HTL - Perg

Höhere Abteilung für Informatik

Diplomarbeit

Mushroom Identifier

Projektteam: Hakan Abbas

Markus Arbeithuber

Jakob Froschauer

Projektbetreuer: Prof. Dipl.-Ing. Christian Aberger

Bearbeitungszeitraum: 01.10.2016 – 30.04.2017

HTL Perg | Machlandstr. 48 | A-4320 Perg | Tel.: +43 (0) 7262 53926

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 unterstützten und uns zur Seite standen.

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

Herzlichen Dank!

Inhalt

1 Impressum

1.1 Schule

HTBLA Perg für Informatik

Machlandstraße 48

4320 Perg

1.2 Schuljahr

2016/2017

1.3 Klasse

5AHIF

1.4 Projektname

Mushroom Identifier

1.5 Projektleiter

Jakob Froschauer

1.6 Projektteam

Hakan Abbas

Markus Arbeithuber

Jakob Froschauer

1.7 Betreuungslehrer

Dipl.-Ing. Christian Aberger

2 Einleitung

2.1 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 Pilzsuche im Wald begegnet man oft Exemplaren, die man schwer ohne unhandliche Pilz Lexika erkennen kann. Selbst mit diesen Büchern ist es schwierig, in vertretbarer Zeit den gefundenen 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 die Farbe und die Form des Pilzes erkannt.

Die dazu nötigen Vergleichsdaten werden lokal gespeichert, um das Problem des schlechten Internetempfangs 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.

2.2 Abstract

The diploma thesis Mushroom Identifier was created during the fifth school year by Hakan Abbas, Markus Arbeithuber and Jakob Froschauer for the diploma exam at the College of Engineering specialised in information technology in Perg. The mobile application represents the future of a Mushroom dictionary. Whilst searching for mushroom in the forest you often encounter different kind of specimen, which can be difficult to recognize without the help of a mushroom lexicon. Even with these books it's hard to identify the found mushroom in a reasonable time. The application Mushroom Identifier shall make this a problem of the past. To the functions: The application user is asked to pick a picture of the given mushroom from the bird's eye view or take a whole new one. Following that, the colour and the form of the mushroom is identified.

The needed data – for comparison – is stored locally on the smartphone to avoid the bad internet connection when walking through a forest. With every identified property, the number of mushrooms in question decreases. If at the end of the image recognition, no mushroom was recognized, the differences of the mushrooms in question are questioned by simple Yes / No user questions. Moreover, it is also determined by machine learning whether it can be a mushroom at all.

2.3 Motivation

Wir sind begeisterte Pilzsammler, jedoch trauten wir uns bisher nur beim Eierschwammerl zuzugreifen, da der Identifikationsprozess von Pilzen bisher ausgesprochen aufwendig war. Außerdem mussten schwere Pilzlexika mitgeschleppt werden. Dieser aufwendige Prozess soll vereinfacht werden. Ein einfaches Foto mit dem Smartphone und eventuell ein paar JA/NEIN Fragen sollen reichen, um Pilze eindeutig zu identifizieren.
Darüber hinaus wird durch das Einsetzen von modernen Technologien ein jüngeres Publikum angesprochen, das sich am Pilzsammeln begeistern kann.

3 Projektdefinition

3.1 Geschäftsziele

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

- 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

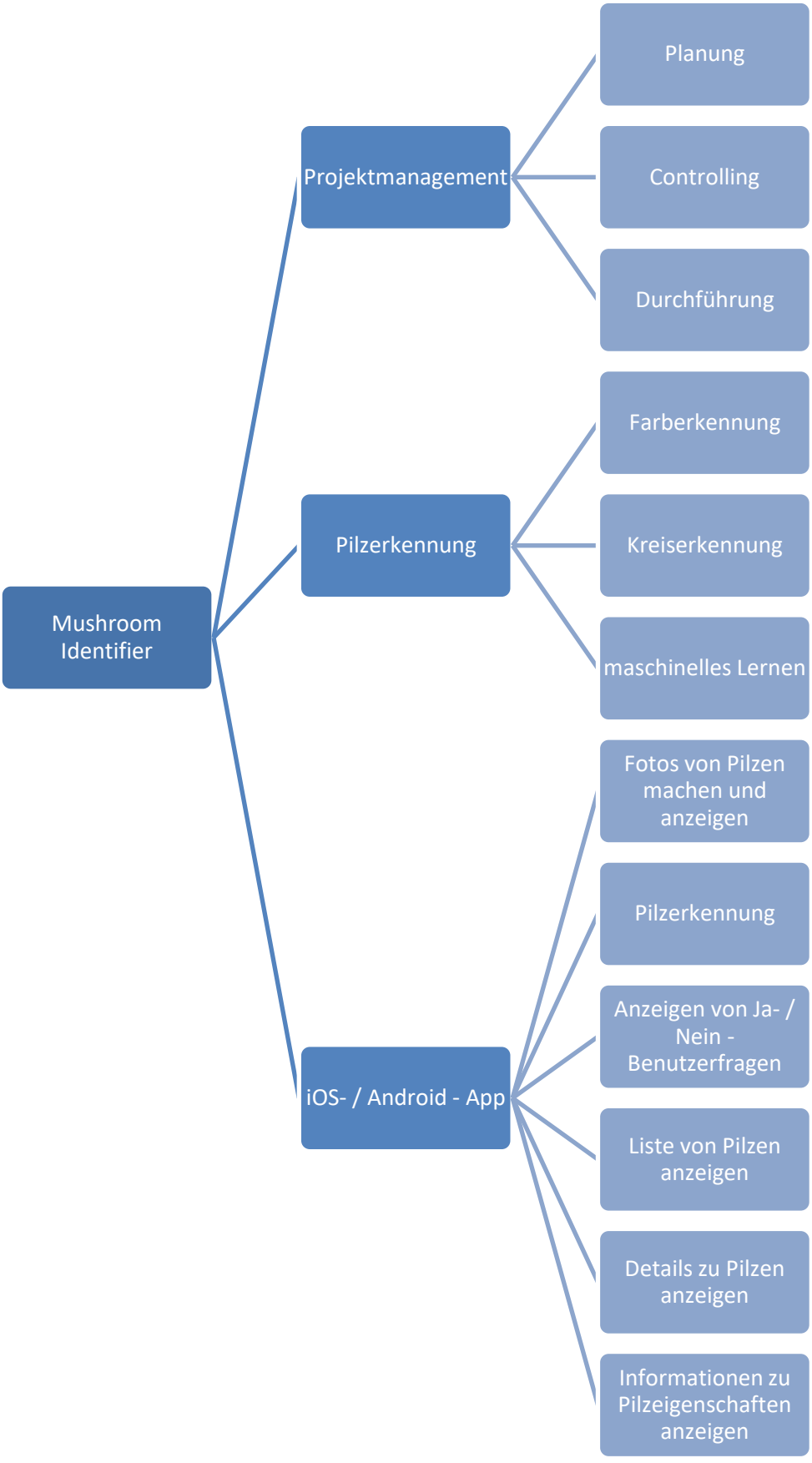
3.2 Projektziele

Es soll eine mobile App entwickelt werden, die Pilzsammlern dabei hilft, Pilze zu identifizieren. Es sollen Fotos von 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 einem Entscheidungsbaum zu einem Ergebnis zu kommen.
Dabei soll es eine Codebasis in C++ geben, auf die sowohl eine native Android App als auch eine native IOS App zugreift. (siehe 7.16 Entscheidung für Native Apps)

3.3 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 als auch der Android Teil zugreifen. Wenn Änderungen am Bilderkennungsalgorithmus durchgeführt werden müssen, sind die Änderungen gleichzeitig auf allen Plattformen, die darauf zugreifen, verfügbar.

3.4 Projektstrukturplan



3.5 Meilensteine

Meilenstein		Andoid	Windows	iOS
Projektstart erfolgt	12.06.2016	12.06.2016	12.06.2016	12.06.2016
Projektinitialisierung abgeschlossen	30.10.2016	28.10.2016	28.10.2016	28.10.2016
Pilz fotografieren ist möglich	13.11.2016	11.08.2016	-	20.08.2016
Pilzfarbe erkennen ist möglich	13.11.2016	-	13.08.2016	-
Daten können von XML File abgerufen werden	18.12.2016	22.02.2017	10.11.2016	-
Form erkennen ist möglich	15.01.2017	-	16.01.2017	-
Bilderkennung auch auf mobilen Geräten möglich	22.01.2017	-	-	10.01.2017
Informationen zum Pilz können vom Benutzer abgefragt werden + Ergebnis wird angezeigt	26.02.2017	04.02.2017	-	15.02.2017
Diplomarbeit fertiggestellt	26.03.2017	04.04.2017	04.04.2017	04.04.2017

3.6 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

Abbildung 1: HTL - Perg

Schulkennzahl: 411457

3.7 Betreuungslehrer

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

Abbildung 2: Dipl.-Ing. Christian Aberger

Kontakt

Dipl.-Ing. Christian Aberger

Softwarepark 37

4232 Hagenberg im Mühlkreis

Tel. 07236 33514200

3.8 Team

3.8.1 Jakob Froschauer

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:	Deutsch (Muttersprache)
	Englisch (fließend)
EDV-Kenntnisse:	Programmieren in Java, C#, C, C++, Typescript, Swift
Sonstiges:	16-stündiger Erste-Hilfe-Kurs
	Führerschein (Klasse B)
	Mitglied des Musikvereins in Naarn (Trompete)
	Mitarbeiter der Bücherei Naarn
	Theatergruppe Naarn

3.8.2 Hakan Abbas

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
Staatsangehörigkeit:	Deutschland

Schulbildung

2004 – 2008	Volksschule Naarn
2008 – 2012	Hauptschule Naarn
2012 – jetzt	HTL für Informatik in Perg

Persönliche Fähigkeiten und Kompetenzen

Sprachen:	Türkisch (Muttersprache)
	Englisch (fließend)
	Deutsch (fließend)
EDV-Kenntnisse:	
	ECDL
	Programmieren in Java, C#, TypeScript
	App Entwicklung in Swift und Java
	Bildbearbeitung (GIMP)
	Sound und Videobearbeitung (Pinnacle)
	Webdesign (HTML, PHP, CSS)
	10-Finger-System
	MS Office

3.8.3 Markus Arbeithuber

Persönliche Daten

Geburtsdaten:	Linz, 6. September 1997
Staatsbürgerschaft:	Österreich
Religionsbekenntnis:	röm.-kath.
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
	Bild- und Videobearbeitung (GIMP)
	Webdesign (HTML, PHP, CSS)
	ECDL
Sonstiges:	Erste-Hilfe-Kurs
	B-Führerschein

3.9 Organisation

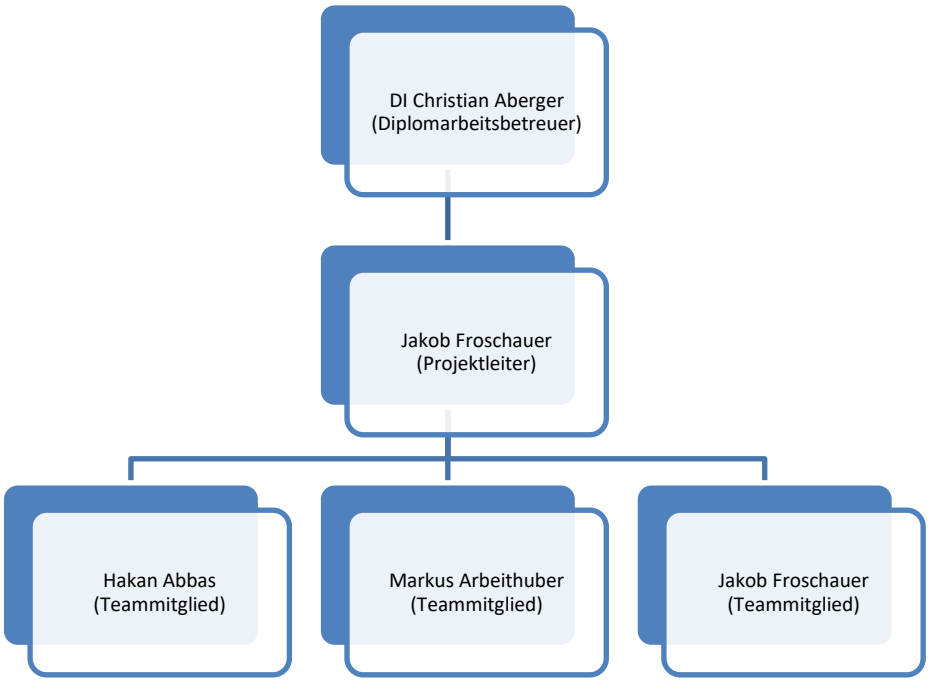


Abbildung 3: Projekthierarchie

4 Entstehung und Planung

4.1 Ideenfindung

Die Idee zu Mushroom Identifier geht auf ein Brainstorming im Projektteam am Ende der 4. Klasse zurück. Dabei hat man 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 „Schwammerlerkennungs-App“ wesentlich mehr interessieren würde. Diese Idee begeisterte sowohl das Projektteam als auch Herrn Direktor Reisinger. Das Diplomarbeitsthema war gefunden.

