.uni.lu/Special:LassyBibDownload?id=453`, 6 2012.

[43] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[44] K. Koskimies and H. Mössenböck, "Designing a framework by stepwise generalization," *Software EngineeringâĂŤESEC'95*, pp. 479–498, 1995.

[45] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Application Frameworks*, ch. Documenting Frameworks, pp. 495–503. Wiley Computer Publishing, 1999.

[46] R. E. Johnson, "Documenting frameworks using patterns," *SIGPLAN Not.*, vol. 27, pp. 63–76, October 1992.

[47] G. Butler and P. Dénommée, "Documenting frameworks," *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, pp. 495–504, 1999.

[48] M. Morrison, "Java 1.1 unleashed." `http://gbengasesan.com/fyp/2/htm/ch29.htm`, 3 2012.

[49] J. Coplien, "Software design patterns: Common questions and answers," *The Patterns Handbook: Techniques, Strategies, and Applications. Cambridge University Press, NY*, pp. 311–320, 1998.

[50] J. Hollingsworth and B. Weide, "Micro-architecture vs. macro-architecture," in *Proceedings of the Seventh Annual Workshop on Software Reuse*, Citeseer, 1995.

[51] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks*, ch. Reusing Hooks, pp. 219–233. Wiley Computer Publishing, 1999.

[52] J. K. Tsotsos and N. D. B. Bruce, "Computational foundations for attentive processes." `http://www.scholarpedia.org/article/Computational_foundations_for_attentive_processes`, 6 2012.

[53] Wikipedia, "Computer vision." `http://en.wikipedia.org/wiki/Computer_vision`, 6 2012.

[54] M. A. Treiber and M. Treiber, "Introduction," in *An Introduction to Object Recognition*, vol. 0 of *Advances in Pattern Recognition*, pp. 1–10, Springer London, 2010. 10.1007/978-1-84996-235-3_1.

[55] J. Tsotsos, "The complexity of perceptual search tasks," (Palo Alto, CA, USA), pp. 1571 – 7, 1989. computer vision;computational complexity;perceptual search tasks;psychology;bottom-up case;NP-complete;.

[56] J. Tsotsos, "Analyzing vision at the complexity level," *Behavioral and Brain Sciences*, vol. 13, no. 3, pp. 423–469, 1990. cited By (since 1996) 130.

[57] P. Parodi, R. Lancewicki, A. Vijh, and J. Tsotsos, "Empirically-derived estimates of the complexity of labeling line drawings of polyhedral scenes," *Artificial Intelligence*, vol. 105, no. 1-2, pp. 47 – 75, 1998/10/. empirically-derived estimates;labeling complexity;line drawings;polyhedral scenes;3D structure;worst-case complexity;random instances;perspective projections;median-case complexity;.

[58] Wikipedia, "Object recognition." `http://en.wikipedia.org/wiki/Object_recognition`, 6 2012.

[59] M. A. Treiber and M. Treiber, "Summary," in *An Introduction to Object Recognition*, vol. 0 of *Advances in Pattern Recognition*, pp. 183–186, Springer London, 2010. 10.1007/978-1-84996-235-3_8.

[60] C. V. Ramamoorthy and H. F. Li, "Pipeline architecture," *ACM Comput. Surv.*, vol. 9, pp. 61–102, Mar. 1977.

[61] R. C. Gonzales and R. E. Woods, "Filtering in the frequency domain," in *Digital Image Processing*, pp. 199–305, Pearson/Prentice Hall, 2008.

[62] R. C. Gonzales and R. E. Woods, "Image restoration and reconstruction," in *Digital image processing* (M. McDonald, A. Dworkin, W. Opaluch, S. Disanno, and R. Kernan, eds.), pp. 311–389, Pearson/Prentice Hall, third ed., 2008.

[63] M.-G. Karlsen, "Object recognition for android," tdt4501 computer science, specialization project, Norwegian University of Science and Technology (NTNU), `http://www.ntnu.edu/`, 12 2011.

[64] J. Revaud, G. Lavoué, Y. Ariki, and A. Baskurt, "Fast and cheap object recognition by linear combination of views," in *Proceedings of the 6th ACM international conference on Image and video retrieval*, CIVR '07, (New York, NY, USA), pp. 194–201, ACM, 2007.

[65] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951 of *Lecture Notes in Computer Science*, pp. 404–417, Springer Berlin / Heidelberg, 2006. 10.1007/11744023_32.

[66] R. C. Gonzales and R. E. Woods, "Fundamentals of spatial filtering," in *Digital Image Processing*, pp. 144–150, Pearson/Prentice Hall, 2008.

[67] R. C. Gonzales and R. E. Woods, "Using the second order derivative for image sharpening - the laplacian," in *Digital Image Processing*, pp. 160–162, Pearson/Prentice Hall, 2008.

[68] J. C.H. Edwards and D. E. Penney, "Higher order determinants," in *Elementary Linear Algebra*, pp. 82–86, Prentice Hall, 2005.

[69] S. Ullman and R. Basri, "Recognition by linear combinations of models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, pp. 992 –1006, oct 1991.

[70] D. Hearn and M. P. Baker, "Matrix representations and homogenous coordinates," in *Computer graphics with OpenGL*, pp. 237–240, Pearson/Prentice Hall, 2004.

[71] J. C.H. Edwards and D. E. Penney, "Orthogonal projections and least square solutions," in *Elementary Linear Algebra*, pp. 220–229, Prentice Hall, 2005.

[72] Wikipedia, "Euclidian distance." `http://en.wikipedia.org/wiki/Euclidian_distance`, 12 2011.

[73] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571, IEEE, 2011.

[74] W. Garage, "Star Detector." `http://pr.willowgarage.com/wiki/Star_Detector`, 6 2012.

[75] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Gool, "A comparison of affine region detectors," *International journal of computer vision*, vol. 65, no. 1, pp. 43–72, 2005.

[76] L. W. Kheng, "Feature Detection and Matchin (Slides)." `www.comp.nus.edu.sg/~cs4243/lecture/feature.pdf`, 6 2012.

[77] P. Meer, "Harris Detecor (Slides)." `http://cronos.rutgers.edu/~meer/TEACH/ADD/Harrisdet.pdf`, 6 2012.

[78] G. Bradski, "Willow Garage, OpenCV, ROS, And Object Recognition." `http://www.ais.uni-bonn.de/~holz/spme/talks/01_Bradski_SemanticPerception_2011.pdf`, 6 2012.

[79] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features.," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[80] Android, "android.renderscript." `http://developer.android.com/reference/android/renderscript/package-summary.html`, 6 2012.

[81] ZiiLabs, "Android." `http://www.ziilabs.com/products/software/android.php`, 6 2012.

[82] G. Forman and J. Zahorjan, "The challenges of mobile computing," *Computer*, vol. 27, pp. 38 –47, apr 1994.

[83] Android, "Activity." http://developer.android.com/reference/android/app/Activity.html, 6 2012.

[84] "Tips: How to install apk files on android emulator." http://openhandsetmagazine.com/2008/01/tips-how-to-install-apk-files-on-android-emulator/, 6 2012.

Part V

Appendices

Appendix A

Google Nexus S Specifications

This is the test device used to do benchmarking. This table gives a short overview of the device specifications so later comparison benchmarking is possible.

Display	
Screen diagonal	4"
Resolution	800x480
Density	High
CPU	
Manufacturer	Samsung
Type	S5PC110
Speed	1 GHz
Hardware	
RAM	512 MB
Internal storage	16 GB
Interface	Touch screen
Camera	
Main camera	5 MP
Front camera	Yes
Flash	LED

Table A.1: Technical specifications of test device[3]

Appendix B

2D object recognition test results

B.1 Test setup

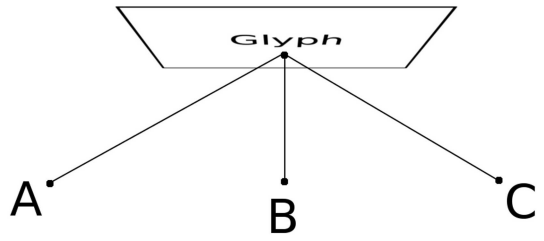


Figure B.1: 2D object recognition test directions

This test quantifies the quality of recognition on simple drawings, glyphs, taken from 3 different directions. The glyphs are printed on sheets of A4 paper and placed on a magnetic board at the same height as the camera. The camera is approximately at arms length from the board. Firstly the glyphs are photographed at the same rotation as the images used for model creation. The three positions are called A, B and C. The situation is shown in figure B.1. Secondly the glyph is randomly rotated and then photographed from the same three positions, now named A', B' and C'. An example of the six views is shown in figure B.2. In all there are 11 glyphs and these can be found in [63]. The same test was performed twice but images were captured at different resolutions; 400x400 and 600x600. Images of the different scenes were only captured once for each test.

Specifications of the test device can be found in appendix A.

