  
HTL - Perg

Höhere Abteilung für Informatik

Diplomarbeit

Mushroom Identifier

Projektteam: Jakob Froschauer

Hakan Abbas

Markus Arbeithuber

Projektbetreuer: Prof. Dipl.-Ing. Christian Aberger

Bearbeitungszeitraum: 01.10.2016 – 30.04.2017

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Unterschrift \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hakan Abbas

Perg, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Unterschrift \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Markus Arbeithuber

Perg, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Unterschrift \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jakob Froschauer

Danksagung

An dieser Stelle möchten wir uns bei allen Personen bedanken, die uns bei der Entstehung der Diplomarbeit unterstützen und uns zur Seite standen.

Besonderer Dank gilt unserem Diplomarbeitsbetreuer Prof. Dipl.-Ing. Christian Aberger, der uns die gesamte Projektdauer für technische und organisatorische Fragen zur Verfügung stand.

Herzlichen Dank!

Inhalt

[2 Impressum 9](#_Toc478021621)

[2.1 Schule 9](#_Toc478021622)

[2.2 Schuljahr 9](#_Toc478021623)

[2.3 Klasse 9](#_Toc478021624)

[2.4 Projektname 9](#_Toc478021625)

[2.5 Projektleiter 9](#_Toc478021626)

[2.6 Projektteam 9](#_Toc478021627)

[2.7 Betreuungslehrer 9](#_Toc478021628)

[3 Einleitung 10](#_Toc478021629)

[3.1 Kurzfassung 10](#_Toc478021630)

[3.2 Abstract 11](#_Toc478021631)

[3.3 Motivation 12](#_Toc478021632)

[4 Projektdefinition 12](#_Toc478021633)

[4.1 Aufgabenstellung 12](#_Toc478021634)

[4.2 Geschäftsziele 12](#_Toc478021635)

[4.3 Projektziele 13](#_Toc478021636)

[4.4 Projektumfang 13](#_Toc478021637)

[4.5 Projektstrukturplan 14](#_Toc478021638)

[4.6 Meilensteine 15](#_Toc478021639)

[4.7 Schule 16](#_Toc478021640)

[4.8 Betreuungslehrer 16](#_Toc478021641)

[4.9 Team 17](#_Toc478021642)

[4.9.1 Jakob Froschauer 17](#_Toc478021643)

[4.9.2 Hakan Abbas 18](#_Toc478021644)

[4.9.3 Markus Arbeithuber 18](#_Toc478021645)

[4.10 Organisation 19](#_Toc478021646)

[5 Entstehung und Planung 19](#_Toc478021647)

[5.1 Ideenfindung 19](#_Toc478021648)

[5.2 Zeitplanung 20](#_Toc478021649)

[5.3 IVM – Matrix 21](#_Toc478021650)

[6 Realisierung 22](#_Toc478021651)

[6.1 Anwendungsfälle 22](#_Toc478021652)

[6.2 Funktionen 23](#_Toc478021653)

[6.2.1 Pilzanalyse 23](#_Toc478021654)

[6.2.2 Pilzauflistung 23](#_Toc478021655)

[7 Darstellung 23](#_Toc478021656)

[7.1 iOS 23](#_Toc478021657)

[7.1.1 Pilzanalyse 23](#_Toc478021658)

[7.1.2 Pilzliste 24](#_Toc478021659)

[7.1 Android 25](#_Toc478021660)

[7.1.1 Startseite 25](#_Toc478021661)

[7.1.2 Foto schießen 25](#_Toc478021662)

[7.1.3 Galerie 27](#_Toc478021663)

[7.1.4 Pilzliste 28](#_Toc478021664)

[8 Technologien 29](#_Toc478021665)

[8.1 Java (Android) 29](#_Toc478021666)

[8.2 Swift/Objective-C (IOS) 29](#_Toc478021667)

[8.3 C++ (OPEN CV) 29](#_Toc478021668)

[8.4 Android Studio 29](#_Toc478021669)

[8.5 Android Studio NDK 30](#_Toc478021670)

[8.6 CMake 30](#_Toc478021671)

[8.7 Gradle 31](#_Toc478021672)

[8.8 Boost ( C++ Libraries ) 31](#_Toc478021673)

[8.9 XCode (IOS) 32](#_Toc478021674)

[8.10 Visual Studio (OPEN CV) 32](#_Toc478021675)

[8.11 OpenCV (Open Source Computer Vision) 32](#_Toc478021676)

[8.12 CMARKUP (XML Lesen) 33](#_Toc478021677)

[8.13 Computer Vision 33](#_Toc478021678)

[8.14 Mensch-Computer-Interaction 33](#_Toc478021679)

[8.15 Haar Cascade Training 33](#_Toc478021680)

[8.16 Entscheidung für Native Apps 34](#_Toc478021681)

[9 Programmierung 35](#_Toc478021682)

[9.1 Bilderkennung 35](#_Toc478021683)

[9.1.1 Kurzerklärung der Schritte 35](#_Toc478021684)

[9.1.2 Computer Vision 36](#_Toc478021685)

[9.1.3 CV\_HOUGH\_GRADIENT: 48](#_Toc478021686)

[Datenspeicherung (XML) 51](#_Toc478021687)

[9.1.4 Maschinelles Lernen 54](#_Toc478021688)

[9.1.5 Tätigkeiten bis zum Endbenutzerprodukt 58](#_Toc478021689)

[9.2 iOS – App 58](#_Toc478021690)

[9.2.1 OpenCVWrapper.h 59](#_Toc478021691)

[9.2.2 OpenCVWrapper.mm 60](#_Toc478021692)

[9.2.3 ViewController 63](#_Toc478021693)

[9.3 Android – App 64](#_Toc478021694)

[9.3.1 Installation 64](#_Toc478021695)

[9.3.2 Übersetzung von C++ Dateien in Android Studio 66](#_Toc478021696)

[9.3.3 Design Implementierung 70](#_Toc478021697)

[10 Qualitätssicherung 78](#_Toc478021698)

[10.1 Qualitätsmerkmale 79](#_Toc478021699)

[10.2 Testen 79](#_Toc478021700)

[80](#_Toc478021701)

[80](#_Toc478021702)

[10.3 Vergleich mit Konkurrenzprodukten 81](#_Toc478021703)

[10.3.1 Meine Pilze (Pilzbestimmung) Entwickler: Meine Pilze 81](#_Toc478021704)

[10.3.2 Pilze Entwickler: Kirill Sidorov 81](#_Toc478021705)

[10.3.3 Pilzführer Nature Lexicon 82](#_Toc478021706)

[10.3.4 Fazit: 82](#_Toc478021707)

[11 Zusammenfassung 82](#_Toc478021708)

[11.1 Ergebnis 82](#_Toc478021709)

[11.2 Resümee 82](#_Toc478021710)

[11.3 Aufwandsverteilung 82](#_Toc478021711)

[12 Literatur und Quellen Verzeichnis 82](#_Toc478021712)

[12.1 Abbildungsverzeichnis 83](#_Toc478021713)

[12.2 Tabellenverzeichnis 83](#_Toc478021714)

[12.3 Literaturverzeichnis 83](#_Toc478021715)

[12.4 Abkürzungsverzeichnis 83](#_Toc478021716)

[13 Im Anhang 83](#_Toc478021717)

[14 Literaturverzeichnis 2 86](#_Toc478021718)

# Impressum

## Schule

HTBLA Perg für Informatik

Marchlandstraße 48

4320 Perg

## Schuljahr

2016/2017

## Klasse

5AHIF

## Projektname

Mushroom Identifier

## Projektleiter

Jakob Froschauer

## Projektteam

Hakan Abbas

Markus Arbeithuber

## Betreuungslehrer

Dipl.-Ing. Christian Aberger

# Einleitung

## Kurzfassung

Die Diplomarbeit Mushroom Identifier ist während des fünften Jahrgangs von Hakan Abbas, Markus Arbeithuber und Jakob Froschauer im Zuge der Reife- und Diplomprüfung an der Technischen Bundeslehranstalt Perg erstellt worden.

Die mobile Anwendung soll die Zukunft des Pilz Lexikons darstellen. Bei der Suche im Wald begegnet man oft Pilzen, die man nicht ohne schwere Pilz Lexika erkennen kann. Auch mit diesen Büchern ist es schwierig in vertretbarer Zeit den gesuchten Pilz zu identifizieren. Mit der App Mushroom Identifier soll dieses Problem der Vergangenheit angehören. Zur Funktionsweise: Der Benutzer wird dazu aufgerufen, ein Pilzfoto aus der Vogelperspektive auszuwählen oder ein neues zu schießen. Daraufhin wird unter anderem die Farbe und die Form des Pilzes erkannt. Auch auffällige Eigenschaften (zum Beispiel viele kleine Punkte in einem großen roten Kreis => Fliegenpilz) werden automatisch erkannt.

Die dazu nötigen Vergleichsdaten werden lokal mit der App mitgeliefert um das Problem des schlechten Internet Empfangs im Wald zu umgehen.

Mit jeder erkannten Eigenschaft verringert sich die Zahl der in Frage kommenden Pilze. Wenn am Ende der Bilderkennung noch kein Pilz feststeht werden die Unterschiede der noch in Frage kommenden Pilze durch Ja/Nein Benutzerfragen abgefragt. Darüber hinaus wird auch durch Maschinelles Lernen festgestellt, ob es sich überhaupt um einen Pilz handeln kann.

## Abstract

Mushroom Identifier is an App, with which you can photograph a mushroom with the smartphone and this app automatically identifies your unknown Mushroom. Furthermore, the app is providing the most important information hosted by Wikipedia. The used Techniques are image recognition, also known as computer vision as well as machine learning. This is a modern technique, where a computer detects the differences of hundreds of positive and negative pictures.

## Motivation

Das gesamte Projektteam sind begeisterte Pilzsammler, jedoch trauten wir uns nur bei dem Eierschwammerl auch zuzugreifen, da der Identifikationsprozess von Pilzen bisher ausgesprochen aufwendig ist. Schwere Pilzlexika mussten mitgeschleppt werden. Dieser aufwendige Prozess soll vereinfacht werden. Ein einfaches Foto mit dem Smartphone und eventuell ein paar JA/NEIN Fragen zu beantworten soll reichen um Pilze eindeutig zu identifizieren.

Darüber hinaus wird durch die modernen Technologien ein jüngeres Publikum angesprochen, das sich am Pilzsammlern begeistern kann und somit die Vorzüge

# Projektdefinition

## Aufgabenstellung

## Geschäftsziele

Ziel dieser Diplomarbeit ist nicht finanziellen Erfolg zu erreichen. Nach Abschluss der Diplomarbeit kann das Projekt:

* an Zukünftige Diplomarbeiten übergeben werden
* an bestehende Pilz-Apps (siehe Punkt „Vergleich mit anderen Pilz-Apps) als zusätzliche Funktion übergeben werden.
* vom Team für zukünftige Projekte verwendet werden.

## Projektziele

Es soll eine mobile App entwickelt werden, die Pilzsammler dabei hilft, Pilze zu identifizieren. Es sollen Fotos von dem für den Benutzer unbekannten Pilzen aufgenommen werden. Diese Fotos werden anschließend mit Methoden der Bilderkennung (Computer Vision) sowie maschinellem Lernen erkannt und klassifiziert. Darüber hinaus werden bei Pilzen, die nicht vollständig anhand eines Fotos erkannt werden können, Fragen an den Benutzer gestellt um in einer Art Entscheidungsbaum zu einem Ergebnis zu kommen.

Dabei soll es eine Codebasis in C++ geben, auf die sowohl eine nativ Android App als auch eine native IOS App zugreift. (siehe 7.16 Entscheidung für Native Apps)

## Projektumfang

Das Ziel von Mushroom Identifier ist es, zu zeigen, wie ein Pilzerkennungsprozess technisch umzusetzen ist. Dabei wird der Umgang mit verschiedensten Methoden der Bilderkennung (Computer Vision) sowie Maschinellem Lernen beschrieben und umgesetzt. Diese App muss sowohl auf iOS, als auch auf Android laufen. Der Bilderkennungsteil soll in C++ umgesetzt werden, worauf sowohl der iOS sowie der Android Teil zugreifen. Wenn Änderungen am Bilderkennungsalgorithmus durchgeführt werden müssen sind die Änderungen gleichzeitig auf allen Plattformen die darauf zugreifen. Was nach dieser Diplomarbeit noch umzusetzen wäre um ein Endbenutzerprodukt zu erstellen, wird in dem Punkt <BITTE PUKT EINFÜGEN> beschrieben.