4.2 Zeitplanung

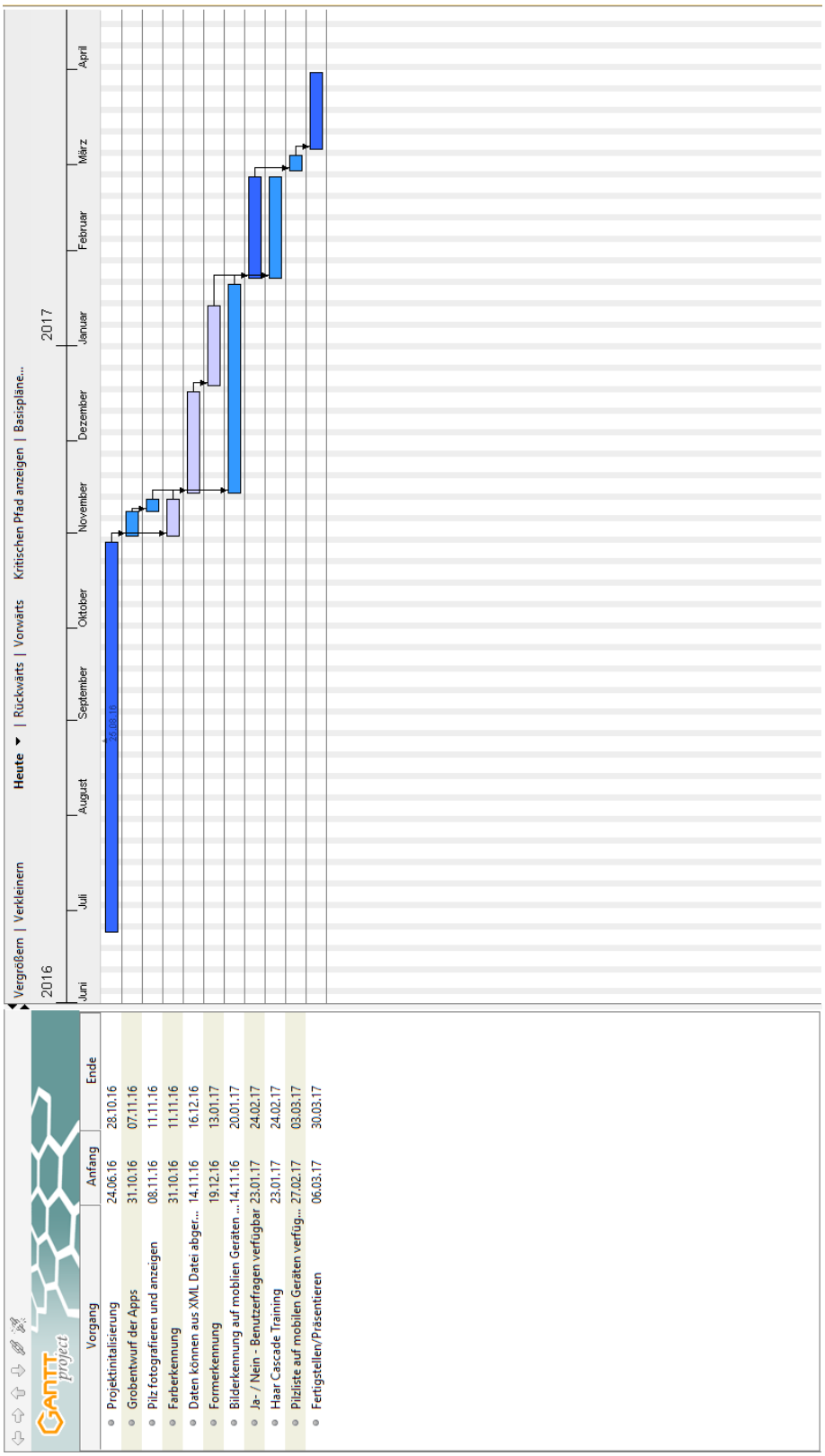


Abbildung 4: Zeitplanung

4.3 IVM – Matrix

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

	Hakan Abbas	Markus Arbeithuber	Jakob Froschauer	Dipl.-Ing Christian Aberger
OpenCV/C++	I	I	V	I
Android App	V	I	I	I
IOS App	I	V	I	I
Maschinelles Lernen	I	V	I	I
Dokumentation	V	V	V	I
Diplomschrift	V	V	V	I

5 Realisierung

5.1 Anwendungsfälle

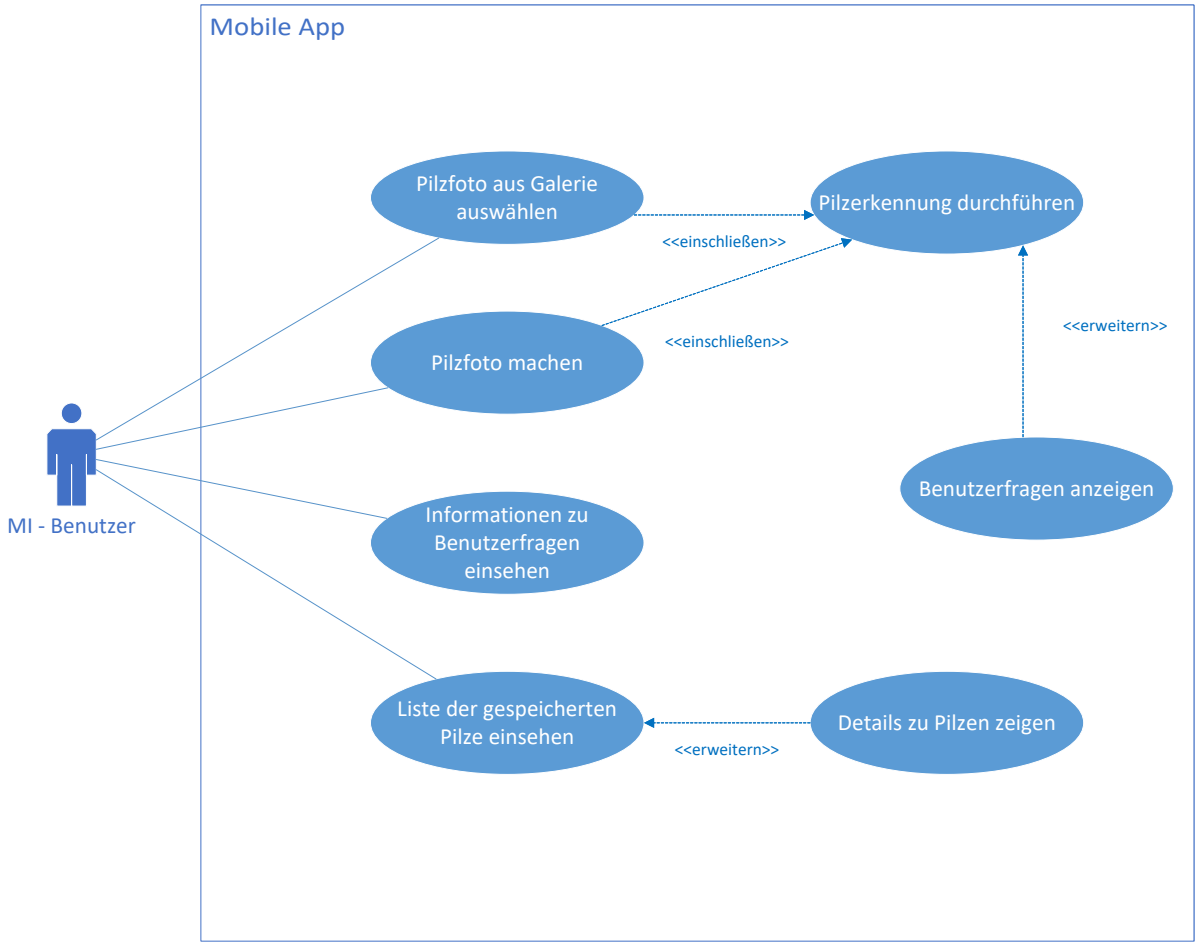


Abbildung 5: Anwendungsfälle

6 Darstellung

6.1 iOS

6.1.1 Pilzanalyse

Abbildung 7: Pilzanalyse mit Benutzerfragen in iOS

Abbildung 6: Pilzanalyse Ergebnis 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.

6.1.2 Pilzliste

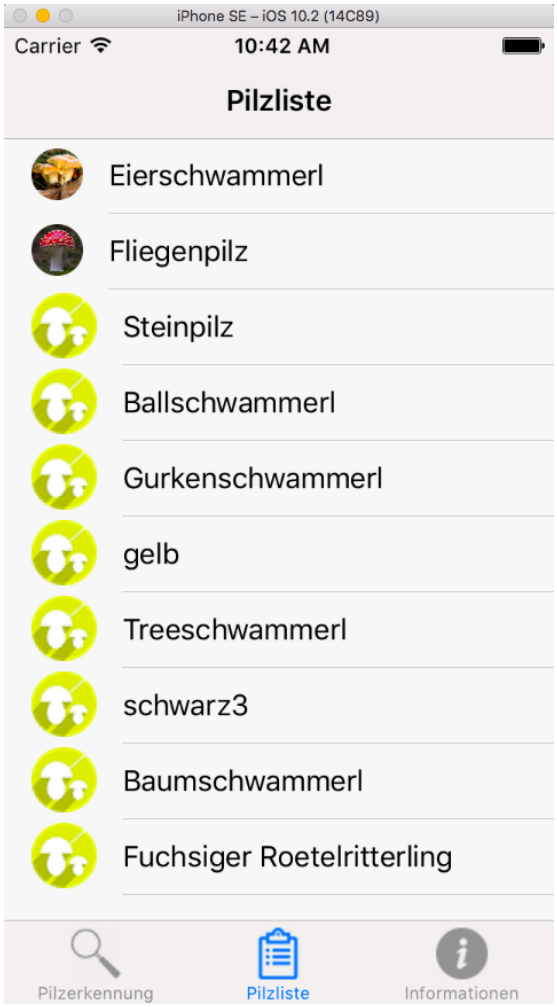


Abbildung 9: Pilzliste iOS

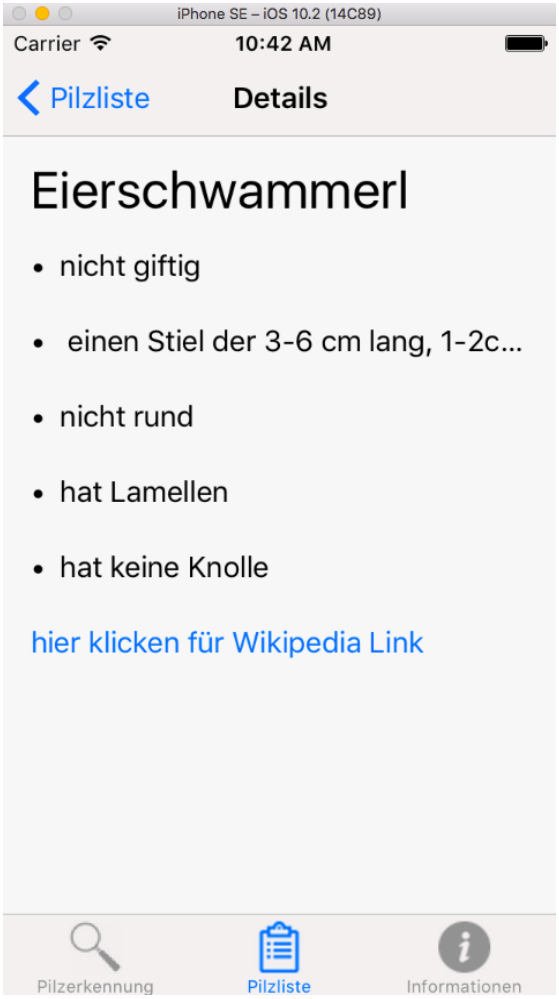


Abbildung 8: Details zu Pilz iOS

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.

6.2 Android

6.2.1 Foto schießen

Drückt man 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.