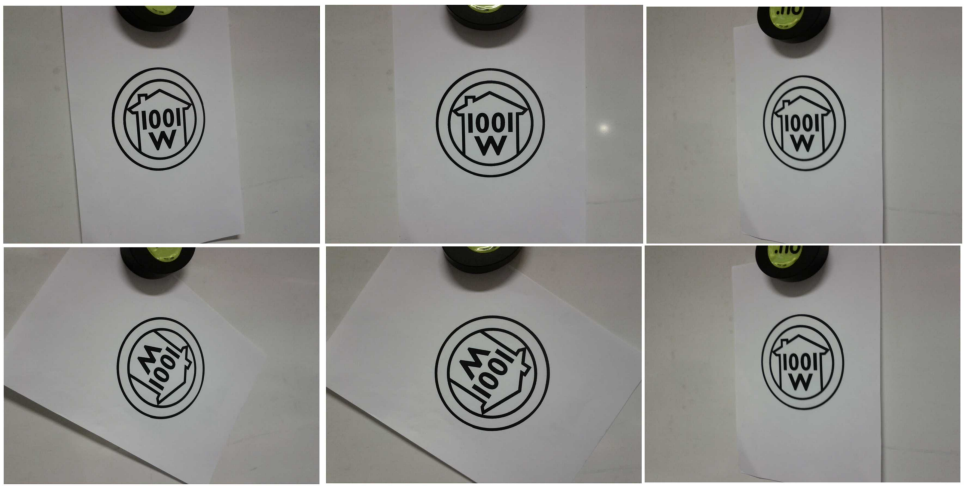


Figure B.2: 2D object recognition test example. *Top row:* Recognition from directions depicted in figure B.1 where object is rotated in same direction as in original model creation images. *Bottom row:* Recognition on random rotated object from directions shown in figure B.1

B.2 Recognition at 400x400

Glyph	A	A'	B	B'	C	C'
1	X	X	X	X	X	O
2	X	X	X	X	X	X
3	W	O	X	X	O	X
4	O	X	W	O	O	O
5	X	X	X	O	X	X
6	X	X	X	X	X	X
7	O	O	X	O	X	O
8	X	O	X	O	X	X
9	X	O	X	X	X	O
10	O	O	X	O	X	O
11	X	O	X	X	X	O

Table B.1: 2D Glyph recognition on figure G.7 results using 400x400 recognition resolution (*X:*Match. *O:*No object matches. *W:*Wrong object matched). A,B,C are images taken from the left, in front of and to the right of the object holding the device oriented the same way as the images used for creating the model. A',B',C' is similar to their counterparts but the device is randomly rotated.

B.3 Recognition at 600x600

Glyph	A	A'	B	B'	C	C'
1	X	X	X	X	X	X
2	X	X	X	X	X	X
3	X	X	X	X	X	X
4	X	O	W	X	X	O
5	X	X	X	X	X	X
6	X	X	X	O	X	X
7	X	X	X	O	X	X
8	X	X	X	X	X	X
9	X	X	X	X	X	X
10	X	X	X	X	X	X
11	X	X	X	X	X	X

Table B.2: 2D Glyph recognition on figure G.7 results using 600x600 recognition resolution (*X:*Match. *O:*No object matches. *W:*Wrong object matched). A,B,C are images taken from the left, in front of and to the right of the object holding the device oriented the same way as the images used for creating the model. A',B',C' is similar to their counterparts but the device is randomly rotated.

B.4 Recognition efficiency

Resolution	$Recognition[s]$
300x300	10
400x400	10
500x500	15
600x600	14
700x700	14
800x800	20
900x900	20
1000x1000	27

Table B.3: Approximate recognition speed on figure G.7 in a database consisting of 25 objects.

Appendix C

Quiz Performance Tests

C.1 Test Setup

This test quantifies the speedup of using an external detector for information extraction. Specifications on the Android device are given in appendix A. The detector ran on Debian squeeze (amd64) using a desktop workstation with 12 GB RAM and 2 quad-core Intel i7-920 processors @ 2.7 GHz. The server was connected to the NTNU campus network while the device was connected to the WIFI (802.11g) NTNU campus network.

The tests were performed using the category initialization activity in the quiz example application. When the recognition process starts the models are preloaded into the device memory. All in all there are 5 models loaded into memory and all are matched against the image data.

Two tests were performed, one where the external server extracted the information and one where the information extraction were done locally on the device. A resolution of 600x600 were used in both tests.

The tests were performed 3 times for each category image. The image IDs in table C.1 and table C.2 are:

1. Figure G.3.
2. Figure G.6.
3. Figure G.4.
4. Figure G.5.
5. Figure G.1.

C.2 Results

Image ID	Test 1[s]	Test 2[s]	Test 3[s]	Avg.[s]
1	8	7	8	8
2	9	11	7	9
3	11	11	8	10
4	10	9	13	11
5	8	7	8	8
Avg.	-	-	-	9

Table C.1: Performance (in seconds) of image information extraction using external detector on images of size 600x600

Image ID	Test 1[s]	Test 2[s]	Test 3[s]	Avg.[s]
1	21	17	19	19
2	19	19	18	19
3	20	21	21	21
4	30	31	29	30
5	17	17	16	17
Avg.	-	-	-	21

Table C.2: Performance (in seconds) of image information extraction using device on images of size 600x600

Image ID	Avg. performance increase[s]	Avg. performance increase [%]
1	11	42
2	10	47
3	11	48
4	20	37
5	9	47
Avg.	12	43

Table C.3: Performance increase using external server for information extraction over information extraction on device using images of size 600x600.

Appendix D

Application Installation

D.1 Phone[1]

Prerequisites:

- Android SDK
- Android USB drivers

For installation of the SDK look at `http://developer.android.com`

For installation of Android USB drivers go to `http://developer.android.com/sdk/win-usb.html`

To be able to install applications from other sources than android market on your phone you need go to Settings → Application Settings and enable Unknown Sources. Also go to Settings → SD Card and Phone Storage → Disable Use for USB Storage. You can enable it again later. Next, open Command Prompt and type: adb install path/file.apk Where path is the full path to the APK file and file is the name of the APK application file. The application is now installed.

D.2 Emulator

Prerequisites:

- Android SDK

For installation of the SDK look at `http://developer.android.com`

Install apk in device emulator [84]

1. Add SDK_ROOT to your system variables pointing to /tools folder under the sdk
2. Run the emulator
3. Copy the apk file to /tools folder
4. Change directory to /tools and run from command line $adb install your_application.apk
   Now check applications list in the emulator and you should see the new application installed and ready.

Appendix E

Framework Installation

E.1 Android

E.1.1 Prerequisites

The CyberlabOD framework requires the following libraries and tools being installed. For installation refer to the links:

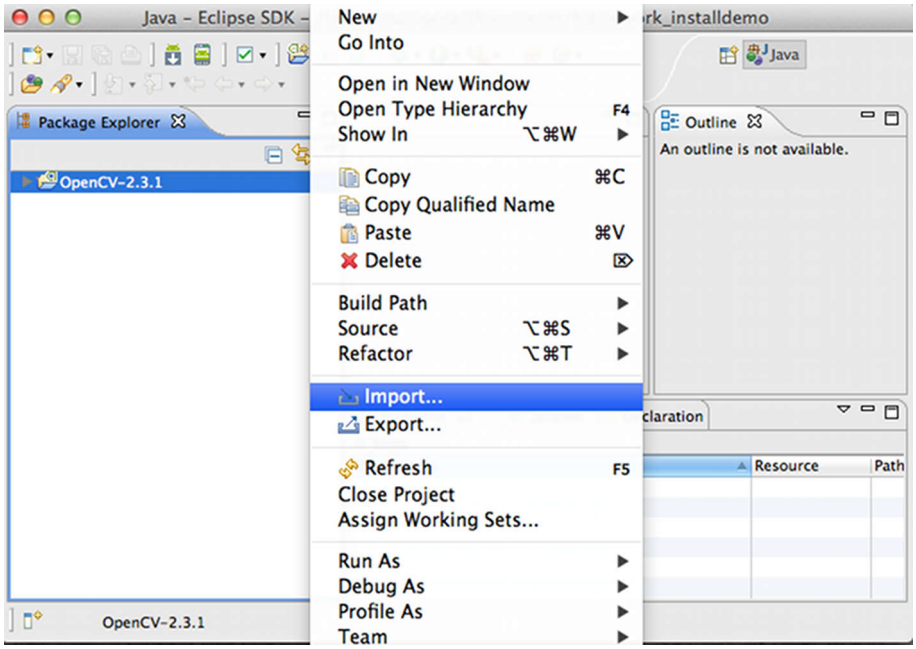
1. **Eclipse IDE** (`http://www.eclipse.org/downloads/`).
2. **Android Development Tools (ADT) for Eclipse**. SDK Platform Android 2.3.1, API 9 (`http://developer.android.com/sdk/installing/index.html`).
3. **Eclipse CDT** (`http://www.eclipse.org/cdt/`).
4. **Android NDK** (`http://developer.android.com/tools/sdk/ndk/index.html`).
5. **OpenCV 2.3.1** (Installation using OpenCV for Android binary package with Eclipse: `http://opencv.itseez.com/doc/tutorials/introduction/android_binary_package/android_binary_package.html#android-binary-package`).

