  
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](#_Toc478814131)

[2.1 Schule 9](#_Toc478814132)

[2.2 Schuljahr 9](#_Toc478814133)

[2.3 Klasse 9](#_Toc478814134)

[2.4 Projektname 9](#_Toc478814135)

[2.5 Projektleiter 9](#_Toc478814136)

[2.6 Projektteam 9](#_Toc478814137)

[2.7 Betreuungslehrer 9](#_Toc478814138)

[3 Einleitung 10](#_Toc478814139)

[3.1 Kurzfassung 10](#_Toc478814140)

[3.2 Abstract 11](#_Toc478814141)

[3.3 Motivation 12](#_Toc478814142)

[4 Projektdefinition 12](#_Toc478814143)

[4.1 Aufgabenstellung 12](#_Toc478814144)

[4.2 Geschäftsziele 12](#_Toc478814145)

[4.3 Projektziele 13](#_Toc478814146)

[4.4 Projektumfang 13](#_Toc478814147)

[4.5 Projektstrukturplan 14](#_Toc478814148)

[4.6 Meilensteine 15](#_Toc478814149)

[4.7 Schule 16](#_Toc478814150)

[4.8 Betreuungslehrer 16](#_Toc478814151)

[4.9 Team 17](#_Toc478814152)

[4.9.1 Jakob Froschauer 17](#_Toc478814153)

[4.9.2 Hakan Abbas 18](#_Toc478814154)

[4.9.3 Markus Arbeithuber 20](#_Toc478814155)

[4.10 Organisation 22](#_Toc478814156)

[5 Entstehung und Planung 22](#_Toc478814157)

[5.1 Ideenfindung 22](#_Toc478814158)

[5.2 Zeitplanung 23](#_Toc478814159)

[5.3 IVM – Matrix 24](#_Toc478814160)

[6 Realisierung 25](#_Toc478814161)

[6.1 Anwendungsfälle 25](#_Toc478814162)

[6.2 Funktionen 25](#_Toc478814163)

[6.2.1 Pilzanalyse 25](#_Toc478814164)

[6.2.2 Pilzauflistung 25](#_Toc478814165)

[7 Darstellung 26](#_Toc478814166)

[7.1 iOS 26](#_Toc478814167)

[7.1.1 Pilzanalyse 26](#_Toc478814168)

[7.1.2 Pilzliste 27](#_Toc478814169)

[7.2 Android 28](#_Toc478814170)

[7.2.1 Startseite 28](#_Toc478814171)

[7.2.2 Foto schießen 28](#_Toc478814172)

[7.2.3 Galerie 30](#_Toc478814173)

[7.2.4 Pilzliste 31](#_Toc478814174)

[8 Technologien 32](#_Toc478814175)

[8.1 Java (Android) 32](#_Toc478814176)

[8.2 Swift/Objective-C (IOS) 32](#_Toc478814177)

[8.3 C++ (OPEN CV) 32](#_Toc478814178)

[8.4 Android Studio 32](#_Toc478814179)

[8.5 JNI – Java Native Interface 32](#_Toc478814180)

[8.6 Android Studio NDK 33](#_Toc478814181)

[8.7 CMake 33](#_Toc478814182)

[8.8 Gradle 34](#_Toc478814183)

[8.9 XCode (IOS) 34](#_Toc478814184)

[8.10 Visual Studio (OPEN CV) 35](#_Toc478814185)

[8.11 OpenCV (Open Source Computer Vision) 35](#_Toc478814186)

[8.12 CMARKUP (XML Lesen) 36](#_Toc478814187)

[8.13 Computer Vision 36](#_Toc478814188)

[8.14 Mensch-Computer-Interaction 36](#_Toc478814189)

[8.15 Haar Cascade Training 36](#_Toc478814190)

[8.16 Entscheidung für Native Apps 37](#_Toc478814191)

[9 Programmierung 38](#_Toc478814192)

[9.1 Bilderkennung 38](#_Toc478814193)

[9.1.1 Kurzerklärung der Schritte 38](#_Toc478814194)

[9.1.2 Computer Vision 39](#_Toc478814195)

[9.1.3 CV\_HOUGH\_GRADIENT: 54](#_Toc478814196)

[Datenspeicherung (XML) 57](#_Toc478814197)

[9.1.4 Maschinelles Lernen 60](#_Toc478814198)

[9.1.5 Erstellung einer Plattform-unabhängigen Bilderkennung in C++ 66](#_Toc478814199)

[9.1.6 Tätigkeiten bis zum Endbenutzerprodukt 67](#_Toc478814200)

[9.2 iOS – App 68](#_Toc478814201)

[9.2.1 OpenCV in iOS 68](#_Toc478814202)

[9.2.2 OpenCVWrapper.h 69](#_Toc478814203)

[9.2.3 OpenCVWrapper.mm 69](#_Toc478814204)

[9.2.4 ViewController.swift 73](#_Toc478814205)

[9.2.5 Libjpeg 73](#_Toc478814206)

[9.3 Android – App 75](#_Toc478814207)

[9.3.1 Installation 75](#_Toc478814208)

[9.3.2 OpenCV Source Code in der Android Native Toolchain kompilieren 75](#_Toc478814209)

[9.3.3 OpenCV Bibliotheken in das Projekt einschließen 80](#_Toc478814210)

[9.3.4 Pilz Klasse 81](#_Toc478814211)

[9.3.5 MushroomDetector.java 82](#_Toc478814212)

[9.3.6 MushroomDetector.h 83](#_Toc478814213)

[9.3.7 MushroomDetector.cpp 83](#_Toc478814214)

[9.3.8 JniUtil.h 87](#_Toc478814215)

[9.3.9 JniUtil.cpp 88](#_Toc478814216)

[9.3.10 MainActivity.java 89](#_Toc478814217)

[10 Qualitätssicherung 91](#_Toc478814218)

[10.1 Qualitätsmerkmale 91](#_Toc478814219)

[10.2 Testen 92](#_Toc478814220)

[92](#_Toc478814221)

[92](#_Toc478814222)

[10.3 Vergleich mit Konkurrenzprodukten 93](#_Toc478814223)

[10.3.1 Meine Pilze (Pilzbestimmung) Entwickler: Meine Pilze 93](#_Toc478814224)

[10.3.2 Pilze Entwickler: Kirill Sidorov 94](#_Toc478814225)

[10.3.3 Pilzführer Nature Lexicon 94](#_Toc478814226)

[10.3.4 Fazit: 94](#_Toc478814227)

[11 Zusammenfassung 95](#_Toc478814228)

[11.1 Ergebnis 95](#_Toc478814229)

[11.2 Resümee 96](#_Toc478814230)

[11.3 Aufwandsverteilung 96](#_Toc478814231)

[12 Literatur und Quellen Verzeichnis 97](#_Toc478814232)

[12.1 Abbildungsverzeichnis 97](#_Toc478814233)

[12.2 Tabellenverzeichnis 97](#_Toc478814234)

[12.3 Literaturverzeichnis 97](#_Toc478814235)

[12.4 Abkürzungsverzeichnis 98](#_Toc478814236)

[13 Im Anhang 98](#_Toc478814237)

# 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.

Die dazu nötigen Vergleichsdaten werden lokal gespeichert 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

## 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

Lebenslauf

**Angaben zur Person**

Name: Hakan Abbas

Geboren am: 12.1.1998

Adresse: Eichenweg 10

4332 Au/Donau

Telefon: +43 676 6722933

E-Mail: [Hakanabbas@live.de](mailto:Hakanabbas@live.de)

Staatsangehörigkeit: Deutschland

**Berufliche Erfahrung**

2015

8. Juli – 31. Juli HABAU Hoch- und Tiefbaugesellschaft mbH in Perg

Aushilfe in der IT-Abteilung

3. August – 31. August RACON Software GmbH in Linz

Software-Entwickler

**Schulbildung**

2004 – 2008 Volksschule Naarn

2008 – 2012 Hauptschule Naarn

2012 – jetzt HTL für Informatik in Perg

**Persönliche Fähigkeiten und Kompetenzen**

Muttersprache Türkisch

Sonstige Sprachen

Englisch

Lesen – ausgezeichnet

Schreiben – gut

Sprechen – ausgezeichnet

Deutsch

Lesen – ausgezeichnet

Schreiben – gut

Sprechen – gut

EDV-Kenntnisse

ECDL

Programmieren in Java

Programmieren in C#

Programmieren in TypeScript

App Entwicklung in Swift und Java

Assembler

Bildbearbeitung (GIMP)

Sound und Videobearbeitung (Pinnacle)

Webdesign (HTML, PHP, CSS)