Abbildung 10: Startseite mit Pilzfoto

6.2.2 Pilzliste

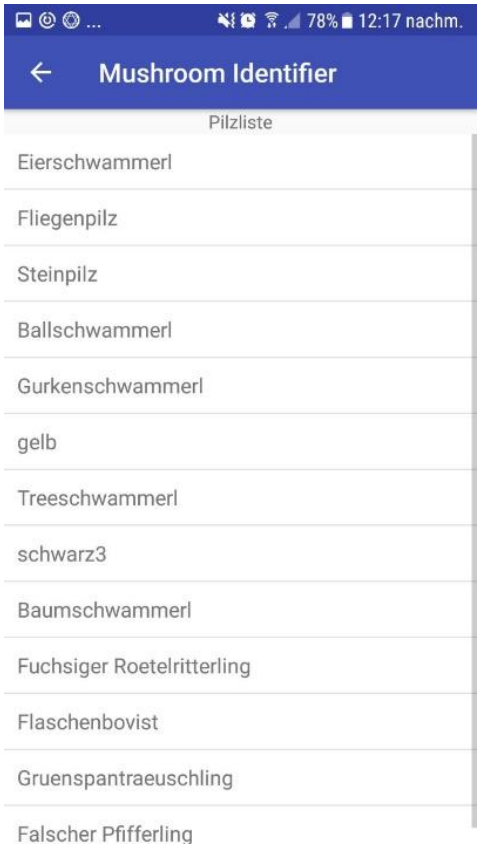


Abbildung 11: Pilzliste

Abbildung 12: Detailansicht vom Pilz

Um Details zu einem Pilz einzusehen, ist nur ein Klick auf diesen notwendig.

7 Technologien

7.1 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.

7.2 C++ (OPEN CV)

C++ ist eine projektorientierte Erweiterung der Programmiersprache C und ermöglicht eine effiziente, maschinennahe Programmierung. Die Programmbibliothek OPENCV, die für den Bilderkennungsprozess verwendet wird, ist in dieser Sprache geschrieben. (vgl. Wikipedia, 2017)

7.3 Android Studio

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

Abbildung 13: Logo Android Studio

Android Studio ist speziell für die Android Entwicklung entwickelt worden. Sie kann frei für Windows, MacOS und Linux heruntergeladen werden.
(vgl. Google , 2017)

7.3.1 Warum ist Android Studio so vorteilhaft?

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

Erweiterte Code Ergänzung: Android Studio liefert präzise Code Ergänzungen, welche für das Programmieren viel Zeit und unnötig langes Fehlerbeheben ersparen.

User Interface: Android Studio hat eine sehr benutzerfreundliche Benutzeroberfläche. Als Einsteiger hat man keine Probleme beim Zurechtfinden im Programm.

Organisierung des Projektes: Android Studio benutzt Module, welche eigene Gradle build Dateien besitzen, die eigene Abhängigkeiten angeben können. Außerdem hat Android Studio eine Funktion, welche das zuletzt geschlossene Projekt beim Start öffnet, was wiederum Zeit spart.

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

Drag and Drop: Android Studio hat eine Drag and Drop Funktion eingebaut, welche über die grafische Benutzeroberfläche benutzt werden kann.
(vgl. Rajput, 2015)

7.4 JNI – Java Native Interface

Dieses Programmiergerüst ermöglicht, dass Java Applikationen mit nativen Bibliotheken kommunizieren und Funktionen aufrufen und umgekehrt. Kurz gesagt, es ermöglicht eine Verbindung zwischen Java und einer fremden Programmiersprache wie zum Beispiel: C oder C++.
(vgl. Wikipedia, JNI, 2011)

7.5 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 zu behandeln, Hardware Sensoren zu benutzen und den Zugang zu Applikationsressourcen anzubieten, wenn man in C/C++ programmiert.
(vgl. Google, 2017)

7.6 CMake

CMake ist eine Open-Source Plattform, welche 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 zu konstruieren.
(vgl. CMake, 2017)

Abbildung 14: Logo CMake (CMake Logo, 2017)

7.7 Gradle

Gradle ist ein Open-Source-Automatisierungssystem.

Abbildung 15: Logo Gradle (Gradle Logo, 2017)

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

Hierfür müsste man selbst ein Skript schreiben, das verschiedene Schritte ausführt.

Gradle ist ein weiteres Konstruktions-System, welches die besten Eigenschaften von anderen Systemen nimmt und diese in einem kombiniert. Es ist möglich, ein eigenes Skript in Java zu schreiben, welches dann von Android Studio benutzt wird.

Gradle kann zum Beispiel ein Verzeichnis in ein anderes Verzeichnis kopieren, noch bevor der eigentliche Konstruktionsprozess passiert.
(vgl. Gradle Inc., 2017)

7.8 XCode (IOS)

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

Abbildung 16: Logo XCode

7.9 Visual Studio (OPEN CV)

Abbildung 17: Logo Visual Studio (Visual Studio Logo, 2017)

Ist eine von Microsoft angebotene Entwicklungsumgebung für verschiedene Hochsprachen (darunter C, C++, C#, Python, HTML, JavaScript und Typescript). Sie wurde für das Implementieren des C++ Bilderkennungs – und maschinelles Lernen-Teiles verwendet.
(vgl. Wikipedia, 2017)

Abbildung 18: Logo OpenCV

7.10 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.
(vgl. Wikipedia, 2017)

Abbildung 19: Logo OpenCV (Wikipedia, 2017)

7.11 CMARKUP (XML esen)

Ist ein simpler XML Parser mit guter Dokumentation und einem „Erklär Video“ wie man ihn benutzen kann.
Dabei handelt es sich um eine einfach zu implementierende Lösung (eine C++ Klasse). Im Grunde ist es ein vereinfachtes DOM Modell, das sich auf das Wesentliche, auf das Lesen von XML, konzentriert.

7.12 Computer Vision

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

7.13 Haar Cascade Training

Wird für das maschinelle Lernen verwendet. Es besteht aus zwei Phasen: Training und Detektion. 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 (vgl. opencv dev team, 2016).

7.14 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 auf 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 für Android auch als IOS zur Verfügung stehen und nicht für beide Systeme neu implementiert werden müssen. Es ist somit innerhalb kürzester Zeit möglich, das System für ein anderes System (beispielsweise Windows Phone, Virtual Reality Brillen, oder was die Zukunft noch bringt) zu implementieren.

8 Programmierung

Der Identifikationsprozess besteht aus insgesamt drei Phasen:

Bilderkennung

Maschinelles Lernen

JA/NEIN Benutzerfragen

8.1 Bilderkennung

8.1.1 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

XML Datei wird mithilfe des Haar Cascade Algorithmus erstellt.

Benutzerfragen

8.1.2 Computer Vision

Funktionsweise Bilderkennung

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

Abbildung 20: Farberkennung Eierschwammerl

8.1.2.1 Einlesen eines Fotos

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

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

8.1.2.2 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")));
```

…

Die Eigenschaften aller Pilze werden eingelesen.

8.1.2.3 Farberkennung

Danach wird ein Quadrat im Zentrum des Bildes nach der Farbe untersucht. Der Mittelwert aus den erkannten Farben wird daraufhin mit den Daten aus der XML Datei verglichen.

Code Snippet:

```
for (int i = -1 * (range); i < range; i++) {
    for (int j = -1 * (range); j < range; 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];
    }
}

//Durchschnittswert der Pixelfarben errechnen
pix[0] = array2[0] / pixels;
pix[1] = array2[1] / pixels;
pix[2] = array2[2] / pixels;
```

8.1.2.4 Konvertierung in HSV Farbraum + Pilzerkennung

Es werden alle Grundfarben (blau, gelb und rot) aus dem Foto, mit den gespeicherten Grundfarben von jedem Pilz verglichen. Wenn bei einem Pilz jede Grundfarbe innerhalb einer Schwelle von 30 liegt, wird der Pilz weiter untersucht.

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

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.

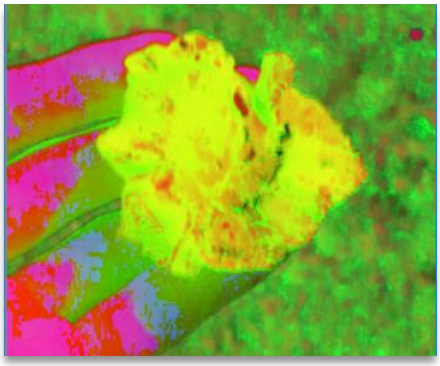


Abbildung 21: Eierschwammerl Darstellung in HSV (OpenCV)

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);

                }
        }
```

8.1.2.5 Weichzeichner

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);
```

8.1.2.5.1 Standard-Weichzeichner

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

Die Faltungsmatrix wird in Folge über jedes Pixel gelegt.

Der errechnete Wert (in diesem Fall 39) wird in ein neues Bild an derselben Stelle eingefügt. Die Idee dahinter ist, dass man mit dem Durchschnittswert von mehreren Farbpixeln rund um ein Bild Unreinheiten beseitigen kann und weichere Übergänge erzielt.

8.1.2.5.2 Gaußscher Weichzeichner

Der Gaußsche Weichzeichner ist eine Erweiterung des Standard-Weichzeichners. Es wird davon ausgegangen, dass Pixel, die sich näher an dem untersuchten Pixel befinden, stärker berücksichtigt werden sollten, als jene, die weiter vom untersuchten Pixel entfernt sind.

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 die Zahlen in der Matrix liegen im Dezimalbereich. Der Prozess wurde jedoch für Darstellungszwecke vereinfacht.

Mit der äußersten Pixelreihe kann dieser Algorithmus nicht durchgeführt werden, weil die Faltungsmatrix einen Pixel umschließen muss. 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 Verfälschungen führen, bei Bildern von mehreren Megapixeln jedoch irrelevant sein. Vorteil: Rechenleistung ersparen.

8.1.2.6 Canny Edge Detector

Um Kanten in einem Bild erkennen zu können wird der Canny Edge Detector verwendet. Dazu wird in einem ersten Schritt ein Weichzeichen-Filter darübergelegt.

Code Snippet:

```
blur(src_gray, detected_edges, Size(3, 3));
Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size);
```

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 möglichst wenig Störgeräusche 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:

8.1.2.6.1 Glättung, um Rauschen zu verhindern

Dazu wird ein einfacher Gaußscher Weichzeichner angewendet (siehe 8.1.2.4.2 Gaußscher Weichzeichner)

(Youtube, 2015)

8.1.2.6.2 Kantendetektion (Sobel Algorithmus)

Dafür wird der Sobel Algorithmus verwendet

Dabei kommen, wie bei den Weichzeichen-Algorithmen, Faltungsmatrizen zum Einsatz.

Faltungsmatrize für x-Achse

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

Faltungsmatrize für y-Achse

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

Bei dem Beispielbild handelt es sich um ein Bild mit einer vertikalen, aber keiner horizontalen Kante. Es sollte also nur eine vertikale Kante gefunden werden.

8.1.2.6.2.1 Beispiel X-Achse

100	100	50	50
100	100	50	50
100	100	50	50
100	100	50	50

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

Ergebnis (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

8.1.2.6.2.2 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.

8.1.2.6.2.3 Gradienten 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:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

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

Aus Performance Gründen wird oft auf eine Annäherung zurückgegriffen:

$$|G| = |Gx| + |Gy|$$

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

$$\theta = \arctan(Gy/Gx)$$

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

8.1.2.6.3 Unterdrückung von Nicht-Maxima

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 Kantennachbarn zu bestimmen. Nach diesem Prozess wird jede Kante in der Mitte als „einpixelig“ erkannt.

8.1.2.6.4 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.

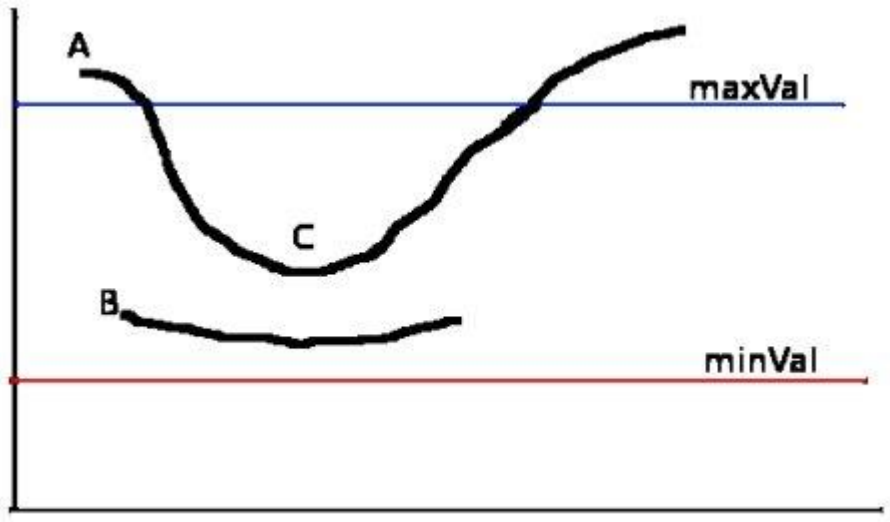


Abbildung 22: Canny maxVal, minVal

Der Teil A in diesem Bild wird generell als 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 auftun.

Die mögliche Linie B wird nicht als Kante erkannt, da sie sich zwischen den zwei Parameter befindet und nicht mit einem Teil über der maxVal zusammenhängt

(vgl. Wagner, Kantenextraktion Klassische Verfahren, 2005/2006), (vgl. Canny, 1986), (vgl. Pound, 2015).

8.1.2.7 Hough Transformation

8.1.2.7.1 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 Kantendetektion. 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. Jeder Punkt in diesem Raum entspricht einem geometrischen Raum im Bild.

(vgl. Wagner, Kantenextraktion Klassische Verfahren, 2005/2006),

8.1.2.7.1.1 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:

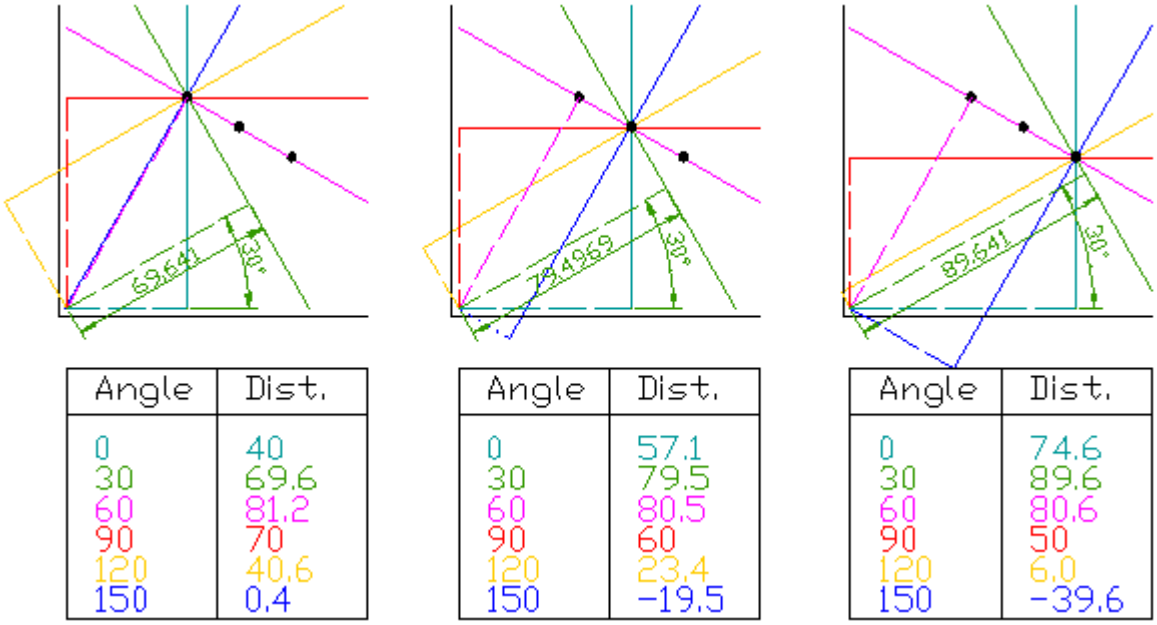
$r= x*\cos(\theta) + y* \sin(\theta)$

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

Dabei ist dieser Algorithmus sehr resistent gegen nicht perfekte Kreise 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.



Angle	Dist.
0	40
30	69,6
60	81,2
90	70
120	40,6
150	0,4

Angle	Dist.
0	57,1
30	79,5
60	80,5
90	60
120	23,4
150	-19,5

Angle	Dist.
0	74,6
30	89,6
60	80,6
90	50
120	6,0
150	-39,6

Abbildung 23: Darstellung Hough Line Transformation 1 (Wikipedia, 2017)

Bei dieser Grafik handelt es sich, vereinfacht ausgedrückt um drei 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 gestrichelten 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 wird als Parameter verwendet.

Diese Tabellen lassen sich auch als Kurven darstellen:

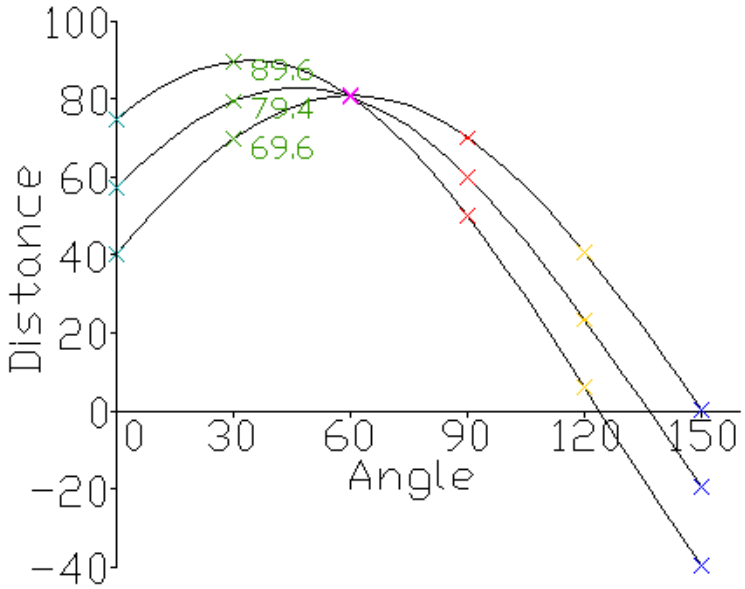


Abbildung 24: Darstellung Hough Line Transformation 2 (Wikipedia, 2017)

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

8.1.2.7.1.2 Hough Circle Transformation

Um diesen Algorithmus für die Kreiserkennung umzuschreiben, muss Folgendes erledigt werden:

Zuerst wird ein Houghraum mit Nullen befüllt.

Mit dem Ausdruck $(i-a)^2+(j-b)^2=r^2$ wird die Mitte des Kreises bestimmt.

r ist dabei der Radius

a und b stellen die Koordinaten des Kreismittelpunktes dar

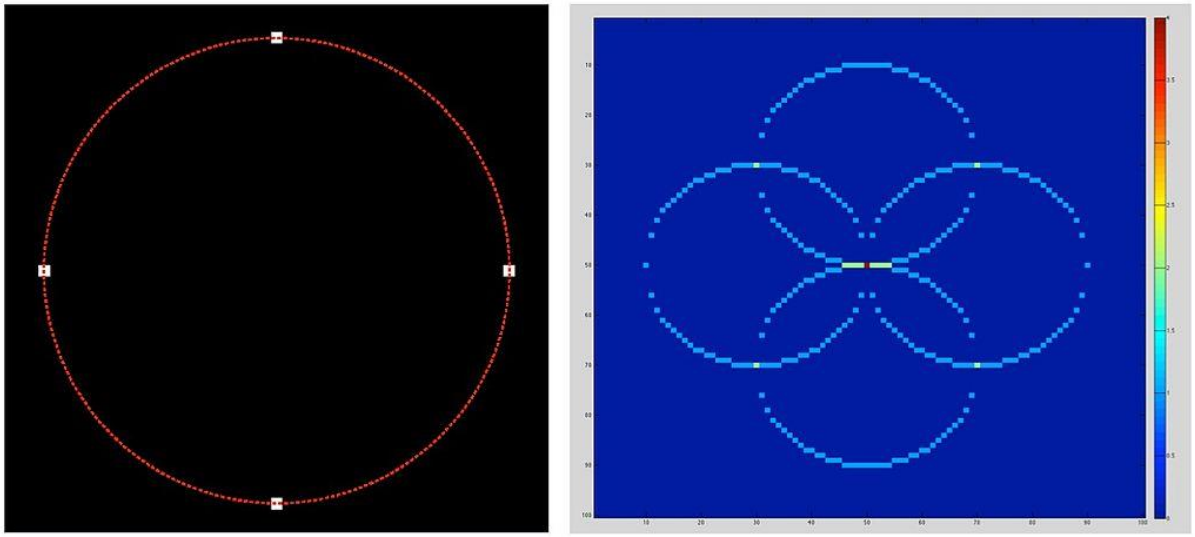


Abbildung 25: Hough Circle Transformation (Wikipedia, 2017)

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 mit verschiedenen Radien durchgeführt werden. Die Werte werden dabei in einem dreidimensionalen Hough Raum gespeichert, um die Werte mit mehreren möglichen Kreisradien speichern zu können.

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());
}
```

8.1.2.8 Findungsprozess der richtigen Parameter:

8.1.2.8.1 Mat src_gray

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

8.1.2.8.2 vector<Vec3f> circles

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

8.1.3 CV_HOUGH_GRADIENT:

Name der Erkennungsmethode (zurzeit ist nur dieser verfügbar)

8.1.3.1.1 DP:

Je kleiner die DP, desto genauer ist die Kreiserkennung.

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

8.1.3.1.2 Min_dist

src_gray.rows/8: Minimale Distanz zwischen erkannten Kreiszentren

8.1.3.1.3 param_1:

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

Entscheidung: 99

8.1.3.1.4 param_2:

Schwellenwert für die Mittenerkennung

Entscheidung: 41

8.1.3.1.5 min_radius:

Minimaler Kreisradius, um erkannt zu werden (0=egal)

8.1.3.1.6 max_radius:

Maximaler Kreisradius, um erkannt zu werden (0=egal)

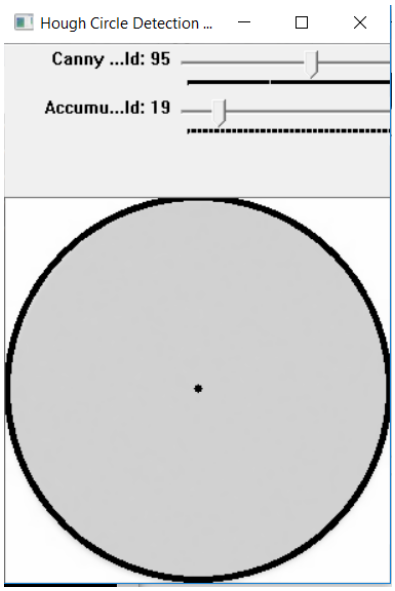


Abbildung 26: Trackbar Kreiserkennung

8.1.3.2 Entscheidung für param_1 und param_2:

Dazu wurden 10 runde Pilze und 10 nicht runde Pilze als Testfälle ausgewählt. Dabei wurde param_1 und param_2 mit Trackbars so verändert, dass 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 29: Eierschwammerl HSV schwarz weiß

Abbildung 28:Eierschwammerl HSV schwarz weiß Weichzeichnen

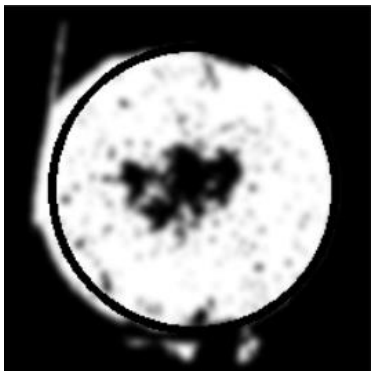


Abbildung 27: Fuchsiger Rötelritterling Kreiserkennung

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

(vgl. Ballard, 1980), (vgl. opencv dev team , 2017)

8.1.3.3 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.

8.1.4 Datenspeicherung (XML)

```
<P1>
    <Farbe b="13" g="138" r="208"> </Farbe>
    <Name>Eierschwammerl</Name>
    <HSV-von h="11" s="145" v="145"> </HSV-von>
    <HSV-bis h="27" s="255" v="255"> </HSV-bis>
    <HSV-von2 h="0" s="0" v="0"> </HSV-von2>
    <HSV-bis2 h="0" s="0" v="0"> </HSV-bis2>
    <Wiki>https://de.wikipedia.org/wiki/Echter_Pfifferling</Wiki>
    <Giftigkeit>0</Giftigkeit>
    <Rund>0</Rund>
    <Lamellen>1</Lamellen>
    <Knolle>0</Knolle>
    <Stiel> einen Stiel der 3-6 cm lang, 1-2cm dick, hellgelb, feinfaserig-schuppig und an der Basis zugespitzt ist</Stiel>
</P1>
```