E.1.2 Install Instructions

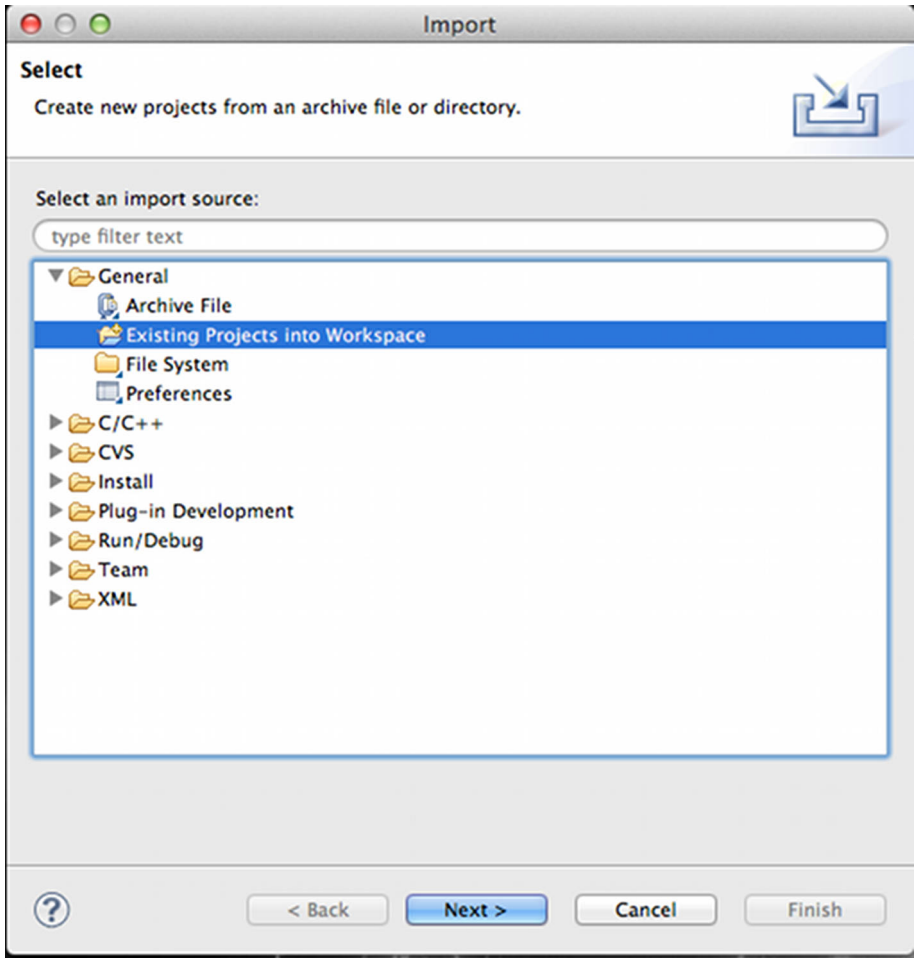
The CyberlabOD library is packed as a ready-for-use Android Library Project. You can simply reference it in your projects.

Each sample included is a regular Android project that already references the CyberlabOD library. Follow the steps to import CyberlabOD and samples into workspace:

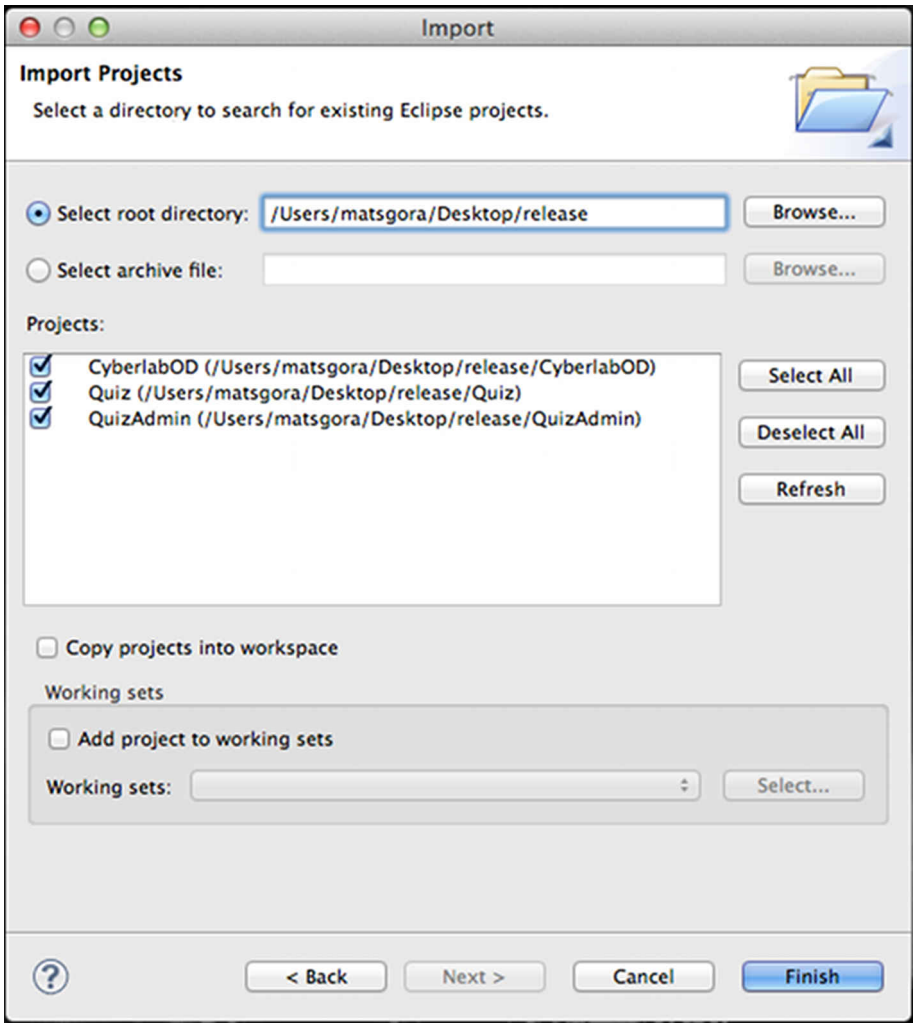
1. Unpack the zip file using your favourite archive utility.
2. Right click on the Package Explorer window and choose the import option from the context menu:



3. In the main panel select General → Existing Projects into Workspace and press the next button:

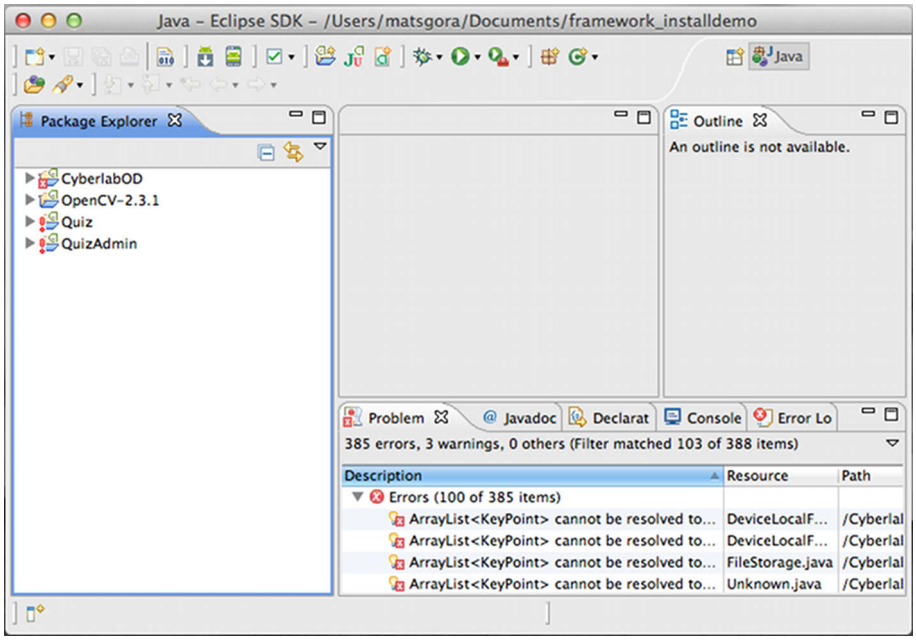


4. Locate your CyberlabOD package folder in the select root directory. (If you have created a workspace in the package directory, then just click the browse button and instantly close the directory, choosing dialog with the OK button!) Eclipse should automatically locate the CyberlabOD library and samples:



5. Click the finish button to complete the import operation.

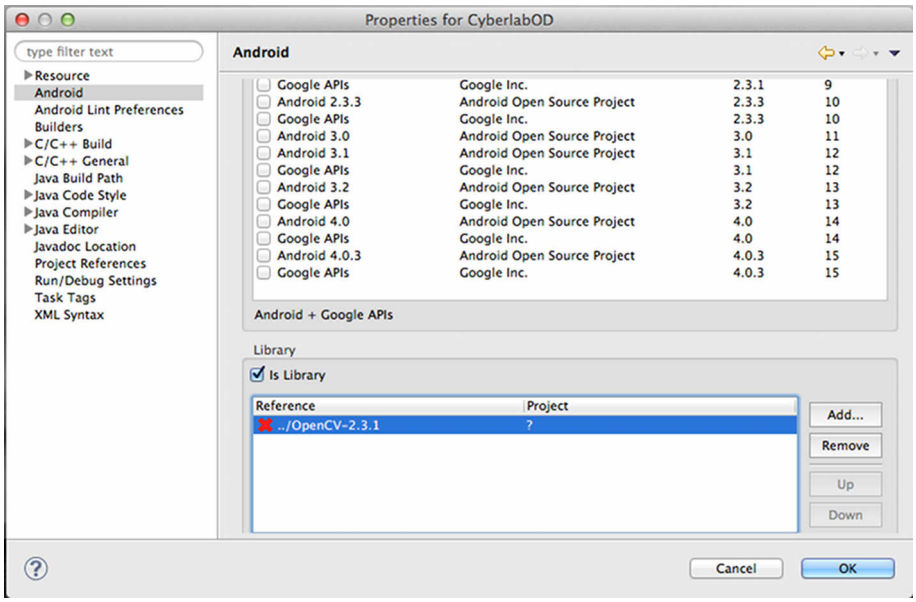
6. The projects will be imported with a bunch of errors:



7. The following steps must be performed to remove the Eclipse errors in the CyberlabOD framework project:

   - Fix the reference to OpenCV workspace project:
     (a) Right click the CyberlabOD project in the package explorer → select properties → select Android.

(b) Under library there will be a red cross indicating an error:



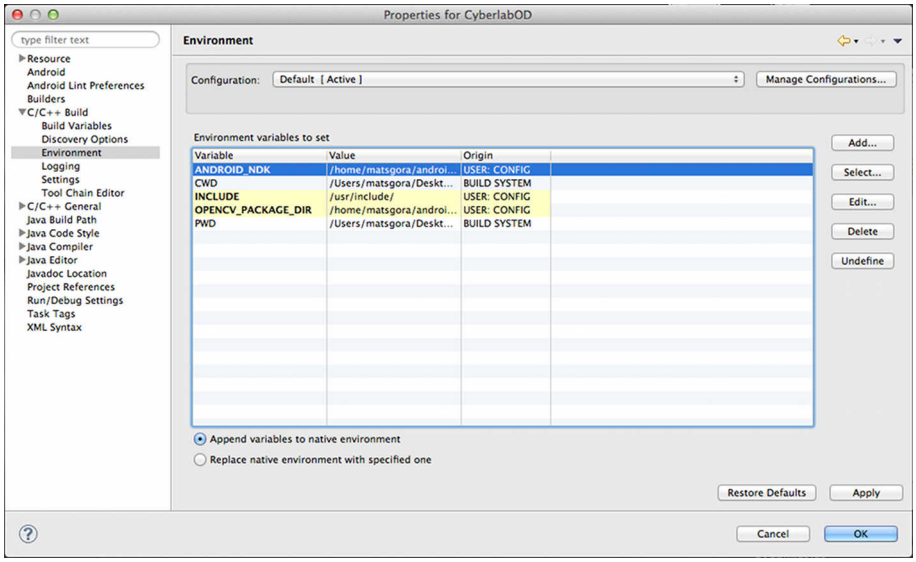
(c) Click remove under the library pane.

(d) Click add.

(e) Select OpenCV.

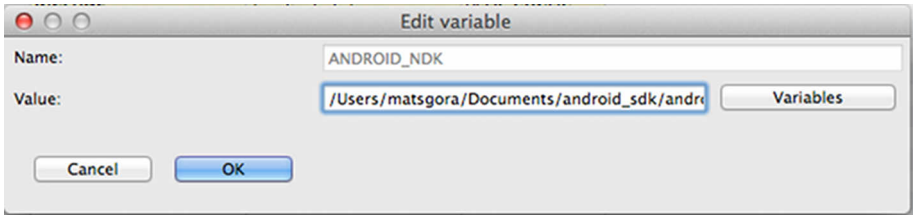
(f) Click OK.

- Fix references to Android NDK, the C/C++ libraries and the OpenCV package directory:

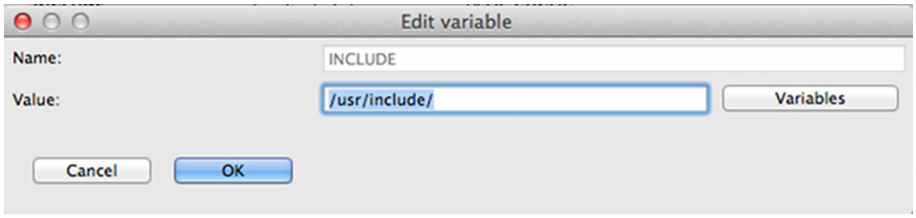


(a) In project properties select C/C++ build → Environment.

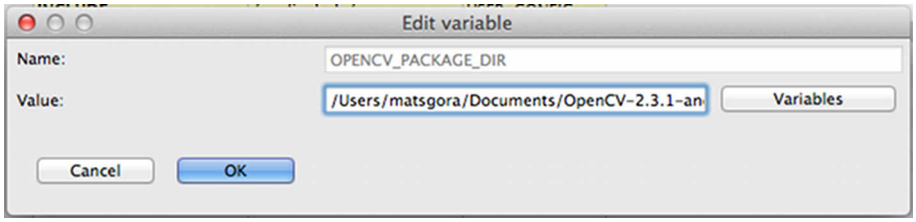
(b) Click on ANDROID_NDK and select edit. Enter the full path to the directory where you installed the Android NDK in your system:



(c) Click on INCLUDE and select edit. Enter the full path to the C/C++ header directory in the system. On UNIX systems this is normally
/usr/include:



(d) Click on OPENCV_PACKAGE_DIR and select edit. Enter the full path to the directory where you installed the OpenCV library:

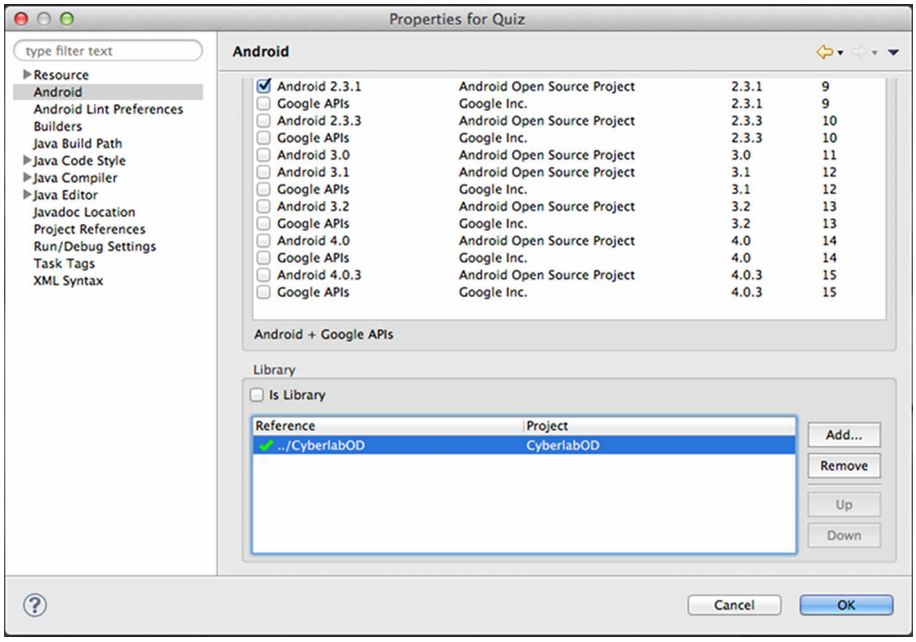


(e) Click on OK, closing the properties window.

- Right click project in package explorer. Select Android Tools → Fix project properties. This should remove invalid properties.
- Select the project. Select project from the Eclipse menu → Build All. The project should build without errors.
- If there are no errors, the framework is ready for use.

8. To remove the Eclipse errors in the example applications, the CyberlabOD library project reference must be updated.

- (a) Right click the example application project in the package explorer → select properties → select Android.
  (b) Under library there will be a red cross indicating an error:



  (c) Click remove under library pane.
  (d) Click add.
  (e) Select CyberlabOD.
  (f) Click OK twice to close the properties dialog.
- Right click the example application and select refresh.
- There should be no errors and the application should be ready to run.

E.2 External support

E.2.1 Detector

The CyberlabOD framework has support for external resources. To date this only includes one component, an image features extractor. The component is implemented as a C++ socket server that listens for clients wanting to use its services.