10-Finger-System

### Markus Arbeithuber

|  |  |  |
| --- | --- | --- |
| **Persönliche Daten** | |  |
|  | |  |
| Geburtsdaten: | | Linz, 6. September 1997 |
| Familienstand: | | ledig |
| Religionskenntnis: | | röm.-kath. |
| Staatsbürgerschaft: | | Österreich |
| Eltern: | | Dietmar Arbeithuber, Polizeibeamter  Ingrid Arbeithuber, Büroangestellte |
| Geschwister: | | 1 Bruder |
|  | |  |
|  | |  |
| **Schulbildung** | |  |
|  | |  |
| Derzeit. 5. Jahrgang | | HTL für Informatik in Perg |
| 4 Jahre | | Hauptschule in Naarn |
| 4 Jahre | | Volksschule in Naarn |
|  | |  |
|  | |  |
| **Kenntnisse und Fähigkeiten** | | |
|  | | |
| Sprachen: | Deutsch (Muttersprache)  Englisch (fließend) | |
|  |  | |
| EDV-Kenntnisse: | Programmieren in Java, C#, C, C++, Typescript, Swift, Objective C  MS Office  Bildbearbeitung (GIMP)  Webdesign (HTML, PHP, CSS)  Sound und Videobearbeitung (Pinnacle) | |

|  |  |
| --- | --- |
| Sonstiges: | ECDL  Erste-Hilfe-Kurs  B-Führerschein |

## Organisation

Abbildung 3: Projekthierarchie

# 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

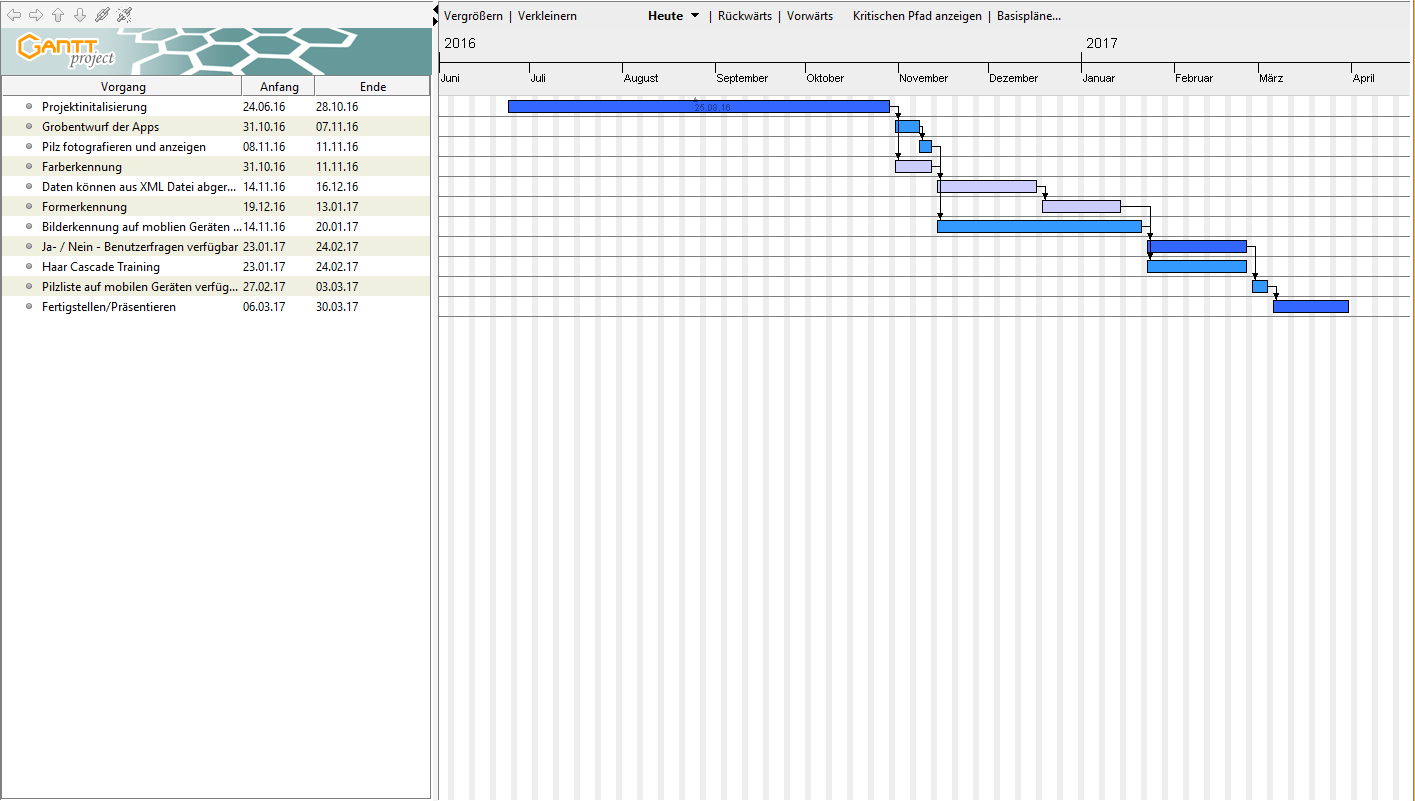


Abbildung 4: Zeitplanung

## 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 | V | V | V | I |

# Realisierung

## Anwendungsfälle



Abbildung 5: Anwendungsfälle

## Funktionen

### Pilzanalyse

### Pilzauflistung

# Darstellung

## iOS

### Pilzanalyse

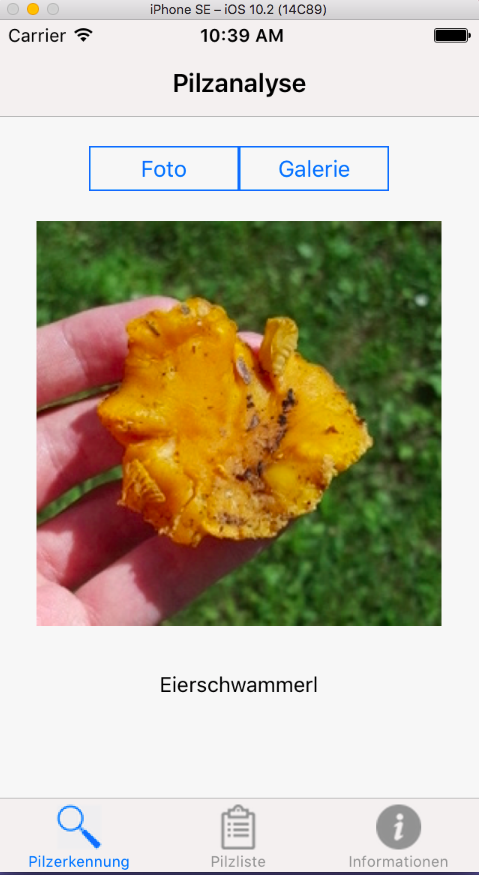


Abbildung : Pilzanalyse Ergebnis in iOS

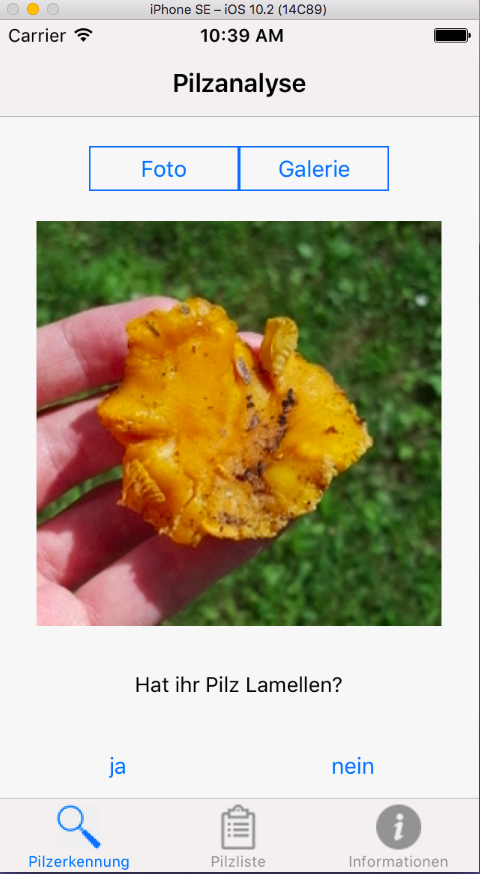


Abbildung : Pilzanalyse mit Benutzerfragen in iOS

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

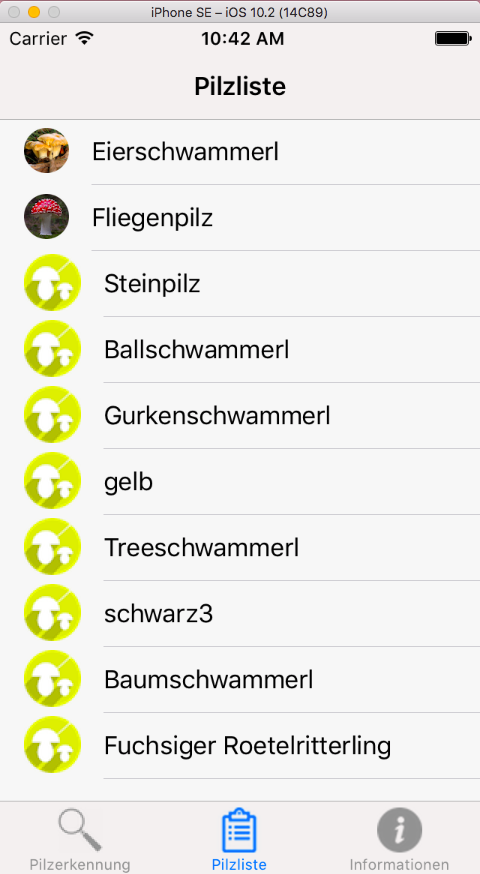
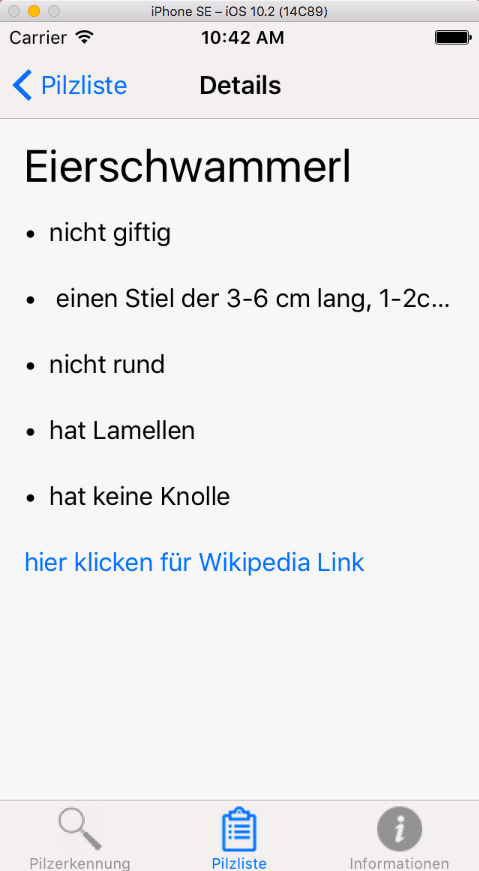