Abbildung 30: Datenspeicherung XML

Die Entscheidung 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;
};
```

8.1.4.1 Probleme

8.1.4.1.1 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.

https://www.youtube.com/watch?v=l4372qtZ4dc (Youtube Video Stand 04.03.2017)

8.1.4.1.2 BGR Farbraum in OpenCV

In OpenCV wird nicht der RGB (Standard) Farbraum, sondern der BGR Farbraum verwendet => führte zu vermeintlich falschen Ergebnissen => Tipp in Forum führte mich zu der Lösung (aardvarkk, 2012)

Die Frage, warum nicht das gebräuchlichere RGB zum Einsatz kommt, hat der Gründer von OPENCV Dr. Gary Bradski in einem Interview folgendermaßen beantwortet:

“Why is 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 der 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.

8.1.4.1.3 HSV Farbraum nur bis zum Wert 180

Der HSV Farbraum ist in OpenCV nur bis 180 gehend (um in uchar zu passen), normale Farbraumumrechner (Word, GIMP, Photoshop, Internetrechner) rechnen mit 360 => Lösung => eigenen Farbraumumrechner für OPENCV gesucht und gefunden (Emami, 2010)

Alternative:

Track Bars selbst implementieren => siehe Circle Detektion

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: Der Wert 181 ist wieder als 1 definiert. 182 als 2,…

Dieses Hilfsmittel kann allerdings nur bis 255 durchgeführt werden, um in uchar zu passen.

8.1.4.1.4 Fliegenpilze komplex

Fliegenpilze eigentlich rund sind mit weißen Flecken teilweise auch an den Rändern.

Diese weißen Flecken werden nicht miterkannt und bei der Untersuchung nach runden Figuren nicht als rund erkannt.

8.1.5 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 beschrieben werden:

8.1.5.1 Training Phase

Die Training Phase besteht darin, 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 angewandte Viola-Jones-Algorithmus durchläuft dabei mehrere Phasen, in denen er bestimmte Merkmale aus den Bildern herausfiltert, an denen man diese erkennen kann. In der folgenden Grafik werden solche Merkmale dargestellt:

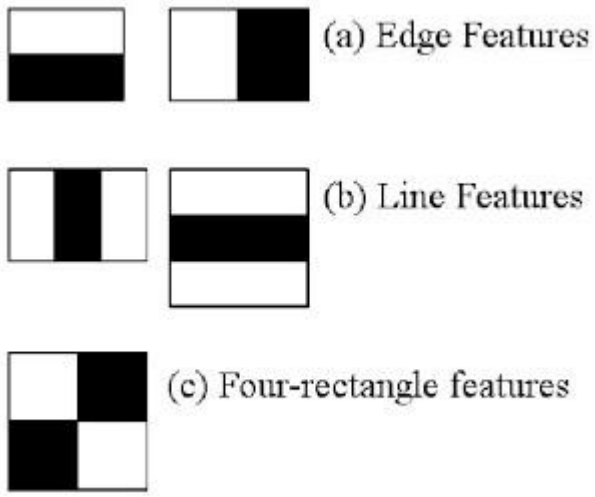


Abbildung 31: Haar Merkmale (opencv dev team, 2017)

Dabei wird eine Summe von Pixel im schwarzen Bereich berechnet und anschließend die Summe der Pixel im weißen Bereich subtrahiert. Diese Merkmale müssen dann für alle Positionen im Bild berechnet werden. Dabei gibt es aber das Problem, dass extrem viele Merkmale berechnet werden müssen (über 160.000 bei einem 24x24 Pixel Bild). Es gibt

jedoch die sogenannten „Integral images“, die das Summieren von Pixeln stark vereinfacht (vgl. opencv dev team, 2017).

Man wandelt also das Originalbild in ein Integralbild um. Dazu wird zuerst am Originalbild eine neue Zeile und eine neue Spalte am Beginn hinzugefügt und mit Nullen aufgefüllt. Anschließend wird jedes Pixel neu berechnet, indem man einfach den alten Wert mit dem Wert des linken und des oberen Nachbarn addiert. Die Berechnung wird in der folgenden Grafik dargestellt.

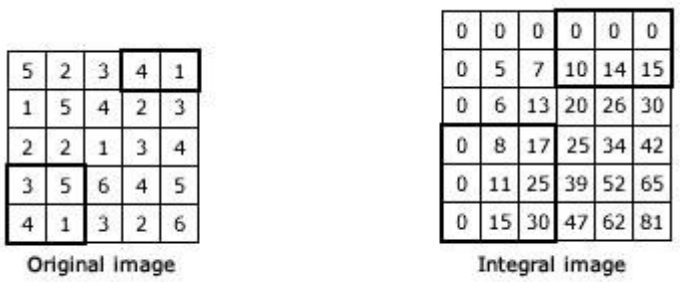


Abbildung 32: Integral image (Sinha, 2010)

Möchte man nun die Summe in einem rechteckigen Bereich von Pixeln des Originalbilds berechnen, nimmt man den zugehörigen Bereich im Integralbild und erweitert diesen links um eine Spalte und oben um eine Zeile. Anschließend wird das rechte untere Pixel mit dem linken oberen Pixel addiert und dann das rechte obere und das linke untere abgezogen. Somit kann die Berechnung von Summen immer mit nur vier Pixeln erfolgen und dadurch deutlich beschleunigt werden (vgl. Sinha, 2010).

Anschließend müssen die relevanten Merkmale, also die, die für die Bilderkennung geeignet sind, herausgefiltert werden. Dies erfolgt über Adaboost. Es wird jedes einzelne Merkmal auf jedes Trainings-Bild angewandt und anschließend werden die Bilder gewichtet und irrelevante Merkmale herausgefiltert. Dann wird derselbe Prozess solange wiederholt, bis die gewünschte Genauigkeit erreicht wird.

Um bei der Erkennung dann nicht alle Merkmale mit jedem Bild vergleichen zu müssen, wurde das Konzept der „Kaskaden“ entwickelt. Dabei werden Merkmale zu Merkmalgruppen zusammengefasst, die dann nach der Reihe auf die zu erkennenden Bilder

angewendet werden. Wenn ein Bild aufgrund einer Merkmalgruppe schon als negativ erkannt werden kann, werden die anderen gar nicht mehr geprüft. Wenn alle Merkmalgruppen ohne negative Erkennung durchlaufen sind, gilt das Bild als positiv (vgl. opencv dev team, 2017).

8.1.5.2 Durchführung des Trainings

Für die gewählte Vorgangsweise, die sich nach dem Tutorial auf http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html richtet, müssen zuerst 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.

Anschließend werden positive und negative Bilder gesucht. 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, da das die Anforderung unserer App ist. 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 der Waldboden zu sehen ist, an. Je mehr qualitativ gute Bilder verwendet werden, desto weniger Falscherkennungen werden bei der Bilderkennung auftreten.

Dann 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. Danach 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 fürnegative 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 von Naotoshi Seo namens „createsamples.pl“, welches über die Eingabeaufforderung aus dem Verzeichnis 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 24 -h 24"

Besondere Aufmerksamkeit muss auf die Parameter -w und -h gelegt werden. Diese beschreiben die Breite und Höhe der Samples und sollten das selbe Seitenverhältnis wie die Trainings-Bilder haben (für genauere Informationen zu den Parametern, siehe opencv dev team, 2017).

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 24 -h 24 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024

Parameter (vgl. opencv dev team, 2017):

-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, desto 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 und -precalcIdxBufSize

Größe des verwendeten Puffers für den Trainingsprozess in MB.

Je höher, desto schneller ist das Training.

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

8.1.5.3 Erkennungs-Phase

Zum Testen dieser XML – Datei wurde ein Programm in C++ geschrieben, das die detectMultiScale – Funktion aufruft. Bei Verwendung der empfohlenen Parameter wurden zunächst extrem viele Fliegenpilzerkennungen erzielt, wie auf den folgenden Abbildungen zu sehen ist:

Abbildung 34: Fliegenpilz mit vielen Erkennungen

Abbildung 33:Viele Falscherkennungen bei Nicht-Fliegenpilz

Leider ist beim Großteil der Erkennungen gar kein Fliegenpilz zu sehen. Mit diesem Ergebnis konnte man sich also noch nicht zufriedengeben, also wurden Schieberegler implementiert, mit deren Hilfe die Parameter einfach und schnell verändert werden können. Bei Erhöhung des minNeighbours – Parameters werden bereits deutlich bessere Ergebnisse erzielt:

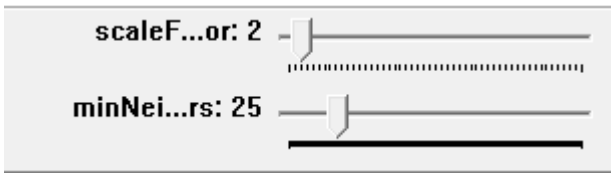


Abbildung 35: Haar Cascade Schieberegler

Abbildung 36: Fliegenpilz1 mit wenigen Erkennungen

Abbildung 37: Nicht-Fliegenpilz ohne Erkennungen

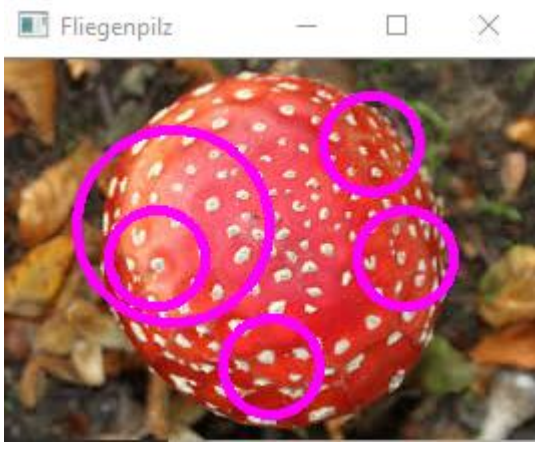


Abbildung 39: Fliegenpilz2 mit wenigen Erkennungen

Abbildung 38: Fliegenpilzähnlicher Pilz mit Falscherkennung

Bei Bildern von Fliegenpilzen hat man immer noch mehrere Erkennungen, bei den meisten anderen Pilzen wird überhaupt kein Fliegenpilz erkannt. Es gibt jedoch einige Pilze, wie zum Beispiel dem Habichtspilz, die dem Fliegenpilz relativ ähnlich schauen. Bei diesen treten daher teilweise Falscherkennungen auf.

8.1.5.4 Probleme

8.1.5.4.1 Geringe Anzahl an positiven und negativen Samples

Grundsätzlich lässt sich sagen: Je mehr Bilder man der Funktion bereitstellt, desto 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 Das Problem hierbei ist, dass es für uns nicht möglich war, genügend unterschiedliche Bilder zu finden, da die Pilze von oben fotografiert sein müssen und über das Internet nicht genügend brauchbare Aufnahmen zu finden waren. Das heißt, die einzige Möglichkeit 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 ca. 15 positive und 60 negative Bilder verwendet.

8.1.5.4.2 Schlechte Ergebnisse aufgrund eines Tippfehlers bei den Parametern

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

8.1.5.4.3 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 reichen, wenn man mit 10 Stages beginnt.
3. Es wurde eine zu hohe Breite und Höhe der Samples gewählt. Anstatt 50x50 reicht auch 20x20.

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

8.1.5.4.4 Probleme beim Finden von Testdaten

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

8.1.6 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.

8.1.6.1 Notwendige Schritte

8.1.6.1.1 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, ergibt 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.

8.1.6.1.2 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 unterschiedliche 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.

8.1.6.1.3 Auslagern von 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.

8.1.7 Tätigkeiten bis zum Endbenutzerprodukt

Aufnahme von mindestens 1000 - besser zwischen 3000 und 5000 Pilze.
Für die 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, einen großen Benutzerumfang aufzubauen, der Fotos schießt. 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.

8.2 iOS – App

8.2.1 OpenCV in iOS

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.

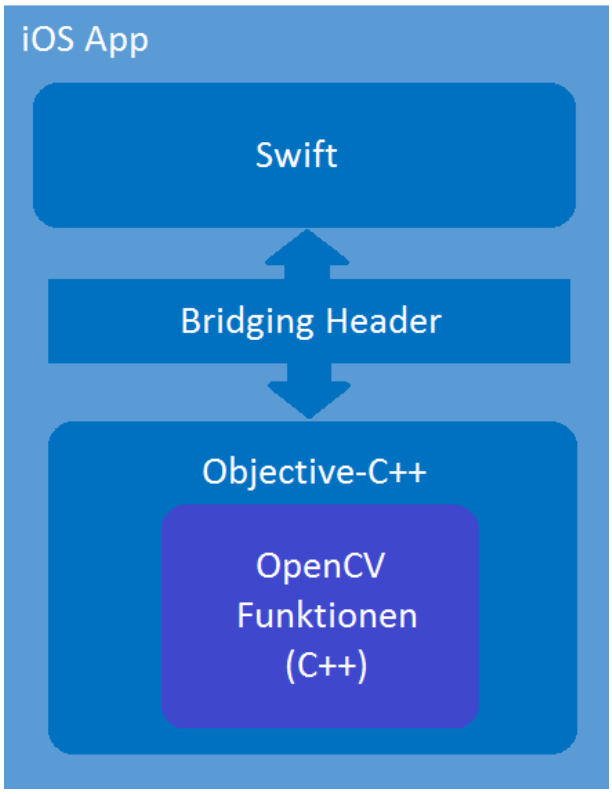


Abbildung 40: Verwendung von C++ in iOS

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 einen Bridging Header erstellen lassen.

8.2.2 OpenCVWrapper.h

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

```
@interface OpenCVWrapper : NSObject
+(NSMutableArray<PilzC *> *) detectMushroom:(UIImage*) img : (NSString*) xmlpath1 : (NSString*) xmlpath2;
+(NSMutableArray<PilzC *> *) allMushrooms: (NSString*) xmlpath;
@end
```

Abbildung 41: Code (OpenCVWrapper.h)

8.2.3 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.

8.2.3.1 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++:

```
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
    string name; //Name des Pilzes
    string wiki; //Wikipedia Link
    int 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
    int nodule; //= Knolle, Eigenschaftswort (z. B. dicke, rundliche etc.)
    string stalk;   //Stiel
};
```

Abbildung 42: Code (Pilzklasse in C++)

Objective C:

```
@interface PilzC:NSObject

@property(nonatomic, readwrite) NSString *name;     // Name des Pilzes
@property(nonatomic, readwrite) NSString *wiki;     // Wikipedia Link
@property(nonatomic, readwrite) int poisonous;      // ist der Pilz giftig, 1 ja, 0 nein
@property(nonatomic, readwrite) int round;          // ist der Pilz rund, 1 ja, 0 nein
@property(nonatomic, readwrite) int lamell;         // hat der Pilz Lamellen, 1 ja, 0 nein
@property(nonatomic, readwrite) int nodule;         // hat der Pilz eine Knolle, 1 ja, 0 nein
@property(nonatomic, readwrite) NSString *stalk;    // Beschreibung des Stiels

@end
```

Abbildung 43: 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.

8.2.3.2 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 die Umwandlung von „UIImage“ in „cv::Mat“ vornimmt.

