  
HTL - Perg

Höhere Abteilung für EDV und Organisation

Diplomarbeit

Mushroom Identifier

Projektteam: Jakob Froschauer

Hakan Abbas

Markus Arbeithuber

Projektbetreuer: Prof. Dipl.-Ing. Christian Aberger

Bearbeitungszeitraum: 01.10.2016 – 30.04.2017

**Eidesstattliche Erklärung**

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

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

(Name)

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

(Name)

Danksagung

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

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

Herzlichen Dank!

[Impressum 5](#_Toc476246168)

[1. Schule 5](#_Toc476246169)

[2. Schuljahr 5](#_Toc476246170)

[3. Klasse 5](#_Toc476246171)

[4. Projektname 5](#_Toc476246172)

[Verwendete Technologien 13](#_Toc476246173)

[Android Studio 13](#_Toc476246174)

[Android Studio NDK 13](#_Toc476246175)

[CMake 14](#_Toc476246176)

[Gradle 14](#_Toc476246177)

[Boost ( C++ Libraries ) 14](#_Toc476246178)

Impressum

1. Schule

HTBLA Perg für Informatik

Marchlandstraße 48

4320 Perg

1. Schuljahr

2016/2017

1. Klasse

5AHIF

1. Projektname

Mushroom Identifier

# Projektleiter

Jakob Froschauer

Projektteam

Hakan Abbas

Markus Arbeithuber

# Betreuungslehrer

Dipl.-Ing. Christian AbergerEinleitung

Kurzfassung

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

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

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

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

Motivation

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

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

Technische Grundlagen

Technologien

Java (Android)

Swift/Objective-C (IOS)

Die grafische Oberfläche der IOS App wurde in Swift programmiert, da es die effizienteste und performanteste Möglichkeit ist Apps für IOS zu programmieren.

C++ (OPEN CV)

Android Studio (Android)

XCode (IOS)

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

Visual Studio (OPEN CV)

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

Entscheidung für Native Apps

Die Anwendung soll dem Nutzer ein Benutzungserlebnis („Look and Feel“) bieten, das er auf den jeweiligen Plattformen gewohnt ist. Trotz vieler APIs für den Kamerazugriff oder Zugriff auf das Filesystem ist es für unsere Anwendung angenehmer und performanter direkt solche Funktionen zuzugreifen.

Für unsere Tätigkeiten wird C++ mit der Programmbibliothek OPENCV verwendet.

OpenCV (Open Source Computer Vision)

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

CMARKUP (XML Lesen)

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

Computer Vision

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

Mensch-Computer-Interaction

Mensch-Computer-Interaction

Haar Cascade Training

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

Organisation

Aufgetretene Probleme

Der offizielle OpenCV Installationsguide 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.

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

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

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

*“Because of Roman horse’s ass!”*

Diese Aussage hat der Blogger und Interviewführer Satya Mallick so interpretiert, dass der Grund für die Verwendung vom BGR Farbformat sei, dass, als OpenCV entwickelt wurde, sowohl Camerahersteller und Softwarehersteller eher das BGR Format 0x00bbggrr verwendeten. Also sei es eine rein historische Entscheidung gewesen.

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

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

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

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

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

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

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

Nicht gewusst, dass Circle Transformation den Canny selbstständig durchführt. Davor habe ich ihn selbstständig davor gemacht

Der Identifiaktionsprozess besteht aus insgesamt drei Phasen:

Die Bilderkennung

Das maschinelle Lernen

Benutzerfragen

Funktionsweise Bilderkennung

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

C++ (einlesen eines Fotos)

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

Einlesen des XMLs:

Lesen von XML Datei (CMARKUP):

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

{

xml.IntoElem();

counter\_str = to\_wstring(counter);

pilz = ws + counter\_str;

//wcout << "pilz: " << pilz;

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")));

...

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

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

Code Snippet:

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];

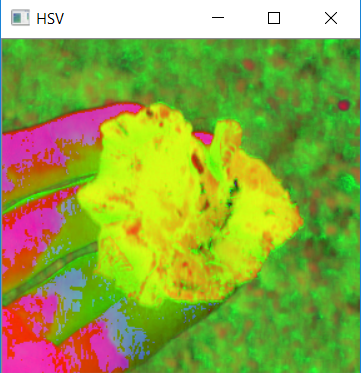
}

}

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

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

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

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.

