

ENTWICKLUNG EINER
OPTICAL-FLOW-BASIERTEN
ANDROID-APPLIKATION ZUR
LANDEPLATZLOKALISIERUNG UNTER
VERWENDUNG VON OPENCV

PROJEKTDOKUMENTATION

HOCHSCHULE BOCHUM
Campus Velbert/Heiligenhaus
Höseler Platz 2
42579 Heiligenhaus

INHALTSVERZEICHNIS

1	Einführung	1
1.1	Motivation	1
1.2	Zielsetzung	1
2	Grundlagen	2
2.1	Android	2
2.1.1	Einführung	2
2.1.2	Warum Android?	3
2.2	OpenCV	4
2.2.1	Einführung	4
2.2.2	Warum OpenCV?	5
2.3	Java Native Interface	5
3	Entwicklungsumgebung	7
3.1	Auswahl der Entwicklungsumgebung	7
3.2	Einrichtung der Entwicklungsumgebung	7
4	Entwicklung der Applikation: Allgemeines OpenCV-Android-Framework	8
4.1	Genereller Aufbau	8
4.2	Teil 1: Java	8
4.2.1	openCVActivity.java	8
4.2.2	openCVFramework.java	14
4.3	Teil 2: C++	16
4.3.1	openCVFramework.h	16
4.3.2	openCVFramework.cpp	17
4.4	Teil 3: XML	20
4.4.1	AndroidManifest.xml	20
4.4.2	Layout	22
5	Entwicklung der Applikation: Visual Based Landing System	23
5.1	Genereller Aufbau	23
5.2	VBLSActivity.java	23
5.3	HoughCircleTransformation.java	26
5.4	HoughCircleTransformation.cpp	27
5.5	USB-Kommunikation	28
5.5.1	Anforderungen an die zu verwendende Bibliothek	28
5.5.2	Einbindung der Bibliothek	28
5.6	PID.java	32
5.7	AndroidManifest.xml	35
6	Versuchsaufbau und Tests	37
6.1	Versuchsaufbau	37
6.2	Mischer	38
6.3	Einstellung des Reglers	39
6.4	Ergebnisse	40
7	Vergleich Zielsetzung-Endergebnis	42
8	Ausblick	43
9	Literatur	45

10 Anhang	46
10.1 Persönliches Fazit	46
10.2 Lizenzen	47
10.3 gitlab-Repository	48
10.4 Eidesstattliche Erklärung	48

ABBILDUNGSVERZEICHNIS

2.1	Lebenszyklus einer Android-Applikation [Goo16a]	2
2.2	Funktionsdiagramm des Java Native Interface, in Anlehnung an [Lia99, S.5]	5
4.1	Funktionsdiagramm des Frameworks	8
5.1	Funktionsdiagramm des Visual Based Landing Systems	23
6.1	Draufsicht auf die Oberseite des Versuchsaufbaus	37
6.2	Draufsicht auf die Unterseite des Versuchsaufbaus	38
6.3	Ansicht des Versuchsaufbaus im Profil	39
6.4	Nahansicht des Versuchsaufbaus im Profil	40

LISTINGS

4.1	Set Up	9
4.2	OpenCV Loader	9
4.3	onCreate	9
4.4	onPause	10
4.5	onResume	10
4.6	Überprüfen der Kamera-Berechtigung	11
4.7	onDestroy	12
4.8	onCameraFrame	12
4.9	Unterscheidung des Bildformats	14
4.10	Laden der vorkompilierten Bibliothek	14
4.11	Konvertierung des Datenformats der Bilder	15
4.12	Implementation des JNI's auf Java-Seite	15
4.13	openCVFramework.h	17
4.14	Deklaration der nativeImageProcession-Funktion	17
4.15	Konvertierung der Java-Datentypen zu nativen Datentypen	18
4.16	Wiederherstellung der Bildmatrix aus dem übergebenen Array	18
4.17	Bildverarbeitung	19
4.18	Konvertierung der Rückgabeargumente	19
4.19	Java-Paket und min. Android-Version	20
4.20	Aufbau der Applikation	20
4.21	Ergänzen der Applikation um weitere Android-Komponenten	21
4.22	Mindestens benötigte SDK-Version	21
4.23	Berechtigungen und Features	21
5.1	Festlegung des Aufnahmeformats	24
5.2	Implementierung von onCameraFrame	24
5.3	calculateNearestCircle	26
5.4	Kreisdetektion	27
5.5	Transformation von Vektor zu Array	27
5.6	Initialisierung des USB-Daten-Arrays	29
5.7	Aufbauen der Service-Connection	29
5.8	Einrichtung des Broadcast-Receivers	30
5.9	Einrichten und Weiterreichen des USB-Handlers	31
5.10	Hilfsklasse für Koordinatentupel	32
5.11	Erläuterung der Parameter	32
5.12	Lesender und schreibender Zugriff auf threadsichere Objekte	33
5.13	Berechnung der Stellgrößen	34
5.14	Aufbau der Applikation ergänzt um Intent-Filter und USB-Service	35
5.15	Verwendete Features	36

1 EINFÜHRUNG

1.1 Motivation

Längst hat der Trend der zunehmenden Automatisierung nicht nur in der Industrie sondern auch im normalen Alltag Einzug gehalten. In Fabriken übernehmen Roboter verschiedenste Aufgaben von der Montage bis hin zu Qualitätskontrolle, auf den Straßen sind Autos in der Lage, Hindernisse selbstständig zu erkennen und eine Notbremsung einzuleiten und in Gebäuden passt sich die Beleuchtung automatisch dem aktuellen Betriebszustand an. Doch um diese Funktionen umsetzen zu können, ist eine geeignete Sensorik bzw. Datenerfassung von essentieller Bedeutung.

Einen Teil dieses Gebiets, der zunehmend an Bedeutung gewinnt, stellt die Bildverarbeitung dar. Für viele Problemstellungen mit Variablen erweist sich diese Art der Datenerfassung als geeigneter als die klassische Sensorik oder gar als einzig mögliche Lösung. In der Industrie kann somit beispielsweise ein Produkt bei der Qualitätskontrolle auf ein bestimmtes Merkmal überprüft werden oder der korrekte Bestückungszustand eines Förderbands kann aus Sicherheitszwecken validiert werden, bevor ein Roboter die Werkstücke greift. Im Alltag kann diese Technik unter anderem bei der erwähnten Hinderniserfassung durch ein Auto zur Verwendung kommen. Gerade im Bereich der Robotik, wo man bei der Entwicklung mobiler Roboter häufig das Problem unbekannter Einsatzumgebungen hat, bieten sich viele Einsatzmöglichkeiten für die Bildverarbeitung, wie zum Beispiel ein Multikopter, welcher von alleine Ladestationen lokalisieren und ansteuern kann, um so über längere Zeit hinweg autonom agieren zu können [vgl. Coc+15, S.1].

Mit dieser Motivation behandelt diese Arbeit einen einfach zugänglichen, flexibel erweiterbaren und kostengünstigen Ansatz mit variablen Einsatzgebieten, welcher auch Privatpersonen ohne ausgeprägte Programmier-Vorkenntnisse die Verwendung von Bildverarbeitungssoftware ermöglichen soll.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung einer Optical-Flow-basierten Applikation, welche eingehende Bilddaten einer Kamera verarbeiten und definierte Umgebungsmerkmale erfassen und herausstellen kann. Anhand dieser soll anschließend ein Ausgangssignal abhängig von der aktuellen Position des Merkmals im Bild erzeugt werden, welches bspw. die Steuerung einer gekoppelten Anwendung übernehmen oder beeinflussen kann (vgl. Spurhalteassistentensysteme in einem Auto). Konkret soll als Zielplattform ein Smartphone auf Android-Basis sowie die Open Source Bibliothek OpenCV verwendet werden. Als Interface soll die USB-Schnittstelle dienen, da die Umwandlung von diesem Format in die meisten etablierten seriellen Datenprotokolle wie beispielsweise UART sehr einfach und kostengünstig umsetzbar ist und ein Großteil der gängigen Entwicklungsplattformen den Standard bereits von sich aus integriert.

Zu Demonstrationszwecken soll neben einem allgemeinen Framework zur Entwicklung von OpenCV-gestützten Bildverarbeitungsapplikationen für Android-Plattformen eine Landeplatzlokalisierung eines UAVs (Abk., engl. für Unmanned Aerial Vehicle) realisiert werden. Als zu erfassendes Merkmal fungiert zu diesem Zweck ein Kreis (vgl. Landeplatzsymbol eines Helikopters). Als Ausgabesignal eines sich unter dem Flugvehikel befindenden Smartphones soll die Stellgröße eines in die Applikation zu integrierenden Reglers dienen. Ziel ist es, dass das UAV sich autonom und ohne weitere Kenntnis über seine Umgebung über einem Landeplatz zu zentriert. Hierbei muss aus Sicherheitsaspekten ein manuelles Eingreifen durch den Menschen jederzeit möglich sein, weshalb eine signalverarbeitende Instanz in Form eines Mischers, welcher die Signale der Fernbedienung und der Applikation überlagert, zu integrieren ist.

2 GRUNDLAGEN

2.1 Android

2.1.1 Einführung

Android Inc. wurde 2003 von Andy Rubin gegründet, ursprünglich mit der Bestrebung, Software für Digitalkameras zu entwickeln. Bald stellte sich jedoch heraus, dass der Markt auf diesem Gebiet nicht groß genug war, so dass Rubin die Zielstellung zur Entwicklung eines für jeden Entwickler frei zugänglichen Betriebssystems für mobile Plattformen verallgemeinerte [vgl. Mar07]. Diese Idee stieß weitgehend auf große Begeisterung, bis schließlich im Juli 2005 Google Android Inc. für 50 Millionen US-Dollar aufkaufte, um seine mobile Sparte zu erweitern [vgl. Man15]. Dem ursprünglichen Ziel von Rubin folgend, gründete sich am 5. November 2007 die Open Handset Alliance bestehend aus Unternehmen wie Sony, Samsung, HTC, T-Mobile, Qualcomm, Texas Instruments und Google selbst mit dem Ziel, einen offenen Standard für Mobilgeräte zu entwickeln - mit Android als Grundlage [vgl. All07]. Daraufhin erschien am 22. Oktober 2008 das HTC Dream als erstes kommerzielles Mobilgerät auf Android-Basis. Seit diesem Zeitpunkt hat sich Android als das weltweit am weitesten verbreitete Betriebssystem für mobile Systeme [vgl. Goo] durchgesetzt mit inzwischen mehr als 1,4 Milliarden aktiven Nutzern [vgl. Cal15, Stand September 2015]. Seit der Veröffentlichung von Android Base, dem ersten offiziellen Release, hat das Betriebssystem eine Vielzahl von Varianten durchlaufen, bis hin zur derzeit (03.11.2016) aktuellsten Version 7.1.1 mit dem Namen Nougat. Neben der offiziellen Version von Google existieren weiterhin eine Vielzahl von modifizierten Distributionen des Betriebssystems wie beispielsweise CyanogenMod.

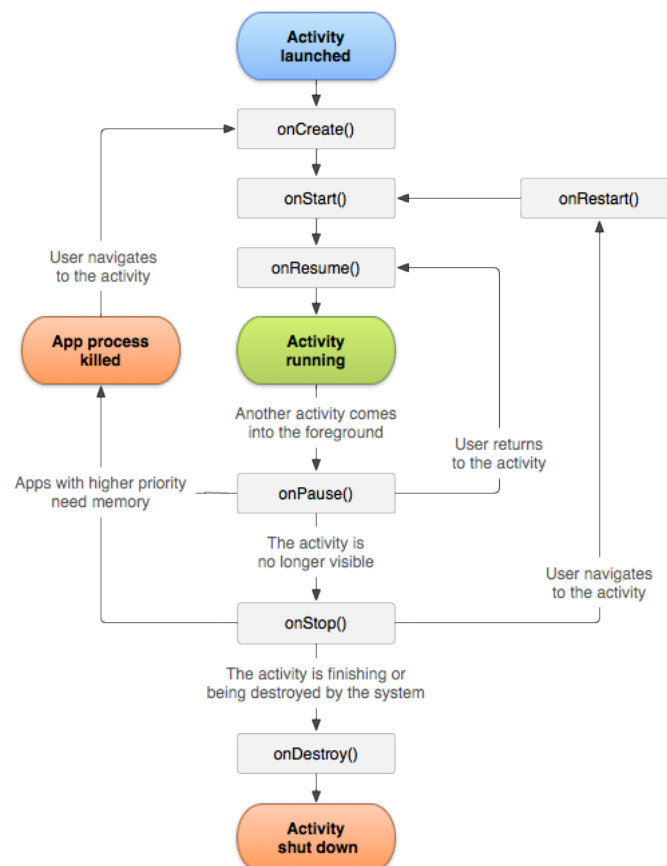


Abbildung 2.1: Lebenszyklus einer Android-Applikation [Goo16a]

Einen großen Anteil an der erfolgreichen Verbreitung von Android hatte (und hat) das frei zugängliche und gut dokumentierte von Google zur Verfügung gestellte Interface zur Erstellung von Applikationen in Kombination mit dem Google Play Store als zentraler Distributionsplattform.

Das Interface kann in Form des Android SDK (Abk., engl. für „Software Development Kit“) (und ggf. des NDK (Abk., engl. für „Native Development Kit“)) direkt von Google bezogen und in viele Entwicklungsumgebungen wie beispielsweise Eclipse direkt integriert werden. Seit Mai 2013 stellt Google mit Android Studio eine eigene IDE (Abk., engl. für „Integrated Development Environment“) zur Verfügung [vgl. Goo16b].

Android trennt bei der Entwicklung sehr stark zwischen dem funktionalen Quellcode und der GUI. Letztere wird in XML-Dateien beschrieben und anschließend mit Funktionen belegt. Zentrale, für die Funktionsfähigkeit der Applikation essentielle Informationen wie beispielsweise Berechtigungen werden dem System mittels des Android-Manifests (ebenfalls in XML) zur Verfügung gestellt [vgl. Goo16c].