## Projektstrukturplan

## Meilensteine

|  |  |
| --- | --- |
| Meilenstein | Datum |
| Projektstart erfolgt | 12.06.2016 |
| Projektinitialisierung abgeschlossen | 30.10.2016 |
| Pilz fotografieren ist möglich | 13.11.2016 |
| Pilzfarbe erkennen ist möglich | 13.11.2016 |
| Daten können von XML File abgerufen werden | 18.12.2016 |
| Form erkennen ist möglich | 15.01.2017 15.01.2017 |
| Bilderkennung auch auf mobilen Geräten möglich | 22.01.2017 |
| Informationen zum Pilz können vom Benutzer abgefragt werden + Ergebnis wird angezeigt | 26.02.2017 |
| Diplomarbeit fertiggestellt | 26.03.2017 |

## Schule

Das Projekt wurde im Rahmen der Abschlussarbeit (Diplomarbeit) für die HTL Perg erstellt.

Kontakt:

**HTBLA Perg**  
Machlandstraße 48  
4320 Perg  
  
Tel. 0 72 62 / 539 26  
Fax. 0 72 62 / 539 26 - 6  
  
Schulkennzahl: 411457

Abbildung : HTL - Perg



Abbildung : Porträt Christian Aberger

## Betreuungslehrer

Als Programmierprofessor seit der vierten Klasse steht uns Herr Professor Dipl.-Ing. Aberger auch als Betreuungslehrer zur Seite.

Kontakt

Dipl.-Ing. Christian Aberger

Softwarepark 37

4232 Hagenberg im Mühlkreis

Tel. [07236 33514200](javascript:void(0))

## Team

### Jakob Froschauer

Lebenslauf

|  |  |  |  |
| --- | --- | --- | --- |
| **Persönliche Daten** | |  | |
| Geburtsdatum | | 24. April 1998 | |
| Staatsbürgerschaft: | | Österreich | |
| Religionsbekenntnis: | | röm.-kath. | |
| Eltern | | Gottfried Froschauer, Bankangestellter  Helga Froschauer, Lehrerin | |
| Geschwister: | | Sara, Journalistin  David, Student, Personalvermittler  Franziska, Studentin | |
| **Schulbildung** | |  | |
| 4 Jahre | | Volksschule in Naarn | |
| 4 Jahre | | Hauptschule in Naarn | |
| seit 2012 | | HTL für Informatik in Perg | |
| **Kenntnisse und Fähigkeiten** | |  | |
| Sprachen: | Englisch (B2) | |
|  |  | |
| EDV-Kenntnisse: | Programmieren in Java, C#, C, C++, Typescript, Swift und Assembler | |
| Sonstiges: | 16-stündiger Erste Hilfe Kurs  Führerschein (Klasse B)  Mitglied des Musikvereins in Naarn (Trompete)  Mitarbeiter der Bücherei Naarn  Theatergruppe Naarn | |

### Hakan Abbas

### Markus Arbeithuber

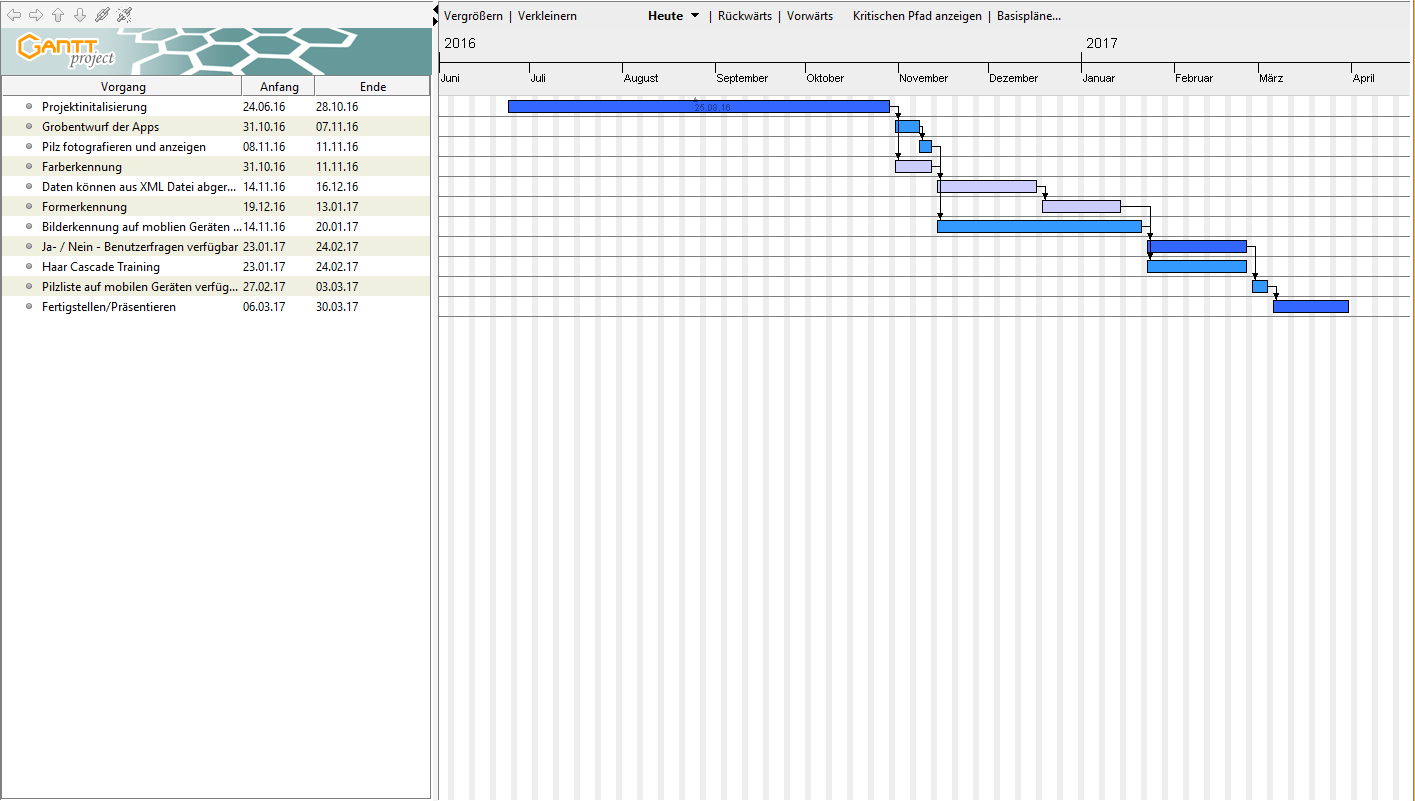
## Organisation

# Entstehung und Planung

## Ideenfindung

Die Idee zu Mushroom Identifier geht auf ein Brainstorming im Projektteam am Ende der 4. Klasse zurück. Dabei wurde sich schnell auf das Thema Bilderkennung geeinigt. Der weitere Entscheidungsprozess stellte sich als aufwendiger heraus, bis wir auf das Thema Blättererkennung kamen. Dieses Stichwort hat unser jetziger Diplomarbeitsbetreuer aufgeschnappt und den Einwand geliefert, dass ihn eine „Schwammerlerkennungsapp“ wesentlich mehr interessieren würde. Diese Idee begeisterte sowohl das Projektteam als auch den Herrn Direktor Reisinger. Das Diplomarbeitsthema war gefunden.

## Zeitplanung



## IVM – Matrix

Die IVM Matrix ermöglicht einen Überblick über verschiedene Teile des Projektes. Dabei wird zwischen Verantwortlichen (V), Mitwirkenden (M) und der Informierten (I) der jeweiligen Teilbereiche unterschieden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Hakan Abbas | Markus Arbeithuber | Jakob Froschauer | Christian Aberger |
| OpenCV/C++ | I | I | V | I |
| Android App | V | I | I | I |
| IOS App | I | V | I | I |
| Maschnielles Lernen | I | V | I | I |
| Dokumentation | M | M | V | I |

# Realisierung

## Anwendungsfälle



## Funktionen

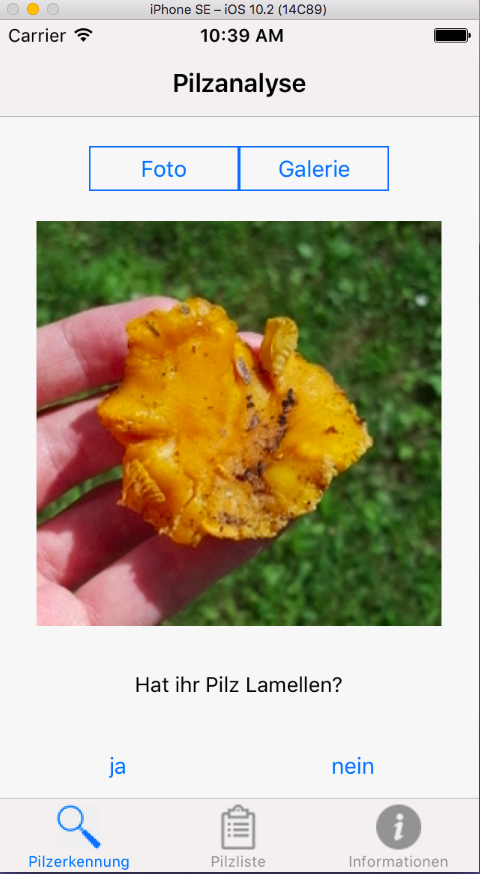
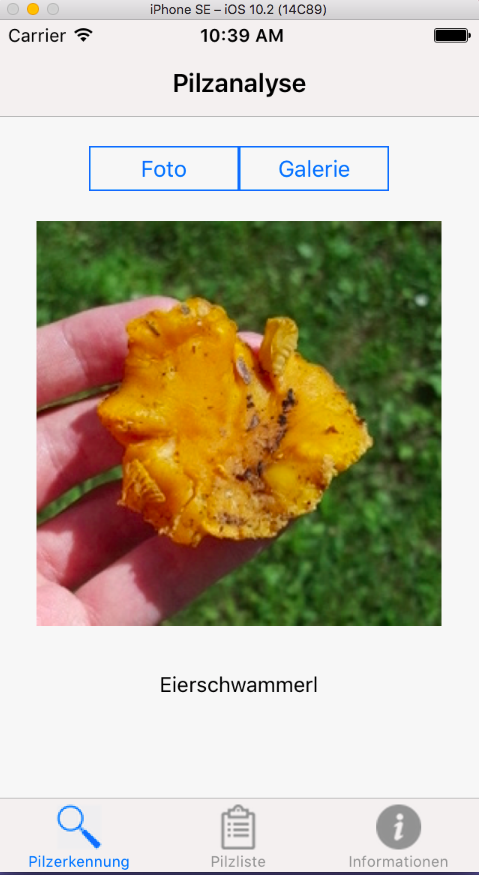
### Pilzanalyse

### Pilzauflistung

# Darstellung

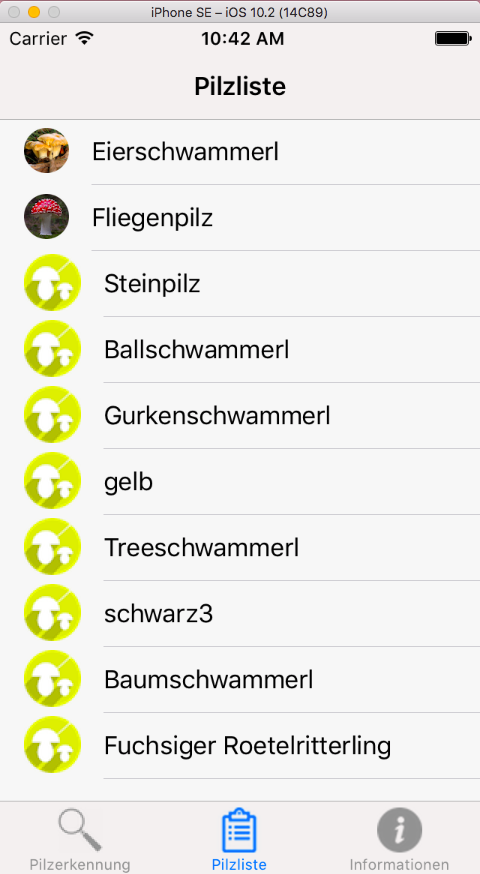
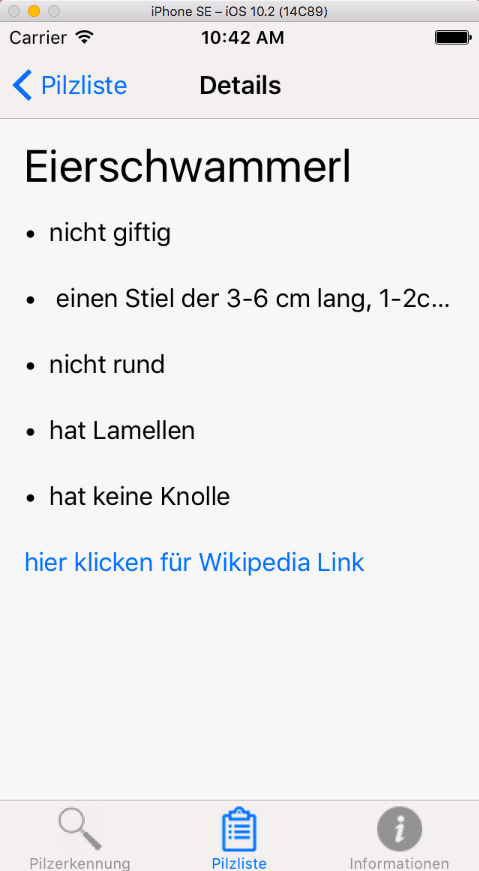
## iOS