```
Mat convertToMat(UIImage* img)
{
    CGColorSpaceRef colorSpace = CGImageGetColorSpace(img.CGImage);
    CGFloat cols = img.size.width;
    CGFloat rows = img.size.height;

    cv::Mat cvMat(rows, cols, CV_8UC4); // 8 bits per component, 4 channels (color channels + alpha)

    CGContextRef contextRef = CGBitmapContextCreate(cvMat.data,                 // Pointer to  data
                                                    cols,                       // Width of bitmap
                                                    rows,                       // Height of bitmap
                                                    8,                          // Bits per component
                                                    cvMat.step[0],              // Bytes per row
                                                    colorSpace,                 // Colorspace
                                                    kCGImageAlphaNoneSkipLast |
                                                    kCGBitmapByteOrderDefault); // Bitmap info flags

    CGContextDrawImage(contextRef, CGRectMake(0, 0, cols, rows), img.CGImage);
    CGContextRelease(contextRef);

    cv::cvtColor(cvMat, cvMat, cv::COLOR_RGB2BGR);

    return cvMat;
}
```

Abbildung 44: Code (convertToMat)

8.2.3.3 detectMushroom

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

```
+(NSMutableArray<PilzC *> *) detectMushroom:(UIImage*) img : (NSString*) xmlpath1 : (NSString*) xmlpath2
{
    std::string xmlpathh1 = std::string([xmlpath1 UTF8String]);
    std::string xmlpathh2 = std::string([xmlpath2 UTF8String]);

    vector<Pilz> mushlist = detectMushroom(xmlpathh1, xmlpathh2, convertToMat(img));

    NSMutableArray<PilzC *> *arr = [[NSMutableArray alloc] init];
    PilzC *c = [[PilzC alloc] init];

    for(int i = 0; i < mushlist.size(); i++){
        c.name = [NSString stringWithUTF8String:mushlist[i].name.c_str()];
        c.wiki = [NSString stringWithUTF8String:mushlist[i].wiki.c_str()];
        c.stiel = [NSString stringWithUTF8String:mushlist[i].stalk.c_str()];
        c.lamellen = mushlist[i].lamell;
        c.knolle = mushlist[i].nodule;
        c.giftigkeitt = mushlist[i].poisonous;
        c.rund = mushlist[i].roud;

        cout << c.name;

        [arr addObject: c];

        c = [[PilzC alloc] init];
    }

    return arr;
```

Abbildung 45: 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.

8.2.3.4 *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.

```
+(NSMutableArray<PilzC *> *) allMushrooms: (NSString*) xmlpath{
    std::string xmlpathh = std::string([xmlpath UTF8String]);

    vector<Pilz> mushlist = readxml(string(xmlpathh)); //Liste aller gelesenen Pilze

    NSMutableArray<PilzC *> *arr = [[NSMutableArray alloc] init];

    PilzC *c = [[PilzC alloc] init];

    for(int i = 0; i < mushlist.size(); i++){
        c.name = [NSString stringWithUTF8String:mushlist[i].name.c_str()];
        c.wiki = [NSString stringWithUTF8String:mushlist[i].wiki.c_str()];
        c.stiel = [NSString stringWithUTF8String:mushlist[i].stalk.c_str()];
        c.lamellen = mushlist[i].lamell;
        c.knolle = mushlist[i].nodule;
        c.giftigkeitt = mushlist[i].poisonous;
        c.rund = mushlist[i].roud;

        cout << c.name;

        [arr addObject: c];

        c = [[PilzC alloc] init];
    }

    return arr;
}
```

Abbildung 46: Code (allMushrooms)

8.2.4 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.

8.2.5 Libjpeg

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

```
Undefined symbols for architecture x86_64:
  "_iconv", referenced from:
      TextEncoding::IConv(void*, int, int) in Markup.o
  "_iconv_open", referenced from:
      TextEncoding::CanConvert(MCD_CSTR, MCD_CSTR) in Markup.o
      TextEncoding::IConv(void*, int, int) in Markup.o
  "_iconv_close", referenced from:
      TextEncoding::CanConvert(MCD_CSTR, MCD_CSTR) in Markup.o
      TextEncoding::IConv(void*, int, int) in Markup.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Abbildung 47: "_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.

8.2.5.1 Probleme

Probleme traten bei der Umwandlung von UIImage in cv::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 umgewandelt werden. OpenCV stellt hierfür die Funktion „COLOR_RGB2BGR“ zur Verfügung.
Weiters musste zuerst das C++-Bilderkennungsprogramm in ein plattformunabhängiges umgeschrieben werden, was bereits weiter oben beschrieben wurde.

8.3 Android – App

8.3.1 Installation

Um mit der Entwicklung des Apps zu beginnen, werden zuerst ein paar Software Programme 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

8.3.2 OpenCV Source Code in der Android Native Toolchain1 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 eigenen Regelungen hat. Um eine funktionierende Sammlung von OpenCV Bibliotheken zu bekommen, 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, einen Ordner im äußersten Projektverzeichnis zu erzeugen. In diesem Ordner werden folgende Dateien erstellt:

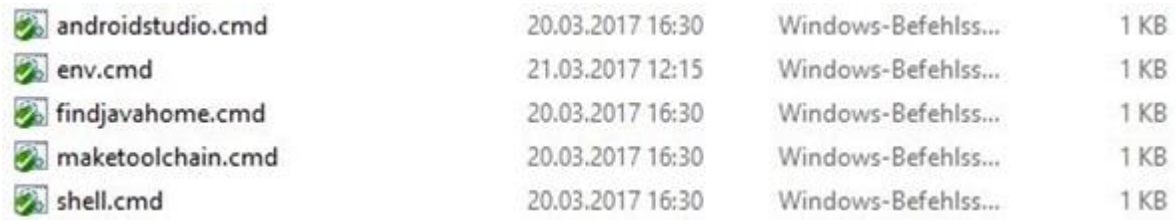


Abbildung 48 Skriptverzeichnis

Bevor weitergemacht wird, ist CMake und Python zu installieren.

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

```
SET ARCH=x86_64
mkdir ndk
pushd ndk
python %NDK%\build\tools\make_standalone_toolchain.py --arch %ARCH% --api 24 --stl=gnustl --unified-headers --install-dir %STANDALONE_TOOLCHAIN%
```

Abbildung 49 maketoolchain.cmd

Parameter:

ARCH: Definiert die Systemarchitektur, für das Toolchain.

NDK und *STANDALONE_TOOLCHAIN* werden später beschrieben.

(vgl. Google, 2017)

[1]Ein Toolchain ist eine Sammlung von Werkzeugen, die ermöglicht ein System zu erzeugen, das unabhängig von seinem Ursprung auf der jeweiligen Entwicklungsumgebung funktioniert.

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.

OpenCV Source Code Zip Datei herunterladen und entpacken.

(vgl. Alexander Alekhin, 2016)

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

```
SET CURDIR=%~dp0
call %CURDIR%findjavahome.cmd

SET STANDALONE_TOOLCHAIN=%CURDIR%ndk\ndk-x86_64

SET ANDROID_SDK=C:\Users\Hakan\AppData\Local\Android\sdk
SET NDK=%ANDROID_SDK%\ndk-bundle
SET ANDROID_NDK=%NDK%

SET PYTHON_HOME=C:\Users\Hakan\AppData\Local\Programs\Python\Python36-32
SET CMAKE_HOME=%ProgramFiles%\CMake

SET OPENCV_SOURCE_CODE_DIR=C:\Users\Hakan\Desktop\5AHIF\Diplomarbeit_MushroomIdentifier\opencv-3.2.0

SET CC=clang
SET CXX=clang++
SET CMAKE_C_COMPILER=clang
SET CMAKE_CXX_COMPILER=clang++
SET CMAKE_MAKE_PROGRAM=%STANDALONE_TOOLCHAIN%\bin\make.exe

SET PATH=%PYTHON_HOME%;%NDK%\bin;%JAVA_HOME%\bin;%SystemRoot%\System32;%SystemRoot%
```

Abbildung 50 Umgebungsvariablen

Parameter:

Curdir: Gibt das aktuelle Verzeichnis an

Im *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 an

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

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 die *shell.cmd* Datei ausgeführt, welche eine Kommandozeile in Windows mit den vorher im *env.cmd* definierten Parametern startet.

```
SET CURDIR=%~dp0
call %CURDIR%\env.cmd
SET PATH=%CMAKE_HOME%\bin;%PATH%
%HOMEDRIVE%
pushd %CURDIR%
start cmd
```

Abbildung 51 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 übersetzen.

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

```
SET ABI=x86_64
SET ANDROID_CMAKE=%ANDROID_SDK%\cmake\3.6.3155560
rm -r build
del /s /q build
mkdir build
pushd build

SET CMAKE_PROG="%ANDROID_CMAKE%\bin\cmake.exe"
%CMAKE_PROG% -G"MinGW Makefiles" -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=%OPENCV_SOURCE_CODE_DIR%/install
-DANDROID_ABI=%ABI% -G"Android Gradle - Ninja" -DANDROID_PLATFORM=android-24 -DCMAKE_TOOLCHAIN_FILE=%NDK%\build\cmake\android.toolchain.cmake
-DANDROID_NDK=%NDK% -DCMAKE_MAKE_PROGRAM=%ANDROID_SDK%\cmake\3.6.3155560\bin\ninja.exe ..
cmake --build . --target install -- -j4
```

Abbildung 52 openCV Bibltiothek 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 führt der CMake Befehle mit den folgenden Parametern aus:

-G"MinGW Makefiles": Das gibt den Generator der CMake an. Der *MinGW Makefiles* Erzeuger ermöglicht, dass erzeugte M*akefiles* 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 im angegebenen Pfad unter *install zu finden sind.*

-DANDROID_ABI: Ist die gewünschte Systemarchitektur

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

-DANDROID_PLATFORM: Die 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 werden.

-DANDROID_NDK: Gibt den Pfad für das NDK an

-DCMAKE_MAKE_PROGRAM: Gibt den Pfad für den ausführbare Ninja Makefile Erzeuger in Android an.

cmake –build . –target install -- -j4: Im *build* Verzeichnis von der entpackten OpenCV Source Code werden die, von *ninja* erzeugten, CMakeFiles gespeichert. Die übersetzten OpenCV Bibliotheken werden alle in das neu generierte *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 befinden sich alle nötigen erzeugten native Bibliotheken. Damit das Android Projekt diese auch benutzen kann, muss im Projekt unter *app/src/main/* ein Verzeichnis mit dem Namen *jnilibs* erstellt und alle Bibliotheken hier hinzugefügt werden.

8.3.3 OpenCV Bibliotheken in das Projekt einschließen

Nun ist die Basis für die Entwicklung einer Android App mit C++ Code gegeben. Aber noch bevor wirklich gestartet werden kann, muss in der *CMakeLists.txt* eine Datei, welche die Projektdateien und die erzeugten OpenCV Bibliotheken 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.

```
find_package(OpenCV REQUIRED)

set(LIBS "")
foreach(libfile ${OpenCV_LIBS})
    find_library(TMP_${libfile} "${libfile}" HINTS "${OPEN_CV_LIB_DIR}" NO_CMAKE_FIND_ROOT_PATH)
    message(STATUS "found ${TMP_${libfile}} for ${libfile}")
    list(APPEND LIBS ${TMP_${libfile}})
    unset(sharedlib)
endforeach(libfile)

include_directories("${OpenCV_INCLUDE_DIRS}")

file (GLOB mushroom_SRC
    "src/main/cpp/*.cpp"
    "src/main/cpp/*.h"
)

add_library(mushroomlib SHARED
    ${mushroom_SRC}
    ${JNI_HEADER_FILE}
)
```

Abbildung 53 CMakeLists.txt

Zuerst wird versucht, die kompilierte OpenCV Bibliothek zu finden und wenn die Bibliothek gefunden ist, dann wird über die gesamte Büchersammlung iteriert und alle gefundenen 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, welcher dann von *CMakeLists.txt* ins Projekt eingeschlossen wird.