Der funktionale Anteil der Anwendung wird in Java geschrieben und implementiert die durch das SDK bereitgestellte Grundstruktur in Form des Activity Lifecycles (vgl. Abbildung 2.1). Activities stellen das Herzstück einer Anwendung dar und repräsentieren voneinander unabhängige Benutzerschnittstellen, die in Kombination die Anwendung bilden. Den Activity Lifecycle bildet ein Satz von Callback-Methoden, bestehend aus

- `onCreate()` (Erstmaliges Starten der Activity)
- `onStart()` (Initiierung der Activity)
- `onPause()` (eine andere Applikation kommt in den Vordergrund)
- `onResume()` (pausierte Activity läuft weiter)
- `onStop()` (Activity ist nicht länger sichtbar)
- `onDestroy()` (Activity wird beendet oder vom System zerstört)
- `onRestart()` (Activity wird nach dem Stoppen erneut aufgerufen)

Der Activity Lifecycle verwaltet den Lebenszyklus der Activity [vgl. Goo16a]. Weiterhin können aus einer Activity Services (über längere Zeit im Hintergrund agierende Funktionen, bspw. Kommunikation mit einem USB-Gerät), Content Providers (Funktionen, um von mehreren Instanzen verwendete Datenquellen wie bspw. SQLite Datenbanken zu verwalten), Broadcast-Receiver und Handler (Funktionen, die systemweite Nachrichten (sogenannte Intents für Anfragen oder Messages für reine Mitteilungen) filtern und je nach Konfiguration bei gewissen Nachrichten Aktionen auslösen) oder andere Activities aufgerufen werden.

Neben der klassischen Programmierung in Java bietet das NDK weiterhin die Option, Elemente nativen Codes mittels des Java Native Interfaces zu implementieren (siehe Kapitel 2.3).

Die eingängige Struktur des Interface erlaubt es auch weniger erfahrenen Programmierern, Applikationen zu erstellen. Für weiterführende Informationen bzgl. der Programmierung von Android-Applikationen kann die offizielle Dokumentation von Google konsultiert werden unter <https://developer.android.com/index.html>.

2.1.2 Warum Android?

Wie bereits in der Einleitung dargestellt, ist das Ziel der Arbeit, einen Ansatz mit variablen Einsatzgebieten zu erarbeiten, der auch Privatpersonen ohne ausgeprägte Programmier-Vorkenntnisse die Verwendung von Bildverarbeitungssoftware ermöglichen soll. Die Einrichtung von Alternativplattformen wie bspw. eines Raspberry Pi kann eine relativ große Herausforderung für Personen ohne Vorkenntnisse darstellen. Dem hingegen ist es verhältnismäßig einfach, mittels der Tutorials von OpenCV, Android Studio oder Eclipse und des im

Rahmen dieses Projekts erstellten Frameworks eine lauffähige Android-Applikation zu erstellen, da sich der eigentliche Programmieraufwand lediglich auf den reinen Bildverarbeitungsanteil beschränkt. Einen weiteren Vorteil bietet das Smartphone als Zielgerät an sich, da es im Normalfall ein Kameramodul, einen Bildschirm, diverse Kommunikationsschnittstellen und ein fertiges internes Bussystem mit sich bringt und sehr kompakt und leicht ist. Weiterhin wird durch die weite Verbreitung von Android mit mehr als 1,4 Milliarden Geräten weltweit [vgl. Cal15, Stand September 2015] die einfache Zugänglichkeit für jeden Nutzer sichergestellt.

2.2 OpenCV

2.2.1 Einführung

OpenCV (Abk. für „Open Source Computer Vision Library“) wurde ursprünglich 1999 als Forschungsprojekt von Intel im Zuge der Entwicklung kommerziellen Bildverarbeitungsanwendungen ins Leben gerufen [vgl. Gar]. Im Laufe der Zeit wechselte das Projekt mehrfach den Entwickler, bis es schließlich im August 2012 von der nicht-profitablen Organisation OpenCV.org übernommen wurde, u. a. mit itseez, welches erst im Mai 2016 von Intel für sein Know-How im Bereich „Computer Vision for IOT“ akquiriert wurde [vgl. Dav16], als treibende Kraft im Hintergrund. Seit August 2014 steht mit OpenCV3.0 die dritte Version des unter der modifizierten BSD-Lizenz veröffentlichten Projekts für maschinelles Lernen und Bildverarbeitung zur öffentlichen Verfügung [vgl. Ale16]. Die Bibliothek stellt über 2500 Algorithmen zur Verfügung, welche genutzt werden können, um:

- Gesichter zu detektieren und zu erkennen
- Objekte zu identifizieren
- menschliche Gesten in Videoaufnahmen zu klassifizieren
- Kamerabewegungen zu verfolgen
- bewegende Objekte zu erfassen
- 3D-Modelle aus Objekten abzuleiten
- 3D-Punktwolken aus Stereo-Kameras zu erzeugen
- Bilder zusammenzuschneiden und ein hochauflösendes Gesamtbild zu erzeugen
- ähnliche Bilder aus einer Datenbank zu filtern
- rote Augen aus Bildern, die mit Blitzlicht aufgenommen wurden, zu entfernen
- Augenbewegungen zu folgen
- Szenarien zu erkennen und Marker zu erzeugen, um diese mit Augmented Reality zu hinterlegen
- Etc.

Ferner beinhaltet das Projekt eine Bibliothek für Maschinelles Lernen zur Implementierung von künstlichen neuronalen Netzen, EM-Algorithmen, Bayes-Klassifikation, Boosting, etc.. OpenCV bietet dabei C++, C, Python, Java und MATLAB Schnittstellen und unterstützt mit Windows, Linux, Android und Mac OS sämtliche großen Betriebssysteme [vgl. its]. Genutzt wird die Bibliothek sowohl in Forschungs- und Regierungsprojekten, als auch von renommierten Unternehmen wie Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda und Toyota für Anwendungen wie das Zusammenschneiden von Streetview-Bildern, Detektierung von Einbrüchen in Überwachungssystemen, Navigieren von Roboterbewegungen, Verkehrsüberwachung, Inspektion von Labeln in der Industrie, Rapid Face Detection u. v. m. [vgl. its].

Für weitere Informationen bzgl. der Programmierung von Bildverarbeitungsprogrammen mit OpenCV sind die Dokumentation des Projekts sowie ausführliche Tutorials zu finden unter <http://opencv.org/documentation.html>.

2.2.2 Warum OpenCV?

OpenCV bietet sich als Grundlage eines Frameworks zur Entwicklung von Bildverarbeitungs-Applikationen an, da es eine Vielzahl an Algorithmen zur Bildverarbeitung in verschiedenen Bereichen und zum Maschinellen Lernen bereitstellt und es damit dem Nutzer ein sehr großes funktionales Spektrum bei der Erstellung einer eigenen Applikation bietet. Außerdem erlaubt die Verfügbarkeit von C++, C und Python-Schnittstellen (Programmiersprachen, die bei der Erstellung von Android-Applikationen neben Java verwendet werden können) die Einbringung persönlicher Programmier-Präferenzen und unterstützt somit eine einfachere Einarbeitung. Für das in dieser Arbeit vorgestellte Framework wurde standardmäßig C++ zur Implementation der Bildverarbeitung genutzt; es ist jedoch möglich, stattdessen eine der anderen Sprachen zu verwenden. Weiterhin steht die Bibliothek unter der modifizierten BSD-Lizenz und trägt somit zu der Zielsetzung der freien öffentlichen Zugänglichkeit und Verwendung des in dieser Arbeit behandelten Projekts bei [vgl. Fre16].

2.3 Java Native Interface

Das JNI (Abk., engl. für „Java Native Interface“) dient dazu, nativen Code in eine Java-Applikation zu integrieren oder andersherum. Man spricht von einem sogenannten „Two-Way-Interface“. Das JNI unterstützt dabei sowohl **native Bibliotheken** als auch **native Applikationen** [vgl. Lia99, S.5].

- Das JNI erlaubt es auf Java-Seite, **native Methoden** äquivalent zu Java-Methoden aufzurufen. Im Hintergrund können diese jedoch in jeder nativen Programmiersprache wie beispielsweise C oder C++ geschrieben sein.
- Auf nativer Seite bietet das JNI ein **Interface** an, welches die Implementierung einer **virtuellen Java-Maschine** in nativen Code erlaubt (siehe Abbildung 2.2). Native Applikationen können Bibliotheken, welche die virtuelle Maschine implementieren, einbinden und somit über das Interface in Java programmierte Elemente aufrufen. So kann beispielsweise eine C++-Instanz von OpenCV den Bildstrom einer mittels Java gesteuerten Kamera abrufen.

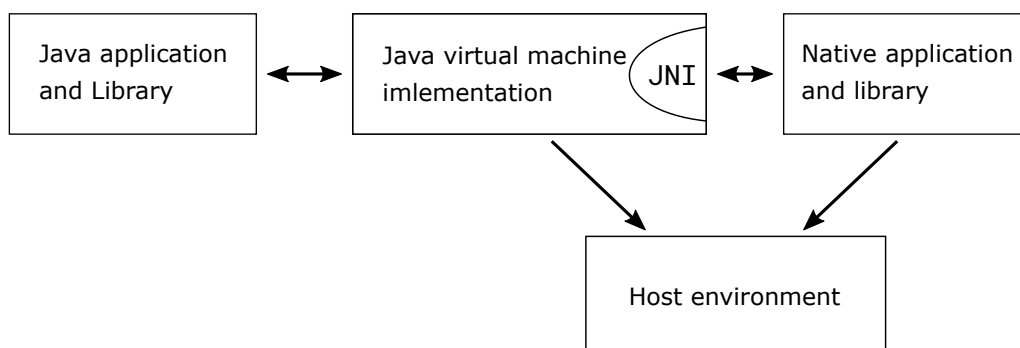


Abbildung 2.2: Funktionsdiagramm des Java Native Interface, in Anlehnung an [Lia99, S.5]

Es ist jedoch zu beachten, dass eine Java-Applikation, welche nativen Code mittels des JNI integriert, zwei wesentliche Vorteile gegenüber der reinen Java-Entwicklung verliert [vgl. Lia99, S.6]:

- Die Applikation kann nicht mehr ohne weiteres in andere Host-Plattformen portiert werden, da der native Anteil jedes mal umgebungsspezifisch neu kompiliert werden muss.
- Während Java architekturgemäß durch die zugrundeliegende JVM (Abk., engl. für „Java Virtual Machine“) eine gewisse Sicherheit mit sich bringt, ist dies für native Sprachen nicht zwangsläufig der Fall und in erster Linie von der Programmierung selbst abhängig.

Ausserdem ist das Java Native Interface lediglich dazu in der Lage, native Datentypen (d. h. Integer, Double, Char, etc., jedoch z. B. keine Matrizen) zu übertragen.

Für weitere Informationen bzgl. der einzuhaltenden Form und Semantik bei der Verwendung des JNI siehe beispielsweise <http://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html>.

3 ENTWICKLUNGSUMGEBUNG

3.1 Auswahl der Entwicklungsumgebung

Für die Entwicklung der Applikation wurde aus persönlichen Präferenzen Eclipse (Download: <https://eclipse.org/downloads/>) als Entwicklungsumgebung verwendet. Alternativ kann ebenfalls Android Studio (Download: <https://developer.android.com/studio/index.html>) gewählt werden. Es ist jedoch zu erwähnen, dass Google seit der Veröffentlichung von Android Studio das ADT (Android Developer Tools) Plugin für Eclipse nicht länger weiterentwickelt [vgl. Goo15]. Dementsprechend ist es zu empfehlen, bei zukünftigen Entwicklungsarbeiten zu Android Studio zu migrieren.

3.2 Einrichtung der Entwicklungsumgebung

Bei der eigentlichen Einrichtung der IDE (Abk., engl. für „Integrated Development Environment“) wurde sich an dem offiziellen Tutorial von OpenCV orientiert, daher wird an dieser Stelle nicht näher darauf eingegangen.

Links zu den offiziellen Tutorials:

- Installation von Java und Einrichtung des SDK (Abk., engl. für „Software Development Kit“) und NDK (Abk., engl. für „Native Development Kit“) für Android:
http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/android_dev_intro.html#android-dev-intro
- Einbindung des OpenCV4Android-SDKs:
http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html#o4a-sdk
- Einrichtung von Eclipse für die Programmierung von OpenCV-Applikationen für Android und Erstellung einer Testapplikation:
http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html#dev-with-ocv-on-android

Es wird empfohlen, eine Testapplikation zu schreiben und zu überprüfen, ob diese sowie die Beispielanwendungen kompiliert und auf einem Testgerät zum Laufen gebracht werden können, um die richtige Einrichtung der IDE zu validieren.

Anmerkung: Beim Kompilieren der VBLS-Applikation (Abk., engl. für „Visual Based Landing System“) ist zu beachten, dass Elemente der verwendeten Bibliothek zur USB-Kommunikation erst ab SDK-Version 12 oder höher verfügbar sind. Dementsprechend muss die Zielplattform unter **Preferences->Android** und in **AndroidManifest.xml** angepasst werden. Des weiteren kann es vorkommen, dass Fehler beim Kompilieren des Layouts auftreten. Dies kann behoben werden, indem die Android Appcompat v7 oder höher als Bibliothek hinzugefügt wird (zu finden im Android SDK).