### Pilzanalyse

Macht man ein Foto von einem Pilz oder wählt einen aus der Galerie aus, kann es sein, dass der Pilz sofort anhand der Farbe, Form und der XML – Datei vom Haar Cascade Algorithmus erkannt werden kann, oder gar kein Pilz in Frage kommt. Falls mehrere Pilze in Frage kommen, werden dem Benutzer noch zusätzlich Fragen gestellt, die er mit ja oder nein beantworten muss.

### Pilzliste

Navigiert man über die Tab Bar zur Pilzliste, wird eine Liste aller Pilze, die durch die App analysiert werden können, angezeigt. Klickt man auf einen Pilz, werden Detailinformationen dazu angezeigt.

## Android

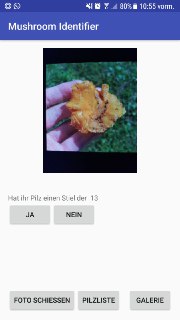
### Startseite



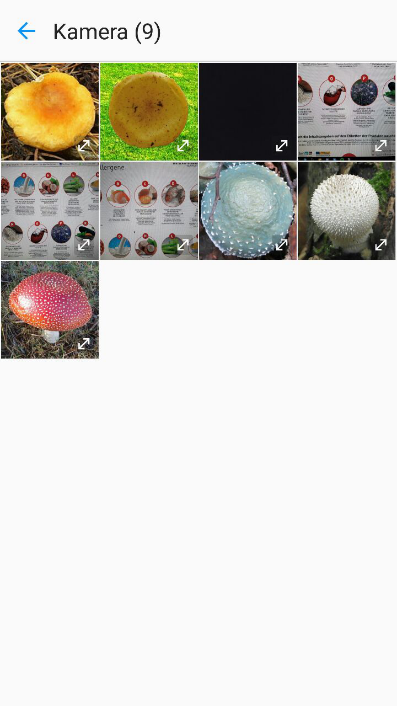
Wenn man die Android App startet, sieht man diese Startseite mit den zwei Buttons: Foto schießen und Galerie.

### Foto schießen

Drückt man nun auf Foto schießen, wird die Kamera des Smartphones geöffnet, und es wird erwartet ein Bild zu machen. Falls man ein gewünschtes Foto hat, wird durch Android für die Standardkamera implementierte Funktion gefragt, ob das Abbild entworfen, gespeichert oder ein neues Abbild geschossen werden soll. Falls die Abbildung gespeichert wird – das „Ja“ Symbol rechts unten – wird man wieder auf die Startseite weitergeleitet und es ist folgendes zu sehen:

Das Bild wurde erfolgreich gespeichert und wird auf der Startseite angezeigt, wenn man will, ist es auch möglich ein neues Bild zu machen.

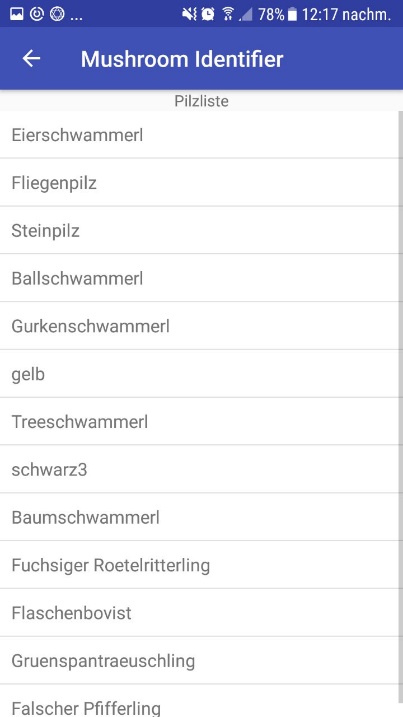
### Galerie



Falls jetzt aber schon ein bestehendes Exemplar analysiert werden soll, bietet die Galerie Funktion aus der Galerie ein Bild herzunehmen.

Und auch hier steht das gewünschte Bild bereit analysiert zu werden. Das Abbild kann jederzeit geändert oder nochmals geschossen werden.

### Pilzliste

Will man jetzt genauer über bestimmte Pilze informiert werden, kann auf die *Pilzliste*-Button gedrückt werden.

Das ist die Detailansicht dazu.

# Technologien

## Java (Android)

## Swift/Objective-C (IOS)

Die grafische Oberfläche der IOS App wurde in Swift programmiert, da es die effizienteste und performanteste Möglichkeit ist Apps für IOS zu programmieren. Die Schnittstelle zwischen den C++ und IOS Teil wurde aufgrund der Kompatibilität in Objective-C realisiert.

## C++ (OPEN CV)



Abbildung : Logo Android Studio

## Android Studio

Android Studio ist die offizielle integrierte Entwicklungsumgebung (IDE) für die Android Plattform.

Android Studio ist speziell für die Android Entwicklung entwickelt geworden. Sie kann frei für Windows, MacOS und Linux heruntergeladen werden.

Wieso Android Studio so vorteilhaft ist:

Gradle Integration: Android Studio benutzt den rasant wachsenden Gradle build System. Gradle automatisiert und liefert bessere Software schneller.

Erweiterte Code Ergänzung: Android Studio liefert präzise Code Ergänzungen, welches für ein Programmiere viel Zeit und unnötiges langes Fehlerbehebung spart.

User Interface: Android Studio hat eine sehr benutzerfreundliche Benutzeroberfläche, als ein Einsteiger hat man keine Probleme beim Zurrechtfinden im Programm.

Organisierung des Projektes: Android Studio benutzt Module, welche alle eine eigene Gradle build Datei besitzen, die seine eigenen Abhängigkeiten angeben kann. Außerdem hat Android Studio eine Funktion, welche das zuletzt geschlossene Projekt beim Start öffnet, welche unnötige Zeit spart.

System Stabilität: Android Studio hat eine stabile Performance hat weniger Software Fehler und die benötigten Systemeigenschaften sind auch sehr niedrig.

Drag and Drop: Android Studio hat eine Drag and Drop Funktion eingebaut, welches über die Grafische Benutzeroberfläche benutzt werden kann.

## Android Studio NDK

Die Native Development Kit ( NDK ) erlaubt es, C und C++ Codes mit Android zu verwenden. Sie bietet Bibliotheken an, welche erlauben die Aktivitäten zu konstruieren, Benutzereingabe behandeln, Hardware Sensoren benutzen und den Zugang zu Applikationsressourcen bieten, wenn man in C/C++ programmiert.



Abbildung : Logo CMake

## CMake

CMake ist eine Open-Source Plattform, welcher Software baut, testet und verpackt.

CMake ist ein erweiterbares Open-Source Programm, welches den Aufbauprozess im Betriebssystem und Übersetzungsunabhängig verwaltet. Mit einer einfachen CMakelists.txt Datei wird die Standard Konstruktions-Datei erzeugt.

Außerdem ist CMake in der Lage Quellcodes zu übersetzen, Bibliotheken zu erzeugen, Programm Verpackungen zu generieren und Ausführbare Dateien konstruieren.



Abbildung : Logo Gradle

## Gradle

Gradle ist ein Open-Source-Automatisierungssystem.

Noch bevor Android Studio wurde Eclipse für die Entwicklung von Android Apps verwendet und mit einer hohen Chance wussten die meisten nicht, wie man eine Android Applikations Paket (APK) konstruierte. Eclipse hat eine eigene Build System, welches eine APK baut, aber was Eclipse nicht hat ist ein automatisiertes Konstruktionssystem.

Hierfür müsste man selbst ein Skript schreiben, der verschiedene Schritte ausführen muss.

Gradle ist ein weiterer Konsturktions-System, welcher die besten Eigenschaften von anderen System nimmt und diese in einem kombiniert. Du kannst deinem eigenen Skript in Java schreiben, welches dann von Android Studio benutzt wird.

Gradle kann zum Beispiel ein Verzeichnis zu einem anderen Verzeichnis kopieren noch bevor der eigentliche Konstruktionsprozess passiert.



## Boost ( C++ Libraries )

Boost ist eine Sammlung von sehr nützlichen und hochqualitativen C++ Bibliotheken, welche die kleinere Standard Bibliotheken ergänzen.

Sie bietet fehlende Werkzeuge an, welche in den Standard C++ Bibliotheken nicht beinhaltet sind. Mit Boost werden etwaige Designs und Implementierungen erspart und die Lösungen können einfach getestet werden. Sie unterstützt viele Entwicklungsumgebungen und pflegt manchmal die Lösungen um effizienter zu arbeiten.



Abbildung : Logo XCode

## XCode (IOS)

Die Standard Entwicklungsumgebung für die Erstellung von IOS Apps. Entwickelt vom Hersteller Apple.



Abbildung : Logo Visual Studio

## Visual Studio (OPEN CV)

Ist eine von dem Unternehmen Microsoft aangebotenen Entwicklungsumgebung für verschiedene Hochsprachen (darunter C, C++, C#, Phyten, HTML, JavaScript und Typescript)

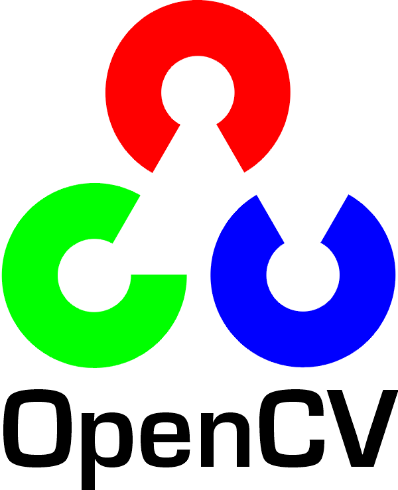


Abbildung : Logo OpenCV

## OpenCV (Open Source Computer Vision)

Diese ist eine in C++ geschriebene Bibliothek, welche ursprünglich von Intel entwickelt wurde, inzwischen jedoch quelloffen unter eine BSD-Lizenz entwickelt wird. Sie umfasst unter anderem Algorithmen für 3D-Funktionalität Gesichtsdetektion und verschiedenste Filter (z. B. Gauß). Mit ihr werden Applikationen erstellt, die sich allgemein mit Computer Vision beschäftigen.

## CMARKUP (XML Lesen)

Simpler XML Parser mit guter Dokumentation und Erklär Videos wie man ihn benützt.

Dabei handelt es sich um ein einfach zu implementierende Lösung (eine C++ Klasse). Im Grunde ist es ein vereinfachtes DOM Modell, dass sich auf das wesentliche, dem lesen von XML konzentriert.

## Computer Vision

Beschreibt die computergestützte Lösung von Aufgaben, die sich an den Fähigkeiten des menschlichen visuellen Sehens orientiert. Wird bei der Automatisierungstechnik bei der Qualitätssicherung, bei Radarfallen bis hin zum selbstfahrenden Auto und in der Sicherheitstechnik eingesetzt.