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]);

cout << "\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);

}

}

if (mushlist2.size() == 0) {

return mushlist2;

}

*Kurzbeschreibung:*

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

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

Code Snippet:

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

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

/// Reduce noise with a kernel 3x3

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

/// Canny detector

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

Und mit dem Ergebnis nach Kreisen gesucht.

Code Snippet:

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

{

std::vector<Vec3f> circles;

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

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

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

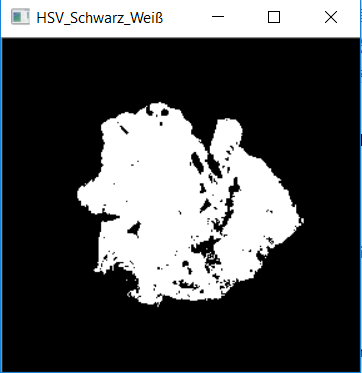
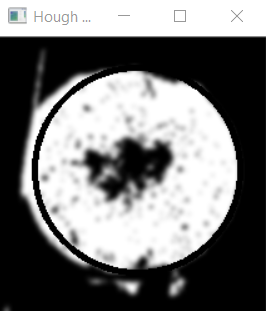
}

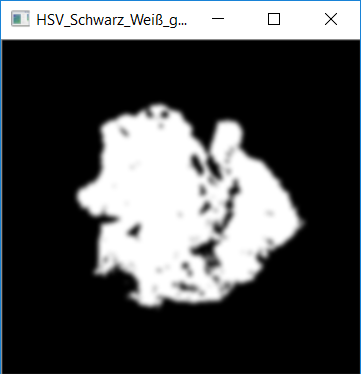
// shows the results

return (circles.size());

}

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





Daraufhin wird der Algorithmus für das Maschinelle Lernen durchgeführt.

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

Folgende Schritte mussten dafür vollzogen werden:

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

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

Python wird benötigt, um ein später beschriebenes Script auszuführen.

Suchen von positiven und negativen Bildern. Das heißt, es werden Bilder, auf denen ein Fliegenpilz zu sehen ist, und solche, auf denen kein Fliegenpilz zu sehen ist, gesucht und in die Ordner „positive\_images“ und „negative\_images“ gespeichert. Alle diese Bilder sollten aus der Vogelperspektive gemacht worden sein. Wichtig ist, dass sie sich auch in Helligkeit und Hintergrund unterscheiden. Als negative Bilder bieten sich Fotos von anderen Pilzen und Fotos, auf denen nur Waldboden zu sehen ist, an.

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

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

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

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

aus.

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

Beispielsweise:

Hat der Pilz Lamellen?

Der Benutzer

Dieses kann mit JA oder NEIN beantwortet werden.

Verwendete Funktionen

imread("C:\\Users\\Jakob\\Documents\\Visual Studio 2015\\Projects\\OPENCV31\\data\\falscher\_pfifferling.jpg");

//umwandlung von BGR in HSV

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

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);

// Gasusscher Weichzeichner

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

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

HoughDetection(src\_gray, src\_gray, 99, 41);

//AUSGABE//////////////////////////////////////////////////////////////////////////////

namedWindow("Pilz\_ohne\_Aenderung", WINDOW\_AUTOSIZE); // Pilz ohne Änderung

int detectAndDisplay(Mat frame); //Maschinelles Lernen; Fliegenpilzerkennung

vector<Pilz> readxml(); //Lesen der PilzXML

void CannyThreshold(int, void\*); //Umrisse werden erkannt

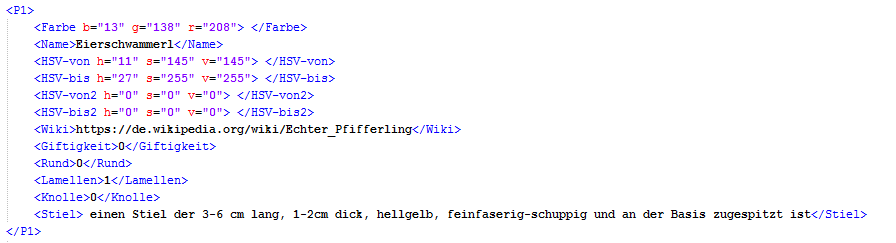
vector<Pilz> oneornull(vector<Pilz> mushlist2, wstring question); // 1/0 Entscheidungsfragen

vector <Pilz> roundornot(vector <Pilz> mushlist, int amountofcircles); //ist der Pilz Rund oder nicht?

vector <Pilz> questions(vector <Pilz>mushlist); //Ausführliche Entscheidungsfragen

int HoughDetection(const Mat& src\_gray, const Mat& src\_display, int cannyThreshold, int accumulatorThreshold); //Hough Circle Detection

Datenspeicherung (XML)



Entscheidung für die Datenspeicherung (XML)

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;

};

Fragenstellung

Hat der Pilz Lamellen?