4 ENTWICKLUNG DER APPLIKATION: ALLGEMEINES OPENCV-ANDROID-FRAMEWORK

4.1 Genereller Aufbau

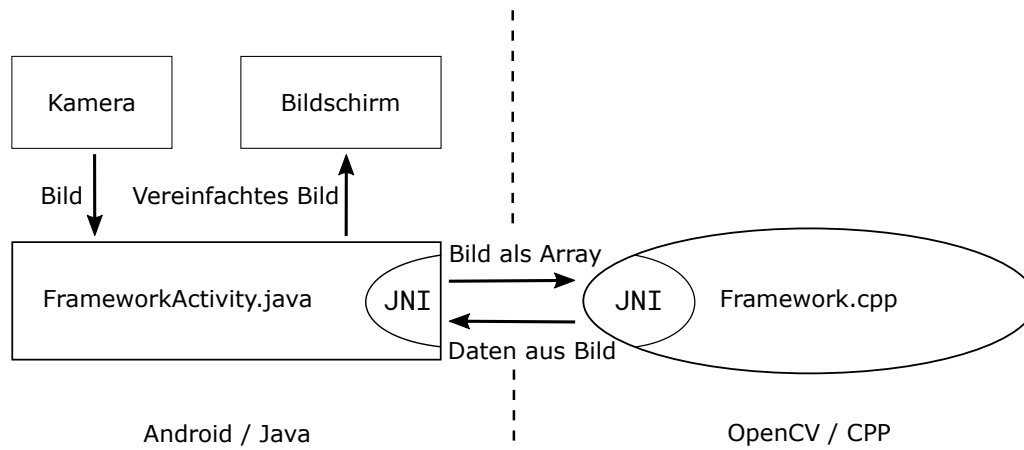


Abbildung 4.1: Funktionsdiagramm des Frameworks

In diesem Kapitel soll zunächst das im Rahmen des Projekts erstellte allgemeine Framework zur Entwicklung von OpenCV-basierten Bildverarbeitungsapplikationen für Android-Systeme betrachtet werden. Wie in dem Architekturdiagramm (4.1) dargestellt, ist das Grundgerüst in drei Bestandteile unterteilt, welche im Folgenden nacheinander betrachtet werden:

- die Android-Schnittstelle
- die Bildverarbeitung auf OpenCV-Basis
- die USB-Bibliothek

Neben den jeweiligen Implementierungen der genannten Kernelemente sollen weiterhin die zwischen den einzelnen Abschnitten kommunizierenden Schnittstellen betrachtet werden. Dabei wird für eine gute Übersichtlichkeit nach verwendeten Programmiersprachen geordnet vorgegangen, beginnend mit Java, gefolgt von C++ und XML. Zentrale Funktionalitäten werden detailliert betrachtet, für nebensächliche oder rein organisatorische Aspekte wird die Kommentierung des Programms als ausreichend erachtet. Der vollständige Quellcode findet sich unter dem in Kapitel 10.3 angeführten Link.

4.2 Teil 1: Java

4.2.1 openCVActivity.java

Die Klasse `openCVActivity.java` bildet das Herzstück der Applikation. Sie implementiert mit dem Activity-Lifecycle (s. Abbildung 2.1) die Android-Schnittstelle und verwaltet die USB-Konfiguration. Gleichzeitig leitet sie die aufgenommenen Bilder über `openCVFramework.java` an `openCVFramework.cpp` (Bildverarbeitung) weiter.

Listing 4.1: Set Up

```

1  /*-----Set Up-----*/
3      final static int IMG_WIDTH = XXXXX;
4      final static int IMG_HEIGHT = XXXXX;
6      // Color - images
7      //private Mat          mRgba;
9      // Gray - images
10     private Mat          mGray;
12  /*-----*/
    
```

Die Klasse ermöglicht ein nutzerspezifisches Set Up. An dieser Stelle kann die gewünschte Kameraauflösung eingestellt und ausgewählt werden, ob mit Farb- oder Graustufen-Bildern gearbeitet werden soll. Je weniger Informationen ein Bild enthält, desto höher ist die Performanz der Applikation. Es ist daher zu empfehlen, die Auflösung nicht höher einzustellen als notwendig und falls möglich mit Graustufen zu arbeiten.

Listing 4.2: OpenCV Loader

```

1  // Implementation of the LoaderCallbackInterface; tries
2  // to load the OpenCV Manager
3  (has to be preinstalled on the device)
4      private BaseLoaderCallback mLoaderCallback = new
5          BaseLoaderCallback(this) {
6              @Override
7              public void onManagerConnected(int status) {
8                  switch (status) {
9                      case LoaderCallbackInterface.SUCCESS:
10                     {
11                         Log.i(TAG, "OpenCV loaded successfully");
12                         mOpenCvCameraView.enableView();
13                     } break;
14                     default:
15                     {
16                         super.onManagerConnected(status);
17                     } break;
18                 }
19             }
20     };
    
```

Der OpenCV-Loader versucht, auf dem Gerät vorinstallierte OpenCV-Bibliotheken zu laden. Diese sind in Form des OpenCV-Managers frei im Google Play Store erhältlich. Alternativ kann die Bibliothek mit der Applikation zusammen installiert werden. Dies ist jedoch aus Speichergründen nicht zu empfehlen, da die Bibliothek ansonsten mit jeder weiteren OpenCV-Applikation neu installiert würde anstatt eine zentrale Instanz zu nutzen.

Listing 4.3: onCreate

```

1  // Called when the activity is first created
2  @Override
3  public void onCreate(Bundle savedInstanceState) {
4      Log.i(TAG, "called onCreate");
5      super.onCreate(savedInstanceState);
6      getWindow().addFlags(WindowManager.LayoutParams.
7          FLAG_KEEP_SCREEN_ON);
    
```

```
9         setContentView(R.layout.openCVFramework_surface_view);

11         mOpenCvCameraView = (CameraBridgeViewBase) findViewById(
12             R.id.openCV_activity_surface_view);

14         // Sets the resolution
15         mOpenCvCameraView.setMaxFrameSize(IMG_WIDTH, IMG_HEIGHT);

17         mOpenCvCameraView.setVisibility(CameraBridgeViewBase.
18             VISIBLE);
19         mOpenCvCameraView.setCvCameraViewListener(this);
20     }
```

Die Funktion `onCreate()` ist Bestandteil des von Android bereitgestellten Interfaces und wird bei erstmaligem Öffnen der Applikation aufgerufen. Zunächst wird versucht, gespeicherte Einstellungen aufzurufen; danach wird die Verbindung zum User Interface der Applikation mittels `setContentView(...)` herzustellen (s. Kapitel 4.4.2). Anschließend wird die Kamera initialisiert und mit `setMaxFrameSize(...)` die Auflösung auf die im Set Up festgelegten Werte gesetzt. Die Funktion `setVisibility(...)` bewirkt die Anzeige der Bilder auf dem Display und kann bei Bedarf deaktiviert werden.

Listing 4.4: onPause

```
1 // Called if the application is moved to the background
2 @Override
3 public void onPause()
4 {
5     super.onPause();
6     if (mOpenCvCameraView != null)
7         mOpenCvCameraView.disableView();
8 }
```

Da es je nach Art und Verwendung der Applikation kritisch sein kann, wenn bei Pausieren der Anwendung beispielsweise die USB-Kommunikation (s. Beispielanwendung Visual Based Landing System) unterbrochen wird, ist `onPause()` standardmäßig derart konfiguriert, dass lediglich die Kamera deaktiviert wird.

Listing 4.5: onResume

```
1 // Called if the application is resumed from the background
2 @Override
3 public void onResume()
4 {
5     super.onResume();
6     if (android.os.Build.VERSION.SDK_INT >= android.os.
7         Build.VERSION_CODES.M){
8         // Android versions above 6.0 work with so called
9         // "Runtime Permissions" allowing the user to change
10        // the permissions granted while the application is
11        // running, so the permissions have to be checked on
12        // every resume
13        int hasCameraPermission = checkSelfPermission(
14            Manifest.permission.CAMERA);
15        if (hasCameraPermission != PackageManager.
16            PERMISSION_GRANTED && counter_denied == 0){
17            requestPermissions(new String[]{Manifest.
18                permission.CAMERA},
19                REQUEST_CODE_CAMERA_PERMISSION);
20        }
```

```

21         else if (hasCameraPermission != PackageManager.
22             PERMISSION_GRANTED && counter_denied != 0){
23             onDestroy();
24         }
25     }
26     if (!OpenCVLoader.initDebug()) {
27         Log.d(TAG, "Internal OpenCV library not found.
28             Using OpenCV Manager for initialization");
29         OpenCVLoader.initAsync(OpenCVLoader.
30             OPENCV_VERSION_3_0_0, this,
31             mLoaderCallback);
32     }
33     else {
34         Log.d(TAG, "OpenCV library found inside package.
35             Using it!");
36         mLoaderCallback.onManagerConnected(
37             LoaderCallbackInterface.SUCCESS);
38     }
39 }

```

Die Funktion `onResume()` ist ebenfalls Teil des von Android zur Verfügung gestellten Interfaces und wird jedes Mal aufgerufen, wenn die Applikation aus dem Hintergrund (pausiert) aufgerufen wird und somit wieder in den Vordergrund rückt, sowie nach dem erstmaligen Starten des Programms (siehe Abbildung 2.1). Zunächst wird `super.onResume()` aufgerufen, wodurch u. a. versucht wird, die in `onPause()` deaktivierte Kamera wieder zu reaktivieren; es ist kein expliziter Aufruf von `mOpenCvCameraView.enableView()` notwendig.

Seit Version 6.0 arbeitet Android jedoch mit sogenannten Runtime Permissions, d. h. es ist möglich, einer Anwendung bestimmte Berechtigungen zu geben und zu nehmen, während diese läuft. Daher muss an dieser Stelle zunächst das Level der Distribution der Plattform überprüft werden und, falls dieses größer als 6.0 ist, ob die benötigte Nutzungsberechtigung für die Kamera aktuell immer noch vorliegt. Dies geschieht mittels der ersten if-Abfrage (`android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.M`) und `checkSelfPermission(...)`. Liegt keine Berechtigung vor, so muss diese aktiv beim Nutzer mittels `requestPermissions(...)` (siehe unten) angefragt werden. Da nach jeder Anfrage einer Berechtigung erneut `onResume()` aufgerufen wird, kann mittels der Einführung der Einführung der Counter-Variablen `counter_denied` eine Fallunterscheidung, ob bereits eine Abfrage stattgefunden hat, implementiert werden. So wird eine rekursive Endlosschleife vermieden, falls der Nutzer die Berechtigung beim Aufruf nicht geben möchte. Die zweite if-Abfrage (`if (!OpenCVLoader.initDebug())`) führt eine Unterscheidung nach der Art der Implementierung der OpenCV-Bibliothek (vorinstalliert mittels des OpenCV-Managers oder mit der Applikation zusammen installiert) durch und versucht, diese zu reinitialisieren.

Listing 4.6: Überprüfen der Kamera-Berechtigung

```

1 // Checks if the camera-permission is granted and terminates
2 // the application if not
3 @Override
4 public void onRequestPermissionsResult(int requestCode,
5     String[] permissions, int[] grantResults){
6     switch (requestCode) {
7         case REQUEST_CODE_CAMERA_PERMISSION:
8             if (grantResults[0] == PackageManager.
9                 PERMISSION_GRANTED){
10                 // Accepted
11                 counter_denied = 0;
12                 return;
13             }

```



```

14         else {
15             // Denied
16             Toast.makeText(VBLSActivity.this, "Camera
17             permission denied! Terminating application!",
18             Toast.LENGTH_SHORT).show();
19             counter_denied++;
20         }
21     }
22 }

```

Die Funktion `onRequestPermissionsResult(...)` wird nach Anfragen einer Berechtigung mittels `requestPermissions(...)` aufgerufen und kann verwendet werden, um Reaktionen abhängig vom zurückgelieferten Ergebnis zu definieren. Da diese Funktion jedoch bei jeder Anfrage einer beliebigen Berechtigung aktiviert wird, ist zunächst mittels `switch(request-Code)` eine Fallunterscheidung bzgl. der Art der angefragten Berechtigung durchzuführen; im konkreten Fall ist die Nutzungsberechtigung für die Kamera von Bedeutung, welche mittels `case REQUEST_CODE_CAMERA_PERMISSION` abgefragt wird. Der erste Eintrag des übergebenen Arrays `grantResults` entspricht dem Ergebnis der Anfrage. Ist die Berechtigung nun also erteilt worden (`grantResults[0] == PackageManager.PERMISSION_GRANTED`), so wird `counter_denied` zurückgesetzt und die Funktion wird ohne weitere Aktionen verlassen (`return`). Als Reaktion auf die Anfrage wird erneut `onResume()` aufgerufen; da die Berechtigung jedoch erteilt wurde, wechselt die Funktion dieses Mal direkt zur Initialisierung der OpenCV-Bibliothek. Ist das Ergebnis der Anfrage dagegen negativ, so wird `counter_denied` erhöht. Dadurch wird beim Aufruf von `onResume()` nach Verlassen von `onRequestPermissionsResult(...)` anstatt einer erneuten Abfrage die Applikation mittels `onDestroy()` terminiert.

Listing 4.7: onDestroy

```

1 // Called if the application is terminated
2 @Override
3 public void onDestroy() {
4     super.onDestroy();
5     mOpenCvCameraView.disableView();
6 }

```

Die Funktion `onDestroy()` terminiert wie zu erwarten die Activity. Standardmäßig wird dabei lediglich `disableView()` zusätzlich zu `super.onDestroy()` aufgerufen um die Kamera zu deaktivieren. An dieser Stelle können/sollten in aus dem Framework abstrahierten Applikationen alle weiteren mit der Activity zusammenhängenden Prozesse (wie beispielsweise weitere Views) beendet werden.