## Mensch-Computer-Interaction

## Haar Cascade Training

Wird für das maschinelle Lernen verwendet. Es besteht aus zwei Phasen: Training und Detection. In der Training Phase werden Bilder mit dem zu erkennenden Objekt und Bilder, auf denen das zu erkennende Objekt nicht zu sehen ist, gesammelt. Über den traincascade Algorithmus werden dann bestimmte Eigenschaften, anhand denen die Objekte erkannt werden können, und solche, die es von den nicht zu erkennenden unterscheiden, ermittelt und in eine XML Datei geschrieben. Anschließend kann mithilfe von diesem die Bilderkennung durchgeführt werden.

## Entscheidung für Native Apps

Die Anwendung soll dem Nutzer ein Benutzungserlebnis („Look and Feel“) bieten, das er auf den jeweiligen Plattformen gewohnt ist. Trotz vieler APIs für den Kamerazugriff oder Zugriff auf das Filesystem ist es für unsere Anwendung angenehmer und performanter direkt solche Funktionen zuzugreifen. Weiters soll es eine gemeinsame Codebasis (in C++) geben, auf die sowohl für den Android und IOS Teil verwendet wird. Dies erleichtert die Wartung eines Programms wesentlich, da bei Änderungen des Bilderkennungsalgorithmus sofort alle Änderungen sofort für Android und IOS zur Verfügung steht und nicht für beide Systeme neu implementiert werden muss. Es ist somit auch innerhalb kürzerer Zeit möglich das System für ein anderes System (Beispielsweise Weise Windows Phone, Virtual Reality Brillen. oder was die Zukunft noch bringt) zu implementieren.

# Programmierung

## Bilderkennung

Der Identifiaktionsprozess besteht aus insgesamt drei Phasen:

### Kurzerklärung der Schritte

Einlesen des XMLs (8.1.1.1)

Pilzeigenschaften werden geladen.

Farberkennung (8.1.1.2)

Die Farbe des Pilzes wird herausgefiltert und mit den gespeicherten Pilzeigenschaften verglichen.

Konvertierung in HSV Farbraum (8.1.1.3)

Um bessere Lichtquellenunabhängigkeit zu erreichen wird der Farbraum verändert. Und ein schwarz-weiß Bild errechnet.

Canny Edge Detector (8.1.1.4)

Hiermit werden die Kanten eines Pilzes gezeichnet.

Hough Circle Transformation (8.1.1.5)

Hiermit werden Kreise aus dem Bild erkannt

Das maschinelle Lernen

Benutzerfragen

### Computer Vision

Funktionsweise Bilderkennung

Abbildung : Farberkennung Eierschwammerl

Zuerst wählt der Benutzer auf dem Smartphone einen quadratischen Bereich aus, in dem sich der Pilz befindet.

#### Einlesen eines Fotos

imread("..\\..\\common\\data\\eiersch.jpg")

Das Bild wird mit der Klasse cv::Mat gespeichert. Jeder Pixel ist somit ein Eintrag in der Matrix. Dabei werden unter anderem die Farbwerte gespeichert, die wir in Folge brauchen werden.

#### Einlesen des XMLs (CMARKUP)

while (xml.FindElem(MCD\_T("Schwammerl")))

{

xml.IntoElem();

counter\_str = to\_wstring(counter);

pilz = ws + counter\_str;

xml.FindElem(MCD\_STR(pilz)); //z. B. P1, P2, P3, ...

xml.IntoElem();

Vec3b bgr;

//Farbe (BGR)

xml.FindElem(MCD\_T("Farbe"));

mush.bgr[0] = std::stoi(xml.GetAttrib(MCD\_T("b")));

mush.bgr[1] = std::stoi(xml.GetAttrib(MCD\_T("g")));

mush.bgr[2] = std::stoi(xml.GetAttrib(MCD\_T("r")));

...

Es wird so lange in dem XML gesucht, bis ein Schwammerl

#### Farberkennung

Dann wird in diesem Quadrat ein Bereich im Zentrum nach der Farbe untersucht. Der Mittelwert aus den erkannten Farben wird daraufhin mit den Daten aus der XML Datei verglichen. Alle Pilze, die eine ähnliche Farbe besitzen kommen in die nähere Auswahl.

Sollte die Farbe einzigartig sein (wie z. B. beim Grünspantäuschling) kann es vorkommen, dass der Erkennungsprozess ab diesem Punkt abgeschlossen ist.

Code Snippet:

//Mehrere (100) Pixel in der Mitte durchsuchen

for (int i = -5; i < 5; i++) {

for (int j = -5; j < 5; j++) {

array2[0] += image.at<Vec3b>(rows\_mid + i, cols\_mid + j)[0];

array2[1] += image.at<Vec3b>(rows\_mid + i, cols\_mid + j)[1];

array2[2] += image.at<Vec3b>(rows\_mid + i, cols\_mid + j)[2];

cout << "yey" << array2[0];

}

}

cout << " rows: " << rows\_mid;

cout << " cols: " << cols\_mid;

//Durchschnittswert der Pixelfarben errechnen

pix[0] = array2[0] / 100;

pix[1] = array2[1] / 100;

pix[2] = array2[2] / 100;

#### Konvertierung in HSV Farbraum

Wenn dem nicht der Fall ist wird das Bild in den HSV Farbraum konvertiert, da dieser wesentlich unempfindlicher für verschiedene Lichtquellen ist als der RGB/BGR-Farbraum.

Abbildung : Eierschwammerl HSV

Code Snippet:

cv::cvtColor(image, hsv\_image, cv::COLOR\_BGR2HSV);

Daraufhin wird nur die Farbe des Pilzes herausgefiltert, sodass der Pilz weiß und der Rest schwarz dargestellt wird.

Code Snippet:

for (int i=0; i<mushlist.size(); i++)

{

if (pix[0]<mushlist[i].bgr[0] + schw && pix[0]>mushlist[i].bgr[0] - schw && pix[1]<mushlist[i].bgr[1] + schw && pix[1]>mushlist[i].bgr[1] - schw && pix[2]<mushlist[i].bgr[2] + schw && pix[2]>mushlist[i].bgr[2] - schw) {

mushlist2.push\_back(mushlist[i]);

wcout << "\n\nSchwammerlname: " << mushlist[i].name;

inRange(hsv\_image, Scalar(mushlist[i].hsv\_v[0], mushlist[i].hsv\_v[1], mushlist[i].hsv\_v[2]), Scalar(mushlist[i].hsv\_b[0], mushlist[i].hsv\_b[1], mushlist[i].hsv\_b[2]), hsv\_first);

inRange(hsv\_image, Scalar(mushlist[i].hsv\_v2[0], mushlist[i].hsv\_v2[1], mushlist[i].hsv\_v2[2]), Scalar(mushlist[i].hsv\_b2[0], mushlist[i].hsv\_b2[1], mushlist[i].hsv\_b2[2]), hsvhelp);

cv::addWeighted(hsv\_first, 1.0, hsvhelp, 1.0, 0.0, hsv\_first);

}

}

Kurzbeschreibung:

Es werden die Grundfarben (Blau, Gelb und Rot), die die mit einer Schwelle von 30 Farbpunkten vom erkannten Pilz mit den gespeicherten Farben aus der XML verglichen. Wenn alle drei Grundfarben innerhalb von einer Schwelle von 30 zu denen zu einem Pilz gespeicherten Farben passen, wird der Pilz in mushlist2 gespeichert. Nur die in mushroom2 gespeicherten Pilze werden weiter untersucht.

Der Pilz wird daraufhin noch Weichgezeichnet und auf diesem Bild wird ein Kreiserkennungsalgorithmus durchgeführt.

Code Snippet:

GaussianBlur(src\_gray, src\_gray, Size(9, 9), 2, 2);

Eingabeparameter

1. Eingabebild
2. Ausgabebild
3. Faltungsmatrix (*convolution kernel)*
4. Standardabweichung

<https://de.wikipedia.org/wiki/Faltungsmatrix>

<https://www.youtube.com/watch?v=C_zFhWdM4ic&feature=youtu.be>

<http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>

Faltungsmatrix

Dabei handelt es sich um Matrizen in ungerader Größe, die über das gesamte Bild gelaufen wird. Im Vergleich zu einem „Standardblur“ (Matrix mit nur Einsen), wird darauf Wert gelegt, die Pixel, die sich in der unmittelbaren Umgebung zu dem bearbeiteten Pixel befinden zu bevorzugen.

Im folgenden Beispiel würde

###### Standardblur

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | 14 | 13 | 09 | 17 |
| 21 | 64 | 62 | 41 | 19 |
| 42 | 54 | 61 | 52 | 40 |
| 41 | 30 | 31 | 34 | 38 |
| 20 | 40 | 38 | 35 | 24 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Ergebnis:

1\*17+1\*14+1\*13+1\*21+1\*64+1\*62+1\*42+1\*54+1\*61=348 / 9 = ca. 39

Dieser Wert wird in ein neues Bild an derselben Stelle eingefügt. Bei dem Gaußschen Weichzeichner will man jedoch einen Unterschied machen von Bildern, die weiter weg vom Ausgangspixel sind, als die direkten Nachbarn.

###### Gaußscher Blur

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | 14 | 13 | 09 | 17 |
| 21 | 64 | 62 | 41 | 19 |
| 42 | 54 | 61 | 52 | 40 |
| 41 | 30 | 31 | 34 | 38 |
| 20 | 40 | 38 | 35 | 24 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

Ergebnis:

1\*17+2\*14+1\*13+2\*21+4\*64+2\*62+1\*42+2\*54+1\*61 = 691 / ( 1+2+1+2+4+2+1+2+1) 17=ca. 41

In der Regel wird eine größere Matrix verwendet und in der Matrix Zahlen im Kommabereich, wurde jedoch für Vorzeigezwecke vereinfacht. Bei größeren Matrizen ist es auch wichtig, dass jenen Pixel mehr Gewicht gegeben wird, die näher am Ausgangspixel liegen.

Mit der äußersten Pixelreihe kann nicht durchgeführt werden. Dafür gibt es mehrere Ansätze:

1. Man erfindet eine zusätzliche Pixelreihe
2. Man ignoriert die äußerste Reihe. Dieser Vorgang würde bei dem oberen Beispiel zwar zu sichtbaren Verfälschung führen, bei Bildern von mehreren Megapixeln jedoch sehr irrelevant sein. Vorteil: Rechenleistung erspart.

#### Canny Edge Detector

Dazu werden mit dem Canny Edge Detector bearbeitet. Dazu wird davor noch einmal ein blur-Filter darübergelegt.

/// Reduce noise with a kernel 3x3

blur(src\_gray, detected\_edges, Size(3, 3));

/// Canny detector

Canny(detected\_edges, detected\_edges, lowThreshold, lowThreshold\*ratio, kernel\_size);

(<http://www.mathematik.uni-ulm.de/stochastik/lehre/ws05_06/seminar/ausarbeitung_wagner.pdf>)

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf

John F. Canny berichtete 1986 in seinem Text “A Computational Approach to Edge Detection” den idealen Kantendetektor entwickelt zu haben, den Canny Edge Detector.

Dafür sollen alle Kanten gefunden werden.

Die Kante sollte möglichst genau auf der Kante erkannt werden, nicht daneben. Weiters sollen kannten laut ihm nicht mehrfach erkannt werden.

In seinem Algorithmus kann es nur 0=keine Kante, oder 1=eine Kante geben. Dazu werden mehrere Schritte durchgeführt:

##### Glättung, um Rauschen zu verhindern

Dazu wird ein einfacher Gaußscher Weichzeichner angewendet (siehe

##### Kantendetektion (Sobel Algorithmus)

Dafür wird der Sobel Algorithmus verwendet

Dafür kommen, wie bei den Blur-Algorithmen, Faltungsmatrizen zum Einsatz.

Faltungsmatrize für x-Achse Faltungsmatrize für y-Achse

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

###### Beispiel X-Achse

|  |  |  |  |
| --- | --- | --- | --- |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Ergebnis:

(-1) \* 100 + 0 \* 100 + 1 \* 50 + (-2) \* 100 + 0 \* 100 + 2 \* 50 + (-1) \* 100 + 0 \* 100 + 1 \* 50 =

-200

Bei der Untersuchung auf der X-Seite ist also ein starker Kontrast zu erkennen. Der Wert ergibt -200

###### Beispiel Y-Achse

Bei der Untersuchung auf der Y-Seite darf keine Kante erkannt werden, der Wert muss 0 ergeben.

|  |  |  |  |
| --- | --- | --- | --- |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |
| 100 | 100 | 50 | 50 |

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Ergebnis:

(-1) \* 100 + (-2) \* 100 + (-1) \* 50 + 0\*100 + 0 \* 100 + 0 \* 50 + 1 \* 100 + 2 \* 100 + 1 \* 50 = 0 Es wurde keine Kante gefunden.