Hat der Pilz eine Knolle?

Hat der Pilz <Lange Frage einfügen>?

//Errechnen, ob der Pilz ein Eierschwammerl sein könnte (Farbe)

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);

}

}

Lesen von XML File (CMARKUP):

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

{

xml.IntoElem();

counter\_str = to\_wstring(counter);

pilz = ws + counter\_str;

//wcout << "pilz: " << pilz;

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")));

...

Verwendete Technologien

Android Studio

Android Studio ist die offizielle integrierte Entwicklungsumgebung ( DIE ) für die Android Plattform. Es wurde am 16. Mai 2013 bei der Google I/O Konferenz angekündigt. Android Studio ist unter dem Apache License 2.0 frei zugänglich.

Android Studio ist speziell für die Android Entwicklung designt geworden. Sie kann frei für Windows, macOS und Linux gedownloadet werden. Android Studio hat Eclipse Android Development Tools ( ADT ) als Googles primäre DIE für nativen Android Applikationen Entwicklung ersetzt.

Wieso Android Studio so vorteilhaft ist:

* **Gradle Integration**: Android Studio benutzt den rasant wachsenden Gradle build System. Gradle automatisiert und liefert bessere Software schneller.
* **Erweiterte Code Ergänzung**: Android Studio liefert präzise Code Ergänzungen, welches für ein Programmiere viel Zeit und unnötiges langes Debugging spart.
* **User Interface**: Android Studio hat eine sehr benutzerfreundliche User Interface als ein Einsteiger hat man keine Probleme beim Zurrechtfinden im Programm.
* **Organisierung des Projektes**: Android Studio benutzt Module, welche alle eine eigene Gradle build File besitzen, die seine eigene Abhängigkeiten angeben kann. Außerdem hat Android Studio eine Funktion, welche das zuletzt geschlossene Projekt beim Start öffnet, welches unnötige Zeit spart.
* **System Stabilität**: Android Studio hat eine stabile Performance hat weniger Bugs und die benötigten Systemeigenschaften sind auch sehr niedrig.
* **Drag and Drop**: Android Studio hat eine Drag and Drop Funktion eingebaut, welches über den GUI ( Graphical User Interface ) benutzt werden kann.

Android Studio NDK

Die Native Development Kit ( NDK ) ist eine Satz von Werkzeugen welches dir erlaubt C und C++ Codes mit Android zu verwenden und er bietet Platform Bibliotheken welches du benutzten kannst um native Aktivitäten und physische Gerätekomponente zu verwalten, wie Sensoren und Touch-Eingabe. Die NDK könnte vielleicht für Android Novizen nicht das richtige Tool sein, welche nur den Java Code und Framework APIs zum entwickeln ihres Apps benötigen. Aber, die NDK kann für ein Paar Fälle nützlich sein:

* Zusätzliche Leistung aus einem Gerät herauszudrücken, um eine geringe Latenz zu erreichen oder rechenintensive Anwendungen wie Spiele oder Physik-Simulationen durchzuführen
* Wiederverwendung Bibliotheken sei es von ihnen oder von anderen Entwicklern.

Mit Android Studio 2.2 und höher können Sie mit dem NDK C und C ++ - Code in eine native Bibliothek kompilieren und in Ihr APK mit dem integrierten Build-System von IDEs durch Gradle verpacken. Ihr Java-Code kann dann Funktionen in Ihrer nativen Bibliothek über das Java Native Interface (JNI) Framework aufrufen.

Das Standard-Build-Tool von Android Studio, um native Bibliotheken zu kompilieren, ist CMake. Android Studio unterstützt auch ndk-build aufgrund der großen Anzahl von vorhandenen Projekten, die das Build-Toolkit verwenden. Allerdings, wenn Sie eine neue native Bibliothek erstellen, sollten Sie CMake verwenden.