Abbildung : Details zu PIlz iOS

Abbildung : Pilzliste iOS

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

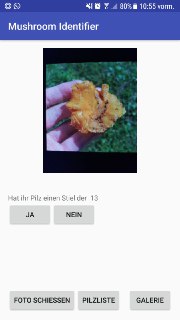
### Startseite



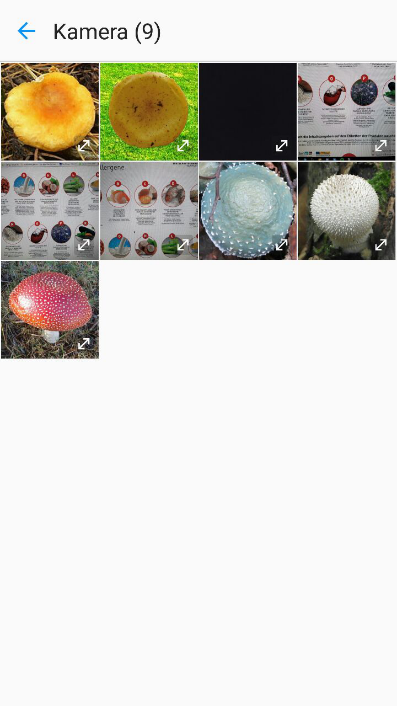
Wenn man die Android App startet, sieht man diese Startseite mit den zwei Buttons: Foto schießen und Galerie.

### 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.

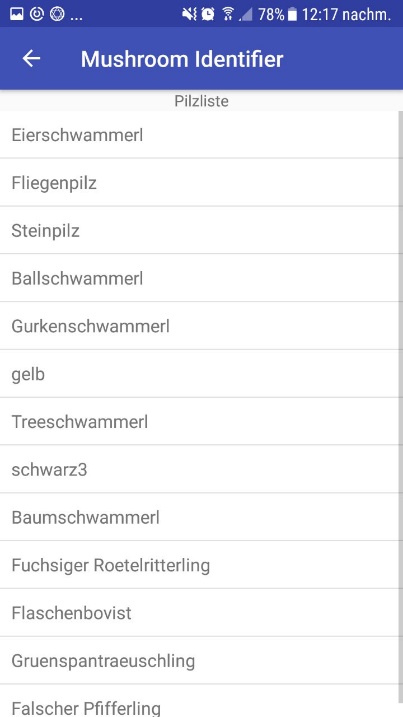
### Galerie



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)

Android Applikationen werden in der Programmiersprache Java geschrieben, da sie eine sehr bekannte Sprache ist. Java ist eine sehr simple Einsteigersprache. Novizen unter den mobilen Entwicklern haben wenig bis keine Schwierigkeiten die Sprache zu erlernen und zu programmieren.

## 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.

## JNI – Java Native Interface

Dieses Programmiergerüst ermöglicht, dass Java Applikationen mit nativen Bibliotheken kommuniziert und Funktionen aufruft und umgekehrt. Kurz gesagt es ermöglicht eine Verbindung zwischen Java und einer fremden Programmiersprache wie zum Beispiel: C oder C++.

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.



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 angebotenen Entwicklungsumgebung für verschiedene Hochsprachen (darunter C, C++, C#, Phyten, HTML, JavaScript und Typescript). Wurde für das implementieren des C++ Bilderkennungs – und Maschinelles Lernen Teil verwendet.

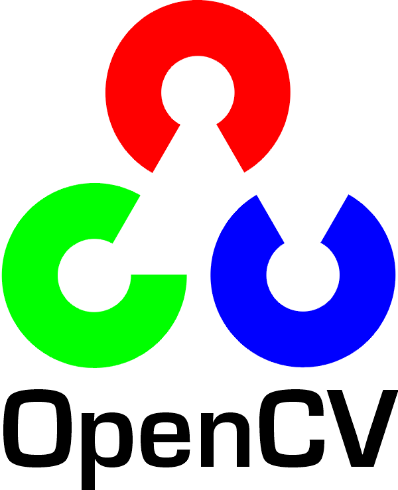


Abbildung : Logo OpenCV

## 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, 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 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.

Code Snippet:

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

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.

Damit sollen alle Kanten, und auch nur diese, gefunden werden.

Die Kante sollte möglichst genau auf der Kante erkannt werden, nicht daneben. Weiters sollen Kanten 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 9.1.2.4.1.2 Gaußscher Blur)

##### 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 |

Bei dem Beispielbild handelt es sich um ein Bild mit einer vertikalen, und keiner Horizontalen Kante. Es sollte als

###### 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 (genau bei drei Pixeln):

(-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 hat 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 18: 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

##### Hough Transformation

Dabei handelt es sich um ein globales Verfahren zur Erkennung von Geraden, Kreisen oder anderen geometrischen Figuren in einem Schwarz-Weiß Bild nach einer Kantendetection.

Dafür wird ein Hough Raum erschaffen. Es werden Bei jedem Punkt, der auf einer Kante liegt, alle möglichen Parameter der zu findenden Figur im Houghraum eingetragen werden. Jeder Punkt in diesem Raum entspricht einem geometrischen Raum im Bild.

Hough Line Transformation

Im Allgemeinen kann die Linie y=k\*x+d als Punkt in diesem Houghraum dargestellt werden, bei vertikalen Linien würden allerdings unendlich viele Werte von K führen. Deshalb wird die Hesse-Normalform angewandt:

r= x\*cos(θ) + y\* sin(θ)

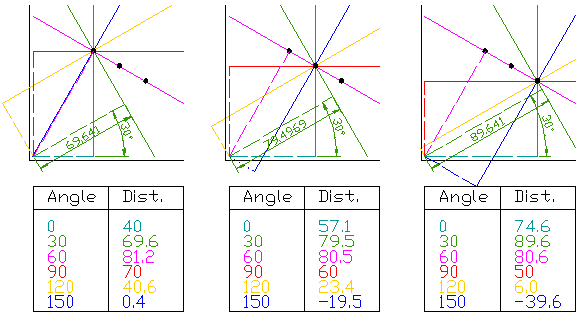
r ist dabei der Abstand vom Ursprung zu einem nächsten Punkt der Geraden.

Und mit dem Ergebnis nach Linien gesucht.

Dabei ist dieser Algorithmus sehr resistent gegen nicht perfekten Kreisen und Bildrauschen. Es wird versucht, einen Kreis zu finden, der mit den zuvor erkannten Kanten am ehesten einem Kreis entspricht.

Funktionsweise:

Dabei wird jeder Punkt der davor gefundenen Kanten durchsucht.



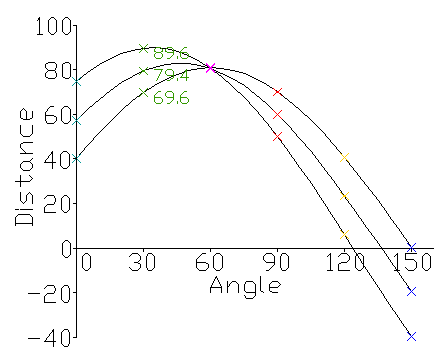
Bei dieser Grafik handelt es sich vereinfacht um eine 3 untersuchte Punkte, die von einem Kantenerkennungsalgorithmus als Kante erkannt worden sind.

Zu jedem Punkt werden durchgezogene Linien in alle Richtungen gezogen, die den Punkt in verschiedene Richtungen scheiden.

Dazu gibt es die geschrichelten Linien, die senkrecht zu den Linien stehen und durch den Bildursprung gehen. Diese Länge gibt die Distanz an.

Der Winkel zwischen der X-Achse und diesen Linien gibt den Winkel an.

Diese Tabellen lassen sich auch als Kurven darstellen:



Je mehr Kurven sich an einem Punkt treffen, desto sicherer handelt es sich an diesem Punkt um einen Gerade.

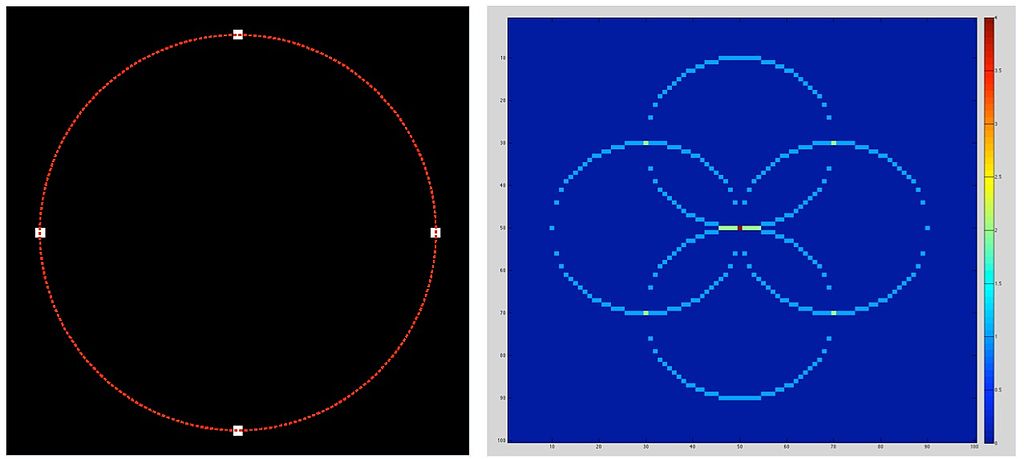
Um diesen Algorithmus für die Kreiserkennung umzuschreiben muss folgendes erledigt werden:

Zuerst wird ein Houghraum mit Nullen befüllt.