###### Gradianten zusammenfügen

Nun hab man 2 Gradienten (Gx, Gy). Diese können dann zusammen kombiniert werden, um die absolute Größe des Gradienten an jedem Punkt und die Orientierung dieses Gradienten zu finden. Die Gradientengröße ist gegeben durch:

Eqn:eqnrob1

Das kann mithilfe des Pythagoras berechnet werden. Dadurch werden unter anderem auch die negativen Gradienten eliminiert.

Aus Performance Gründen wird auch oft auf eine Annäherung zurückgegriffen:

Eqn:eqnrob2

Die Richtung der einzelnen Kantenpixel kann mithilfe des Arkustangens berechnet werden.

Eqn:eqnsob3

Da jeder Pixel allerdings nur 8 Nachbarn hat, werden die Kantenrichtungen gerundet auf 0°, 45°, 90° und 135°

##### Unterdrückung von Nicht-Maxima

Gerade, wenn davor der Gaußsche Weichzeichner angewendet wird, werden beim reinen Canny mehrere Kanten erkannt. Dabei werden Kanten, deren Breite mehr als ein Pixel beträgt auf nur ein Pixel „ausgedünnt“. Es sollen nur die Maxima, also die größten Kantenwerte erhalten bleiben. Für jedes Pixel aus G(x,y) wird der Wert mit dem Wert links und rechts verglichen. Nur der Maximalwert wird als Kante erkannt bleiben, der Rest wird auf Null gesetzt. Dafür ist auch die Kantenrichtung aus dem Sobel Algorithmus wichtig, um damit die echten Kantennachbaren zu bestimmen. Nach diesem Prozess wir jede Kante in der Mitte als „einpixelig“ erkannt.

##### Hysterese

Nun werden noch an zu vielen Orten Kanten erkannt. Um das zu verhindern gibt es zwei Schwellwerte minVal und maxVal. Alle Werte unterhalb von minVal werden automatisch nicht als Kante erkannt, alle Pixel, die über maxVal liegen, werden automatisch als Kante erkannt. Alle Pixel dazwischen werden nur dann als Kante erkannt, wenn sich direkt daneben ein „Kantenpixel“ befindet.



Abbildung 11: Canny maxVal, minVal

http://docs.opencv.org/trunk/da/d22/tutorial\_py\_canny.html

Der Teil A in diesem Bild wird generell als Bild Kante erkannt. Den Teil C, also jenen, der unter der oberen Schwelle liegt, wird nur deshalb auch als Kante erkannt, weil die Kante mit dem A-Teil der Kante zusammenhängt. Wäre diese Kurve unter die minVal gegangen, würde sich eine Lücke bei der Kante auftuen.   
Die mögliche Linie B, wird nicht als Kante erkannt, da sie sich zwischen den 2 Parameter befindet und nicht mit einem Teil über der maxVal zusammenhängt.

#### Hough Circle Transformation

Und mit dem Ergebnis nach Kreisen gesucht.

Code Snippet:

int HoughDetection(const Mat& src\_gray, const Mat& src\_display, int cannyThreshold, int accumulatorThreshold)

{

// will hold the results of the detection

std::vector<Vec3f> circles;

// runs the actual detection

HoughCircles(src\_gray, circles, HOUGH\_GRADIENT, 1, src\_gray.rows / 8, cannyThreshold, accumulatorThreshold, 0, 0);

// clone the colour, input image for displaying purposes

Mat display = src\_display.clone();

for (size\_t i = 0; i < circles.size(); i++)

{

Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));

int radius = cvRound(circles[i][2]);

// circle center

circle(display, center, 3, Scalar(0, 255, 0), -1, 8, 0);

// circle outline

circle(display, center, radius, Scalar(0, 0, 255), 3, 8, 0);

}

// shows the results

imshow(windowName, display);

return (circles.size());

}

#### Findungsprozess der richtigen Parameter:

##### Mat src\_gray

Eingabebild (in diesem Fall ein weichgezeichnetes Schwarz-Weiß-Bild von einem Pilz)

##### vector<Vec3f> circles

Vektor (Vergleichbar mit Liste in anderen Sprachen), in der Kreisinformationen gespeichert werden. Koordinaten von Kreismitte und Radius.

### CV\_HOUGH\_GRADIENT:

Name der Erkennungsmethode (zurzeit ist nur dieser Verfügbar)

##### DP:

Je kleiner die DP, desto genauer ist die Kreiserkennung.

Je genauer die Kreiserkennung ist, desto schneller werden nicht perfekte Pilze nicht als Kreis, oder mehrere Kreise bei dickere Kanten erkannt.

##### Min\_dist

src\_gray.rows/8: Minimale Distanz zwischen erkannten Kreiszentren

##### param\_1:

Obere Schwelle für den internen Canny Edge Detector (siehe Canny Edge Detector)

Entscheidung: 99

x

##### param\_2:

Schwellenwert für die Mittenerkennung

Entscheidung: 41

##### min\_radius:

Minimaler Kreisradios um erkannt zu werden (0=egal)

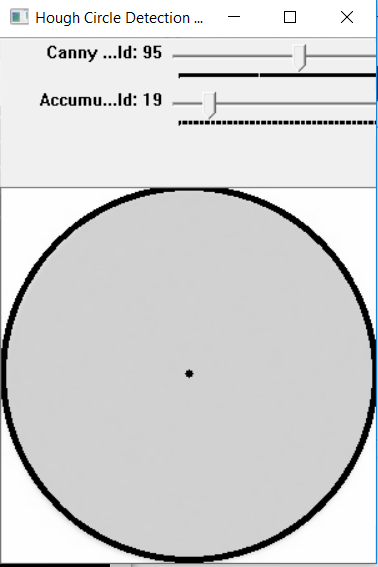


Abbildung : Trackbar Kreiserkennung

##### max\_radius:

Maximaler Kreisradius um erkannt zu werden (0=egal)

#### Entscheidung für param\_1 und param\_2:

Dazu wurden 10 runde Kreise / Pilze und 10 nicht runde Pilze als Testfälle bestimmt. Dabei wurde param\_1 und param\_2 mit Trackbars so verändert, dass richtiger Weise entweder ein Kreis oder kein Kreis erkannt wird. Dier Fall trat bei den Parametern 95 für den Parameter param\_1 und 41 bei dem Parameter param\_2 ein.



Abbildung : Fuchsiger Rötelritterling Kreiserkennung



Abbildung :Eierschwammerl HSV schwarz weiß Weichzeichnen

Abbildung : Eierschwammerl HSV schwarz weiß

Bei dem Eierschwammerl werden (zurecht) keine Pilze gefunden. Bei dem immer runden Ruchsigen Rötelritterling wird dagegen zurecht ein Kreis erkannt.

#### Ja/Nein Benutzerfragen

Wird daraufhin noch kein Pilz erkannt, werden dem Benutzer so lange JA/Nein Benutzerfragen gestellt, bis ein Pilz eindeutig identifiziert ist.

Beispielsweise:

Hat der Pilz Lamellen?

Der Benutzer

Dieses kann mit JA oder NEIN beantwortet werden.

### Datenspeicherung (XML)



Entscheidung für die Datenspeicherung (XML)

Der Grund für die Speicherung der Daten im XML Format lokal auf dem Gerät und gegen eine ausgelagerte Datenbank ist, dass im Wald oftmals keine ausreichende Internetverbindung verfügbar ist. Das kann dazu führen, dass die Bilderkennung nicht, oder erst zu spät durchgeführt werden könnte.

Dazugehörige Klasse in C++:

class Pilz { //Pilzklasse

public:

Vec3b bgr; //BGR Farbe

Vec3b hsv\_v; //HSV Bereich Begin (von)

Vec3b hsv\_b; //HSV Bereich Ende (bis)

Vec3b hsv\_v2;//HSV Bereich Begin (von) für Rottöne

Vec3b hsv\_b2;//HSV Bereich Ende (bis) für Rottöne

wstring name; //Name des Pilzes

wstring wiki; //Wikipedia Link

wstring lamell; //1 für es gibt Lamellen, 0 für es gibt keine Lamellen, Eigenschaftswort für "Hat der pilz ... Lamellen?"

int roud; //ist der Pilz Rund, 1 ja, 0 nein

int poisonous; //ist der Pilz giftig, 1 ja, 0 nein

wstring nodule; //= Knolle, Eigenschaftswort (z. B. dicke, rundliche etc.)

wstring stalk;

};

#### Probleme

##### Veraltete OpenCV Installationsanleitung

Die offizielle OpenCV Installationsanleitung ist veraltet und aus jetziger Sicht sehr inkorrekt <http://docs.opencv.org/2.4/doc/tutorials/introduction/windows_install/windows_install.html>

Lösung:

aktuellere Guides gesucht und gefunden => Problem: nicht für die aktuellste Version=> Guide für meine Version angepasst.

##### BGR Farbraum in OpenCV

In OpenCV wird nicht der RGB (Standard) Farbraum, sondern der BGR Farbraum verwendet => führte zu vermeintlich falscher Ergebnisse => Tipp in Forum führte mich zu der Lösung

<http://stackoverflow.com/questions/8932893/accessing-certain-pixel-rgb-value-in-opencv>  
Die Frage wieso nicht das gebräuchlichere RGB zum Einsatz kommt hat der Gründer von OPENCV Dr. Gary Bradski in einem Interview folgender Maßen geantwortet:

“Why is the the US standard railroad gauge 4 feet, 8.5 inches?”…

“Because of Roman horse’s ass!”

Diese Aussage hat der Blogger und Interviewführer Satya Mallick so interpretiert, dass der Grund für die Verwendung vom BGR Farbformat sei, dass, als Dr. Gary Bradski begonnen hat, OpenCV zu entwickeln, sowohl Kamerahersteller und Softwarehersteller eher das BGR Format 0x00bbggrr verwendeten als das RGB Format. Also sei es eine rein historische Entscheidung gewesen.

https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/

##### HSV Farbraum nur bis zum Wert 180

Der HSV Farbraum ist in OpenCV nur bis 180 (um in ein uchar zu passen) gehend, normale Farbraumumrechner (Word, GIMP, Photoshop, Internetrechner) rechnen mit 360 => Lösung => eigenen Farbraumumrechner für OPENCV gesucht und gefunden

<http://www.shervinemami.info/colorConversion.html>

Alternative:

Trackbars selbst implementieren => siehe Circle Detection

Farbraum

Im HSV Farbraum ist die Farbe Rot (z. B. Fliegenpilz) 2-Geteilt=>

Lösung 1. Zuerst linken Farbraum untersuchen, dann rechten Farbraum untersuchen dann beide Bilder zusammenfügen

Lösung 2. Man kann über H Wert 181=1 und so weiter => geht allerdings nur bis 255 (um in uchar zu passen)

##### Fliegenpilze komplex

Fliegenpilze (eigentlich rund) sind mit weißen Flecken (auch teilweise an den Rändern übersäht, diese weiße Flecken werden nicht miterkannt und bei der Untersuchung nach Runden Figuren nicht als Rund erkannt => PROBLEM

##### Circle Transformation führt Canny selbstständig durch

Nicht gewusst, dass Circle Transformation den Canny selbstständig durchführt. Davor habe ich ihn selbst auch gemacht und somit doppelt.

### 

### Maschinelles Lernen

Für die Bilderkennung wurde der Haar Cascade Algorithmus für den Fliegenpilz implementiert. Als Ausgangspunkt wurde das Beispiel auf <https://github.com/mrnugget/opencv-haar-classifier-training> verwendet.

Folgende Schritte mussten dafür vollzogen werden:

Als erstes müssen Cygwin, Perl und Python installiert werden, falls das noch nicht schon vorher gemacht wurde.

Mit Cygwin können Linux Konsolenbefehle in der Windows Eingabeaufforderung verwendet werden.

Perl und Python werden benötigt, um zwei später beschriebene Scripts auszuführen.