Listing 4.8: onCameraFrame

```

1 // Called on every frame received from the input - stream
2 // of the camera
3 public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
4     // Color - images
5     /*mRgba = inputFrame.rgba();
6     byte[] jImageData = transformationTools.jMatToArray(
7     mRgba);
8     double[] jImageDataCircles = transformationTools.detect(
9     jImageData);*/
11
12     // Gray - images
13     mGray = inputFrame.gray();
14
15     byte[] jImageData = null;
16     try {
17         jImageData = transformationTools.jMatToArray(mGray);

```

```

17     }
18     // Terminates the application if the image data was damaged
19     // at some point
20     catch (IOException e){
21         Log.e(TAG, e.getMessage());
22         Log.e(TAG, "Transformation fault");
23         this.onDestroy();
24     }

26     double[] jImageDataCircles = null;
27     try {
28         jImageDataCProcessed = transformationTools.process(
29             jImageData);
30     }
31     // Terminates the application if the image data was damaged
32     // at some point
33     catch (IOException e){
34         Log.e(TAG, e.getMessage());
35         this.onDestroy();
36     }
37     // Fault in the image procession or conversion via JNI if
38     // jImageDataCircles is still null at this point
39     if (jImageDataProcessed == null){
40         Log.e(TAG, "Image processing fault");
41         this.onDestroy();
42     }

44     /* Possibly do stuff with your processed image here*/
45     /*
46     *
47     *
48     *
49     * */

51     // Color - images
52     //return mRgba;

54     // Gray - images
55     return mGray;
56 }
    
```

Die Funktion `onCameraFrame(...)` ist eine weitere Callback-Methode des von Android bereitgestellten Interfaces. Sie wird jedes Mal aufgerufen, wenn die Kamera ein Bild aufnimmt und bekommt dieses als Übergabeargument in Form eines `CvCameraViewFrames` übergeben. Zunächst muss auch an dieser Stelle wieder ausgewählt werden, ob im weiteren Verlauf mit einem Farb- oder Graustufenbild gearbeitet werden soll. Je nachdem wird das Bild mittels `rgba()` oder `gray()` von `CvCameraViewFrames` zu `Mat` (Abk., kurz für Matrix), einem Datentyp bestehend aus einem Array von Vektoren, welche wiederum die Informationen jedes Pixels des Bilds enthalten (d. h. für Farbbilder die RGB- sowie den Alpha-Anteil und für Graustufenbilder den Weißanteil), konvertiert.

Da die eigentliche Bildverarbeitung jedoch in `openCVFramework.cpp` stattfindet und das JNI, welches als Kommunikationsschnittstelle zwischen Java und C++ fungiert, lediglich native Datentypen unterstützt, muss weiterhin die Matrix zu einem Array aus Bytes umgewandelt werden. Dies geschieht mit `jMatToArray(...)` (s. Kapitel 4.2.2). Das so erhaltene Array wird in `jImageData` gespeichert. Schlägt die Umwandlung fehl, z. B. wenn die Bilddaten beschädigt sein sollten, so wirft `jMatToArray(...)` eine `IOException`, die an dieser Stelle

abgefangen wird und mit einem entsprechenden Log-Eintrage zur Terminierung der Applikation führt. Anschließend wird mit `process(...)` die eigentliche Bildverarbeitung initiiert. Das in Form des Byte-Arrays übergebene Bild wird per JNI an `openCVFramework.cpp` weitergeleitet und dort verarbeitet. Standardmäßig ist `process(...)` so eingerichtet, dass die Rückgabe in Form eines Double-Arrays erfolgt, so dass beispielsweise Positionen oder Größen von detektierten Objekten übergeben werden können (s. Kapitel 4.3.2). Alternativ ist es auch möglich ein bearbeitetes Bild (ebenfalls wieder in Byte-Form) zurückzugeben und anschließend anzeigen zu lassen.

Treten bereits vor der Kommunikation mittels JNI Komplikationen auf, so wirft auch `process(...)` eine Fehlermeldung, die ebenso wie bei `jMatToArray(...)` zu einer Beendigung der Applikation führt; kommt es zu einem Fehler während der eigentlichen Bildverarbeitung, so liefert `openCVFramework.cpp` einen Nullpointer als Rückgabewert zurück, der in einer if-Abfrage abgefangen wird. So kann im Fehlerfall beim Debugging leichter zwischen den Ursachen differenziert werden.

Anschließend an die Verarbeitung besteht die Möglichkeit, weitere Aktionen durchzuführen. So können die durch `process(...)` erhaltenen Daten bspw. per USB ausgegeben oder ereignisbasierte Nachrichten dargestellt werden. Durch den Zugriff auf die Java-Bibliotheken auf Android-Seite vervielfältigen sich die Möglichkeiten. Abschließend wird mittels `return` das Bild, welches auf dem Display angezeigt werden soll, in Matrix-Darstellung zurückgegeben.

Standardmäßig ist die Applikation derart eingerichtet, dass das aufgenommene Bild in Graustufen-Form dargestellt wird. Wurde das Bild im Rahmen der Anwendung jedoch verändert (z. B. indem bestimmte detektierte Objekte eingerahmt wurden), so kann an dieser Stelle auch die modifizierte Variante übergeben werden. Es ist jedoch anzumerken, dass es rechen-technisch sehr aufwendig ist, das an die Bildverarbeitung übergebene Bild dort zu verändern, innerhalb von `openCVFramework.cpp` wieder von einer Matrix in ein Byte-Array umzuwandeln, anschließend per JNI zurückzugeben und erneut zu einer Matrix zu konvertieren. Daher ist es aus Performanz-Gründen zu empfehlen, falls möglich lediglich die Detektion der Objekte im Bild in der Verarbeitung auszuführen, deren Positionen anschließend zurückzugeben und auf Java-Seite direkt in das Bild einzuzichnen (auch wenn dort ohne die Einbindung einer weiteren externen Bibliothek lediglich triviale Aktionen zur Verfügung stehen).

4.2.2 openCVFramework.java

Die Klasse `openCVFramework.java` stellt die Schnittstelle zwischen `openCVActivity.java` (Android) und `openCVFramework.cpp` (OpenCV) dar und wird von `openCVActivity.java` eingebunden.

Listing 4.9: Unterscheidung des Bildformats

```
1 // Color - image
2 // private static final int ch = 4;

4 // Gray - image
5 private static final int ch = 1;
```

Wie auch in der übergeordneten Activity kann unterschieden werden, ob ein Farbbild oder ein Graustufen-Bild verarbeitet wird. Je nachdem sind jedem Pixel des Bildes unterschiedlich viele Informationen zugeordnet; im Falle des Farbbilds sind dies jeweils die RGB-Anteile und zusätzlich der Alpha-Anteil, bei einem Graustufen-Bild wird lediglich der Weißanteil des Pixels angegeben.

Listing 4.10: Laden der vorkompilierten Bibliothek

```
1 public openCVFramework() {
2     System.loadLibrary("openCVFramework");
3 }
```

Mittels des dargestellten Quellcodes wird die Bibliothek `openCVFramework.so` aufgerufen. Diese ist notwendig für die Verwendung des JNIs und wird bei Kompilieren der Applikation automatisch aus `openCVFramework.h` erstellt, da Android aus Gründen der Speicheroptimierung lediglich vorkompilierte Bibliotheken erlaubt. Für eine manuelle Anleitung zur Erstellung der Bibliothek über die Kommandozeile siehe Kapitel 4.3.1.

Listing 4.11: Konvertierung des Datenformats der Bilder

```

1  // Convert the given Mat to a byte-Array for passing it through
2  // jni (jni doesn't communicate non - primitive data types)
3  byte[] jMatToArray(Mat img) throws IOException {
4      // Read the matrix - parameters (they are needed later on to
5      // rebuild the image inside the jni - method)
6      columns = img.cols();
7      rows = img.rows();
8      channels = img.channels();
9      depth = img.depth();
10     // Check if the image data is not damaged (or rather if the
11     // dimensions make sense)
12     if (columns != 0 && rows != 0 && channels == ch && depth ==
13         0){
14         int size = columns*rows*channels;
15         byte[] imageData = new byte[size];
16         // Convert the image data from the matrix
17         img.get(0, 0, imageData);
18         return imageData;
19     }
20     else {
21         IOException e = new IOException("Damaged image data");
22         throw e;
23     }
24 }

```

Wie bereits in Kapitel 2.3 erwähnt, können über das JNI lediglich native Datentypen kommuniziert werden. Da Android jedoch mit dem Datenformat Mat (s. Kapitel 4.2.1) arbeitet, muss zunächst eine Konvertierung stattfinden, bevor die Bilder an `openCVFramework.cpp` weitergeleitet werden können. Diese Funktionalität bietet die Funktion `jMatToArray(...)`. Dabei werden zunächst die Parameter des Bildes (Höhe, Breite, Kanäle und Format der Matrix) eingelesen und überprüft. Somit wird vermieden, dass ein beschädigtes Bild verarbeitet wird und einen Systemabsturz verursacht. Sind die Parameter in Ordnung wird anschließend ein Byte-Array erzeugt und der Inhalt der Matrix wird mittels `img.get(...)` in das Array übertragen. Ist die if-Bedingung nicht erfüllt, wirft die Funktion eine Exception.

Listing 4.12: Implementation des JNIs auf Java-Seite

```

1  // Calls the native method containing the image processing
2  // through jni
3  double[] process(byte[] img) throws IOException {
4      // Check if the image data is not damaged (or rather if
5      // the dimensions make sense)
6      if (img != null){
7          return nativeImageProcesson(img, rows, columns,
8              channels, depth);
9      }
10     else {
11         IOException e = new IOException("Damaged image data");
12         throw e;
13     }

```

```

14     }

16     // Declaration of the image processing method called above (
17     // in detect()); the "native" parameter signals that a
18     // jni - function is to be called
19     private static native double[] nativeImageProcesson(byte[]
20     image, int r, int c, int ch, int d);

```

Die Funktionen `process(...)` und `nativeImageProcesson(...)` stellen die Implementierung des JNI auf Java-Seite dar. Erstere Funktion ruft `nativeImageProcesson(...)` mit dem übergebenen Bild und den zugehörigen Eigenschaften auf, wodurch der Aufwand zur Verwendung der Funktion reduziert wird. Das Schlüsselwort `native` in der Deklaration von `nativeImageProcesson(...)` signalisiert, dass es sich um eine über das JNI aufgerufene Funktion handelt und erlaubt es dem Interface, die Verknüpfung mit der Bibliothek `openCVFramework.h` herzustellen. Standardmäßig wird von der in `openCVFramework.cpp` implementierten Funktion ein Double-Array über das JNI zurückgeliefert, wodurch Positionen oder Größen von detektierten Gegenständen übergeben werden können. Kommt es während der Bildverarbeitung jedoch zu einem Fehler, wird stattdessen ein Nullpointer übergeben.

4.3 Teil 2: C++

4.3.1 openCVFramework.h

In diesem Kapitel soll beschrieben werden, wie aus einer eine oder mehrere native Funktionen implementierenden Java-Klasse, in diesem Falle `openCVFramework.java`, die für die Verknüpfung mittels JNI benötigte native Bibliothek erstellt werden kann. Zunächst muss dafür aus der betreffenden .java Datei aus ihrem Verzeichnis heraus mittels

- `javac -cp Pfade_der_implementierten_Bibliotheken Name_der_Klasse.java`

kompiliert werden. Aus der resultierenden .class Datei wiederum kann nun aus dem Source-Verzeichnis der Applikation durch

- `javah -cp Pfad_der_implementierten_Bibliothek_1: Pfad_der_implementierten_Bibliotheken_2: Pfade_der_implementierten_Bibliothek_n: Pfad_vom_Source-Verzeichnis_der_betreffenden_Klasse.class`

die zugehörige Bibliothek generiert werden. Diese .h Datei ist notwendig, damit das JNI die Referenz von der nativen Implementation einer Funktion zu deren Aufruf in Java herstellen kann. Im konkreten Fall lauten die Aufrufe wie folgt:

- `javac -cp ~/OpenCV-android-sdk/sdk/java/src/ openCVFramework.java`
- `javah -cp ~/OpenCV-android-sdk/sdk/java/src: org.opencv.openCVFramework.java`

Die resultierende Bibliothek `org_opencv_openCVFramework.h` hat anschließend folgende Form:

Listing 4.13: `openCVFramework.h`

```

1  // This header is the interface between openCVFramework.java
2  // and openCVFramework.cpp which is used for the call through
3  // jni. The file is generated from openCVFramework.java.

5  /* DO NOT EDIT THIS FILE - it is machine generated */
6  #include <jni.h>
7  /* Header for class org_opencv_openCVFramework */

9  #ifndef _Included_org_opencv__openCVFramework
10 #define _Included_org_opencv__openCVFramework
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14 /*
15  * Class:      org_opencv__openCVFramework
16  * Method:     nativeFindCircles
17  * Signature:  ([BIIII)[B
18  */
19 JNIEXPORT jdoubleArray JNICALL
20     Java_org_opencv_openCVFramework_nativeImageProcession
21     (JNIEnv *, jclass, jbyteArray, jint, jint, jint, jint);

23 #ifdef __cplusplus
24 }
25 #endif
26 #endif
    
```

Es ist darauf zu achten, dass die so erstellte `.h` Datei sich in dem `jni`-Ordner der Applikation befinden muss, damit das Java Native Interface diese bei Aufruf der nativen Funktion auf Java-Seite findet. Bei der Kompilierung der Applikation wird aus dem nativen Anteil der Anwendung schließlich eine `.so` Datei erstellt, die in `openCVFramework.java` (s. Kapitel 4.2.2) aufgerufen wird. Dieser Vorgang kann in der `Android.mk` und in der `Application.mk` (beide ebenfalls im JNI-Verzeichnis) definiert werden.

4.3.2 `openCVFramework.cpp`

Die Klasse `openCVFramework.cpp` beinhaltet die eigentliche Bildverarbeitung der Applikation und implementiert die in `openCVFramework.java` definierte native Funktion `nativeImageProcession(...)` (s. Kapitel 4.2.2).