The socket support server is not ready for a production environment until the following features/issues has been added/resolved:

- Encrypt socket communication.
- Add log features.
- Recycle socket to allow for immediate server restart. Crash the server and restart. The socket server is not allowed to restart immediately. Ref. table F.3.
- Fix the image extractor bug. Ref. table F.3.
- Stress test system in a realistic environment.
- Define the maximum number of concurrent clients the socket server supports.
- Create make file scripts, removing the dependency to Eclipse.
- Add parameters to detector and operating system wrappers, allowing control of starting and stopping as well as parameters for socket timeout, server listen port numbers, maximum number of concurrent clients.

Prerequisites

The detector requires the following libraries and tools. For installation refer to the links and your specific platform:

1. **Posix system.** The detector currently doesn't support Windows sockets.
2. **Eclipse IDE** (`http://www.eclipse.org/downloads/`).
3. **Eclipse CDT** (`http://www.eclipse.org/cdt/`).
4. **OpenCV >= 2.3.1.** The system relies on OpenCV for extendability and cross platform support (`http://opencv.willowgarage.com/wiki/InstallGuide`).
5. Platform specific **C/C++ development tool chain.** (Refer to platform specific installation instructions. Can be found by using a search engine or system manuals).

6. **Boost C++ thread library.** Platform independent threading library. Should ease cross platform development. (`http://www.boost.org/`)

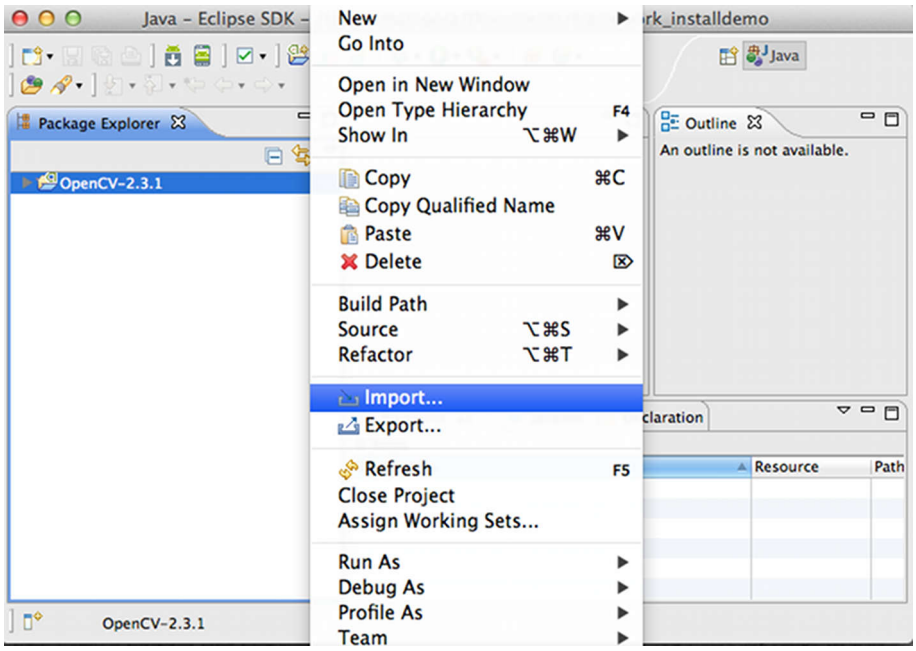
The author recommends Debian or a similar system that offers installation of the dependencies through a package management system.

Install Instructions

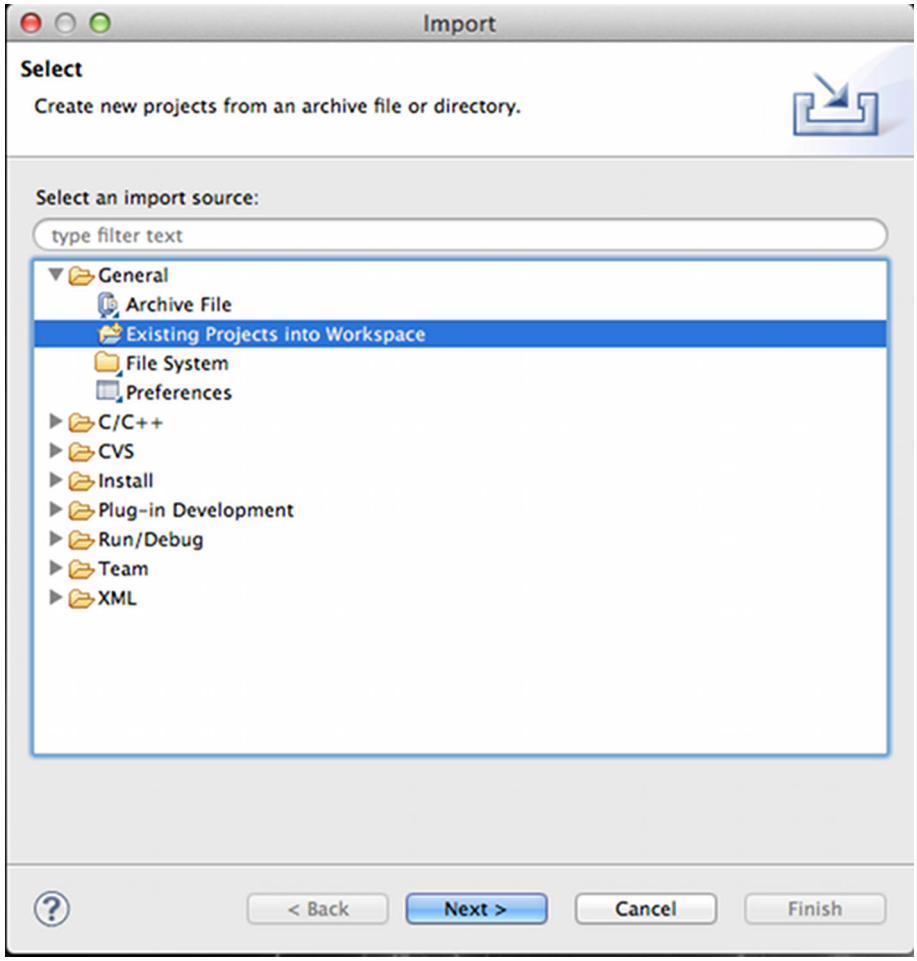
The detector is a regular C++ eclipse project.

Follow the steps to import the Detector into the workspace:

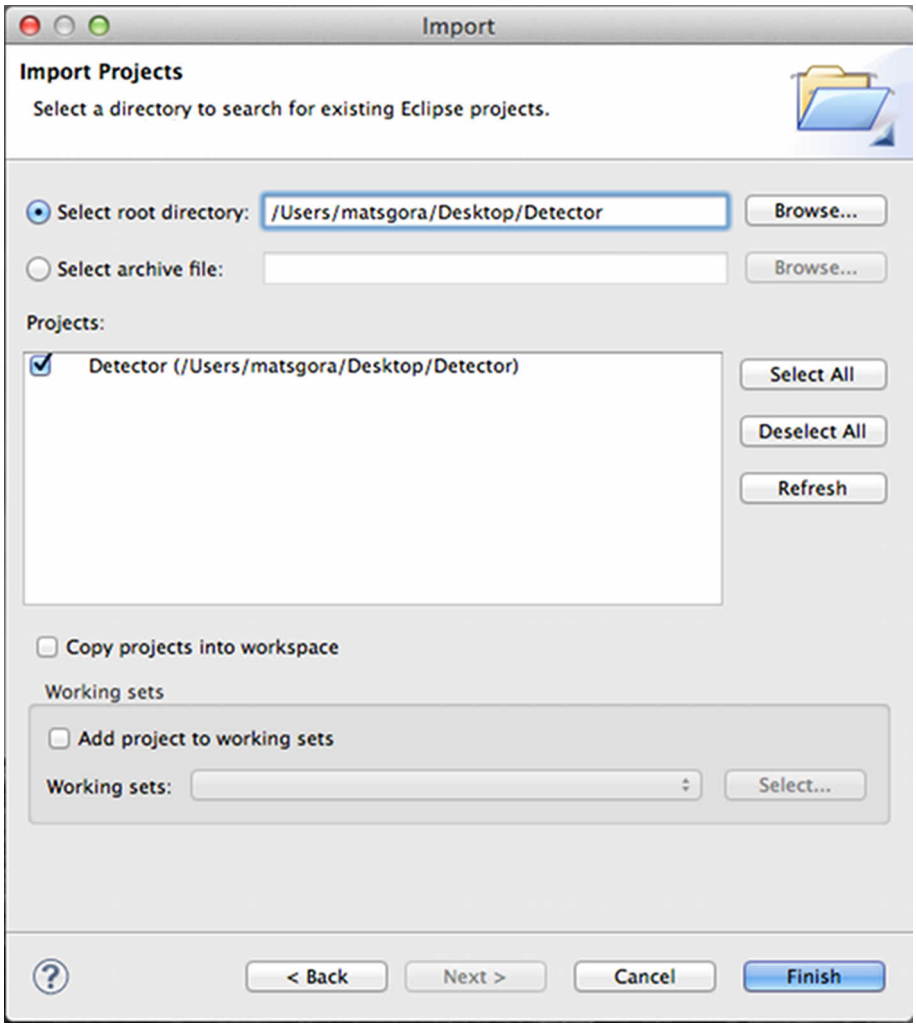
1. Unpack the zip file using your favourite archive utility.
2. Right click on the Package Explorer window and choose the import option from the context menu:



3. In the main panel select General → Existing Projects into Workspace and press the next button:

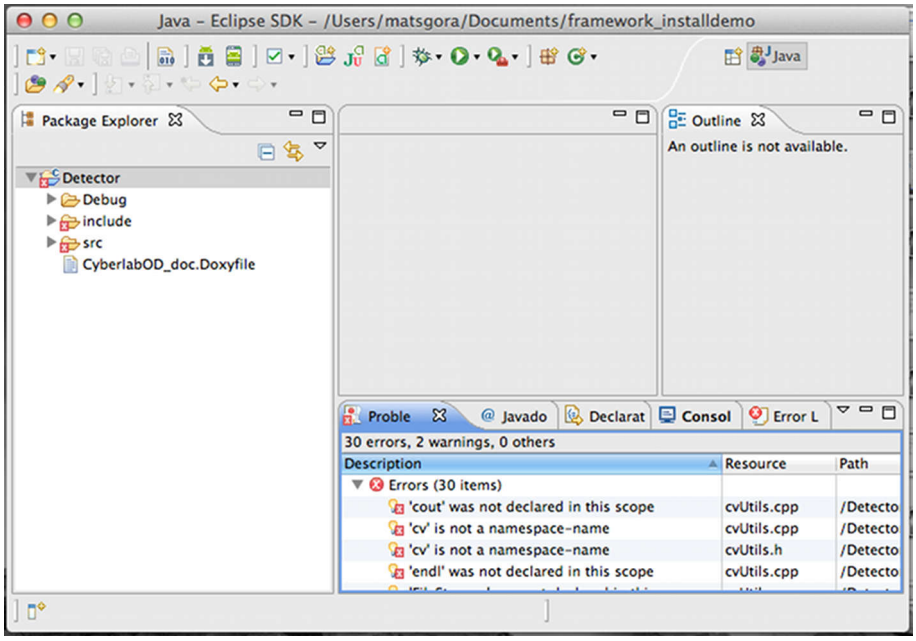


4. Locate your Detector package folder in select root directory. (If you have created a workspace in the package directory, then just click the browse button and instantly close the directory choosing dialog with the OK button!) Eclipse should automatically locate Detector project:



5. Click the finish button to complete the import operation.

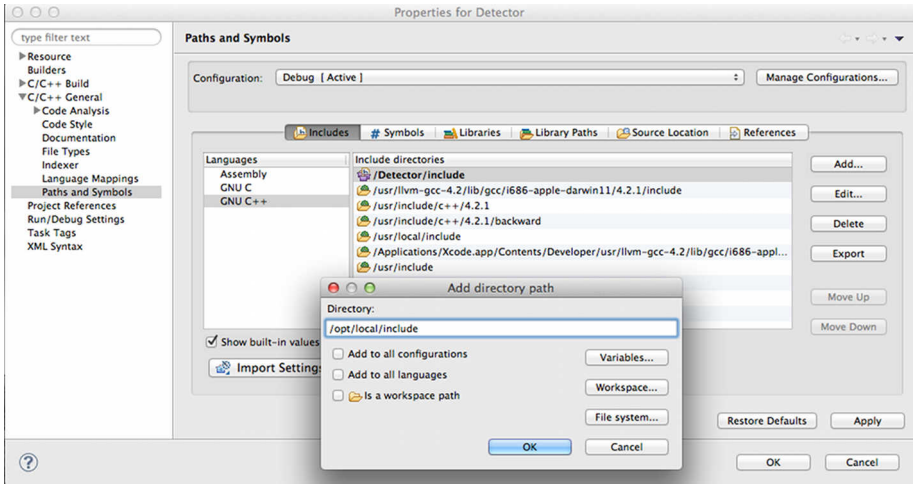
6. The project will be imported with a bunch of errors:



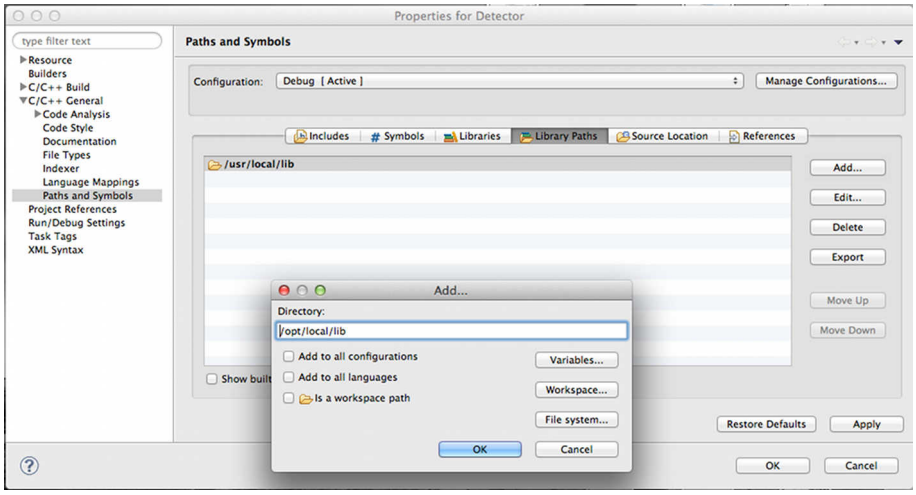
7. The following steps must be performed to remove the Eclipse errors in the Detector project,:

   (a) Right click the Detector project in the package explorer → select properties → select C/C++ General → Paths and Symbols.

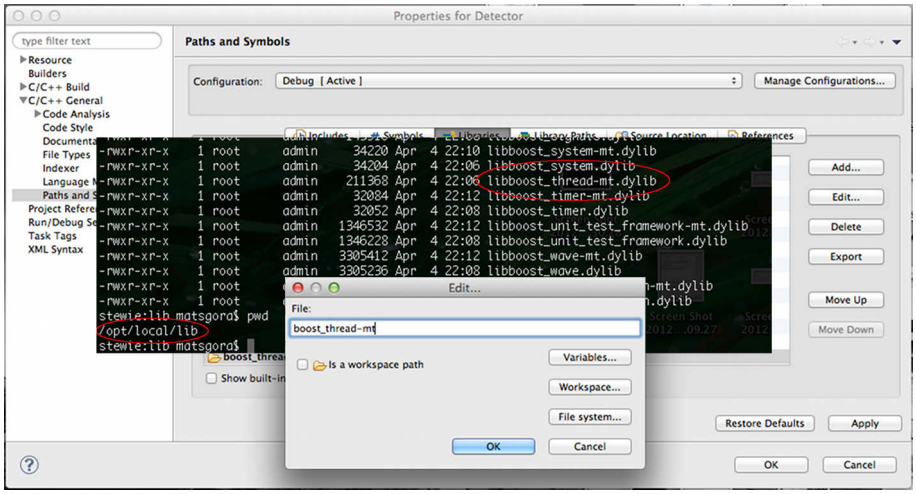
(b) Under the includes tab make sure that both GNU C and GNU C++ has references to the header directories for the STD library (on linux normally */usr/include*), OpenCV (on Linux normally in */usr/include*, but varies on the different platforms) and Boost thread (on Linux normally in */usr/include*, but varies on the different platforms).



(c) Select library paths. Make sure that this points to the STD libraries (on Linux normally in */usr/local/lib*), OpenCV libraries (on Linux normally in */usr/local/lib*, but varies on the different platforms) and Boost (on Linux normally in */usr/local/lib*, but varies on the different platforms).

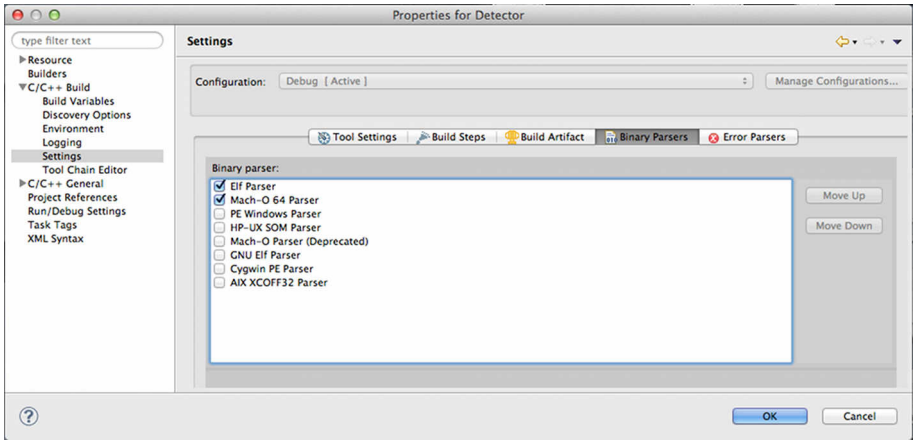


(d) Select libraries. Make sure that the libraries listed corresponds with the file names in the library folders in the previous step. **NB! These names can vary.** The naming convention here is: *opencv_core* points to the file *libopencv_core.dylib*. So if the name of a library file is *libboost_thread-mt.dylib*, the name in libraries should be *boost_thread-mt*.