In den nächsten Schritten werden alle sich im Projekt unter *src/main/cpp* befindlichen C++ Codes und Headers in die globale Variable *mushroom_SRC* gespeichert und 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.

8.3.4 Pilz Klasse

Java

```
public class Mushroom {

    private byte[] color = new byte[3];
    private byte[] hsv_v = new byte[3];
    private byte[] hsv_b = new byte[3];
    private byte[] hsv_v2 = new byte[3];
    private byte[] hsv_b2 = new byte[3];
    private String name;
    private String wiki;
    private boolean poisonous;
    private boolean round;
    private boolean lamella;
    private boolean nodule;
    private String stalk;
```

Abbildung 54 Pilzklasse in Java

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

C++

```
class Mushroom {
public:
    /** default constructor */
    Mushroom();
    /** copy constructor */
    Mushroom(const Mushroom&);

    /** assignment operator */
    Mushroom& operator=(const Mushroom& other);

    /** BGR color */
    Vec3b bgr;
    Vec3b hsv_v; //HSV Bereich Begin (von)
    Vec3b hsv_b; //HSV Bereich Ende (bis)
    Vec3b hsv_v2;//HSV Bereich Begin (von)
    Vec3b hsv_b2;//HSV Bereich Ende (bis)
    string name;
    string wiki; //Wikipedia Link
    int lamell;
    int round; //ist der Mushroom Rund, 1 ja, 0 nein
    int poisonous; //ist der Mushroom giftig, 1 ja, 0 nein
    int nodule; //= Knolle, Eigenschaftswort (z. B. dicke, rundliche etc.)
    string stalk;
```

Abbildung 55 Pilzklasse in C++

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

8.3.5 MushroomDetector.java

```
public class MushroomDetector {
    static {
        System.loadLibrary("mushroomlib");
    }
    public native Mushroom[] computeSchwammerlType(Mushroom[] templates, String imagePath);
}
```

Abbildung 56 Überbrückungsmethode zwischen Java und C++

Diese Klasse dient als 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 diese Pilze.

String imagePath: Der Pfad – wo das Bild gespeichert ist – wird mitgegeben

8.3.6 MushroomDetector.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_mushroom_android_cpptest_MushroomDetector */

#ifndef _Included_com_mushroom_android_cpptest_MushroomDetector
#define _Included_com_mushroom_android_cpptest_MushroomDetector
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_mushroom_android_cpptest_MushroomDetector
 * Method:    computeSchwammerlType
 * Signature: ([Lcom/mushroom/android/cpptest/Mushroom;Ljava/lang/String;)[Lcom/mushroom/android/cpptest/Mushroom;
 */
JNIEXPORT jobjectArray JNICALL Java_com_mushroom_android_cpptest_MushroomDetector_computeSchwammerlType
  (JNIEnv *, jobject, jobjectArray, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

Abbildung 57 Header Datei für die Überbrückungsmethode

Das ist die automatisch generierte Header-Datei, die 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 erstellte Array zu *Mushroom[] templates*.

jstring: Das automatisch erstellte jstring zum Pfad des Bildes

8.3.7 MushroomDetector.cpp

Das ist die C++ Klasse, welche das Feld von Pilzen von Java in ein 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 die Bilderkennungsmethode nicht aufrufen kann:

```
#include <jni.h>
#include <cstring>
#include <iostream>
#include <opencv2/highgui/highgui.hpp>

#include <Mushroom.cpp>

#include "JniUtil.h"
#include "GrayScaler.h"
#include "Mushroom.h"
#include "MushroomDetector.h"
```

Abbildung 58 Implementierung der Analysedatei

Die Hauptmethode besteht aus zwei verschiedenen Bereichen:

Aus Java Pilz ein C++ Pilz

```
extern "C"
JNIEXPORT jobjectArray JNICALL Java_com_mushroom_android_cpptest_MushroomDetector_computeSchwammerlType
        (JNIEnv *env, jobject mushroomDetector, jobjectArray templates, jstring imagePath) {
    JniUtil util(env);
    MushroomMarshaller marshaller(env);

    std::string imagePathString = util.toString(imagePath);

    int length = env->GetArrayLength(templates);
    vector<Mushroom> mushrooms;
    jclass elementClass = NULL;
    for (int i = 0; i < length-1; i++) {
        jobject templateElement = env->GetObjectArrayElement(templates, i);
        if (elementClass == NULL) {
            elementClass = env->GetObjectClass(templateElement);
        }
        Mushroom mushroom = marshaller.fromJavaObject(templateElement);
        mushrooms.push_back(mushroom);
    }
    cv::Mat image = readImageFromPath(imagePathString);

    vector<Mushroom> detectedShrooms = detectMushroom(mushrooms, image);
```

Aufruf der Methode für die Bilderkennung in C++

Abbildung 59 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

jstringt imagePath: Der Pfad, wo das Bild gespeichert ist

Da ein *Mat* für die Bilderkennung benötigt wird, wird der Pfad zum Bitmap, welcher lokal im Smartphone temporär gespeichert wurde in *readImageFromPath()* aufgerufen, welche dann über den Pfad aus dem gespeicherten Bild ein *Mat* erstellt.

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

Aus C++ Pilz ein Java Pilz

```
jobjectArray objs = env->NewObjectArray(mushrooms.size(), elementClass, NULL);
jsize index = 0;
for(vector<Mushroom>::iterator it = detectedShrooms.begin(); it != detectedShrooms.end(); it++) {
    jobject object = marshaller.asJavaObject(elementClass, *it);
    env->SetObjectArrayElement(objs, index++, object);
}
return objs;
```

Abbildung 60 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.

8.3.8 MushroomMarshaller

```
class MushroomMarshaller {
public:
/**
 * @param env the jni environment
 */
    MushroomMarshaller(JNIEnv *env) : env(env) {}
/**
 * for signatures see the constants see the output genterated from javap
 * @param obj the Mushroom java object
 * @return the converted c++ Mushroom object
 */
    Mushroom fromJavaObject(jobject obj);
    jobject asJavaObject(jclass, const Mushroom&);
private:
    JNIEnv *env;
};
```

Abbildung 61 Klasse, welche die Umwandlungsmethoden definiert

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

8.3.9 fromJavaObject

Bei dieser Methode wird das *Mushroom*[] Array von Java in einzelne *Mushroom*s für die 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.

```
vector<unsigned char> hsvBS = util.getByteArrayField(obj, "hsv_b2");
for (int i = 0; i < bytes.size(); i++) {
    mushroom.hsv_b2[i] = hsvBS[i];
}
mushroom.name = util.getStringField(obj, "name");
mushroom.stalk = util.getStringField(obj, "stalk");
mushroom.wiki = util.getStringField(obj, "wiki");
mushroom.poisonous = util.getBooleanField(obj, "poisonous");
mushroom.round = util.getBooleanField(obj, "round");
mushroom.lamell = util.getBooleanField(obj, "lamella");
mushroom.nodule = util.getBooleanField(obj, "nodule");
return mushroom;
```

Abbildung 62 Umwandlung einzelner Eigenschaften von Pilzen 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 wurden.

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

8.3.10 AsJavaObject

```
jobject MushroomMarshaller::asJavaObject(jclass clazz, const Mushroom& mushroom) {
    JniUtil util(env);

    jmethodID fid = env->GetMethodID(clazz, "<init>", "()V");
    jobject object = env->NewObject(clazz, fid);

    util.setStringField(object, "name", mushroom.name.c_str());
    util.setStringField(object, "nodule", mushroom.stalk.c_str());
    util.setStringField(object, "wiki", mushroom.wiki.c_str());
    util.setBooleanField(object, "poisonous", mushroom.poisonous);
    util.setBooleanField(object, "round", mushroom.round);
    util.setBooleanField(object, "lamell", mushroom.lamell);
    util.setBooleanField(object, "stalk", mushroom.nodule);

    return object;
}
```

Abbildung 63 Umwandlung einzelner Eigenschaften von Pilzen 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 wieder einmal die Namen der Eigenschaften der Pilze aus der Seite von Java, damit die übergebenen Eigenschaften der Pilze aus der Seite von C++ in die einzeln definierten Variablen in der *Mushroom.java* Klasse gespeichert werden können. Zusätzlich wird das *object*, in dem der umgewandelte Pilz gespeichert wird, und der zu speichernde Wert der Eigenschaften, mitgegeben.

8.3.11 JniUtil.h

```
class JniUtil {
public:
    JniUtil(JNIEnv *env) : env(env) {
    }
    string toString(jstring jniString);
    vector<unsigned char> getByteArrayField(jobject object, const char *name);
    void setBytearrayField(jobject, const char *name, char *bytes, int length);

    string getStringField(jobject object, const char *name);
    void setStringField(jobject object, const char *name, const char *value);

    int getBooleanField(jobject object, const char *name);
    void setBooleanField(jobject, const char *name, int value);
private:
    JNIEnv *env;
};
```

Abbildung 64 Definierung der Methoden für die Umwandlung einzelner Eigenschaften

Die Methoden für die Umwandlung einzelner Pilzeigenschaften werden hier definiert.

8.3.12 JniUtil.cpp

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

getByteArrayField()

```
vector<unsigned char> JniUtil::getByteArrayField(jobject obj, const char *name) {
    jclass klass = env->GetObjectClass(obj);
    jfieldID fid = env->GetFieldID(klass, name, "[B");

    jobject colorObject = env->GetObjectField(obj, fid);
    jbyteArray& byteArray = reinterpret_cast<jbyteArray&>(colorObject);
    int length = env->GetArrayLength(byteArray);
    jbyte *color = env->GetByteArrayElements(byteArray, NULL);
    vector<unsigned char> bytes;
    for (int i = 0; i < length; i++) {
        bytes.push_back(color[i]);
    }
    env->ReleaseByteArrayElements(byteArray, color, 0);
    return bytes;
}
```

Abbildung 65 Java byte[] in C++ vec3b

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

Set/GetStringField()

```
string JniUtil::getStringField(jobject obj, const char *name) {
    jclass klass = env->GetObjectClass(obj);
    jfieldID fid = env->GetFieldID(klass, name, "Ljava/lang/String;");
    jstring stringObject = (jstring)env->GetObjectField(obj, fid);
    return stringObject ? toString(stringObject) : "";
}
void JniUtil::setStringField(jobject object, const char *name, const char *value) {
    jclass klass = env->GetObjectClass(object);
    jfieldID fieldId = env->GetFieldID(klass, name, "Ljava/lang/String;");
    jstring val = env->NewStringUTF(value);
    env->SetObjectField(object, fieldId, val);
}
```

Abbildung 66 Methoden für die Umwandlung von Textvariablen

Bei diesen Methoden erfolgt die Umwandlung jeweils von Java String zu C++ String und wieder umgekehrt. Dazu wird wieder die Signatur und das Objekt, welches mitgegeben wird, benötigt.

Set/GetBooleanField()

```
int JniUtil::getBooleanField(jobject object, const char *name) {
    jclass klass = env->GetObjectClass(object);
    jfieldID fid = env->GetFieldID(klass, name, "Z");
    jboolean val = env->GetBooleanField(object, fid);
    if(val){
        return 1;
    } else{
        return 0;
    }
}

void JniUtil::setBooleanField(jobject object, const char *name, int value) {
    jclass klass = env->GetObjectClass(object);
    jfieldID fid = env->GetFieldID(klass, name, "Z");
    if(value == 1){
        env->SetBooleanField(object, fid, JNI_TRUE);
    }else{
        env->SetBooleanField(object, fid, JNI_FALSE);
    }
}
```

Abbildung 67 Methoden für Umwandlung von boolean Variablen

Bei 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 einfaches 0 und 1 in C++ gespeichert und umgekehrt.

8.3.13 Bitmap zu Mat

```
private String saveToInternalStorage(Bitmap bitmapImage){
    ContextWrapper cw = new ContextWrapper(getApplicationContext());
    // path to /data/data/yourapp/app_data/imageDir
    File directory = cw.getDir("imageDir", Context.MODE_PRIVATE);
    // Create imageDir
    File mypath=new File(directory,"profile.jpg");

    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(mypath);
        // Use the compress method on the BitMap object to write image
        bitmapImage.compress(Bitmap.CompressFormat.PNG, 100, fos);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return directory.getAbsolutePath();
}
```

Abbildung 68 Umwandlung von Bitmap zu Mat

Bilder in Android Studio werden als *Bitmap* gespeichert, für die Bilderkennung in C++ wird ein *Mat* benötigt, dafür wird im Smartphone mit dieser *Methode* eine temporäre Datei erstellt, das *Bitmap* in dieser gespeichert und der Pfad in die *JNI* Methode mitgegeben. In dieser Methode wird eine *Mat* Variable erstellt und mit der C++ Funktion *imread(picturePath) das gespeicherte Bild in ein* Mat eingelesen.