Listing 4.14: Deklaration der `nativeImageProcession`-Funktion

```

1  #include "org_opencv_openCVFramework.h"

3  [...]

5  JNIEXPORT jdoubleArray JNICALL
6  Java_org_opencv_openCVFramework_nativeImageProcession
7  (JNIEnv *env, jclass thisObject, jbyteArray inJNIArray, jint
8  jrows, jint jcolumns, jint jchannels, jint jdepth){
    
```

Damit das JNI den Bezug von der nativen Funktion zu der Bibliothek und darüber zu deren Einbindung in Java herstellen kann, ist es nötig, dass die native Klasse den selben Namen hat, wie die Java-Klasse, in der die Funktion aufgerufen wird, d. h. wenn die Java-Klasse

`opencvFramework.java` heißt, muss der Name des nativen Äquivalents `opencvFramework.cpp` sein. Der Funktionsaufruf der nativen Funktion selbst muss demjenigen aus `org_opencv_opencvFramework.h` gleichen, damit die Referenz zu der Bibliothek aufgelöst werden kann. Das Schlüsselwort `JNIEXPORT` zeigt an, dass der nachfolgende Datentyp (in diesem Fall `jdoubleArray`) zurückgegeben wird; `JNICALL` bedeutet, dass die Funktion über das JNI aufgerufen wird.

Listing 4.15: Konvertierung der Java-Datentypen zu nativen Datentypen

```
1  Mat src, src_gray;
2  int  crowds, ccolumns, cchannels, cdepth, mat_size, pos, posr,
3  posc, vec_size, sub_vec_size;
4  vector<Vec3f> circles;

6  crowds = jrows;
7  ccolumns = jcolumns;
8  cchannels = jchannels;
9  cdepth = jdepth;
10 mat_size = crowds*ccolumns*ccchannels;

12 // Convert the incoming JNI bytearray to C's byte[] and jints
13 // to ints
14 signed char *inCArray= (env)->GetByteArrayElements(inJNIArray,
15 NULL);
16 if (inCArray == NULL){
17     return NULL;
18 }
```

Da die Speicherstruktur der Datentypen in Java sich teilweise von derjenigen in C++ unterscheidet, muss zunächst eine Konvertierung vorgenommen werden, um anschließend die Daten verwenden zu können. Das JNI stellt dafür die Datentypen `jint`, `jbyteArray`, etc. für die von Java übergebenen Argumente sowie Funktionen zur Umwandlung wie `GetByteArrayElements(...)` zur Verfügung.

Listing 4.16: Wiederherstellung der Bildmatrix aus dem übergebenen Array

```
1  // Read the image (re - construct the matrix from the byte -
2  // array)
3  // Color - image
4  // Check if the image format is correct (reduces the chance of
5  // an error that might occur whilst the jni - passing)
6  /*if (ccchannels == 4 && cdepth == 0){
7      src = Mat(crows,ccolumns, CV_8UC4);
8      for (int i = 0; i < crows; i++) {
9          for (int j = 0; j < ccolumns; j++) {
10             for (int k = 0; k < cchannels; k++){
11                 pos = i*ccolumns*ccchannels+j*ccchannels+k;
12                 // The matrix contains a vector of the given channel -
13                 // size for each pixel of the image, thus each vector
14                 // has to be filled one after another.
15                 src.at<Vec4b>(i, j)[k] = inCArray[pos];
16             }
17         }
18     }
19 }*/

21 // Gray - image
22 // Check if the image format is correct (reduces the chance of
23 // an error that might occur whilst the jni - passing)
```

```

24     if (cchannels == 1 && cdepth == 0){
25         src = Mat(crows, ccolumns, CV_8UC1);
26         for (int i = 0; i < crows; i++) {
27             for (int j = 0; j < ccolumns; j++) {
28                 pos = i*ccolumns+j;
29                 src.at<signed char>(i, j) = inCArray[pos];
30             }
31         }
32     }
33     else {
34         return NULL;
35     }

37     if (!src.data){
38         return NULL;
39     }
    
```

Vor der Verarbeitung muss aus dem übergebenen Bild-Array erneut eine Matrix erzeugt werden, da die OpenCV-Funktionalitäten den Datentyp Mat als Übergabewert erwarten. Dafür werden die die Informationen eines Bildpixels widerspiegelnden Einträge des Arrays (s. Kapitel 4.2.1) jeweils in den dem Pixel zugeordneten Vektor geschrieben. Entsprechen die Bildparameter nicht den dem gewählten Bildformat zugehörigen oder wird die Konvertierung nicht korrekt durchgeführt, so gibt die Funktion einen Nullpointer zurück.

Listing 4.17: Bildverarbeitung

```

1     // Color - image
2     // Convert image to gray
3     // cvtColor(src, src_gray, CV_BGR2GRAY);

5     /* Do stuff with the image here*/
6         /*
7             *
8             *
9             *
10            *
11            *
12            *
13            *
14            *
15            *
16            * */
    
```

Für den Fall, dass ein Farbbild übergeben wurde, besteht auch an dieser Stelle noch einmal die Möglichkeit, das Bild zu einem Graustufenbild zu konvertieren, um den Rechenaufwand bei der Bildverarbeitung zu reduzieren. Anschließend folgt das Herzstück der Applikation. An dieser Stelle kann die eigentliche Bildverarbeitung implementiert werden. Für eine ausführliche Dokumentation der durch OpenCV zur Verfügung gestellten Funktionen siehe auch <http://opencv.org/documentation.html>.

Listing 4.18: Konvertierung der Rückgabeargumente

```

1     // Return your stuff if you want to
2     double *outCArray;
3     outCArray = (double *) malloc((XXXXXXXXXX)); // Malloc to
4     // prevent the application from using new memory space on
5     // every call of this method

7     // Return the results
    
```



```

8     jdoubleArray outJNIArray = (env)->NewDoubleArray(XXXXXXXXXX);
9     // allocate
10    if (NULL == outJNIArray){
11        free(outCArray);
12        return NULL;
13    }

15    (env)->SetDoubleArrayRegion(outJNIArray, 0 , XXXXXXXXXXXX,
16    outCArray); // copy
17    free(outCArray);

19    return outJNIArray;
20 }

```

Abschließend besteht die Möglichkeit, Argumente, die über das JNI an Java zurückgegeben werden sollen, festzulegen. Standardmäßig ist die Rückgabe eines Double-Arrays implementiert. Dabei muss jedoch auch an dieser Stelle zunächst eine Konvertierung von einem C++-Array in die von Java verwendete Arraystruktur erfolgen. Dies geschieht, indem entsprechender Speicherplatz per `malloc(...)` reserviert, der Zeiger des Output-Arrays (in diesem Fall vom Typ `jdoubleArray`) auf diesen Speicherbereich gesetzt wird und anschließend der Inhalt des C++-Arrays per `SetDoubleArrayRegion(...)` in das Java-Array geschrieben wird. Misslingt die Speicherbelegung, so wird ein Nullpointer zurückgegeben.

4.4 Teil 3: XML

4.4.1 AndroidManifest.xml

Jede Android-Applikation benötigt ein Android-Manifest. In diesem werden dem System vorab essentielle Informationen bezüglich der Anwendung mitgeteilt. Ohne diese ist die Applikation nicht lauffähig [vgl. Gool6c]. Das Manifest ist in XML verfasst und enthält unter anderem folgendes [vgl. Gool6c]:

- Das Java-Paket, in dem sich die Applikation befindet sowie die mindestens auf dem ausführenden Gerät benötigte Android-Version:

Listing 4.19: Java-Paket und min. Android-Version

```

1     package="org.opencv"
2     android:versionCode="301"
3     android:versionName="3.01"

```

Die mindestens benötigte Android-Version orientiert sich dabei an den in der Applikation verwendeten Features.

- Die Grundeigenschaften der Applikation wie Name und Icon sowie die einzelnen Bestandteile wie Activities, Services, Intent-Filter, etc.:

Listing 4.20: Aufbau der Applikation

```

1     <application
2         android:label="@string/app_name"
3         android:icon="@drawable/ic_launcher"
4         android:theme="@android:style/Theme.NoTitleBar.
5         Fullscreen" >

6
7         <activity android:name="openCVActivity"
8             android:label="@string/app_name"
9             android:screenOrientation="landscape"
10            android:configChanges="keyboardHidden|

```

```

11         orientation"
12     >
13 </activity>

15 </application>
    
```

Im Falle des Frameworks existiert auf Android-Seite standardmäßig lediglich eine Activity. Besteht jedoch im Rahmen der Implementierung des Frameworks je nach abzudeckender Anwendung die Notwendigkeit, weitere Android-Komponenten hinzuzufügen, so sind diese im Manifest nach folgendem Muster zu ergänzen:

Listing 4.21: Ergänzen der Applikation um weitere Android-Komponenten

```

1  <application
2      ...

4      <activity android:name="XXX"
5          ...>
6          <intent-filter>
7              ...>
8          </intent-filter>
9      </activity>
10     <service
11         ...
12     </service>

14 </application>
    
```

Mittels des `<application>` Blocks werden innerhalb des Betriebssystems beim Starten der Applikation die Beziehungen zwischen den einzelnen Teilkomponenten hergestellt.

- Die mindestens zum Kompilieren der Applikation benötigte SDK-Version:

Listing 4.22: Mindestens benötigte SDK-Version

```

1  <uses-sdk android:minSdkVersion="12" />
    
```

Ebenso wie die mindestens benötigte Android-Version orientiert sich die SDK-Version an den in der Applikation verwendeten Features. Standardmäßig ist 12 angegeben, da diese Version bereits die Verwendung der meisten Kommunikations-Schnittstellen wie beispielsweise USB erlaubt, so dass diese bei Bedarf leicht implementiert werden können.

- Die für das Ausführen der Applikation benötigten Berechtigungen und die verwendeten Features:

Listing 4.23: Berechtigungen und Features

```

1  <uses-permission android:name="android.permission.
2  CAMERA"/>

4  <uses-feature android:name="android.hardware.camera"
5  android:required="true"/>
6  <uses-feature android:name="android.hardware.camera.
7  autofocus"
8  android:required="true"/>
    
```

Standardmäßig wird für die Applikation lediglich die Kamera des Mobilgeräts sowie die zugehörige Berechtigung benötigt. Werden in der nutzerspezifisch erstellten Anwendung weitere Features verwendet, sind diese an dieser Stelle im Manifest zu ergänzen.

Des Weiteren können im Android-Manifest bei komplexeren Applikationen festgelegt werden, mit welcher Activity die Anwendung startet und gegen welche Bibliotheken sie gelinkt werden muss [vgl. Goo16c].

4.4.2 Layout

Ein Layout definiert die visuelle Strukturierung eines User-Interfaces wie beispielsweise einer Activity. Das Layout kann dabei auf zwei Arten definiert werden [vgl. Goo16d]:

- Das Layout kann in einer XML-Datei deklariert werden, wodurch eine klare Trennung zwischen Code und optischer Gestaltung erreicht werden kann.
- Alternativ können die visuellen Elemente bei Ausführung der Anwendung initialisiert werden. Dazu können in der Applikation selbst sogenannte Views erstellt werden.

Wichtig ist dabei, dass neben einem übergeordneten Layout, in dem „globale“ (d. h. für die ganze Applikation gültige) Parameter deklariert werden, ein Layout für jedes erstellte User Interface angelegt werden muss. Im Falle des Frameworks sind dies die Dateien `activity_openCV.xml` und `openCVFramework_surface_view.xml`. Die Dateien sind grundsätzlich selbsterklärend, da die Parameter automatisch von der nächsten übergeordneten Instanz, d. h. vom Betriebssystem bezogen werden und sich ansonsten an der Standard-Displayauflösung des Mobilgeräts orientiert wird.

Weiterhin ist zu erwähnen, dass, falls im Android-Manifest ein Icon definiert wurde, die zugehörige Bilddatei ebenfalls im `res`-Ordner in den `drawable-XXX`-Verzeichnissen abgelegt werden muss.

5 ENTWICKLUNG DER APPLIKATION: VISUAL BASED LANDING SYSTEM

5.1 Genereller Aufbau

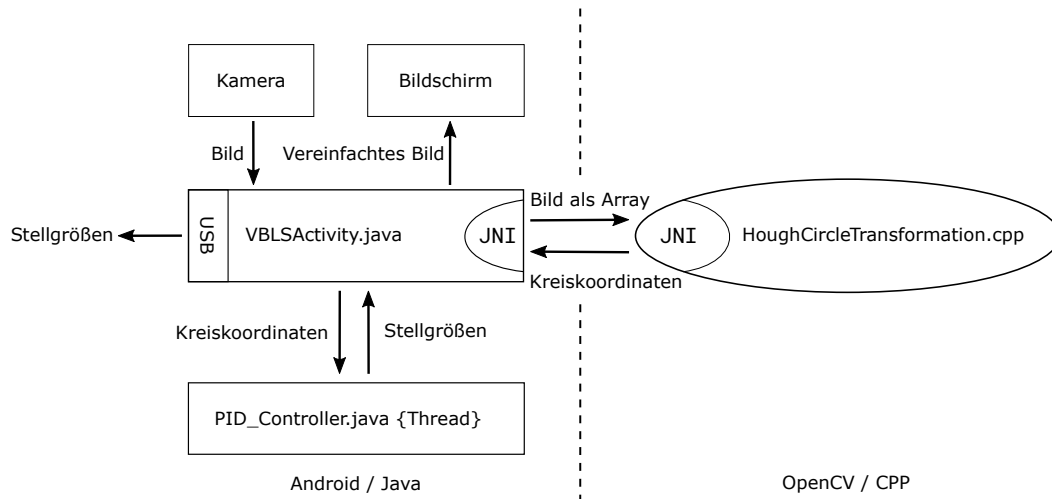


Abbildung 5.1: Funktionsdiagramm des Visual Based Landing Systems

Auf Basis der in Kapitel 4 entwickelten Vorlage wurde mit Visual Based Landing System (VBLS) zu Demonstrationszwecken eine Applikation für die automatische Landeplatzlokalisierung von UAVs (Abk., engl. für Unmanned Aerial Vehicle) entwickelt. Mögliche Anwendungsfälle wären beispielsweise die Ermöglichung der selbstständigen Detektion von Landestationen als Bestandteil eines autonomen Betriebs [vgl. Coc+15, S.1] oder das Auffinden eines vordefinierten Landeplatzes in ansonsten unbekanntem oder unwegsamem Gelände. Als Kennzeichnung des Landeplatzes wurde in Anlehnung an die Markierung eines Helikopter-Landeplatzes ein Kreis gewählt. Als Grundlage implementiert die Anwendung das OpenCV-Android-Framework. Um die Unterscheidbarkeit von den generalisierten Klassen des Frameworks zu gewährleisten wurden die für diese Applikation spezialisierten Klassen entsprechend eindeutig benannt. Die Namenskonvention lässt sich dabei wie folgt interpretieren:

- `openCVActivity.java` -> `VBLSActivity.java`
- `openCVFramework.xxx` -> `HoughCircleTransformation.xxx` (Hintergrund: Hough-Circle-Transformation ist die zur Detektion der Kreise genutzte Bildverarbeitungsfunktion)

Weiterhin wurden die Möglichkeit, dass das Mobilgerät als USB-Host gegenüber verbreiteten Mikrocontrollern wie dem Arduino fungieren und über diese Schnittstelle die erhaltenen Daten kommuniziert werden können, sowie ein PID-Regler zur Regelung der Ausgabe-werte in Abhängigkeit von der Position des Landeplatzes im Kamerabild, implementiert.