Mit dem Ausdruck wird die Mitte des Kreises bestimmt.

r ist dabei der Radius

a und b stellen den die Koordinaten des Kreismittelpunktes dar.



Erkennung mit bekanntem Radius:

Es werden an den erkannten Kanten des Bildes Kreise mit dem Radius gezeichnet. Der Kantenpunkt stellt dabei den Mittelpunkt eines Kreises dar. Ein Kreis wird dann erkannt, wenn sich die Kreise so, wie in dem Blauen Bild zu sehen ist, in der Mitte treffen.

Ist der Radius nicht bekannt wie in unserem Fall, muss dieser Algorithmus in mehreren Iterationen und verschiedenen Radien durchgeführt werden. Die Werte werden dabei in einem Dreidimensionalen Hough Raum gespeichert um die Werte mit mehreren möglichen Kreisradien speichern zu können.

<http://web.eecs.umich.edu/~silvio/teaching/EECS598/papers/Ballard.pdf>

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

##### param\_2:

Schwellenwert für die Mittenerkennung

Entscheidung: 41

##### min\_radius:

Minimaler Kreisradios um erkannt zu werden (0=egal)

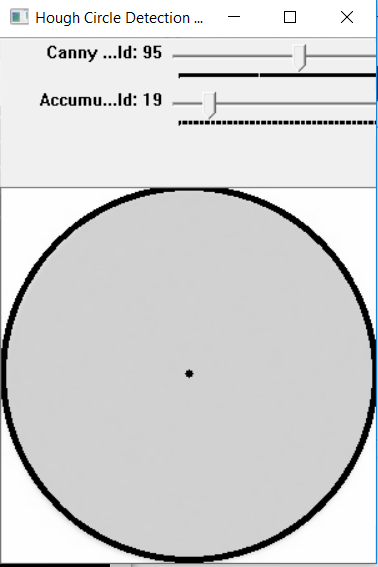


Abbildung : Trackbar Kreiserkennung

##### 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. Dieser 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?

Hat der Pilz eine Knolle?

Diese Fragen kann der Benutzer mit JA oder NEIN beantworten.

### 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

(aardvarkk, 2012)  
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!” (Bradski, 2015)

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.

##### 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.

### 

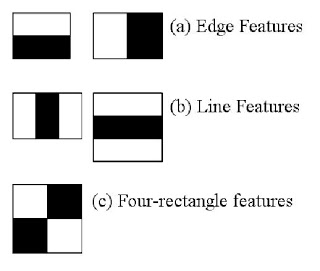
### Maschinelles Lernen

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, wie sie im Link <http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html> beschrieben werden:

#### Training Phase

Die Training Phase besteht daraus, dass positive und negative Bilder, also solche, die erkannt werden sollen, und solche, die nicht erkannt werden sollen, gesucht werden. Anschließend wird mithilfe von maschinellem Lernen aufgrund dieser Bilder eine XML Datei erstellt, über die dann später die Bilderkennung erfolgen kann. Der Algorithmus durchläuft dabei mehrere Phasen, in denen er bestimmte Eigenschaften aus den Bilder herausfiltert, an denen man diese erkennen kann. In der folgenden Grafik werden solche möglichen Eigenschaften beschrieben.



Dabei wird eine Summe von Pixel

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.

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. Es erstellt positive Samples, indem es negative Bilder als Hintergrund und positive als Vordergrund verwendet.

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 40 -h 40"

Parameter:

<http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html#negative-samples>

-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 negativen Samples an

-numStages

beschreibt die Anzahl der durchlaufenen Phasen

je mehr Phasen durchlaufen werden, umso genauer wird das Ergebnis sein, jedoch dauert das Training dann auch dementsprechend länger.

-minHitRate

Minimale Trefferquote pro Phase

-maxFalseAlarmRate

Maximale Falschalarmrate

-numPos

Anzahl der positiven Samples, sollte jedoch niedriger angegeben werden als die tatsächliche Anzahl

-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

Als Ergebnis erhält man eine XML-Datei, aufgrund der dann die Bilderkennung implementiert werden kann.

#### Probleme

##### Geringe Anzahl an positiven und negativen Samples

Grundsätzlich lässt sich sagen: Je mehr Bilder man der Funktion bereitstellt, umso genauer wird die Bilderkennung. Bei der Gesichtserkennung zum Beispiel werden tausende von positiven und negativen Bildern benötigt, um wirklich brauchbare Ergebnisse zu erzielen. Nicht anders sieht das bei Pilzen aus. Sogar für den Menschen ist es schwierig, gewisse Pilze zu unterscheiden. Das Problem hierbei ist, dass es nicht möglich war, genügend Bilder zu finden, da die Pilze von oben fotografiert sein müssen und über das Internet nicht sehr viele brauchbare Aufnahmen zu finden waren. Das heißt, die einzige Möglichkeit um dieses Problem zu lösen, wäre, selbst in den Wald zu gehen und Pilze zu suchen. Das würde jedoch den Rahmen der Diplomarbeit sprengen und somit wurde ein Versuch mit den im Internet gefundenen Pilzen gemacht. Dazu wurden 30 positive und 60 negative Bilder verwendet.

##### Schlechte Ergebnisse aufgrund eines Tippfehlers bei den Parametern

Beim Angeben der Parameter ist ein Tippfehler gemacht worden. Es wurde anstatt von „-numNeg 60“ „numNeg 600“ eingegeben. Am Ergebnis konnte das vorerst nicht gemerkt werden, da damit gerechnet wurde, dass die Bilderkennung nicht zu hundert Prozent funktionieren kann. Es waren durchaus Tendenzen zu erkennen, dass Fliegenpilze erkannt werden und andere nicht. Bei späterer Betrachtung des Aufrufs wurde der Fehler jedoch gefunden und ein weiterer Durchlauf gestartet.

##### Lange Dauer beim Ausführen des Training-Algorithmus

Die erste Durchführung des Trainings-Algorithmus hat über 50 Stunden in Anspruch genommen. Das hatte mehrere Ursachen:

1. Beim Angeben der Parameter ist ein Tippfehler gemacht worden. Es wurde anstatt von „-numNeg 60“ „numNeg 600“ eingegeben.
2. Es wurde eine hohe Anzahl an Training Stages gewählt, nämlich 20. Für erste Tests sollte es jedoch auch reichen, wenn man mit 10 Stages beginnt.
3. Es wurde eine zu hohe Breite und Höhe der Samples gewählt. Anstatt von 50x50 reicht auch 20x20.

In einem zweiten Durchgang, nach Ausbesserung der Parameter, wurde eine deutlich bessere Zeit erreicht, nämlich ca. 20 Stunden.

##### Probleme beim Finden von Testdaten

Aufgrund der Tatsache, dass nicht sehr viele positive und negative Bilder gefunden wurden, war es auch schwierige, zusätzlich noch Bilder zum Testen zu finden. Ein Großteil der gefundenen Bilder wurde bereits für das Training verwendet und daher konnte nicht mehr mit sehr vielen Bildern getestet werden.

### Erstellung einer Plattform-unabhängigen Bilderkennung in C++

Am Anfang wurde die Bilderkennung in Windows implementiert. Da man aber nicht drei verschiedene Versionen für Windows, Android und iOS haben wollte, weil sonst Änderungen immer den dreifachen Aufwand bedeuten würden, hat man sich als Ziel gesetzt, eine Datei zu schreiben, die auf allen drei Plattformen funktioniert. Dazu wurde zunächst versucht, die Bilderkennung in iOS zum Laufen zu bringen.

#### Notwendige Schritte

##### Ändern der Funktionsparameter

In der Windows Version wird das zu analysierende Bild in der main-Funktion vom lokalen Speicher aus eingelesen. Da in den Apps jedoch die Bilder mit der Kamera gemacht und dann im Programm gespeichert werden, macht es mehr Sinn, sie von dort aus an die Bilderkennungsfunktion zu übergeben. Weiters übergibt man noch den Pfad zur XML-Datei für die Bilderkennung und den Pfad zur XML-Datei für das maschinelle Lernen.

##### Ersetzen von wstring durch string

Ein wstring (Wide String) ist eine Liste von wchars. Der Unterschied zwischen einem wchar und einem normalen char ist, dass ein wchar größer als ein normaler char ist und dadurch mehr verschiedene Zeichen abbilden kann. In XCode werden jedoch keine wstrings unterstützt und deshalb müssen die in der Bilderkennung verwendeten wstrings durch gewöhnliche strings ersetzt werden Anschließend müssen noch alle wcouts, also Ausgaben für wstrings in normale couts geändert werden.

##### Auslagern von einigen Funktionen in die jeweilige Plattform

In der Windows Version werden Bilder direkt im Bilderkennungsalgorithmus als Fenster ausgegeben. Da dies aber in allen drei Plattformen anders funktioniert, wird diese Funktion herausgenommen und für Windows, Android und iOS separat gelöst. Dasselbe gilt für die Benutzerfragen.