Suchen von positiven und negativen Bildern. Das heißt, es werden Bilder, auf denen ein Fliegenpilz zu sehen ist, und solche, auf denen kein Fliegenpilz zu sehen ist, gesucht und in die Ordner „positive\_images“ und „negative\_images“ gespeichert. Alle diese Bilder sollten aus der Vogelperspektive gemacht worden sein. Wichtig ist, dass sie sich auch in Helligkeit und Hintergrund unterscheiden. Die negativen Bilder sollten möglichst ähnlich wie die positiven aussehen, mit dem Unterschied, dass das zu erkennende Objekt nicht darauf zu sehen ist. Dafür bieten sich Fotos von anderen Pilzen und Fotos, auf denen nur Waldboden zu sehen ist, an.

Es werden viel mehr negative als positive Bilder benötigt. Je mehr Bilder verwendet werden, desto weniger Falscherkennungen werden bei der Bilderkennung auftreten.  
Als Richtwert für eine akkurate Erkennung können 40 positive und 600 negative Bilder verwendet werden. Natürlich spielt auch die Qualität der Bilder eine Rolle.

Anschließend müssen alle positiven und negativen Bilder in zwei Textdokumente geschrieben werden. Dazu öffnet man die Eingabeaufforderung und wechselt über den „cd“ Befehl in das „open-cv-haar-classifier-training“ - Verzeichnis. Dann führt man die Befehle

find ./positive\_images -iname "\*.jpg" > positives.txt

find ./negative\_images -iname "\*.jpg" > negatives.txt

aus.

Im nächsten Schritt benötigt man Samples, sowohl für positive als auch negative Bilder. Für die negativen Bilder kann das bereits erstellte Textdokument verwendet werden. Bei den positiven Bildern sieht das anders aus: Es wird die von OpenCV bereitgestellt „opencv\_createsamples“ Funktion verwendet (arbeitet mit Rotationen und Transformationen). Im „bin“ – Ordner gibt es ein Perl – Script namens „createsamples.pl“, welches über die Eingabeaufforderung aus dem Verzeichnis aus ausgeführt wird.

perl bin/createsamples.pl positives.txt negatives.txt samples 1500 " opencv\_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"

Parameter:

-bgcolor

-bgtresh

-maxxangle

-maxyangle

-maxzangle

-maxidev

-w

gibt die Breite in Pixel der zu erstellenden Samples an

-h

gibt die Höhe in Pixel der zu erstellenden Samples an

Dann wird das mergevec.py Script im „open-cv-haar-classifier-training“ - Verzeichnis ausgeführt, um die Samples zu fusionieren:

python ./tools/mergevec.py -v samples/ -o samples.vec

Anschließend kann mit dem Training begonnen werden. Dazu wird die Funktion opencv\_traincascade verwendet:

opencv\_traincascade -data classifier -vec samples.vec -bg negatives.txt -numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 1500 -numNeg 60 -w 80 -h 40 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024

Parameter:

-data

gibt das Verzeichnis an, in dem der trainierte Classifier gespeichert wird

-vec

gibt die “.vec” – Datei mit den positiven Samples an

-bg

gibt die “.txt” – Datei mit den negative Samples an

-numStages

Anzahl der durchlaufenen Phasen

-minHitRate

Minimale Trefferquote pro Phase

-maxFalseAlarmRate

Maximale Falschalarmrate

-numPos

Anzahl der positiven Samples

-numNeg

Anzahl der negativen Samples

-w

Breite der Samples (muss gleich sein wie bei der „opencv\_createsamples“ – Funktion)

-h

Höhe der Samples (muss gleich sein wie bei der „opencv\_createsamples“ – Funktion)

-mode

gibt die Art von Haar Eigenschaften an, die im Training verwendet werden

“ALL“ verwendet im Vergleich zu „BASIC“ zusätzlich zu den aufrechten Funktionen auch die um 45 Grad gedrehten

-precalcValBufSize

-precalcldxBufSize

#### Probleme

### Tätigkeiten bis zum Endbenutzerprodukt

Aufnahme von mindestens 1000 - besser zwischen 3000 und 5000 Pilze.

Für unsere Diplomarbeit mit einem Zeitraum von unter einem Jahr verbunden mit der Rechenleistung, die mit Privaten Rechnern möglich ist, ist es unmöglich ein zu 100 Prozent funktionierendes Maschinelles Lernen umzusetzen.

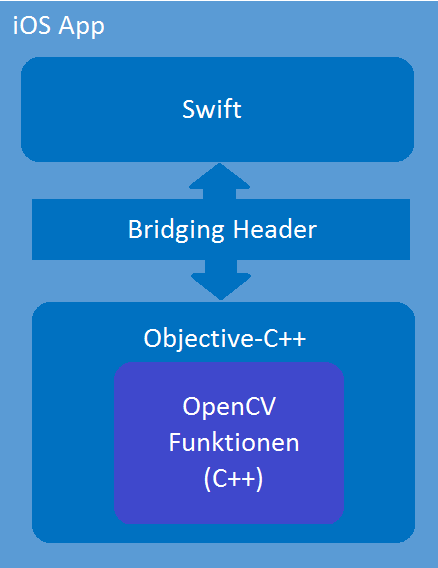
Die perfekte Lösung wäre es, einen großen Benutzerumfang aufzubauen, die Fotos schießen. Diese Bilder werden dann mit den anderen Erkennungsmethoden wie der Bilderkennung und den Ja/Nein Benutzerfragen vom Benutzer identifiziert. Diese Fotos werden mit den Ergebnissen dann auf einen sehr Leistungsfähigen „Mushroom Identifier Server“ übertragen und in den Maschinelles Lernen Algorithmus integriert. Erst ab einer bestimmten Anzahl von integrierten Bildern (ca. 500 positive) wird ein Pilz für den Maschinelles Lernen Algorithmus freigeschaltet.

## iOS – App

Um mit der Entwicklung in iOS zu beginnen werden zunächst einige Technologien und Werkzeuge benötigt:

XCode

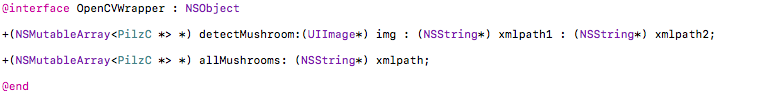
OpenCV

Die App selbst ist in der Programmiersprache Swift geschrieben. In Swift gibt es jedoch keine Möglichkeit C++ Code zu verwenden. Die OpenCV – Bilderkennung ist aber in C++ geschrieben. Die Lösung für dieses Problem ist Objective C++, wodurch man Objective C und C++ Code verwenden kann.

Um den Bilderkennungsalgorithmus einzubauen muss die Source.cpp Datei ins Projektverzeichnis kopiert werden. Anschließend erstellt man eine .mm Datei und dazu eine .h Datei. Automatisch kann man sich dazu einen Bridging Header erstellen lassen.

### OpenCVWrapper.h

Header Dateien in Objective C haben die Dateiendung „.h“. Sie sind dafür da, um Methoden zu definieren.



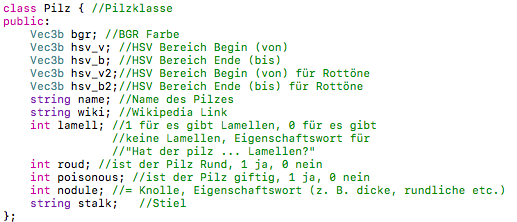
### OpenCVWrapper.mm

In der OpenCVWrapper.mm Datei werden die Methoden aus der Header Datei implementiert. Im Gegensatz zu „.m“ Dateien kann bei einer „.mm“ Datei nicht nur Objective C, sondern zusätzlich auch C++ Code verwendet werden.

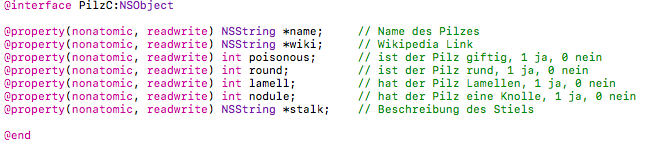
#### Pilz Klasse

Als Rückgabeparameter muss ein Objective C Datentyp zurückgegeben werden. Da die Bilderkennung jedoch einen C++ Datentyp zurückgibt, wird eine neue Objective C Klasse benötigt.

**C++:**



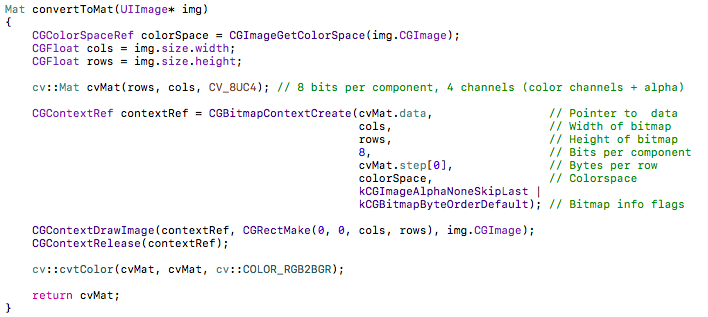
**Objective C:**



Die Pilz – Klasse in Objective C beinhaltet nicht mehr alle Eigenschaften eines Pilzes, sondern nur mehr die, die später in der App angezeigt werden sollen.

#### convertToMat

In Swift werden Bilder als „UIImage“ gespeichert. Diesen Datentyp gibt es jedoch in Objective C++ nicht. Stattdessen wird „cv::Mat“ verwendet, wozu allerdings eine Methode benötigt wird, die für die Umwandlung von „UIImage“ in „cv::Mat“ vornimmt.



#### detectMushroom

In dieser Methode wird der Bilderkennungsalgorithmus aufgerufen und die Ergebnisliste vom C++ Datentyp „Pilz“ in den Objective C Datentyp „PilzC“ konvertiert.



**Parameter:**

UIImage\* img: das zu analyserende Pilz - Bild

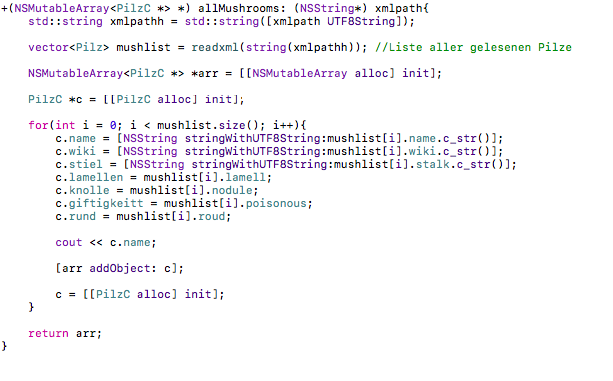
NSString\* xmlpath1: gibt den Pfad zur xml – Datei für die Bilderkennung an

NSString\* xmlpath2: gibt den Pfad zur xml – Datei für den Haar Cascade Algorithmus an

Zuerst werden die übergebenen xml – Pfade in C++ Strings konvertiert. Anschließend wird der Bilderkennungsalgorithmus aufgerufen und ein Pilz in C++ zurückgegeben. Dazu ist es notwendig, das zu analysierende Bild mittels der convertToMat Funktion umzuwandeln. Als Ergebnis erhält man dann eine Liste aller Pilze, die nach Durchlaufen des Bilderkennungsalgorithmus noch in Frage kommen. Allerdings ist das eine Liste von C++ Pilzen, also müssen diese umgewandelt werden. Das geschieht in einer Schleife, in der alle Elemente in eine neue Liste von Objective C Pilzen konvertiert werden.

#### allMushrooms

Mit dieser Methode wird eine Liste aller Pilze, die in der xml – Datei für die Bilderkennung stehen, zurückgegeben. Auch hier muss die Liste wieder in Objective C umgewandelt werden.



### ViewController

#### Probleme

## Android – App

### Installation

Um mit der Entwicklung des Apps zu beginnen, werden zuerst ein Paar Softwares benötigt.

Zuerst wird Android Studio benötigt, das ist die Entwicklungsumgebung für Android Applikationen.

Aktuellste Version ist hier zu finden:

<https://developer.android.com/studio/index.html?gclid=CMKrwYyFvdICFVEz0wodCWwB2w>

Dann wird die Android NDK benötigt, damit C/C++ Codes übersetzt und geschrieben werden können.

Hier ist eine Anleitung, wie die NDK installiert werden muss:

<https://developer.android.com/studio/projects/add-native-code.html>

Da Android Studio NDK nicht alle C++ Bibliotheken unterstützt und dadurch benötigte Funktionalitäten fehlen wird die Boost C++ Bibliothek benötigt.