5.2 VBLSActivity.java

Die Klasse `VBLSActivity.java` wurde im Gegensatz zu `openCVActivity.java` um die Verarbeitung der in `HoughCircleTransformation.cpp` aus dem übergebenen Bild extrahierten Daten erweitert. Weiterhin wurde ein Regler in Form der Klasse `PID.java`, welcher die Zentrierung des UAVs über dem detektierten Landeplatz bewirkt, implementiert und die USB-Kommunikation realisiert.

Anmerkung: Da die USB-Kommunikation in Kapitel 5.5 separat betrachtet wird, wird an dieser Stelle nicht weiter darauf eingegangen.

Listing 5.1: Festlegung des Aufnahmeformats

```
1 // Color - images
2 //private Mat mRgba;

4 // Gray - images
5 private Mat mGray;

7 final static int IMG_WIDTH = 320;
8 final static int IMG_HEIGHT = 240;
```

Da die Natur der Anwendung eine möglichst hohe Echtzeitfähigkeit (d. h. in diesem Falle einfach möglichst viele Bildverarbeitungszyklen pro Zeitintervall) fordert, wurde die Priorität an dieser Stelle klar auf möglichst geringen Rechenaufwand gelegt und standardmäßig Graustufen als Format gewählt. Die Auflösung sollte für die beste Performanz möglichst so gering eingestellt werden, dass der als Landeplatz dienende Kreis aus der durchschnittlichen Flughöhe gerade noch so detektiert wird.

Listing 5.2: Implementierung von onCameraFrame

```
1 // Called on every frame received from the input - stream of
2 // the camera
3 public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
4     // Color - images
5     /*mRgba = inputFrame.rgba();
6     byte[] jImageData = transformationTools.jMatToArray(mRgba);
7     double[] jImageDataCircles = transformationTools.detect(
8     jImageData);*/

10    // Gray - images
11    mGray = inputFrame.gray();

13    byte[] jImageData = null;
14    try {
15        jImageData = transformationTools.jMatToArray(mGray);
16    }
17    // Terminates the application if the image data was damaged
18    // at some point
19    catch (IOException e){
20        Log.e(TAG, e.getMessage());
21        Log.e(TAG, "Transformation fault");
22        this.onDestroy();
23    }

25    double[] jImageDataCircles = null;
26    try {
27        jImageDataCircles = transformationTools.detect(jImageData);
28    }
29    // Terminates the application if the image data was damaged
30    // at some point
31    catch (IOException e){
32        Log.e(TAG, e.getMessage());
33        this.onDestroy();
34    }
35    // Fault in the image procession or conversion via JNI if
```

```

36 // jImageDataCircles is still null at this point
37 if (jImageDataCircles == null){
38     Log.e(TAG, "Image processing fault");
39     this.onDestroy();
40 }

42 // Calculates the nearest circle and writes the circle -
43 // coordinates in the USB - write - buffer
44 if (jImageDataCircles.length > 0){
45     jImageDataCircles = transformationTools.
46     calculateNearestCircle(jImageDataCircles, IMG_WIDTH,
47     IMG_HEIGHT, VEC_SUB_SIZE);

49 // PID - controller
50 pid.set_values(jImageDataCircles[0], jImageDataCircles[1]);

52 // Set the USB-output-data to the current actuating
53 // variables
54 double[] actuating_variables_copy = pid.
55 get_actuating_variables();
56 double x_temp = actuating_variables_copy[0]*(255.0/
57 IMG_WIDTH)+127.0;
58 double y_temp = actuating_variables_copy[1]*(255.0/
59 IMG_HEIGHT)+127.0;
60 if (x_temp > 255.0) {
61     usb_data[0] = (byte) 255.0;
62 }
63 else if (x_temp < 0.0) {
64     usb_data[0] = (byte) 0.0;
65 }
66 else {
67     usb_data[0] = (byte) x_temp;
68 }
69 if (y_temp > 255.0) {
70     usb_data[1] = (byte) 255.0;
71 }
72 else if (y_temp < 0.0) {
73     usb_data[1] = (byte) 0.0;
74 }
75 else {
76     usb_data[1] = (byte) y_temp; // Scaling to img_width to
77 // ensure identical x- and y-resolution (1 step in x
78 // equals the same distance in y)
79 }

81 usbService.write(usb_data);
82 }

84 // Color - images
85 //return mRgba;

87 // Gray - images
88 return mGray;
89 }
    
```

Die konkrete Implementation von `onCameraFrame(...)` sieht vor, dass nach der eigentli-

chen Kreisdetektion durch `HoughCircleTransformation.cpp` (vgl. Kapitel 5.4) zunächst überprüft wird, ob überhaupt Kreise (und somit potentielle Landeplätze) gefunden wurden (`jImageDataCircles.length > 0`). Ist dies der Fall, so wird mit Hilfe der Funktion `calculateNearestCircle(...)` (vgl. Kapitel 5.3) der Kreis, der sich am Nächsten zum Bildmittelpunkt befindet, ermittelt und zurückgegeben. Somit wird verhindert, dass bei mehreren Einträgen des Arrays `jImageDataCircles` immer nur der erste und damit bedingt durch die Natur des Suchalgorithmus der am weitesten links oben im Bild angeordnete Kreis verwendet wird. Die derart erhaltenen Koordinaten werden an den PID-Regler (vgl. Kapitel 5.6) übergeben, der wiederum auf dieser Basis die Stellgrößen anpasst. Diese Stellgrößen werden mittels `get_actuating_variables()` ausgelesen, auf einen Wertebereich von 0 bis 255 umgerechnet (Wertebereich eines Bytes; notwendig, damit bei der Konvertierung von `Double` zu `Byte` keine Informationen verloren gehen) und per `write(...)` über die USB-Schnittstelle ausgegeben. Wichtig ist, dass sowohl in x - als auch in y -Richtung ein fester Offset von 127 auf die Ausgabegrößen beaufschlagt werden muss, da diese sich um 0 herum bewegen, der Flugcontroller jedoch Werte um 127 herum erwartet.

Anmerkung: Der PID-Regler arbeitet threadbasiert. Um Nebenläufigkeitskomplikationen auszuschließen, ist es daher notwendig, beim Zugriff auf die Stellgrößen eine einmalige lokale Kopie dieser anzulegen und im Anschluss mit dieser Kopie zu arbeiten, anstatt sowohl für x - als auch für y -Richtung jeweils einzeln `get_actuating_variables()` aufzurufen. Sonst läuft man Gefahr, dass die Stellgrößen zwischen dem ersten und dem zweiten Zugriff aktualisiert wurden und die beiden ausgelesenen Werte nicht mehr miteinander korrelieren.

5.3 HoughCircleTransformation.java

`HoughCircleTransformation.java` implementiert im Gegensatz zu `openCVFramework.java` zusätzlich die Funktion `calculateNearestCircle(...)`.

Listing 5.3: `calculateNearestCircle`

```
1 // Calculates the circle closest to the middle of the image
2 double[] calculateNearestCircle(double[] circles, int
3 img_width, int img_height, int vec_sub_size){
4     int index_nearest_circle = 0;
5     double min_dist = Math.sqrt(Math.pow((circles[
6     index_nearest_circle]-0.5*img_width), 2)+Math.pow((circles[
7     index_nearest_circle+1]-0.5*img_height), 2));
8     double dist;
9     for(int i = 0; i < circles.length; i += vec_sub_size){
10         dist = Math.sqrt(Math.pow((circles[i]-0.5*img_width), 2)+
11         Math.pow((circles[i+1]-0.5*img_height), 2));
12         if (dist < min_dist){
13             min_dist = dist;
14             index_nearest_circle = i;
15         }
16     }
17     double[] nearest_circle = {circles[index_nearest_circle],
18     circles[index_nearest_circle+1]};
19     return nearest_circle;
20 }
```

Die Funktion `calculateNearestCircle(...)` überprüft anhand der Länge des die Kreiskoordinaten enthaltenden Arrays, ob bei der Bildverarbeitung mehrere Kreise detektiert wurden. Falls dies der Fall sein sollte, wird der dem Bildmittelpunkt am Nächsten gelegene Kreis ermittelt und zurückgegeben. Dazu wird zunächst der Abstand des ersten Positions-Eintrags des Arrays als minimaler Abstand festgelegt und anschließend für alle weiteren Kreiskoordinaten überprüft, ob deren Abstand zum Bildmittelpunkt geringer ist. Ist dies der Fall, wird

der entsprechende Eintrag als neue Referenz gespeichert. Somit bleibt am Ende derjenige Kreis mit der geringsten Entfernung zur Bildmitte übrig. Dies ist notwendig, da ansonsten immer der am weitesten links oben angeordnete Kreis als Ziel gewählt würde, da die Bildverarbeitung das Bild von der linken oberen Ecke aus durchläuft und somit dessen Koordinaten an erster Stelle im Array stehen würden.

Anmerkung: Der Offset von $0.5 \cdot \text{img_width}$ bzw. $0.5 \cdot \text{img_height}$ ergibt sich dadurch, dass der Nullpunkt des Bildes standardmäßig in der oberen linken Ecke liegt.

5.4 HoughCircleTransformation.cpp

`HoughCircleTransformation.cpp` basiert auf der Klasse `openCVFramework.cpp` des Frameworks und implementiert die OpenCV-Methoden zur Lokalisierung von Kreisen und damit potentiellen Landeplätzen.

Listing 5.4: Kreisdetektion

```

1  // Reduce the noise so false circle detection is avoided (or
2  // rather reduced)
3  GaussianBlur(src_gray, src_gray, Size(9, 9), 2, 2);

5  // Apply the Hough Transform to find the circles
6  HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT, 1.0,
7  (double) src_gray.rows/8, 150.0, 75.0, 0, 0);
    
```

Nachdem das in Byte-Array-Form per JNI übergebene Bild erneut zurück in eine Matrix umgewandelt wurde (s. Kapitel 4.3.2, wird zunächst mittels `GaussianBlur(...)` eine Reduzierung des Rauschens im Bild durchgeführt. Dafür werden jeweils die Pixel in direkter Nachbarschaft zum aktuell betrachteten Bildpunkt untersucht und deren Farb- oder Helligkeitswert (je nachdem, ob Farb- oder Graustufenbilder verwendet werden) in Relation gesetzt. Ist also beispielsweise das aktuelle Pixel schwarz, alle benachbarten Bildpunkte jedoch weiß, so wird der Wert des Pixels ebenfalls auf weiß gesetzt. Diese Verminderung des Rauschens ist notwendig, um bei der nachfolgenden Objektllokalisierung bessere Ergebnisse erzielen zu können.

Anschließend wird mittels `HoughCircles(...)` die eigentliche Kreisdetektion durchgeführt. Diese basiert ebenfalls darauf, dass Cluster von benachbarten, gleichfarbigen Pixeln betrachtet werden und zu jedem dieser Cluster ein zugehöriger Kreisradius und -mittelpunkt interpoliert werden. Liegen genügend Mittelpunkte von Clustern mit gleichem Radius in einem gewissen Bereich, so wird angenommen, dass diese Pixel zu einem Kreis gehören. Die Attribute der gefundenen Kreise (bestehend aus den Koordinaten des Mittelpunkts und dem Radius) werden in einem Array von Vektoren, `circles`, gespeichert. Die letzten fünf Übergabeparameter der Funktion stellen dabei konfigurierbare Parameter wie beispielsweise minimale und maximale Radien oder Thresholds für die Kantendetektion dar.

Für genauere Informationen kann die Dokumentation der Funktion unter http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles konsultiert werden.

Listing 5.5: Transformation von Vektor zu Array

```

1  // Convert the vector containing the coordinates and the size
2  // of the detected circles to the output-array
3  vec_size = circles.size();
4  sub_vec_size = 3;
5  double *outCArray;
6  outCArray = (double *) malloc((vec_size*sub_vec_size)*sizeof(
7  double)); // Malloc to prevent the application from using
8  // new memory space on every call of this method
9  for (int i = 0; i < vec_size; i++) {
    
```



```
10     for (int j = 0; j < sub_vec_size; j++) {  
11         outCArray[i*sub_vec_size+j] = cvRound((double)circles[i][  
12             j]);  
13     }  
14 }
```

Um die extrahierten Daten anschließend wieder über das JNI an `VBLSAActivity.java` zurückgeben zu können, muss die Vektordatenstruktur von `circles` in ein Array eines nativen Datentyps, in diesem Falle `Double`, konvertiert werden. Dafür wird zunächst die Größe des benötigten Feldes berechnet und der entsprechende Speicherplatz per `malloc(...)` reserviert. Anschließend werden die einzelnen den detektierten Kreisen zugehörigen Vektoren nacheinander ausgelesen und deren Inhalt in das Array geschrieben. Die `sub_vec_size` von drei kommt daher, dass jedem Kreis sowohl der Radius als auch die Koordinaten des Mittelpunkts zugeordnet werden. Anschließend wird das Array wie in `openCVFramework.cpp` (s. Kapitel 4.3.2) per JNI zurückgegeben.