### 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

### OpenCV in iOS

<https://www.extendi.it/blog/2015/5/22/46-how-to-add-opencv-2-4-11-in-your-ios-project>

Um mit der Entwicklung in iOS zu beginnen, wird zunächst XCode benötigt, also die Entwicklungsumgebung, in der der Code geschrieben wird. Damit OpenCV verwendet werden kann, muss zusätzlich das OpenCV Framework heruntergeladen und ins Projektverzeichnis kopiert werden. Es ist unter folgendem Link zu finden:

<https://sourceforge.net/projects/opencvlibrary/files/opencv-ios/2.4.9/opencv2.framework.zip/download>

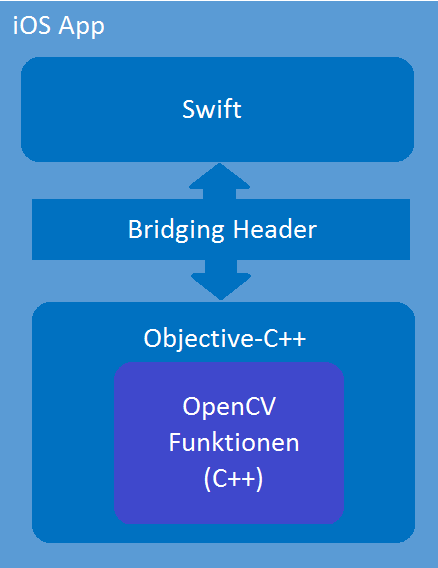
Die App selbst ist in der Programmiersprache Swift geschrieben. In Swift gibt es jedoch keine Möglichkeit, C++ Code zu verwenden. Die OpenCV – Bilderkennung ist aber in C++ geschrieben. Die Lösung für dieses Problem ist Objective C++, wodurch man Objective C und C++ Code verwenden kann.

Abbildung : Verwendung von C++ in iOS

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.

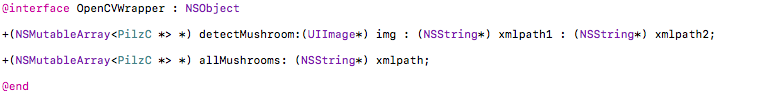


Abbildung 24: Code (OpenCVWrapper.h)

### 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++:**

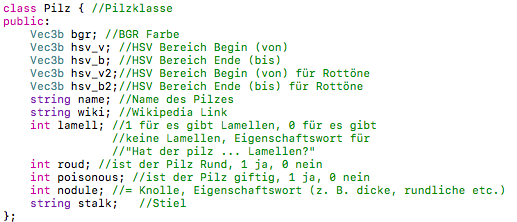


Abbildung 25: Code (Pilzklasse in C++)

**Objective C:**

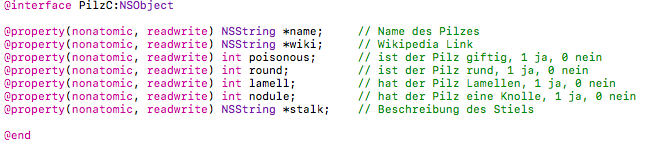


Abbildung 26: Code (Pilzklasse in Objective C)

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.

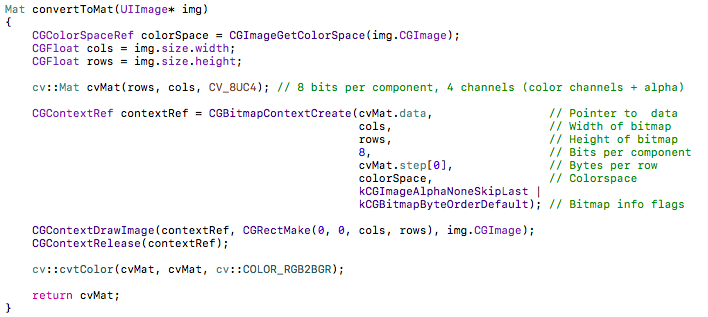


Abbildung 27: Code (convertToMat)

#### detectMushroom

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



Abbildung 28: Code (detectMushroom)

**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.

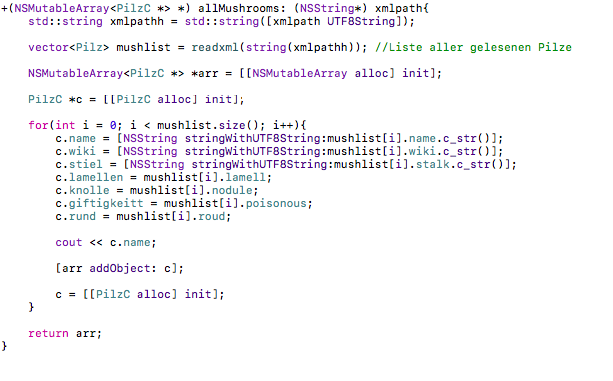


Abbildung 29: Code (allMushrooms)

### ViewController.swift

Im ViewController.swift, also dem Teil, der die Funktionen hinter der grafischen Benutzeroberfläche beinhaltet, kann dann der Pilzanalyse-Algorithmus aufgerufen werden. Als Rückgabewert erhält man eine Liste aller Pilze, die nach Durchlauf dieser Funktion noch in Frage kommen. Diese wird dann von Objective C in Swift konvertiert.

### Libjpeg

Beim Aufruf des Bilderkennungs-Algorithmus treten zuerst einige Fehler auf, die sich alle auf „\_iconv“ beziehen:

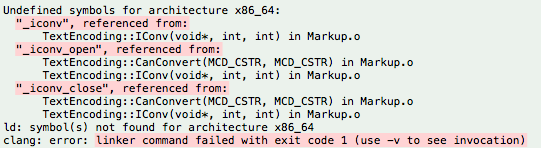


Abbildung : "\_iconv" Fehlermeldung

Um dieses Problem zu lösen, muss „libjpeg-turbo“ installiert werden. Dazu kann man folgenden Link verwenden:

<https://sourceforge.net/projects/libjpeg-turbo/files/1.4.0/>

Anschließend muss man in den Projekteinstellungen unter „Build Settings“ „linking“ „other linker flags“ auf das „+“-Symbol klicken und liconv hinzufügen. Wenn das erledigt ist, sollte die Bilderkennung Kompilieren.

#### Probleme

##### Umwandlung von UIImage in Mat

Probleme traten bei der Umwandlung von UIImage in ein Mat auf, da ein UIImage im klassischen RGB-Farbraum gespeichert ist, ein Mat jedoch im BGR-Farbraum. Daher musste das Ergebnis noch von RGB in BGR umwandeln. OpenCV stellt hierfür die Funktion „COLOR\_RGB2BGR“ zur Verfügung.

## 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/ndk/guides/index.html>

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/>

Zum Schluss wird noch ein Projekt mit C++ Unterstützung erstellt.

Link zum Tutorial:

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

### OpenCV Source Code in der Android Native Toolchain[[1]](#footnote-2) kompilieren

Für die Bilderkennung in Windows werden die OpenCV Bibliotheken benötigt, da diese alle nötigen Analysemethoden enthalten. Daher müssen diese auch in Android Studio importiert werden, da sonst wichtige Funktionen fehlen würden. Die Bibliotheken, welche für Windows erstellt wurden können nicht für Android eingesetzt werden, weil Android Studio seine eigene Regelungen hat. Um eine funktionierende Sammlung von OpenCV Bibliotheken zu haben, muss diese über die Android Native Toolchain konstruiert werden.

Um das zu verwirklichen müssen ein paar Schritte befolgt werden:

Um nicht den Überblick zu verlieren, wird empfohlen ein Ordner im äußersten Projektverzeichnis zu erzeugen. In diesem Ordner werden folgende Dateien erstellt:



Abbildung 31: Skriptverzeichnis

Bevor weitergemacht wird, ist CMake und Python zu installieren.

In der *maketoolchain.cmd* wird der Android Native Toolchain erstellt.



Abbildung 32: maketoolchain.cmd

**Parameter:**

[Hier](https://developer.android.com/ndk/guides/standalone_toolchain.html) werden alle Eigenschaften, welche für die richtige Toolchain benötigt werden, bestens beschrieben.

*ARCH*: Definiert die Systemarchitektur, für das Toolchain.

*NDK* und *STANDALONE\_TOOLCHAIN* werden später beschrieben.

Eine benutzerdefinierte Toolchain wird mittels ….\**make\_standalone\_toolchain.py** erstellt, welches für ein 64-Bit System operiert. Der API-Level wird auf 24 gesetzt, weil sonst standardmäßig das Minimum für die ausgewählte Architektur ausgewählt wird. (21 für ein 64-Bit System)

Der *unified-headers* ermöglicht, dass die libc Header, welche für die native Kodierung benötigt werden, für jede API-Level funktionieren. Am Ende der Kodezeile wird noch mit *install-dir* angegeben, wo man das erstellte Toolchain gespeichert haben will.

Aus diesem [Link](https://github.com/opencv/opencv/releases) wurde der OpenCV Source Code Zip Datei heruntergeladen und in ein eigenes Verzeichnis entpackt.