CMake

CMake ist eine Open-Source-, Cross-Plattform-Familie von Tools entwickelt, um zu bauen, zu testen und Software zu verpacken. CMake wird verwendet, um den Software Compilationsprozess mit einfachen Plattform- und Compiler-unabhängigen Konfigurationsdateien zu steuern und native Makefiles und Workspaces zu erzeugen, die in der Compiler-Umgebung Ihrer Wahl verwendet werden können. Die Suite von CMake-Tools wurde von Kitware als Antwort auf die Notwendigkeit einer leistungsstarken, plattformübergreifenden Buildumgebung für Open-Source-Projekte erstellt.

CMake ist Teil der Kitware-Sammlung von kommerziell unterstützten Open-Source-Plattformen für die Softwareentwicklung.

Gradle

Gradle ist ein Open-Source-Automatisierungssystem, das auf den Konzepten von Apache Ant und Apache Maven aufbaut und eine groovy-basierte Domain-spezifische Sprache (DSL) anstelle des von Apache Maven verwendeten XML-Formulars zur Deklaration der Projektkonfiguration einführt. Gradle verwendet einen gerichteten azyklischen Graphen ("DAG"), um die Reihenfolge zu bestimmen, in der Aufgaben ausgeführt werden können.

Gradle wurde für Multi-Projekt-Builds entworfen, die sehr groß werden können, und unterstützt inkrementelle Builds, indem sie intelligent bestimmen, welche Teile des Build-Baums aktuell sind, so dass jede Aufgabe, die von diesen Teilen abhängig ist, nicht benötigt wird – nicht wieder ausgeführt werden

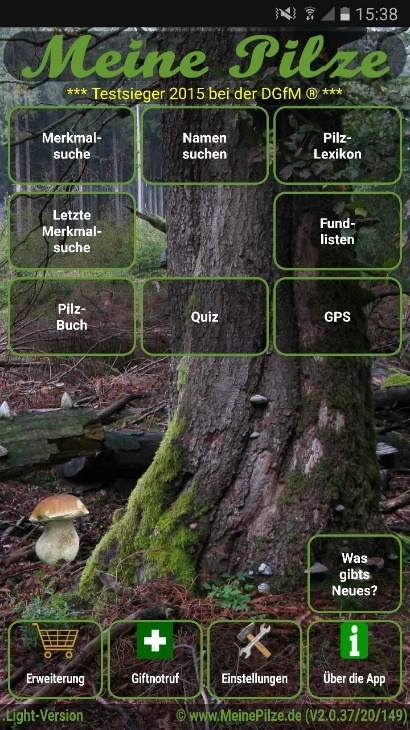
Die ersten Plugins konzentrieren sich primär auf Java, Groovy und Scala Entwicklung und Implementierung, aber mehr Sprachen und Projekt-Workflows im Ziel.

Boost ( C++ Libraries )

Boost ist ein Satz von Bibliotheken für die C ++ - Programmiersprache, die Unterstützung für Aufgaben und Strukturen wie lineare Algebra, Pseudozufallszahlengenerierung, Multithreading, Bildverarbeitung, reguläre Ausdrücke und Unit-Tests bieten. Es enthält über achtzig einzelne Bibliotheken.

Die meisten Boost-Bibliotheken sind unter der Boost Software-Lizenz lizenziert und sollen Boost sowohl mit kostenlosen als auch proprietären Softwareprojekten nutzen. Viele von Boosts Gründern sind auf dem C ++ - Normenausschuss, und mehrere Boost Bibliotheken wurden für den Einbau in den C ++ Technical Report 1 und den C ++ 11 Standard akzeptiert.

Vergleich mit anderen Programmen

Meine Pilze (Pilzbestimmung) Entwickler: Meine Pilze

Eine App, bei der man nach Pilznamen, nach Pilzeigenschaften suchen kann.

Vorteile:

Eine Fundliste kann gefüllt werden

(in der Kostenpflichtigen Variante) Größere Pilzdatenbank

Zusätzliche Quiz Fragen um sich mit weiterzubilden

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Sehr umständlich Pilze zu bestimmen.

Pilze Entwickler: Kirill Sidorov

Vorteile:

Größere Pilzdatenbank

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Keine Möglichkeit der Merkmalsuche.

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

Pilzführer Nature Lexicon

Vorteile:

Größere Pilzdatenbank

Nachteile:

Keine Bilderkennung

Design (entspricht nicht den Designrichtlinien der Plattformen)

Pilzerkennung

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

Fazit:

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

Der Einzige erkennbare Nachteil gegenüber den anderen Pilzen ist die kleinere Pilzdatenbank, die jedoch nicht Teil dieser Diplomarbeit ist.

Quellen

Cascade Training:

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

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