Die aktuellste Boost Version:

<https://sourceforge.net/projects/boost/files/boost/1.63.0/>

Und hier eine Anleitung wie Boost zu installieren ist:

<http://www.boost.org/doc/libs/1_63_0/more/getting_started/windows.html>

Notiz: In der Diplomarbeit wird Boost 1.60 verwendet, da zu dem Zeitpunkt der Installation 1.63 ( 26.Dezember.2016 ) noch nicht existierte.

Link: <http://www.boost.org/doc/libs/1_60_0/more/getting_started/windows.html>

OpenCV Source Code in der Android Native Toolchain kompilieren

1. CMake installieren
2. Python installieren
3. [Android NDK standalone toolchain](https://developer.android.com/ndk/guides/standalone_toolchain.html) durchlesen und die Parameter in der Datei maketoolchain.cmd anpassen.
   1. <https://developer.android.com/ndk/guides/standalone_toolchain.html>
   2. Mein maketoolchain.cmd: *python %NDK%\build\tools\make\_standalone\_toolchain.py --arch x86\_64 --api 24 --stl=stlport --unified-headers --install-dir %STANDALONE\_TOOLCHAIN%*
4. OpenCV [Source Code](https://github.com/opencv/opencv/releases) downloaden und in ein Verzeichnis entpacken.
   1. <https://github.com/opencv/opencv/releases>
   2. Zip Datei herunterladen
5. Die Datei env.cmd in einem Ordner erzeugen und folgende Pfade am eigenen Rechner anpassen
   1. Android\_SDK: Das ist der Pfad zur SDK von Android Studio ( meistens unter User/User/AppData/Local/Android/sdk
   2. ANDROID\_NDK: Pfad zum heruntergeladenen NDK
   3. PYTHON\_HOME: Pfad zum heruntergeladenen Python
   4. STANDALONE\_TOOLCHAIN: Der erzeugte Toolchain wird dann auf diesem Pfad gespeichert
   5. CMAKE\_HOME: Pfad zum installierten CMake
6. Diese angegebene Pfade dann mit *SET PATH=..*  speichern
7. Eine Shell öffnen und ins erzeugte Unterverzeichnis wechseln und dann von dieser Commandline aus maketoolchain.cmd ausführen, dies erstellt auf dem angebenen Pfad der *STANDALONE\_TOOLCHAIN* ein Toolchain -Verzeichnis ndk
8. Eine .cmd Datei erstellen den folgenden Inhalt hat
   1. SET OPENCV\_DIR=C:\Users\Hakan\Desktop\5AHIF\Diplomarbeit\_MushroomIdentifier\opencv-3.2.0
   2. cmake -G"MinGW Makefiles" -DCMAKE\_TOOLCHAIN\_FILE=%OPENCV\_DIR%\platforms\android\android.toolchain.cmake -DCMAKE\_MAKE\_PROGRAM=%STANDALONE\_TOOLCHAIN%\bin\make.exe ..
   3. cmake --build . -- -j8
   4. cmake --build . --target install
9. Im immer noch offenen Kommandozeile das oben erstellte .cmd Datei ausführen
10. Dieses so bebaute neuem Build-Vorgang neu erstelle install Verzeichnis enthält jetzt die include und library-Dateien für die in der Toolchain gewählte Architektur.

Visual Studio wird benötigt um die C++ Applikation auszutesten noch bevor in Android Studio übersetzt wird.

Hier ist ein Link zum freien Download der Community Version:

<https://www.visualstudio.com/downloads/>

Ein Projekt mit C++ Unterstützung muss erstellt werden.

Link zum Tutorial:

<https://developer.android.com/studio/projects/add-native-code.html>

### Übersetzung von C++ Dateien in Android Studio

1. Bestehende C++ Quelldatei in einen eigenen Ordner reinkopieren



1. CMake Konstruktionsskript erzeugen, damit die Quelldatei in eine Bibliothek errichtet wird
   1. Cmake\_miminum-requiews( …. ) gibt an, wenn die nötige Version niedriger ist als die jetzige Version, dann wird der Prozess des Projektes gestoppt und es wird eine Meldung zurückgegeben
      1. Je höher die Version desto mehr Features können angewendet werden
   2. Add\_library ( openlib SHARED src/main/openCV/Source.cpp )
      1. Openlib ist der Bibliotheksname in dem die Quelldateien gespeichert werden
      2. SHARED Bibliotheken sind dynamisch verknüpft und werden erst zur Laufzeit geladen
      3. Die C++ Quelldatei wird in Form eines Pfades angegeben und sie gibt der CMake Datei an, wo der zu übersetzende Datei sich befindet



* 1. Die NDK Bibliotheken muss mit der Bibliothek verknüpfen werden, wo die Quelldatei übersetzt wird, da die NDK Bibliotheken schon bereits übersetzt worden ist, ist nur mehr die Pfadangabe in CMake nötig.
     1. Include(…)
        1. Gibt den CMake Code an, welcher geladen und ausgeführt wird
     2. Find\_library
        1. Diese Funktion gibt die Lokation der NDK Bibliothek an, damit diese Implementiert und benutzt werden
     3. Target\_include\_directories(….)
        1. Gibt an welche zusätzlichen Bibliotheken importiert werden sollen
        2. Openlib
           1. Zielbibliothek
        3. PUBLIC
           1. Gibt an, ob die importierten Bibliotheken zu den andern Bibliotheken hinzugefügt werden sollen, oder ob in einen eigenen Ordner implementiert werden sollen.
        4. Target\_link\_libraries
           1. Verknüpft die Bibliotheken, welche Implementiert werden sollen mit den eingeschlossenen NDK Bibliotheken



1. Gradle mit der CMake Datei verbinden, damit Gradle die Quelldatei in das Android Studio Projekt importieren und die Bibliotheken mit der C++ Datei in eine APK verpacken kann.
   1. Dies kann manuell durchgeführt werden, einfach die *build.gradle* der Datei öffnen
   2. *CMakeLists.txt* eintippen



Hier kommt noch wie ich die Schnittstelle zum OpenCV Code mache

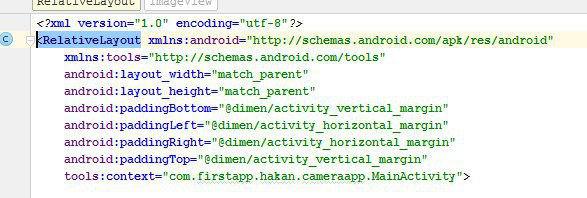
### Design Implementierung

#### View / Benutzersicht

Um die Gestaltung zu muss in die jeweilige xml-Datei der Aktivität gewechselt werden.

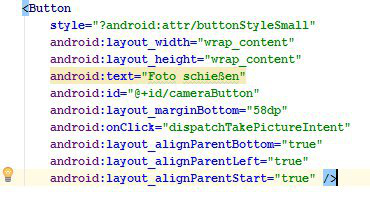


Zuerst wird der Layout Typ festgelegt. Wir verwenden ein RelativLayout, weil es ermöglicht, die einzelnen Objekte je nach Vorgänger oder Nachgänger zu orientieren.



* Layout\_width=“match\_parent“
  + Gibt an, dass die Nachfolger Objekte maximal so groß wie ihre Vorgänger Objekte es angibt, sein dürfen.
* Padding Left/Right/Top
  + Gibt an wie viel Platz nach dem letzten Objekt in der RelativeLayout noch frei ist.
* tools:context
  + Gibt an unter welchen Kontext die Benutzeroberfläche dargestellt werden soll

#### Foto schießen Button



* Style
  + Gibt den Style des Buttons an
* Layout -> wrap\_content
  + Der Button orientiert sich nicht nach Vorgänger Objekte, sondern nach der Größe des Inhaltes, in unserem Fall ist das der *Text.* Je länger der Text desto mehr Platz wird für den Button reserviert.
* android:text
  + Gibt an mit welchem Text der Button angezeigt werden soll
* marginBottom
  + Gibt an wie viel Abstand zum unteren Rand der Smartphone gehalten werden soll
* alignParent…. True
  + Gibt die Positionierung des Objektes in Bezug seines Vorgängers an
* android:onClick
  + Diese Funktion ruft eine Methode im Programm auf, der eine bestimme Aufgabe erfüllt
* onClick Methode dispatcTakePictureIntent
  + Android arbeitet mit Aktionen wie zum Beispiel *onClick* indem er einen Aufruf zu einem Absicht ( intent ) macht
  + Mit takePictureIntent wird eine Absicht zum Schießen eines Bildes gemacht.
  + Um aber diese Absicht auch auführen zu können muss zuerst gefragt werden, ob die Applikation überhaupt eine Aktivität hat, der auch frei ist, diese Intention auszuführen. *startActivityFoResult(…)*

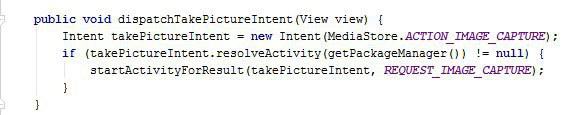
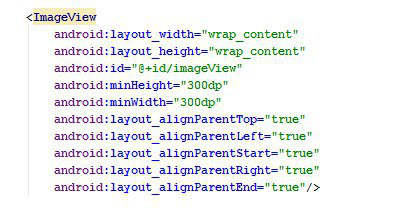


Bild anzeigen

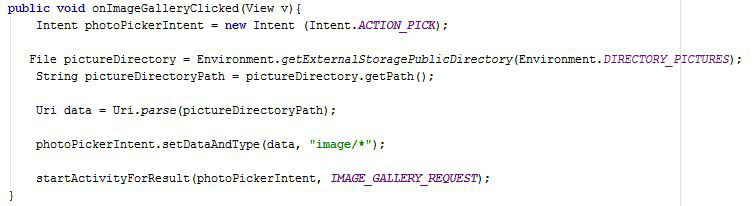


* Android:id
  + Dies ist benötigt um dieses Objekt auch in der Klasse, wo es bearbeitet wird, anreden zu können
* minHeight/Width
  + Gibt die minimale Größe und Weite der *ImageView* an

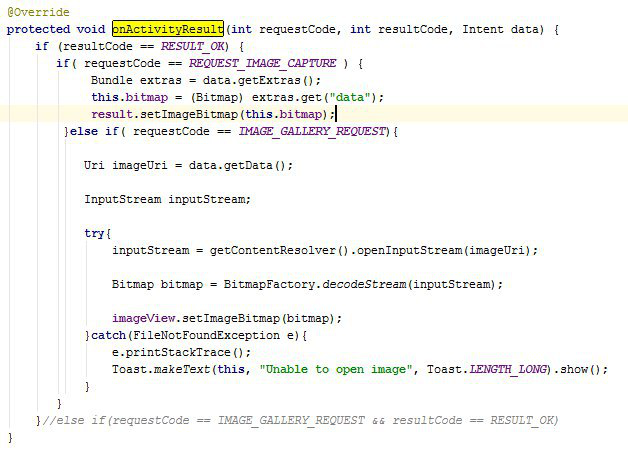
#### Button für die Galerie



* onClick
  + Ein Verweis zu der jeweiligen Methode, welche dann eine Aufgabe erfüllt
* Sobald auf den Button gedrückt wird, wird diese Methode aufgerufen
  + Es wird wieder eine Absicht ersellt mit *Intent.ACTION\_PICK*. Das heißt, dass ein Bild aus der Galerie gewählt werden kann, sobald die Absicht ausgeführt wird.
  + Dann wird der Pfad – also die Lokation der Galerie – in eine Variable gespeichert. *String pictureDirectoryPath*
  + Dieser Pfad wird dann in eine Uri umgewandelt
  + Dann wird angegeben, welche Arten von Bilder alle erlaubt sind mit *image/\** werden alle Arten erlaubt. ( Wie zum Beispiel .jpg, .png etc. )
  + Zum Schluss wird die Absicht gestartet und der Benutzer wählt einen Bild aus.



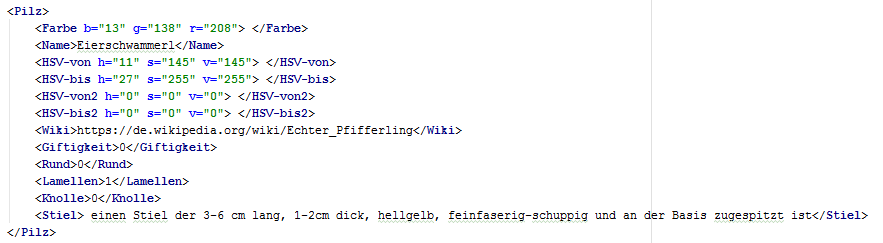
#### Was passiert nachdem die Methoden ausgeführt wurden?