In der ausführbaren *env.cmd* Datei werden alle benötigten Pfade angegeben:

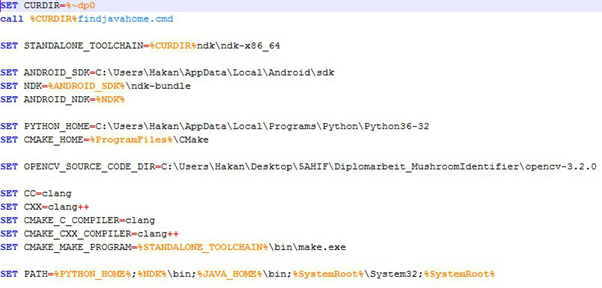


Abbildung 33: Umgebungsvariablen

**Parameter:**

*Curdir:* Gibt das aktuelle Verzeichnis an

In der *call* Aufruf wird - der im aktuellen Verzeichnis vorhandene - *findjavahome.cmd* ausgeführt, welche dann die aktuelle Version vom Java Development Kit kontrolliert und den Pfad dieser in *JAVA\_HOME* speichert.

*STANDALONE\_TOOLCHAIN:* Gibt den Pfad an, wo der erstellte Toolchain gefunden werden kann

*ANDROID\_SDK:* Gibt den Pfad an wo das Entwicklungswerkzeug für Android Applikationen gefunden werden kann

*ANDROID\_NDK:* Pfad für den Stammverzeichnis für die native Android Entwicklung, wo die wichtigsten Komponente für eine Android Applikation mit nativer Kodierung enthalten sind.

*PYTHON\_HOME:* Pfad für das Stammverzeichnis für Python

*CMAKE\_HOME:* Pfad für das Stammverzeichnis für CMake

*OPENCV\_SOURCE\_CODE\_DIR:* Pfad für Stammverzeichnis der heruntergeladenen OpenCV Source Codes

*CC: CLANG -* Gibt den Kompilierer für C Standardbibliotheken

*CXX: CLANG++ -* Gibt den Kompilierer für C und C++ Standardbibliotheken

*CMAKE\_C\_COMPILER:* Gibt den Übersetzer für C Standardbibliotheken über CMake an

*CMAKE\_CXX\_COMPILER:* Gibt den Übersetzer für C und C++ Standardbibliotheken über CMake an

*CMAKE\_MAKE\_PROGRAM:* Über CMake wird der Android Toolchain aufgerufen

*PATH:* Systemumgebungsvariablen werden gesetzt

Um die benutzerdefinierte Android Toolchain zu erbauen, wird der *shell.cmd* Datei ausgeführt, welche eine Kommandozeile in Windows mit den vorher im *env.cmd* definierten Parametern startet.

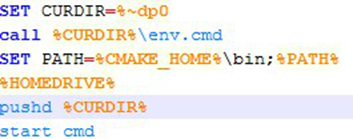


Abbildung 34: Kommandozeile mit definierten Umgebungsvariablen

In der Kommandozeile lassen sich mit *ls* alle Dateien im aktuellen Verzeichnis anzeigen.

*Maketoolchain.cmd* wird ausgeführt um den Toolchain für Android zu erzeugen.

Ein ndk Verzeichnis mit der Toolchain wurde produziert.

Mit der neu erzeugten Toolchain ist es jetzt möglich den OpenCV Source Code für Android Studio zu übersetzten.

Um den Übersetzungsprozess zu starten wird ein *.cmd* Datei erstellt:



Abbildung 35: openCV Bibliothek wird übersetzt

**Parameter:**

*ANDROID\_CMAKE:* Gibt den Pfad für den in den *SKD-Tools* installierte CMake Verzeichnis an

Falls schon ein *build* Verzeichnis im Ordner existiert, wird dieses gelöscht und mit einem neuen leeren ersetzt.

*CMAKE\_PROG:* Pfad der ausführbaren Datei *cmake.exe* wird hier gespeichert

In der nächsten Zeile wird der CMake Befehle mit den folgenden Parametern ausgeführt:

*-G“MinGW Makefiles“*: Das gibt den Generator der CMake an. Der *MinGW Makefiles* Erzeuger ermöglicht das, dass erzeugte *makefile* die Kommandozeile von Windows benutzen. Also ist es nicht mehr nötig eigene Befehlszeilen herunterzuladen.

*-DBUILD\_SHARED\_LIBS=ON:* Gibt an, ob die erzeugten Bibliotheken gemeinsam benutzt werden sollen

*-DCMAKE\_INSTALL\_PREFIX*: Installiert alle erzeugten Bibliotheken die in der angegeben Pfad unter *install zu finden ist*

*‑DANDROID\_ABI:* Ist die gewünschte Systemarchitektur

*-G“Android Gradle – Ninja“:* Gibt den *makefile* Generator für Android an

*-DANDROID\_PLATFORM:* Das gewünschte Android Plattform

*-DCMAKE\_TOOLCHAIN\_FILE:* Der Pfad für den vorher erzeugten Android Toolchain wird angegeben, damit die OpenCV Bibliotheken nicht durch eine Standardmäßig definierte Toolchain gebaut wird.

*-DANDROID\_NDK*: Gibt den Pfad für das NDK an

*-DCMAKE\_MAKE\_PROGRAM:* Gibt den Pfad für das ausführbare Ninja makefile Erzeuger für Android an

*cmake –build . –target install -- -j4:* Im *build* Verzeichnis von der entpackten OpenCV Source Code warden die von *ninja* erzeugten CMakeFiles gespeichert. Und die übersetzten OpenCV Bibliotheken werden alle in der neu generierten *install* Verzeichnis gesichert.

Wenn alle Parameter richtig gesetzt wurden, dann werden nach einer 15-Minütigen Pause alle für die Native Android Programmierung nötigen OpenCV Bibliotheken übersetzt.

Im *OpenCV/install/sdk/native/libs/ABI* Verzeichnis sind alle nötigen erzeugte native Bibliotheken. Damit das Android Projekt diese auch benutzten kann, muss im Projekt unter *app/src/main/* ein Verzeichnis mit dem Namen *jnilibs* erstellt und die Bibliotheken alle hier hinzugefügt werden.

### OpenCV Bibliotheken in das Projekt einschließen

Nun ist der Basis für die Entwicklung einer Android app mit C++ Code gegeben, aber noch bevor wirklich gestartet werden kann muss im Projekt in der *CMakeLists.txt* Datei welche die Projektdateien und die erzeugten OpenCV Bibkiotheken in ein gemeinsames Packet zusammenfügt erstellt werden.

*CMakeLists.txt* wird im Verzeichnis *Projekt/app* erzeugt.

In dieser Datei werden alle vorher in der *env.cmd* definierten Parameter gesetzt, damit CMake auf diese zugreifen kann.

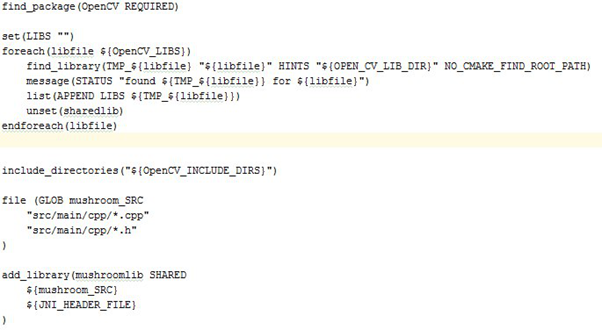


Abbildung 36: CMakeLists.txt in Android

Zuerst wird versucht das kompilierte OpenCV Bibliothek zu finden und wenn die Bibliotheken gefunden wird dann wird über die gesamte Büchersammlung iteriert und alle gefundene Bibliotheken in den *LIBS* Parameter gespeichert. In den vorherigen Schritten wurde OpenCV kompiliert und nach dieser Übersetzung wurden von OpenCV CMake automatisch ein Paar zusätzliche Parameter gesetzt eines von diesen ist *OpenCV\_INCLUDE\_DIRS* diese Variable zeigt auf den *Include* Ordner in OpenCV, welche dann von *CMakeLists.txt* ins Projekt eingeschlossen wird.

In den nächsten Schritten werden alle sich im Projekt unter *src/main/cpp* befindliche C++ Codes und Headers in die Globale Variable *mushroom\_SRC* gespeichert. Dann in der gemeinsamen Bibliothek *mushroomlib* mit der JNI Header File zusammengefasst und gespeichert. Der JNI Header File ist ein Header File der dann die Methode für die Kodierung in nativen C++ und Java angibt.

Im letzten Schritt wird dann automatisch eine Header-Datei für Java erstellt, über die dann in der definierten Methode in Java und C++ entwickelt werden kann.

Die OpenCV Bibliotheken werden dann in der Gesamtsammlung *mushroomlib* wo alle C++ und Header Dateien gespeichert sind zusammengeführt. Dadurch können die Bibliotheken eingeschlossen und die Funktionalitäten benutzt werden.

### Pilz Klasse

**Java**

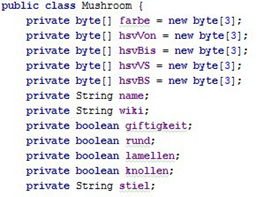


Abbildung 37: Pilzklasse in Java

Das sind die Eigenschaften von einem Pilz, die dann später für das Lesen aus einer XML-Datei und als Übergabeparameter für die Überbrückungsmethode zwischen Java und C++ benötigt wird.

**C++**



Abbildung 38: Pilzklasse in C++

Hier werden die Eigenschaften eines Pilzes in C++ definiert, das wird benötigt, damit in JNI die Umwandlung erfolgen kann.

### MushroomDetector.java

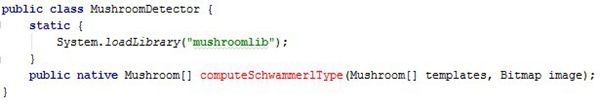


Abbildung 39: Überbrückungsmethode zwischen Java und C++

Diese Klasse dient als die JNI Brücke zwischen Java und C++.

**Parameter**

*Mushroom[] templates:* Ein Feld von Pilzen, die von der XML-Datei eingelesen wurden. Das C++ Quelldatei von Windows analysiert dann mit diesen Pilzen.