8.3.14 MainActivity.java

In dieser Klasse erfolgen die allgemeinen Funktionen zum Aufbauen der App. Wenn ein Foto geschossen oder von der Galerie übernommen wird, wird der Pfad dieser in einer String Variable gespeichert. Die JNI-Bridge-Methode – also *computeSchwammerlType* – wird daraufhin aufgerufen und das Feld von Pilzen und der gespeicherte Pfad des Bildes

übergeben. Das Ergebnis wird dann gespeichert und je nachdem ob das Ergebnis eindeutig war oder nicht, werden Fragen zum Pilz gestellt oder der analysierte Pilz sofort angezeigt.

8.3.15 Probleme

8.3.15.1.1 Standardbibliotheken von Android unterstützen wichtige native C++ Funktionalitäten nicht

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

Diese Funktion wandelt einen *String* Text in ein *Integer um*.

8.3.15.1.2 Lösung:

Nach Absprache mit Diplomarbeitsbetreuer Professor Aberger, wurde die Methode, welche die Pilze aus der XML-Datei einliest, in eine eigene C++ Datei gespeichert.

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

8.3.16 Grund für die Nicht-Fertigstellung der Android-App

Die Methode für die Bilderkennung in C++ hat eine Liste/*vector von* Pilzen mit keinen Werten zurückgegeben. Also wurden nie Pilze erkannt, die dann tatsächlich zurückgegeben werden. Es gab auch Fehler bei der Umwandlung von C++ *vectoren* in *Arrays* in Java, besonders *NullPointerExceptions*, welche vom vorherigen Problem resultierten.

Ein Grund für diesen Fehlschlag kann durch eine fehlerhafte Speicherung des Bildes legen, von dem ein Pfad als Parameter übergeben wird, welches dann als ein C++ image *Mat* durch die Methode *imread(pfad)* gespeichert wird.

9 Qualitätssicherung

Eierschwammerl

Abbildung 69: Testen unterschiedlicher Eierschwammerl (IOS)

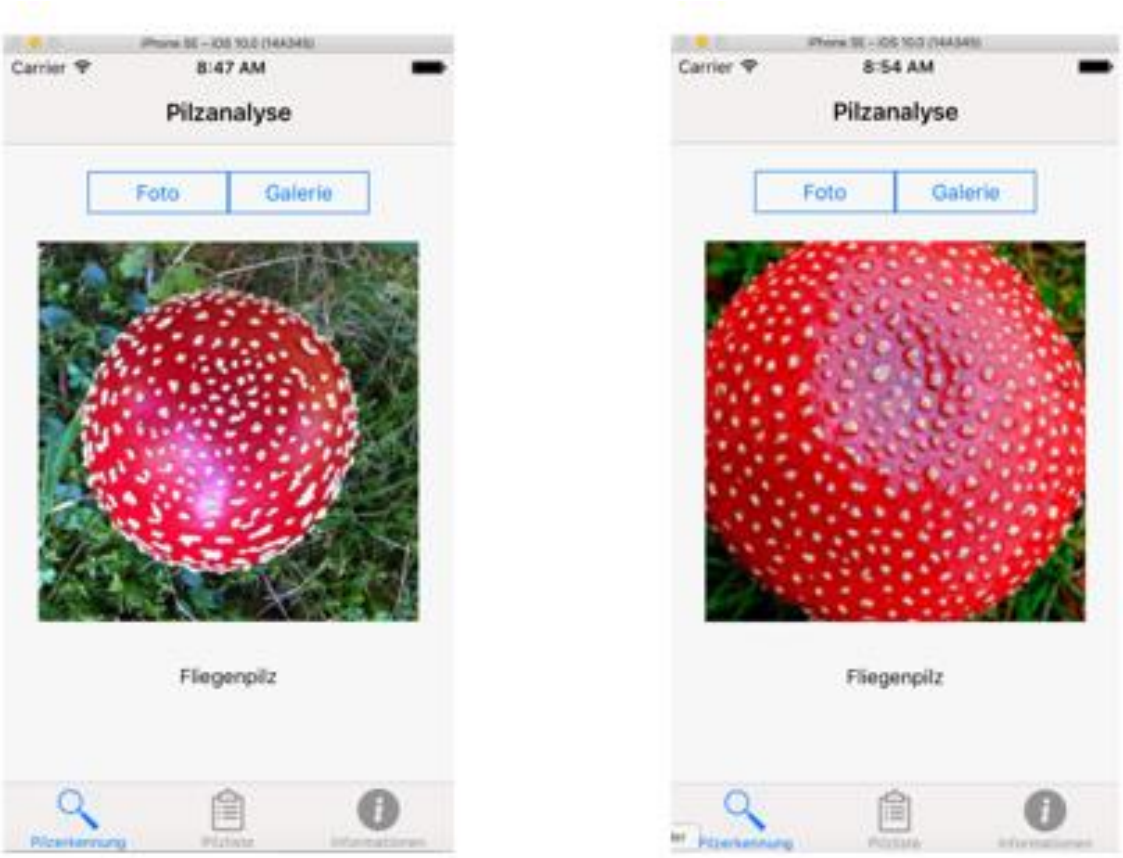


Abbildung 70: Testen unterschiedlicher Fliegenpilze (IOS)

9.1 Vergleich mit Konkurrenzprodukten

9.1.1 Meine Pilze (Pilzbestimmung) Entwickler: Meine Pilze

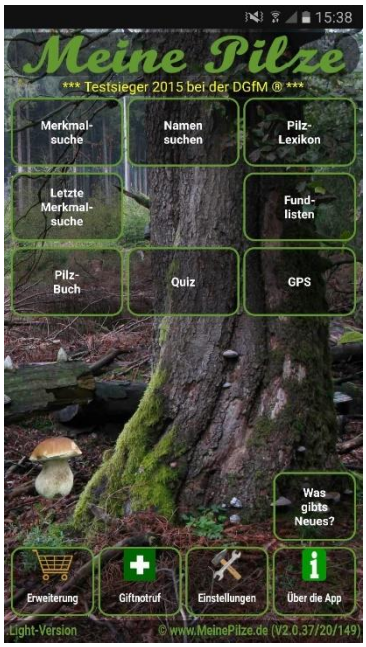


Abbildung 71: Meine Pilze App Screenshot

Vorteile:

Eine Fundliste kann gefüllt werden

(in der kostenpflichtigen Variante) Größere Pilzdatenbank

Zusätzliche Quiz Fragen, um sich 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“

9.1.2 Pilze Entwickler: Kirill Sidorov

Abbildung 72: Pilze App Screenshot

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 Pilz, wird der Wikipedia-Text zu dem Pilz angezeigt.

9.1.3 Pilzführer Nature Lexicon

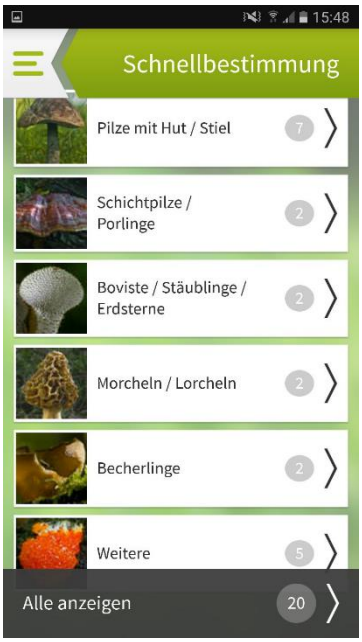


Abbildung 73: Pilzführer Nature Lexicon App Screenshot

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 / Lorcheln) in einer Baumstruktur untergliedert.

9.1.4 Fazit:

Mushroom Identifier ist die innovativste und am besten designte App, die es für Smartphones gibt.

Der einzige erkennbare Nachteil gegenüber den anderen Apps ist die kleinere Pilzdatenbank in Mushroom Identifier. Eine größere ist jedoch nicht Teil dieser Diplomarbeit. Dieses Problem kann in einem Folgeprojekt gelöst werden, indem mit einem der bestehenden Pilzapp-Hersteller kooperiert wird und auf deren Daten zugegriffen wird.

Vorteil für Mushroom Identifier: Es kann auf bestehende Daten zugegriffen werden.

Vorteil für bestehende Apps: zusätzlich automatische Pilzerkennung

10 Zusammenfassung

10.1 Ergebnis

Als Ergebnis ist nun eine App für IOS entstanden, die mithilfe von Computervision, maschinellem Lernen und Benutzerfragen Pilze erkennt.
Dabei greift die Apps auf ein C++ Programm zu, das in Windows programmiert wurde. Es wurde also eine plattformunabhängige Standardversion für die Bilderkennung erstellt.
Der ebenfalls geforderte Android Teil scheiterte

10.2 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 und um das beste Ergebnis zu erzielen. Darüber hinaus war es ein schwieriges Unterfangen den C++ Code sowohl auf IOS als auch vor allem für die Android App zu kompilieren, um die gemeinsame Codebasis zu erreichen.
Der Android Teil konnte in der Zeit der Diplomarbeit aufgrund der in 8.3.16 beschriebenen Problemen leider nicht vervollständigt werden.
Beim maschinellen Lernen war uns Anfangs nicht klar, wie viel Zeit das Durchlaufen des Trainings-Algorithmus in Anspruch nimmt. Da jedoch früh genug damit angefangen wurde, war dieser zeitliche Aufwand kein Problem und es konnte in der Zwischenzeit an der App selbst gearbeitet werden.
Bei der Zeitplanung wurde vor allem der Aufwand des Kompilierens des C++ Codes für die Android Plattform unterschätzt.

11 Literatur und Quellen Verzeichnis

11.1 Abbildungsverzeichnis

11.2 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

CMake. (04. April 2017). *CMake*. Von https://cmake.org/ abgerufen

CMake Logo. (2017). Von https://mspoweruser.com/for-developers-windows-phone-gets-cmake-support/ abgerufen

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:. Von http://www.shervinemami.info/colorConversion.html abgerufen

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

Gradle Inc. (04. April 2017). *Gradle*. Von https://gradle.org/ abgerufen

Gradle Logo. (2017). Von http://blog.iteedee.com/2014/01/android-gradle-tutorial/ 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. (16. 12 2016). *docs.opencv.org*. Von http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html 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

opencv dev team. (3. 4 2017). *docs.opencv.org*. Von http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html#cascade-training abgerufen

Pound, D. M. (11. 11 2015). *https://www.youtube.com*. Von https://www.youtube.com/watch?v=sRFM5IEqR2w&t=188s abgerufen

Rajput, Mehul. (20. Mai 2015). *dzone*. Von https://dzone.com/articles/why-android-studio-better 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

Sinha, U. (2010). *aishack.in*. Von http://aishack.in/tutorials/integral-images-opencv/ 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

Visual Studio Logo. (2017). Von https://commons.wikimedia.org/wiki/File:Visual_Studio_2012_logo_and_wordmark.svg abgerufen

Wagner, C. (2005/2006). *Kantenextraktion Klassische Verfahren.* Von http://www.mathematik.uni-ulm.de/stochastik/lehre/ws05_06/seminar/ausarbeitung_wagner.pdf abgerufen

Wagner, C. (WS 2005, 2006). *Kantenextraktion*. Von Klassische Verfahren. abgerufen

Wikipedia. (10. 03 2017). *Wikipedia C++*. Von https://de.wikipedia.org/wiki/C%2B%2B abgerufen

Wikipedia. (04. 04 2017). *Wikipedia OpenCV*. Von https://de.wikipedia.org/wiki/OpenCV abgerufen

Wikipedia. (03. 03 2017). *Wikipedia Visual Studio*.

Wikipedia, Faltungsmatrix. (27. 03 2017). *Faltungsmatrix*. Von https://de.wikipedia.org/wiki/Faltungsmatrix abgerufen

Wikipedia, JNI. (20. 02 2011). *Wikipedia, JNI*. Von https://en.wikipedia.org/wiki/Java_Native_Interface abgerufen

XCode Logo. (2017). Von http://logonoid.com/xcode-logo/ abgerufen

Youtube. (02. 10 2015). *How Blurs & Filters Work - Computerphile*. Von https://www.youtube.com/watch?v=C_zFhWdM4ic abgerufen

12 Im Anhang

12.1 Source Code

- OpenCV/C++
- Haar Cascade
- Windows
- Android
- iOS

12.2 Dokumentation

- Projektstrukturplan
- Projektdefinition
- Use-Case-Diagramm
- Zeitplan
- Risikomanagement
- Aufwandsschätzung
- Stundenaufwand