5.5 USB-Kommunikation

5.5.1 Anforderungen an die zu verwendende Bibliothek

Wie bereits in Kapitel 5.1 beschrieben, verfügen Android-Mobilgeräte standardmäßig nicht über die Möglichkeit, als Host gegenüber anderen USB-Geräten zu fungieren. Für diesen Zweck stellt das Betriebssystem das sogenannte Open Accessory zur Verfügung [vgl. Sch16, S.199ff.]. Dabei simuliert das Android dem angeschlossenen USB-Gerät, der Host zu sein, obwohl in Realität letzteres die Host-Funktion übernimmt. Diese Funktionalität ist bisher jedoch nur in wenigen Chips von FTDI integriert und somit sehr unflexibel hinsichtlich der Gerätekompatibilität.

Alternativ zu Open Accessory besteht die Möglichkeit, eine von mehreren zur Verfügung stehenden Bibliotheken inklusive Treibern für die anzusprechenden Endgeräte in die Applikation zu integrieren. Von zentraler Wichtigkeit bei der Auswahl der passenden Bibliothek ist deren Lizenz. Um der in der Zielsetzungen des Projekts (s. Kapitel 1.2) geforderte möglichst einfache Zugänglichkeit zu gewährleisten, ist es wichtig, ein Projekt unter einer Open Source-Lizenz auszuwählen. Von technischer Seite aus sollte besonderes Augenmerk auf die Flexibilität bzgl. der USB-Chips, die jede der Bibliotheken unterstützt, gelegt werden, um eine möglichst große Anzahl an Endgeräten unterstützen zu können. Weiterhin ist es wichtig, dass asynchrone Kommunikation unterstützt wird, um die Applikation nicht unnötig zu verlangsamen (die Bildverarbeitung im UI-Thread kann weiter stattfinden, während der USB-Thread kommuniziert). Zuletzt ist eine gute Dokumentation und Verständlichkeit der Bibliothek wichtig, um die Implementierung einfach und korrekt durchführen zu können.

Auf Grundlage dieser Kriterien wurde sich für die USB-Serial-Bibliothek von Felipe Herranz, zu finden unter <https://github.com/felHR85/UsbSerial>, unter der MIT-Lizenz ausgewählt.

5.5.2 Einbindung der Bibliothek

Bei der Entwicklung von Android-Applikationen wird bzgl. des funktionalen Codes zwischen aktivem Code (Activities) und Programmabschnitten, die dauerhaft im Hintergrund ablaufen (Services), unterschieden. In letztere Kategorie fällt auch die USB-Kommunikation mit einem anderen Endgerät. Die Verbindung wird von einem im Hintergrund ablaufenden Service aufrechterhalten, der in einer Activity initiiert und gebunden wird. Dementsprechend lässt sich auch die Implementierung der USB-Serial-Bibliothek in zwei Abschnitte unterteilen:

- Erstellung des Services
- Einbindung in die Activity

Die der Implementierung des Services dienliche Klasse `USB_Service.java` orientiert sich stark an der der Bibliothek beiliegenden Beispiel-Implementation (ebenfalls zu finden unter <https://github.com/felHR85/UsbSerial>) und ist an dieser Stelle gut dokumentiert. Dementsprechend soll an dieser Stelle nicht weiter darauf eingegangen, sondern sich lediglich auf die Integration in `VBLSAActivity.java` fokussiert werden. Einzig erwähnenswert an dieser Stelle ist, dass mittels der zu Beginn des Services definierten Variable `BAUD_RATE` die für die Kommunikation zu verwendende Baud-Rate eingestellt werden kann.

Anmerkung: Für Nutzer, die neu in der Android-Entwicklung sind, ist es grundsätzlich empfehlenswert, sich die offizielle Dokumentation zu Services unter <https://developer.android.com/guide/components/services.html> durchzulesen.

Listing 5.6: Initialisierung des USB-Daten-Arrays

```
1      byte[] usb_data;

3      [...]

5      usb_data = new byte[2];
```

Zunächst wird ein Daten-Array initialisiert, in das die zu versendenden Daten geschrieben werden. Dies dient zur Bufferung bis zur eigentlichen Übertragung.

Listing 5.7: Aufbau der Service-Connection

```
1      private final ServiceConnection usbConnection = new
2      ServiceConnection() {
3          @Override
4          public void onServiceConnected(ComponentName arg0,
5          IBinder arg1) {
6              usbService = ((USB_Service.UsbBinder) arg1).
7              getService();
8              usbService.setHandler(usbHandler);
9          }

11         @Override
12         public void onServiceDisconnected(ComponentName arg0) {
13             usbService = null;
14         }
15     };

17     [...]

19     private void startService(Class<?> service, ServiceConnection
20     serviceConnection, Bundle extras) {
21         if (!USB_Service.SERVICE_CONNECTED) {
22             Intent startService = new Intent(this, service);
23             if (extras != null && !extras.isEmpty()) {
24                 Set<String> keys = extras.keySet();
25                 for (String key : keys) {
26                     String extra = extras.getString(key);
27                     startService.putExtra(key, extra);
28                 }
29             }
30             startService(startService);
31         }
32         Intent bindingIntent = new Intent(this, service);
33         bindService(bindingIntent, serviceConnection, Context.
```

```
34         BIND_AUTO_CREATE);  
35     }
```

Mittels der Service-Connction sowie der Funktion `startService` wird der Service aus der Activity heraus gestartet und an die Activity gebunden. Die Bindeanfrage wird dabei mittels des Intents `bindingIntent` über die Funktion `bindService` kommuniziert. `startService` wird sowohl bei erstmaligem Erstellen der Applikation als auch, wenn die Anwendung aus gestopptem Zustand heraus fortgesetzt wird mittels `onResume()` aufgerufen. Bei Pausieren (`onPause()`) wird der Service gestoppt und der zugehörige Broadcast-Receiver mittels `unregisterReceiver(...)` entfernt.

Listing 5.8: Einrichtung des Broadcast-Receiver

```
1     private final BroadcastReceiver mUsbReceiver = new  
2     BroadcastReceiver() {  
3         @Override  
4         public void onReceive(Context context, Intent intent) {  
5             switch (intent.getAction()) {  
6                 case USB_Service.ACTION_USB_PERMISSION_GRANTED:  
7                     // USB PERMISSION GRANTED  
8                     Toast.makeText(context, "USB Ready", Toast.  
9                         LENGTH_SHORT).show();  
10                    break;  
11                case USB_Service.  
12                ACTION_USB_PERMISSION_NOT_GRANTED: // USB  
13                // PERMISSION NOT GRANTED  
14                    Toast.makeText(context, "USB Permission not  
15                        // granted", Toast.LENGTH_SHORT).show();  
16                    break;  
17                case USB_Service.ACTION_NO_USB: // NO USB  
18                // CONNECTED  
19                    Toast.makeText(context, "No USB connected",  
20                        Toast.LENGTH_SHORT).show();  
21                    break;  
22                case USB_Service.ACTION_USB_DISCONNECTED: // USB  
23                // DISCONNECTED  
24                    Toast.makeText(context, "USB disconnected",  
25                        Toast.LENGTH_SHORT).show();  
26                    break;  
27                case USB_Service.ACTION_USB_NOT_SUPPORTED: // USB  
28                // NOT SUPPORTED  
29                    Toast.makeText(context, "USB device not  
30                        // supported", Toast.LENGTH_SHORT).show();  
31                    break;  
32            }  
33        }  
34    };  
35  
36    [...]  
37  
38    private void setFilters() {  
39        IntentFilter filter = new IntentFilter();  
40        filter.addAction(USB_Service.  
41        ACTION_USB_PERMISSION_GRANTED);  
42        filter.addAction(USB_Service.ACTION_NO_USB);  
43        filter.addAction(USB_Service.ACTION_USB_DISCONNECTED);  
44        filter.addAction(USB_Service.ACTION_USB_NOT_SUPPORTED);
```

```

45         filter.addAction(USB_Service.
46             ACTION_USB_PERMISSION_NOT_GRANTED);
47         registerReceiver(mUsbReceiver, filter);
48     }
    
```

Um miteinander zu kommunizieren, nutzen Activities und Services die Broadcast-Funktionalität. Dabei kann jede Komponente sogenannte Intents oder Messages über das Broadcast-System versenden. Andere Komponenten können Broadcast-Receiver (s. Kapitel 2.1.1) einrichten, um allen auf diese Art im System gesendeten Nachrichten zu lauschen und Aktionen für bestimmte Inhalte definieren. Mittels `setFilters()` können die zu empfangenden Intents oder Messages festgelegt werden; die Komponente *abboniert* sie. Anschließend wird der Receiver mittels `registerReceiver(...)` registriert. Im konkreten Fall werden lediglich Nachrichten des Services empfangen.

Listing 5.9: Einrichten und Weiterreichen des USB-Handlers

```

1    /*
2     * This handler will be passed to UsbService. Data received
3     * // from serial port is displayed through this handler
4     */
5     private static class usbHandler extends Handler {
6         private final WeakReference<VBLSActivity> mActivity;

7
8         public usbHandler(VBLSActivity activity) {
9             mActivity = new WeakReference<>(activity);
10        }

11
12        @Override
13        public void handleMessage(Message msg) {
14            switch (msg.what) {
15                case USB_Service.MESSAGE_FROM_SERIAL_PORT:
16                    String data = (String) msg.obj;
17                    Log.d(TAG, data);
18                    break;
19                case USB_Service.CTS_CHANGE:
20                    Toast.makeText(mActivity.get(), "CTS_CHANGE",
21                        Toast.LENGTH_LONG).show();
22                    break;
23                case USB_Service.DSR_CHANGE:
24                    Toast.makeText(mActivity.get(), "DSR_CHANGE",
25                        Toast.LENGTH_LONG).show();
26                    break;
27            }
28        }
29    }

30    [...]

31    [...]

32    private usbHandler usbHandler;

33    [...]

34    usbHandler = new usbHandler(this);

35    [...]

36    usbService.setHandler(usbHandler);
    
```

Eine weitere Möglichkeit der Kommunikation zwischen Activity und Service besteht neben dem Versenden von Intents über das Broadcast-System im Überliefern von Messages mittels Handlern. Wie bereits in Kapitel 2.1.1 beschrieben, besteht die Aufgabe von Intents darin, Anfragen zu übermitteln, während Messages reine Mitteilungen darstellen. Die Klasse `usbHandler` dient zur Verarbeitung der zwischen beiden Komponenten versendeten Messages. In ihr werden verschiedene Aktionen für bestimmte Nachrichten wie bspw. das Ausgeben der vom Endgerät empfangenen Daten bei `USB_Service.MESSAGE_FROM_SERIAL_PORT` (für die konkrete Anwendung uninteressant, da nur Daten versendet und nicht empfangen werden) definiert. Anschließend wird das in der Activity erzeugte und mittels `WeakReference` gebundene Objekt per `setHandler(...)` an den Service übergeben, wodurch es diesem ermöglicht wird, Messages über den Handler zu versenden.

5.6 PID.java

Wie bereits in der Zielsetzung (vgl. Kapitel 1.2) definiert, soll die zu Demonstrationszwecken dienende Applikation zuverlässig einen Landeplatz zu lokalisieren und es dem UAV ermöglichen, sich über diesem zu zentrieren. Da es sich jedoch bei einem UAV im Normalfall um ein träges, nicht mechanisch geführtes System handelt, das weiterhin von Einflüssen umwelttechnischer oder mechanischer Art wie bspw. Windböen oder einer ungenauen Kalibrierung der Sensorik betroffen sein kann, ist es für eine sichere Positionierung über einem detektierten Landeplatzes unerlässlich, eine Positionsregelung zu implementieren, um die Fluggeschwindigkeit zu regulieren und starke Überschwinger zu vermeiden. Dies geschieht mittels der Klasse `PID.java` in Form eines klassischen PID-Reglers.

Listing 5.10: Hilfsklasse für Koordinatentupel

```
1  // Immutable class to represent a coordinate tuple
2  final class Coordinates_Immutable {
3      final private double x;
4      final private double y;

6      Coordinates_Immutable (double x, double y) {
7          this.x = x;
8          this.y = y;
9      }

11     final double get_x () {
12         return x;
13     }

15     final double get_y () {
16         return y;
17     }
18 }
```

Da für die Realisierung des Reglers ein threadbasierter Ansatz gewählt wurde und im Bezug auf die Stellgrößen auf der Basis von Referenzen gearbeitet wird (s. u.), wird eine Klasse benötigt, deren Objekte es erlauben, ein Koordinatenpaar zu übergeben und auf diese Werte zuzugreifen. Diesen Zweck erfüllt `Coordinates_Immutable`, da Java standardmäßig keine Tupel unterstützt. Die Klasse ist aus Gründen der Zugriffssicherheit nach dem Entwurfsmuster „Immutable“ (einmalige schreibende Zuweisung der Werte, ansonsten lediglich lesender Zugriff) gestaltet.