(e) For Mac OS X systems:

i. Select C/C++ Build → Settings → Binary Parsers:



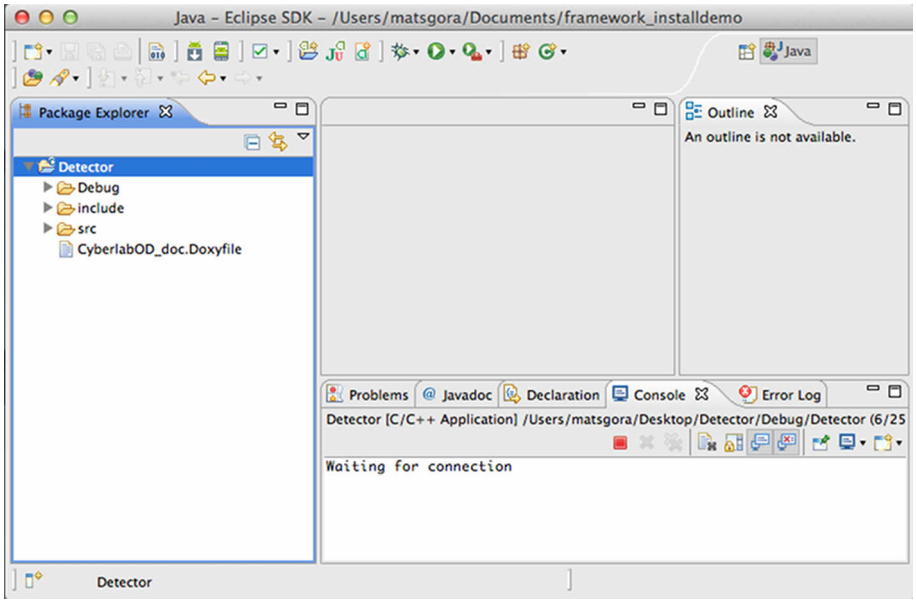
ii. Tick Mach-O 64 Parser. This enables generation of the binaries.

(f) Click the OK button in the properties dialog box, exiting back to Eclipse.

(g) Select Project → Clean from the menu.

(h) Select Project → Build All from the menu.

(i) If the project builds without errors you are done and can run the project as a Local C\C++ Application.



E.2.2 HTTP database synchronizator

The example applications implement support data synchronization by uploading and downloading their SQLite3 database to a server using HTTP. By running their own local instance of the database, the applications are capable of running in isolated environments. The server is by the author considered outside the scope of this thesis, but will elaborate on a minimal solution for providing the server synchronization support.

Prerequisites

- **HTTP server** (i.e. Apache) with mod_rewrite (allow password protection) and PHP support.
- **PHP file upload script.**

Configuration

- In the public accessible folder add a file called .htaccess with the following code:

```
1 AuthUserFile <path to private folder>/.htpasswd
2 AuthType Basic
3 AuthName "<Custom text>"
4 Require valid-user
```

- In the private folder create .htpasswd using the htpasswd command. This encrypts the password.

- Create a PHP file upload script. Here is the script used by the author:

```
1  <?php
2      $target_path  = "<path to files folder>";
3      $target_path = $target_path . basename( $_FILES[
           'uploadedfile']['name']);
4      if(move_uploaded_file($_FILES['uploadedfile']['
           tmp_name'], $target_path)) {
5          echo "The file ". basename( $_FILES['
               uploadedfile']['name']).
6          " has been uploaded";
7      } else{
8      echo "There was an error uploading the file , 
           please try again!";
9      //echo "Here is some more debugging info:";
10     //print_r($_FILES);
11     //print_r($_SERVER);
12  }
13 ?>
```

Appendix F

Framework Bugs

This chapter collects the known bugs in the framework, the example application and the external socket detector.

F.1 CyberlabOD

Bug description	Why is it not resolved?	Complexity
The "Dynamic" feature detectors are only used by the Android version of OpenCV and doesn't exist on other platforms. This feature is not disabled in the *SocketLocalFeaturesExtractor* advanced constructor and trying to use a "Dynamic" feature detector crashes the socket server. The constructor should reject attempts to use these by either throwing an exception, falling back to the non dynamic type of the feature detector or convert the "Dynamic" requests in the same manner as the Android OpenCV library.	Time constraints	Easy

Table F.1: The known bugs in the core framework, the reason for them not being fixed and the authors assessment on their repair difficulty (*easy*, *medium* and *hard*)

F.2 Framework examples

Bug description	Why is it not resolved?	Complexity
When camera intents return, dependent upon the device, the application activity is restated. The operating system stop activities in the background to free available memory. (Ref activity life cycle in official Android documentation). This causes the applications to loose track of its state resulting in behavior like restarting the camera or not displaying the image returned. The bug appears frequently on HTD Desire HD devices and it is consequent on Samsung Galaxy SII devices. The author recommends using the latter. Possible solutions: Implement custom camera activity, create singleton to store the activity state or find where to store the state in the activities using the provided methods in the Android framework.	Time constraints The bug doesn't appear on the device available to the author or on the Android emulator, resulting in time consuming and difficult debugging.	Medium to Hard (Debug device dependent)

Table F.2: The known example bugs, the reason for them not being fixed and the authors assessment on their repair difficulty (*easy*, *medium* and *hard*)

F.3 External Socket Detector

Bug description	Why is it not resolved?	Complexity
If the server crashes it cannot be immediately restated. This can be fixed by recycling the socket when this happens. The Detector documentation mentions this problem and indicates where in the source code the fix should be implemented.	Time Constraints	Easy
The "Dynamic" feature detectors are only used by the Android version of OpenCV and doesn't exist on other platforms. This feature is not disabled in the *CvUtils* advanced constructor and trying to use a "Dynamic" feature detector crashes the server. The constructor should reject attempts to use these by either throwing an exception or falling back to the non dynamic type of the feature detector.	Time constraints	Easy

Table F.3: The known external detector bugs, the reason for them not being fixed and the authors assessment on their repair difficulty (*easy*, *medium* and *hard*)

Appendix G

Quiz Recognition Images

The images listed in this Appendix chapter can be used to test the Quiz application.