*Bitmap image:* Das Bild, welches geschossen oder aus der Galerie entnommen wurde

### MushroomDetector.h



Abbildung 40: Header Datei für die Überbrückungsmethode

Das ist der automatisch generierte Header-Datei, der im vorher beschrieben CMake Schritt gezeigt wurde. Hier wird dem System bekannt gegeben, dass die *computeSchwammerlType* Methode in C++ geschrieben ist.

**Parameter:**

*JNIEnv \*:* Das ist ein Zeiger zu allen gespeicherten JNI Funktionalitäten

jobjectArray*:* Das automatisch erstelle Array zu *Mushroom[] templates*

*jobject:* Das automatisch erstellte jobject zu *Bitmap*

### MushroomDetector.cpp

Das ist die C++ Klasse, welche den Feld von Pilzes von Java in Feld von Pilzen in C++ umwandelt und wieder zurück.

Also von **Mushroom[]** zu **vector<Mushroom>** und wieder zurück.

Zuerst muss aber die C++ Quelldatei eingeschlossen werden, da man sonst nicht die Bilderkennungsmethode aufrufen kann:

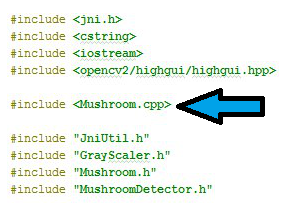


Abbildung 41: Implementierung der Analysedatei

Die Hauptmethode besteht aus zwei verschiedene Bereiche

**Aus Java Pilz ein C++ Pilz**



Abbildung 42: Umwandlung von Java in C++

**Parameter:**

*\*env:* Ein Zeiger zu den JNI Funktionalitäten

*jobject mushroomDetector:* Automatisch erstelltes Objekt zu *mushroomDetector.java*

*jobjectArray templates:* Die Felder aller Pilze, welche in Java übergeben werden sind hier gespeichert

*jobject image:* Das Bild, welche in Java übergeben wird ist hier gespeichert

In diesem Bereich wird das Feld von Pilzen in einer Schleife iteriert und jeder einzelner Pilz wird in der *fromJavaObject()* Methode in ein C++ Pilz umgewandelt und in ein *vector<Mushroom>* gespeichert.

**Aus C++ Pilz ein Java Pilz**

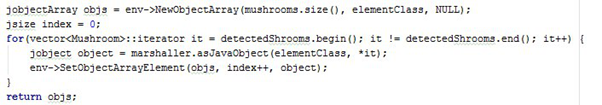


Abbildung 43: Umwandlung von C++ in Java

Ein neues *jobjectArray* mit der Länge von den zurückgegebenen Vektoren von Pilzen wird erstellt. Anschließend wird durch dieses Feld in einer Schleife iteriert und die einzelnen Pilze werden in *jobjectArray* gespeichert und anschließend für die App als ein normales *Mushroom[]* zurückgegeben.

#### MushroomMarshaller

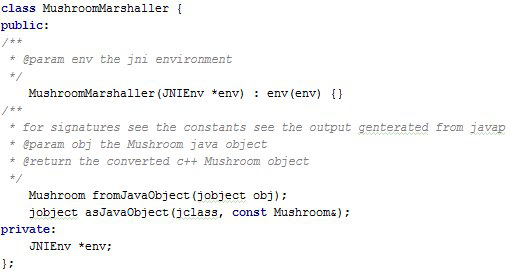


Abbildung 44: Klasse, welche die Umwandlungsmethoden definiert

Diese Klasse in *MushroomDetector.cpp* definiert die Methoden für das Umwandeln von Java Objekten.

##### fromJavaObject

In Dieser Methode wird das *Mushroom*[] Array von Java in einzelne *Mushroom*s für das C++ Quelldatei umgewandelt

Die Variable *mushroom* ist ein Platzhalter wo die einzelnen eingelesenen Pilze gespeichert und zurückgegeben werden.

Die *util(env)* Variable beinhaltet alle Umwandlungsmethoden für die einzelnen Eigenschaften der Pilze.

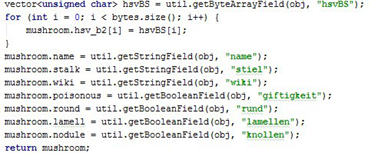


Abbildung 45: Umwandlung einzelner Eigenschaften von Java in C++

Die einzelnen Methoden die aufgerufen werden sind alle in der *JniUtil* Klasse definiert, als Parameter übergibt man das Objekt wo der Pilz gespeichert ist und die *Namen* der einzelnen Eigenschaften der Pilze, welche in der *Mushroom.java* Klasse definiert wurde.

Die genaue Funktionalität, wie die Umwandlung von den einzelnen Eigenschaften funktioniert wird später in der *JniUtil* Klasse beschrieben.

**Parameter:**

*jobject obj:*  Das ist das übergebene *Mushroom*[] Feld von Java

##### AsJavaObject

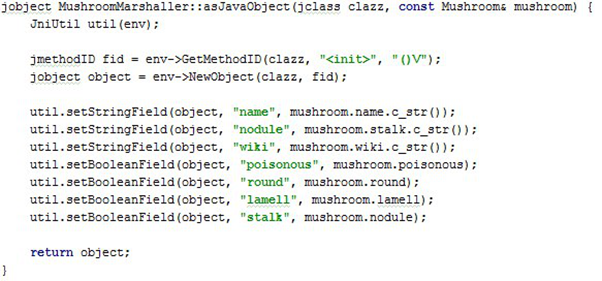


Abbildung 46: Umwandlungs einzelner Eigenschaften von C++ in Java

Diese Methode dient zur Rückumwandlung von einzelnen Pilzen vom C++ Code in Java.

Hier erfolgt die Rückumwandlung von C++ in Java Pilzen. Die einzelnen Methoden, welche aufgerufen werden benötigen dieses Mal die Namen der Eigenschaften der Pilze aus der Seite von C++ Quelldatei aus, da von C++ in Java umgewandelt wird. Zusätzlich wird der *object*, wo der umgewandelte Pilz gespeichert wird noch der zu speichernde Wert der Eigenschaften mitgegeben.

### JniUtil.h

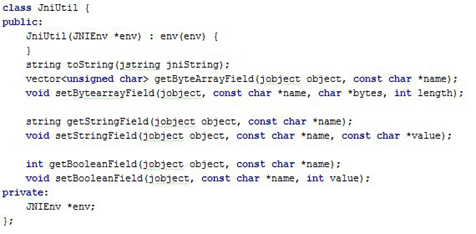


Abbildung 47: Definierung der Methoden für die Umwandlung einzelner Eigenschaften

Die Methoden für die Umwandlung einzelner Pilzeigenschaften werden hier definiert.

### JniUtil.cpp

In dieser C++ Datei werden alle – in der JniUtil.h - definierten Methoden geschrieben

**getByteArrayField()**

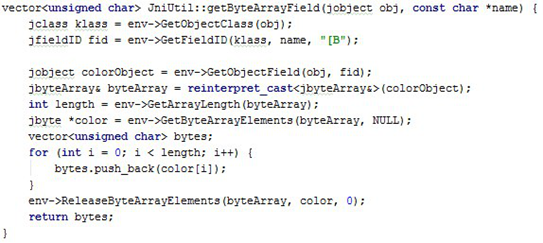


Abbildung 48: Java byte[] in C++ vec3b

In dieser Methode wird die Umwandlung von Java byte Array in C++ byte Array durchgeführt. Aus dem Java Pilz Objekt wird die genaue Eigenschaft mit der *Name* und mit der Signatur – also die Identifikation - der Datentyp für die jeweilige Eigenschaft der Pilz gesucht und gespeichert. Dann erfolgt die Umwandlung von Java zu C++ und der umgewandelte byte Feld wird zurückgegeben.

**Set/GetStringField()**

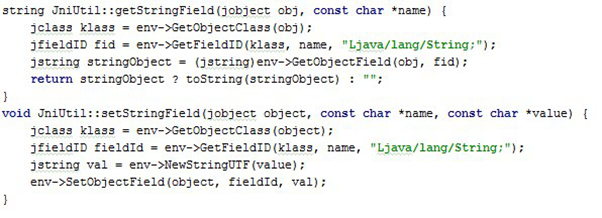


Abbildung 49: Methoden für Umwandlung von Textvariablen:

In diesen Methoden erfolgt die Umwandlung jeweils von Java String zu C++ String und wieder umgekehrt. Dazu wird wieder die Signatur und der Objekt welcher mitgegeben wird benötigt.

**Set/GetBooleanField()**

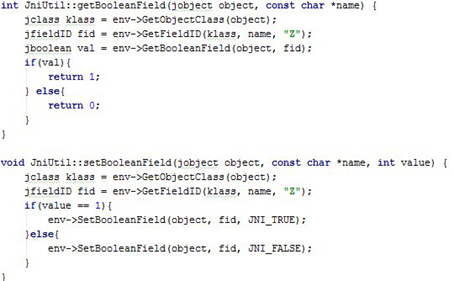


Abbildung 50: Methoden für Umwandlung von boolean Variablen

In diesen Methoden erfolgt die Umwandlung von Java Boolean zu C++ Integer und umgekehrt. Da manche Eigenschaften in C++ in Integer gespeichert sind, weil sie sonst nicht analysiert werden können, wird hier der boolean Wert in Java eingelesen aber ein einfach 0 und 1 in C++ gespeichert und umgekehrt.