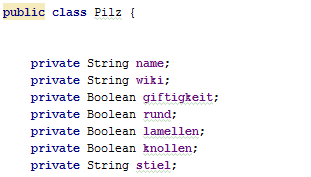
* In dieser Methode wird am Anfang sofort gefragt ob etwas die Aktivität aufgerufen hat. Falls das der Fall ist, wird gefragt, mit welcher Absicht das gemacht wurde.
* Fall 1: *REQUEST\_IMAGE\_CAPTURE* – Foto schießen
  + In diesem Fall wird das aufgenommene Bild in einen *Bundle* gespeichert. Ein Bundle wird verwendet um Daten zwischen Aktivitäten zu speichern.
  + Das Bild wird in eine *Bitmap*gespeichert und später auf dem Bildschirm des Benutzers angezeigt.
  + Ein Bitmap speichert ein Bild als Serien von winzigen Pixel.
* Fall 2: IMAGE\_GALLERY\_REQUEST
  + Hier wird geprüft, ob das Abbild welches gemacht wurde geöffnet werden kann.
    - Falls ja > Bild auf dem Bildschirm anzeigen
    - Falls nein > Fehlermeldung ausgeben

#### Pilzliste

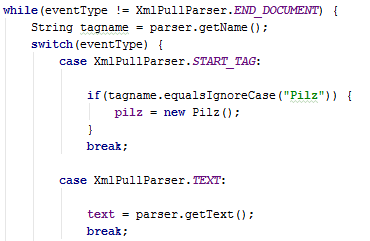
Die Pilze sind in einem XML-File gespeichert.



Android Studio bietet die Klasse XmlPullParserHandler an um aus einer XML-Datei Daten auszulesen und in eine Liste zu speichern. Damit das aber möglich ist, ist eine Klasse der zu speichernden Pilze zu erstellen.



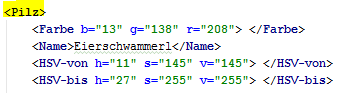
Diese Klasse hat eine *parse()* Methode die wichtig ist. In dieser Methode wird das XML-File eingelesen und anschließend in die Liste gespeichert.



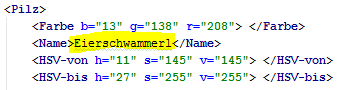
Die Datei wird in der *while()* Schleife solange eingelesen bist *END\_DOCUMENT* erreicht wurde. *END\_DOCUMENT* bedeutet, dass das Ende der Datei erreicht wurde. Eine Hilfsvariable *String tagname* wird erstellt um den eingelesenen *Tag[[1]](#footnote-1)* zu speichern.

Durch ein *switch[[2]](#footnote-2)* werden verschiedene Abfragen bearbeitet:

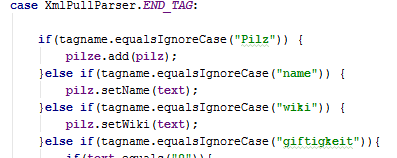
1. Fall eins: Wenn als Starter-Tag ein „Pilz“ gelesen wird, dann wird dieser als ein Pilz erkannt und auch als eine gespeichert.



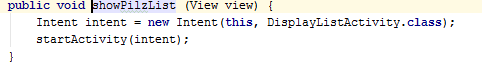
1. Fall zwei: Falls ein Text im *Tag* vorkommt wird dieser eingelesen und gespeichert.



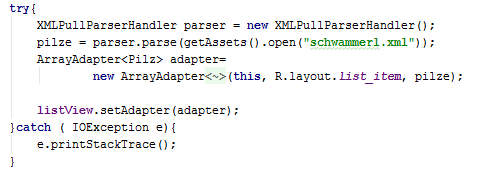
1. Fall drei: Wenn die eingelesene Identifikation dieselbe Name der Objekte in der Klasse erstellen Pilz hat, dann wird der Inhalt gespeichert und in die Liste der Pilze eingefügt.



Auf der Hauptseite der Applikation ist ein *Button* mit dem Namen „Pilzliste“ zu finden, wenn man auf diesen klickt wird die Methode *showPilzList* aufgerufen, welcher auf eine weitere Ansicht weiterleitet und die Pilze in der Liste anzeigt.



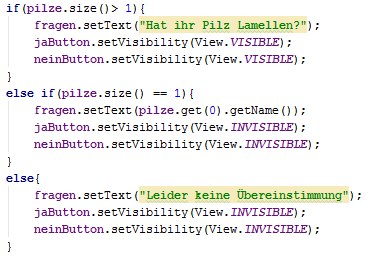
In dieser Ansicht wird die Methode für das Lesen der Pilze aus der XML-Datei aufgerufen und die Liste mit Pilzen befüllt.



Klickt man auf einer der Pilze, wird man wieder auf eine Ansicht mit den Informationen weitergeleitet. Die Eigenschaften werden mit dem Klicken auf den Button mitgeschickt.

#### Benutzerfragen

Sofort nach dem Aufnehmen eines Bildes werden alle Treffer in eine Liste gespeichert. Wenn es mehr als ein Treffer gibt, dann wird dem Benutzer mit Benutzerfragen geholfen.



Die Frage wird gestellt und der Benutzer kann mit „Ja“ oder „Nein“ auf diese antworten.

Je nachdem wie geantwortet wurde, wird die Liste der Pilze laufend aktualisiert bis entweder nur mehr ein Treffer existiert oder garkeines.

#### Probleme

##### Datenverlust bei Vertikal/Horizontaler Rotation

Wenn der Bildschirm vom Smartphone rotiert wird, kommt es standardmäßig zum Datenverlust, was dazu führte, dass zum Beispiel das gespeicherte Bild nicht mehr zu sehen war oder dass Button gefehlt haben.

##### Lösung:

Nach langer Internetrecherche konnte festgestellt werden, dass es zwei Funktionen gibt, die dieses Problem beheben.

**Kurz**: Die Instanz - mit allen Daten – musste gespeichert werden, welches dann später nach einer Rotation wieder aufgerufen wurde.

<http://stackoverflow.com/questions/5179686/restoring-state-of-textview-after-screen-rotation>

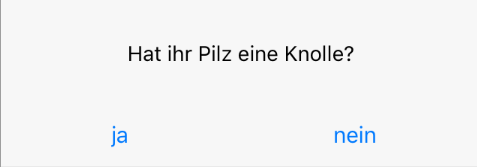


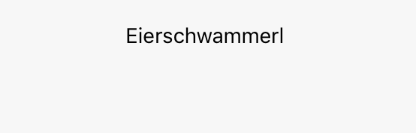
# Qualitätssicherung

## Qualitätsmerkmale

## Testen

Eierschwammerl





## 

## 



## Vergleich mit Konkurrenzprodukten

### Meine Pilze (Pilzbestimmung) Entwickler: Meine Pilze

Abbildung : Meine Pilze App Screenshot

Vorteile:

Eine Fundliste kann gefüllt werden

(in der Kostenpflichtigen Variante) Größere Pilzdatenbank

Zusätzliche Quiz Fragen um sich mit weiterzubilden

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Sehr umständlich Pilze zu bestimmen.

Funktionsweise:

Der mit Mushroom Identifier vergleichbare Teil ist der Punkt „Merkmalsuche“



Abbildung : Pilze App Screenshot

### Pilze Entwickler: Kirill Sidorov

Vorteile:

Größere Pilzdatenbank

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Keine Möglichkeit der Merkmalsuche.

Funktionsweise:

Es wird nur nach Ungenießbar, Giftig und Essbar unterschieden. Entscheidet man sich für eine Kategorie werden Bilder mit den dazugehörigen Namen aufgelistet, entscheidet man sich für einen wird der Wikipediatext zu dem Pilz angezeigt.

### Pilzführer Nature Lexicon

Abbildung : Pilzführer Nature Lexicon App Screenshot

Vorteile:

Größere Pilzdatenbank

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Funktionsweise:

Der Pilz wird durch das Aussehen der Pilze (z. B. Pilze mit Hut / Stiel oder Morcheln / Lorchen) in einer Baumstruktur untergliedert.

### Fazit:

Mushroom Identifier ist die innovativste und am besten Designte App die es für Smartphones gibt. Darüber hinaus sollte die Pilzerkennung am intuitivsten von statten gehen.

Der Einzige erkennbare Nachteil gegenüber den anderen Apps ist die kleinere Pilzdatenbank in Mushroom Identifier. Eine größere ist jedoch nicht Teil dieser Diplomarbeit.

# Zusammenfassung

## Ergebnis

Als Ergebnis sind nun zwei funktional gleiche Apps für Android und IOS entstanden die mithilfe von Computervision, Maschinellem Lernen und Benutzerfragen Pilze erkennen.

Dabei greifen diese Apps beide auf ein C++ Programm zu, das in Windows programmiert wurde. Es wurde also eine plattformunabhängige Standartversion für die Bilderkennung erstellt.

## Resümee

Bilderkennung ist ein schwieriges Unterfangen. Es erfordert sich genau in die „Denkweise“ eines Rechners hineinzuversetzen, denn das „sehen“ eines Computers kann nicht mit dem des menschlichen Auges verglichen werden. Es erfordert intensive Recherche und viel Testen um die besten Algorithmen und Parameter zu finden, um das Beste Ergebnis zu erzielen. Darüber hinaus war es ein schwieriges Unterfangen den C++ Code sowohl auf IOS sowie vor allem für die Android App zu kompilieren, um die gemeinsame Codebasis zu erreichen.

Das Maschinelle Lernen

Bei der Zeitplanung wurde vor allem der Aufwand des kompilieren des C++ Codes für die Android Plattform unterschätzt.

## Aufwandsverteilung

# Literatur und Quellen Verzeichnis

## Abbildungsverzeichnis

## Tabellenverzeichnis

## Literaturverzeichnis

## Abkürzungsverzeichnis

# Im Anhang

Quellen

Cascade Training:

<http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html>

<http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>

Bilder

<https://cdn.xebialabs.com/assets/files/plugins/cmake.jpg>

<https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Visual_Studio_2012_logo_and_wordmark.svg/2000px-Visual_Studio_2012_logo_and_wordmark.svg.png>

<http://logoall.info/uploads/posts/2016-05/0_xcode_logo.jpg>

<http://www.slashslash.info/wp-content/uploads/2013/10/get-boost.png>

<https://upload.wikimedia.org/wikipedia/commons/5/53/OpenCV_Logo_with_text.png>

<https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Android_Studio_icon.svg/1000px-Android_Studio_icon.svg.png>

<http://www.aberger.at/blog/assets/images/abs.jpg>

Abbildungsverzeichnis

[Abbildung 1: HTL - Perg 15](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295675)

[Abbildung 2: Logo Android Studio 22](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295676)

[Abbildung 3: Logo CMake 23](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295677)

[Abbildung 4: Logo Gradle 24](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295678)

[Abbildung 5: Logo XCode 25](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295679)

[Abbildung 6: Logo Visual Studio 25](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295680)

[Abbildung 7: Logo OpenCV 25](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295681)

[Abbildung 8: Farberkennung Eierschwammerl 27](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295682)

[Abbildung 9: Eierschwammerl HSV 29](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295683)

[Abbildung 10: Canny maxVal, minVal 31](#_Toc477295684)

[Abbildung 11Trackbar Kreiserkennung 33](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295685)

[Abbildung 12: Fuchsiger Rötelritterling Kreiserkennung 34](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295686)

[Abbildung 13:Eierschwammerl HSV schwarz weiß Weichzeichnen 34](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295687)

[Abbildung 14: Eierschwammerl HSV schwarz weiß 34](file:///D:\git\Mushroom-Identifier\Mushroom-Identifier\Diplomschrift\Diplomschrift.docx#_Toc477295688)

# Literaturverzeichnis 2

Canny, J. (1986). *A Computational Approach to Edge Detection.* Von http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf abgerufen

Wagner, C. (2005/2006). *Kantenextraktion Klassische Verfahren.* Von http://www.mathematik.uni-ulm.de/stochastik/lehre/ws05\_06/seminar/ausarbeitung\_wagner.pdf abgerufen

1. Eine Art von Identifikation, die auch Informationen erhalten kann [↑](#footnote-ref-1)
2. Mehrere Bedingungen können mit einem *switch* zusammengefasst werden, das erspart mühsames schreiben und ist im generellen übersichtlicher [↑](#footnote-ref-2)