G.1 Category Initialization

Figure G.1: Science category

Figure G.2: Science category

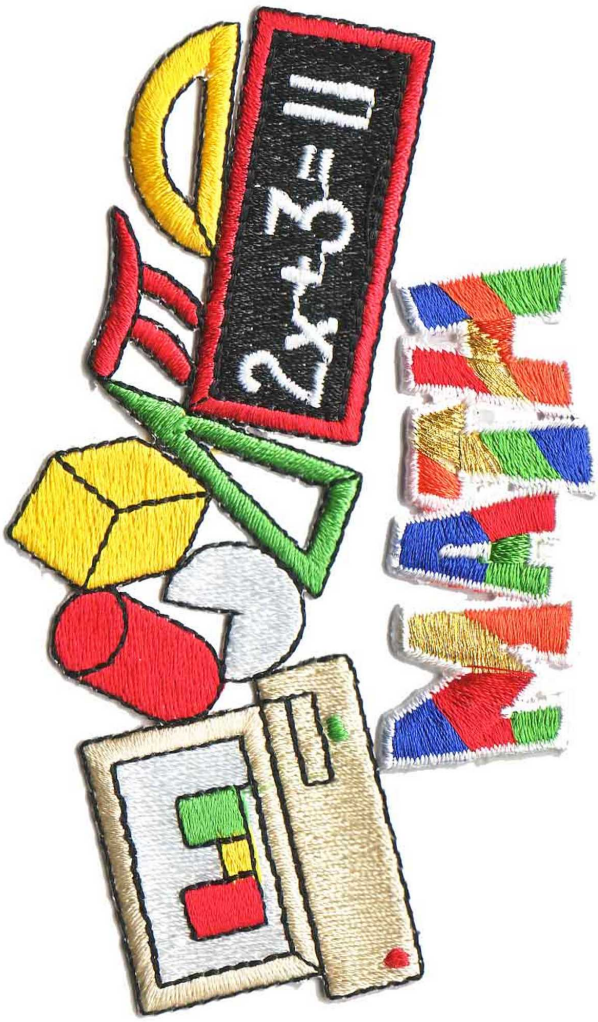


Figure G.3: Math category

Figure G.4: Geography category

Figure G.5: History category

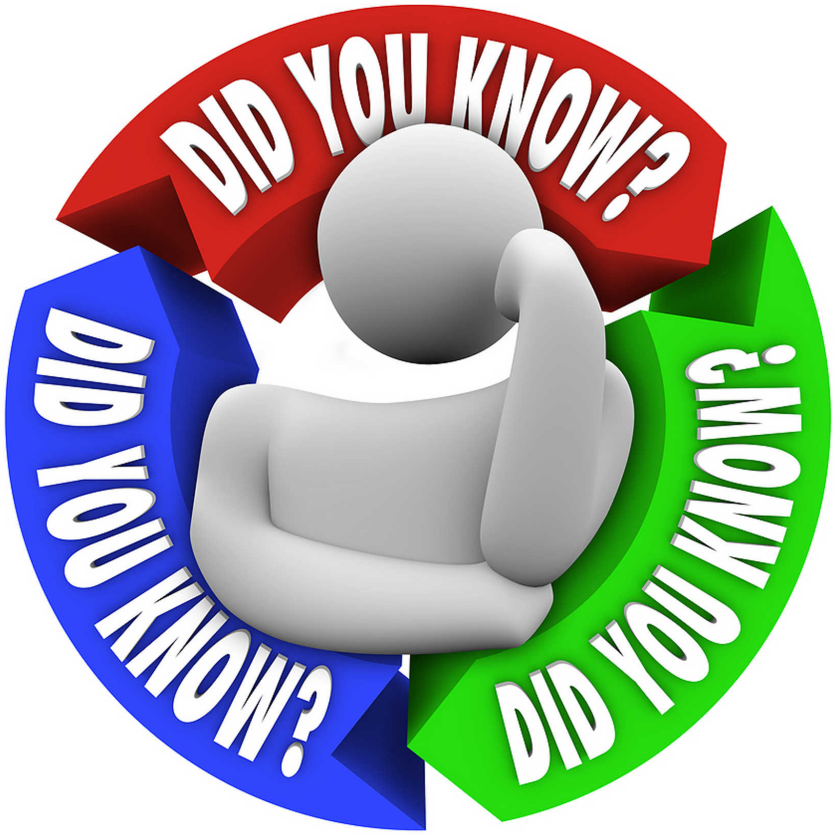


Figure G.6: Trivia category

G.2 Recognition Question Answers

Figure G.7: Science answer

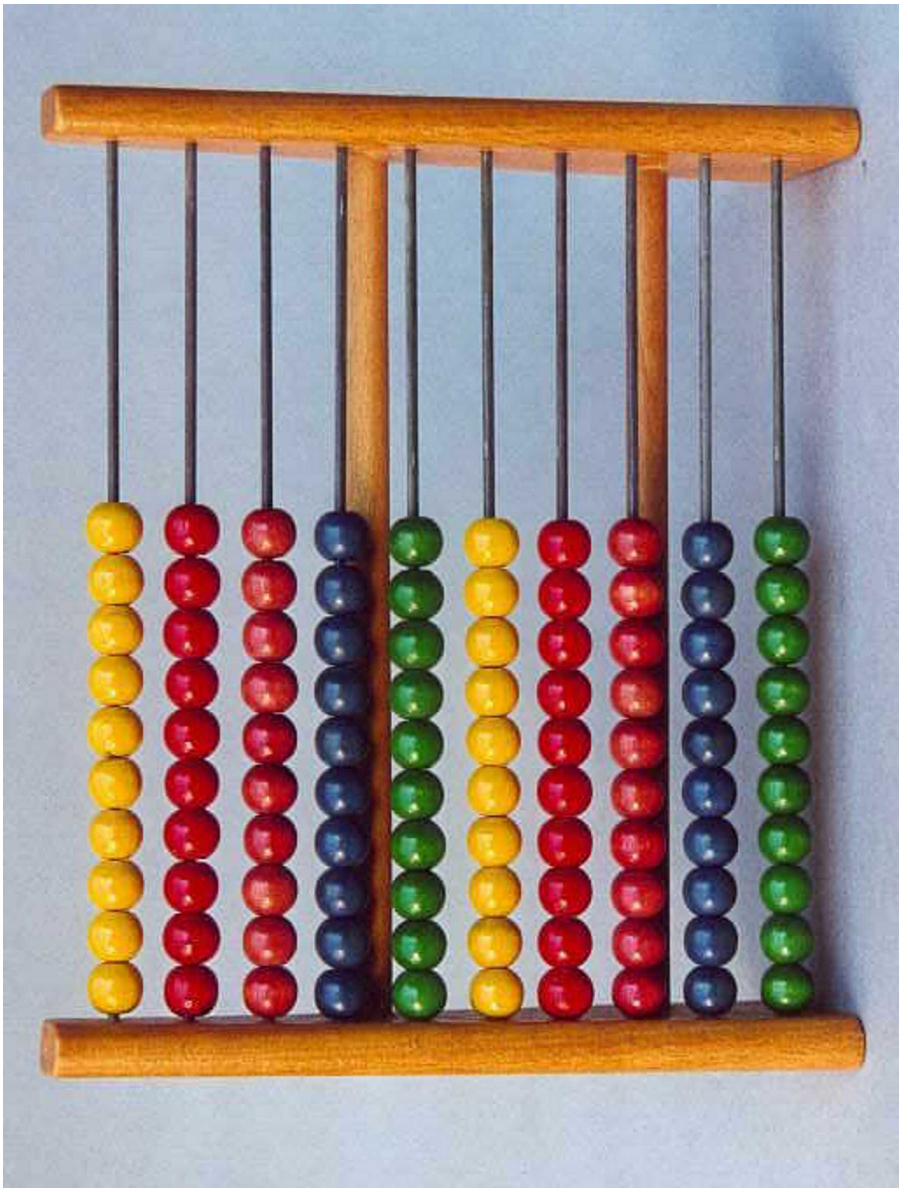


Figure G.8: Math answer

Figure G.9: Geography answer

G.2.1 Images Of Real World Objects

For convenience here are images taken of real world objects used for answering recognition questions.

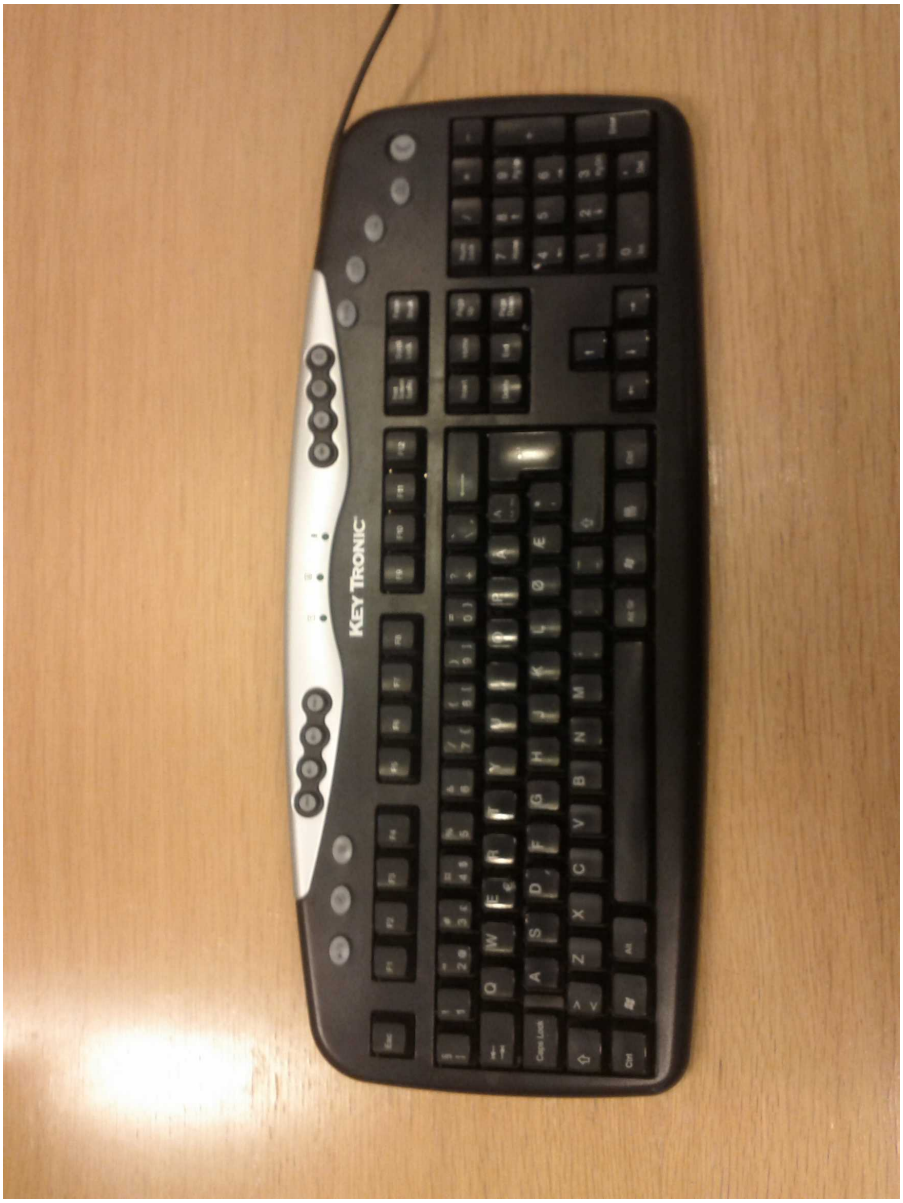


Figure G.10: History answer

Figure G.11: Trivia answer