### MainActivity.java

In dieser Klasse erfolgen die allgemeinen Funktionen, zum Aufbauen der App. Wenn ein Foto geschossen oder von der Galerie übernommen wird, wird dieser in ein *Bitmap* gespeichert und die JNI-Bridge-Methode – also *computeSchwammerlType* – aufgerufen und das Feld von Pilzen und der gespeicherte Bild übergeben. Das Ergebnis wird dann gespeichert und je nachdem ob das Ergebnis eindeutig war oder nicht, werden Fragen zur Pilz gestellt oder das analysierte Pilz sofort angezeigt.

#### Probleme

##### Standardbibliotheken von Android unterstützen wichtige native Funktionalitäten nicht

Die Analysemethode in der C++ Quelldatei hat die Pilze gleich von der XML-Datei gespeichert und die Funktionen, die hier verwendet wurden, werden in Android Studio nicht unterstützt, was zu Übersetzungsfehler führte. Ganz spezifisch ist es die Funktionalität *stoi() gewesen*.

Diese Funktion wandelt ein einfaches *String* Text in ein *Integer*.

##### Lösung:

Unser Betreuungslehrer Herr Professor Aberger, hat vorgeschlagen die Methode, welches die Pilze aus der XML-Datei einliest, in eine eigene C++ Datei zu speichern.

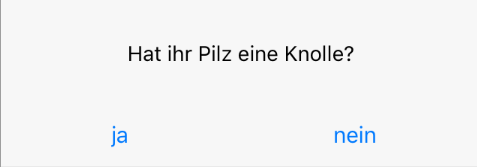
Die OpenCV Windows Datei und iOS greifen auf diese Datei zusätzlich zu, während Android eine eigene Umwandlungsmethode für das Lesen der Pilze schreibt.

# Qualitätssicherung

## Qualitätsmerkmale

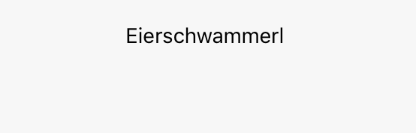
## Testen

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

aardvarkk, (. (19. 01 2012). *stackoverflow.com*. Von http://stackoverflow.com/questions/8932893/accessing-certain-pixel-rgb-value-in-opencv abgerufen

Alessani, M. (22. 5 2015). *www.extendi.it*. Von https://www.extendi.it/blog/2015/5/22/46-how-to-add-opencv-2-4-11-in-your-ios-project abgerufen

Alexander Alekhin. (23. Dezember 2016). *GitHub*. Von https://github.com/opencv/opencv/releases abgerufen

Ball, T. (22. 7 2013). *coding-robin.de*. Von http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html abgerufen

Ballard, D. H. (23. 09 1980). *www.web.eecs.umich.edu.* Von http://web.eecs.umich.edu/~silvio/teaching/EECS598/papers/Ballard.pdf abgerufen

Bradski, D. G. (27. 9 2015). Why does OpenCV use BGR color format ? (S. Mallick, Interviewer) Von https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/ abgerufen

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

Doxygen; Dimitri van Heesch. (14. 03 2017). *docs.opencv.org.* Von Canny Edge Detection: http://docs.opencv.org/trunk/da/d22/tutorial\_py\_canny.html abgerufen

Emami, S. (04. 10 2010). Viewing OpenCV's HSV color space:.

Google . (2. März 2017). *Developer Android*. Von https://developer.android.com/studio/index.html?gclid=CMKrwYyFvdICFVEz0wodCWwB2w abgerufen

Google. (27. März 2017). *Developer Android*. Von https://developer.android.com/ndk/guides/index.html abgerufen

Google. (27. März 2017). *Developer Android*. Von https://developer.android.com/studio/projects/add-native-code.html abgerufen

Google. (27. März 2017). *Developer Android*. Von https://developer.android.com/ndk/guides/standalone\_toolchain.html abgerufen

Hughes, K. (kein Datum). *github.com*. Von https://github.com/mrnugget/opencv-haar-classifier-training abgerufen

Microsoft. (27. März 2017). *Visual Studio*. Von https://www.visualstudio.com/de/downloads/ abgerufen

opencv dev team. (25. 3 2017). *docs.opencv.org*. Von http://docs.opencv.org/2.4/doc/user\_guide/ug\_traincascade.html#negative-samples abgerufen

Rhody, H. (11. 10 2005). *www.cis.rit.edu.* Von https://www.cis.rit.edu/class/simg782/lectures/lecture\_10/lec782\_05\_10.pdf abgerufen

Slashdot Media. (2017). *sourceforge.net*. Von https://sourceforge.net/projects/opencvlibrary/files/opencv-ios/2.4.9/opencv2.framework.zip/download abgerufen

Slashdot Media. (2017). *sourceforge.net*. Von https://sourceforge.net/projects/libjpeg-turbo/files/1.4.0/ abgerufen

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

## 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>

<http://opencv-python-tutroals.readthedocs.io/en/latest/_images/haar_features.jpg>

Abbildungsverzeichnis

[Abbildung 1: HTL - Perg 16](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380011)

[Abbildung 2: Porträt Christian Aberger 16](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380012)

[Abbildung 3: Projekthierarchie 22](#_Toc478380013)

[Abbildung 4: Zeitplanung 23](#_Toc478380014)

[Abbildung 5: Anwendungsfälle 25](#_Toc478380015)

[Abbildung 6: Pilzanalyse Ergebnis in iOS 26](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380016)

[Abbildung 7: Pilzanalyse mit Benutzerfragen in iOS 26](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380017)

[Abbildung 8: Details zu PIlz iOS 27](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380018)

[Abbildung 9: Pilzliste iOS 27](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380019)

[Abbildung 10: Logo Android Studio 32](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380020)

[Abbildung 11: Logo CMake 33](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380021)

[Abbildung 12: Logo Gradle 34](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380022)

[Abbildung 13: Logo XCode 34](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380023)

[Abbildung 14: Logo Visual Studio 35](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380024)

[Abbildung 15: Logo OpenCV 35](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380025)

[Abbildung 16: Farberkennung Eierschwammerl 39](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380026)

[Abbildung 17: Eierschwammerl HSV 41](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380027)

[Abbildung 18: Canny maxVal, minVal 50](#_Toc478380028)

[Abbildung 19: Trackbar Kreiserkennung 55](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380029)

[Abbildung 20: Fuchsiger Rötelritterling Kreiserkennung 55](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380030)

[Abbildung 21:Eierschwammerl HSV schwarz weiß Weichzeichnen 55](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380031)

[Abbildung 22: Eierschwammerl HSV schwarz weiß 55](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380032)

[Abbildung 23: Verwendung von C++ in iOS 68](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380033)

[Abbildung 24: Code (OpenCVWrapper.h) 69](#_Toc478380034)

[Abbildung 25: Code (Pilzklasse in C++) 70](#_Toc478380035)

[Abbildung 26: Code (Pilzklasse in Objective C) 70](#_Toc478380036)

[Abbildung 27: Code (convertToMat) 71](#_Toc478380037)

[Abbildung 28: Code (detectMushroom) 71](#_Toc478380038)

[Abbildung 29: Code (allMushrooms) 72](#_Toc478380039)

[Abbildung 30: "\_iconv" Fehlermeldung 73](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380040)

[Abbildung 31: Skriptverzeichnis 76](#_Toc478380041)

[Abbildung 32: maketoolchain.cmd 76](#_Toc478380042)

[Abbildung 33: Umgebungsvariablen 77](#_Toc478380043)

[Abbildung 34: Kommandozeile mit definierten Umgebungsvariablen 78](#_Toc478380044)

[Abbildung 35: openCV Bibliothek wird übersetzt 78](#_Toc478380045)

[Abbildung 36: CMakeLists.txt in Android 80](#_Toc478380046)

[Abbildung 37: Pilzklasse in Java 81](#_Toc478380047)

[Abbildung 38: Pilzklasse in C++ 82](#_Toc478380048)

[Abbildung 39: Überbrückungsmethode zwischen Java und C++ 82](#_Toc478380049)

[Abbildung 40: Header Datei für die Überbrückungsmethode 83](#_Toc478380050)

[Abbildung 41: Implementierung der Analysedatei 84](#_Toc478380051)

[Abbildung 42: Umwandlung von Java in C++ 84](#_Toc478380052)

[Abbildung 43: Umwandlung von C++ in Java 85](#_Toc478380053)

[Abbildung 44: Klasse, welche die Umwandlungsmethoden definiert 85](#_Toc478380054)

[Abbildung 45: Umwandlung einzelner Eigenschaften von Java in C++ 86](#_Toc478380055)

[Abbildung 46: Umwandlungs einzelner Eigenschaften von C++ in Java 87](#_Toc478380056)

[Abbildung 47: Definierung der Methoden für die Umwandlung einzelner Eigenschaften 87](#_Toc478380057)

[Abbildung 48: Java byte[] in C++ vec3b 88](#_Toc478380058)

[Abbildung 49: Methoden für Umwandlung von Textvariablen: 88](#_Toc478380059)

[Abbildung 50: Methoden für Umwandlung von boolean Variablen 89](#_Toc478380060)

[Abbildung 51: Meine Pilze App Screenshot 93](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380061)

[Abbildung 52: Pilze App Screenshot 94](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380062)

[Abbildung 53: Pilzführer Nature Lexicon App Screenshot 94](file:///C:\Users\Hakan\Downloads\Diplomschrift.docx#_Toc478380063)

1. Ein Toolchain ist eine Sammlung von Werkzeugen, die ermöglicht ein System zu erzeugen, das unabhängig von seiner Ursprung auf der jeweiligen Entwicklungsumgebung funktioniert. [↑](#footnote-ref-2)