Listing 5.11: Erläuterung der Parameter

```
1  // Constructor
2  public PID_Controller (double Kp, double Ki, double Kd, double
3      set_point_x, double set_point_y, long dt) {
```

```

4      error = new AtomicReference<Coordinates_Immutable>();
5      actuating_variables = new AtomicReference
6      <Coordinates_Immutable>();

8      set_point = new double[2];
9      preError = new double[2];

11     this.Kp = Kp;
12     this.Ki = Ki;
13     this.Kd = Kd;

15     this.set_point_x = set_point_x;
16     this.set_point_y = set_point_y;

18     this.dt = dt;

20     set_values(set_point_x, set_point_y); // Initialize error so
21     // the first access in run won't throw a nullpointer-
22     // exception
23 }
    
```

Wie bereits zu Beginn der Sektion angeführt, weist der implementierte Regler in seiner Grundform eine standardmäßige PID-Charakteristik auf. Folglich entsprechen die übergebenen Werte für K_p , K_i und K_d den Koeffizienten für den Proportional-, Integral- und Differentialanteil, sowie `set_point_x` und `set_point_y` dem Arbeitspunkt. Da sowohl der I-, als auch der D-Anteil eines PID-Reglers zeitabhängig sind, wurde zur Realisierung dieser Funktionalität ein threadbasierter Ansatz gewählt. Die `Run`-Methode des Threads führt dabei jeweils die Berechnung aus. Anschließend schläft der Thread bis zu seinem nächsten Aufruf und gibt belegte Ressourcen frei (`sleep(...)`). Mittels `dt` kann die Schrittweite bzw. das Zeitintervall, in dem der Regler aufgerufen wird, festgelegt werden (in Millisekunden).

Da nun sowohl schreibender (auf die Eingangsgrößen) als auch lesender (auf die Stellgrößen) Zugriff aus einem anderen Thread (dem Main-Thread) auf Elemente des Regler-Threads erfolgen soll, ist es notwendig, die betroffenen Objekte threadsicher zu implementieren. Ansonsten kann es zu Nebenläufigkeitsproblematiken kommen, z. B. erster Wert der Stellgröße wird ausgelesen -> Aktualisierung der Stellgröße von Seiten des Reglers -> zweiter Wert wird ausgelesen -> ausgelesene Werte korrelieren nicht miteinander. Zu diesem Zweck werden die betroffenen Elemente als threadsichere Referenzen, sogenannten **AtomicReferences**, auf Objekte der Klasse `Coordinates_Immutable` implementiert. Somit erfolgt der Zugriff auf die Objekte selbst threadsicher und es wird immer auf beide betroffenen Werte (x - und y -Richtung) gleichzeitig zugegriffen.

Listing 5.12: Lesender und schreibender Zugriff auf threadsichere Objekte

```

1  // Gets the actual actuating variable
2  public double[] get_actuating_variables () {
3      Coordinates_Immutable actuating_variables_copy =
4      actuating_variables.get(); // Get a local copy of the
5      // reference first to avoid access-issues
6      double[] actuating_variables_copy_values = {
7      actuating_variables_copy.get_x(), actuating_variables_copy.
8      get_y()};
9      return actuating_variables_copy_values;
10 }

12 // Sets error values through creating a new
13 // Coordinates_Immutable-object and setting the reference of
14 // error towards it
    
```

```

15     public void set_values (double x_value, double y_value) {
16         error.set(new Coordinates_Immutable(x_value-set_point_x,
17         y_value-set_point_y));
18         // notify(); // Use for sole P - controllers
19     }

```

Lesender Zugriff auf die durch threadsichere Referenzen gekapselten Elemente erfolgt, indem zunächst mittels `get()` das Objekt selbst und anschließend dessen Inhalt zurückgegeben wird. Dabei ist auch an dieser Stelle darauf zu achten, dass um Nebenläufigkeitsproblematiken zu vermeiden, `get()` genau einmal aufgerufen und anschließend mit einer Kopie der Objekte gearbeitet werden sollte. Eine flache Kopie (lediglich die Referenz) ist ausreichend, da bei einer Aktualisierung der Stellgrößen nicht die Werte der Inhalt des Objekts verändert werden (nicht möglich, da immutable), sondern ein komplett neues Objekt erzeugt wird (genau ein schreibender Aufruf und nicht zwei und damit erneut Anfälligkeit für Nebenläufigkeitsproblematiken).

Dieses Vorgehen kann man ebenfalls bei der Funktion `set_values(...)` sehen, mittels der aus dem Main-Thread bei jedem verarbeiteten Bild die Eingangsgrößen geschrieben werden. Auch an dieser Stelle werden nicht die Werte verändert, sondern die `AtomicReference` auf ein neues Objekt der Klasse `Coordinates_Immutable` mit den neuen Eingangsgrößen gesetzt. Das „alte“ Element wird somit nicht länger verwendet und vom Garbage Collector der JVM freigegeben.

Listing 5.13: Berechnung der Stellgrößen

```

1     // Thread to calculate the actuating variables for a given
2     // relative positioning towards a given point
3     @Override
4     public void run () {
5         Coordinates_Immutable error_copy;
6         while (true) {
7             error_copy = error.get(); // Get a local copy of the
8             // reference first to avoid access-issues
9             integral = integral+(error_copy.get_x()*dt/
10             MILLISEC_TO_SEC);
11             derivative = (error_copy.get_x()-preError[0])*
12             MILLISEC_TO_SEC/dt;

14             integral = integral+(error_copy.get_y()*dt/
15             MILLISEC_TO_SEC);
16             derivative = (error_copy.get_y()-preError[1])*
17             MILLISEC_TO_SEC/dt;

19             actuating_variables.set(new Coordinates_Immutable(Kp*
20             error_copy.get_x()+Ki*integral+Kd*derivative, Kp*
21             error_copy.get_y()+Ki*integral+Kd*derivative));

23             preError[0] = error_copy.get_x();
24             preError[1] = error_copy.get_y();

26             // Use for sole P - controllers
27             /*try {
28                 wait();
29             }
30             catch (InterruptedException e) {
31                 e.printStackTrace();
32             }*/

```



```

34         // Use for every other controller
35         try {
36             sleep(dt);
37         }
38         catch (InterruptedException e) {
39             e.printStackTrace();
40         }
41     }
42 }
    
```

Die Berechnung der Stellgrößen erfolgt innerhalb der Run-Methode der Klasse. Dafür wird jeweils zu Beginn eine Kopie der aktuellen Eingangsgrößen angelegt. Auf dieser Grundlage können anschließend die jeweiligen Proportional-, Integral- und Differentialanteile berechnet und mittels `set(...)` als neues Objekt der Klasse `Coordinates_Immutable` der thread-sicheren Referenz `actuating_variables` zugewiesen werden. Ist die Berechnung abgeschlossen, versucht der Thread für das bei der Initialisierung übergebene Zeitintervall die belegten Ressourcen mittels `sleep(...)` freizugeben.

5.7 AndroidManifest.xml

Durch das Hinzukommen der Kommunikation über die USB-Schnittstelle des Mobilgeräts ändern sich sowohl die allgemeine Struktur der Applikation als auch die verwendeten Android-Features. Dies muss dementsprechend auch im Android-Manifest angepasst werden.

Listing 5.14: Aufbau der Applikation ergänzt um Intent-Filter und USB-Service

```

1     <application
2         android:label="@string/app_name"
3         android:icon="@drawable/ic_launcher"
4         android:theme="@android:style/Theme.NoTitleBar.
5         Fullscreen" >

6
7         <activity android:name="VBLSActivity"
8             android:label="@string/app_name"
9             android:screenOrientation="landscape"
10            android:configChanges="keyboardHidden|
11            orientation"
12            >
13            <intent-filter>
14                <action android:name="android.intent.action.MAIN"
15                />
16                <category android:name="android.intent.category.
17                LAUNCHER" />
18            </intent-filter>
19        </activity>
20        <service
21            android:name="org.opencv.visual_based_landing_system.
22            USB_Service"
23            android:enabled="true"
24            >
25        </service>
26    </application>
    
```

Zusätzlich zu der eigentlichen Activity implementiert die Anwendung einen Service, um die USB-Schnittstelle handzuhaben (s. Kapitel 5.5). Weiterhin wird in `VBLSActivity.java` ein Intent-Filter eingebunden, um zwischen Service und Activity kommunizieren zu können. Dies spiegelt sich in dieser Form auch in der in `<application>` eingebetteten Struktur der Anwendung wieder.

Listing 5.15: Verwendete Features

```
1 <uses-feature android:name="android.hardware.usb.host"  
2 android:required="true"/>  
3 <uses-feature android:name="android.hardware.camera"  
4 android:required="true"/>  
5 <uses-feature android:name="android.hardware.camera.  
6 autofocus" android:required="true"/>
```

Des Weiteren muss der Liste der verwendeten Features die USB-Schnittstelle hinzugefügt werden.

6 VERSUCHSAUFBAU UND TESTS

6.1 Versuchsaufbau

Dieses Projekt teilt sich einen gemeinsamen Versuchsaufbau mit einem Parallelprojekt, der Entwicklung eines echtzeitfähigen RC-Mischers auf Arduino-Basis¹. Im Folgenden soll ein Überblick über die Anordnung der Komponenten gegeben werden.

Der Versuchsaufbau besteht aus einem Quadrokopter in X-Anordnung. Die Hauptdiagonale zwischen zwei Rotoren beträgt 415mm, der Rotordurchmesser beträgt 210mm.

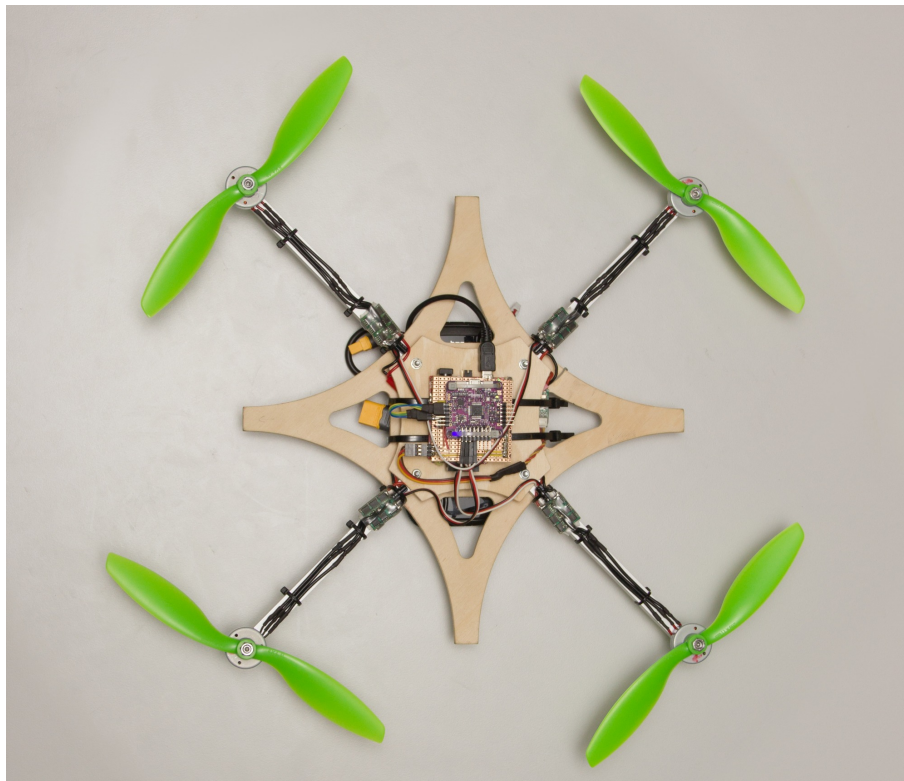


Abbildung 6.1: Draufsicht auf die Oberseite des Versuchsaufbaus

Die Masse des Aufbaus beträgt 375g. Zu dieser kommen weitere 98g für den verwendeten zwei-Zellen-LiPo vom Typ „Gens ace 1800mAh-20C-7.4V-2S1P“, sowie 142g für das verwendete Mobiltelefon vom Typ „HTC One S“ hinzu. Für die Gesamtmasse des flugbereiten Aufbaus ergibt sich folglich ein Gewicht von 515g. Die maximale Tragfähigkeit des Quadrokopters ist nicht bekannt, das Flugverhalten bei diesem Gewicht suggeriert jedoch eine Auslastung von etwa 70%-80%.

Anhand von Abbildung 6.3 kann der schichtweise Aufbau des Testgeräts gut zu nachvollzogen werden:

- Naze32-kompatible Flugsteuerung.
- Arduino-Leonardo-kompatibler Olimexino 32u4 mit eigens angefertigter Adapterplatine, um die Verdrahtung der Komponenten entsprechend den Anforderungen des vibrationsträchtigen Aufbaus sicher zu gestalten (siehe Kapitel 6.2).

¹LeonardoMixerIO - <https://gitlab.cvh-server.de/lf.ps/vbls/tree/master/LeonardoMixerIO>



Abbildung 6.2: Draufsicht auf die Unterseite des Versuchsaufbaus

- Der ursprünglich von Weber im Rahmen eines Vorgängerprojekts konstruierte Quadrocopter mit neu ausgeführter Verdrahtung und Ergänzungen [vgl. Web].
- Akku
- Mobiltelefon mit VBLS-Applikation

6.2 Mischer

Um das UAV außerhalb der autonomen Landeplatzsuche steuern, sowie bei unerwarteten Situationen auch während des autonomen Betriebs manuell in das Flugverhalten eingreifen zu können, ist ein Mischer notwendig. Dieser hat den Zweck, die Eingangssignale der Fernbedienung und die Ausgabedaten der Applikation miteinander zu verarbeiten und in ein einheitliches, von dem Flugkontroller des UAVs verstandenes Protokoll zu konvertieren. Die angeführten Funktionalitäten werden durch den im Rahmen der Veranstaltung „Systemtechnik“ entstandenen echtzeitfähigen RC-Mischer auf Arduino-Basis realisiert. An dieser Stelle soll daher lediglich auf die Dokumentation dieses eigenständigen Projekts verwiesen werden:



<https://gitlab.cvh-server.de/lf.ps/vbbs/tree/master/LeonardoMixerIO>

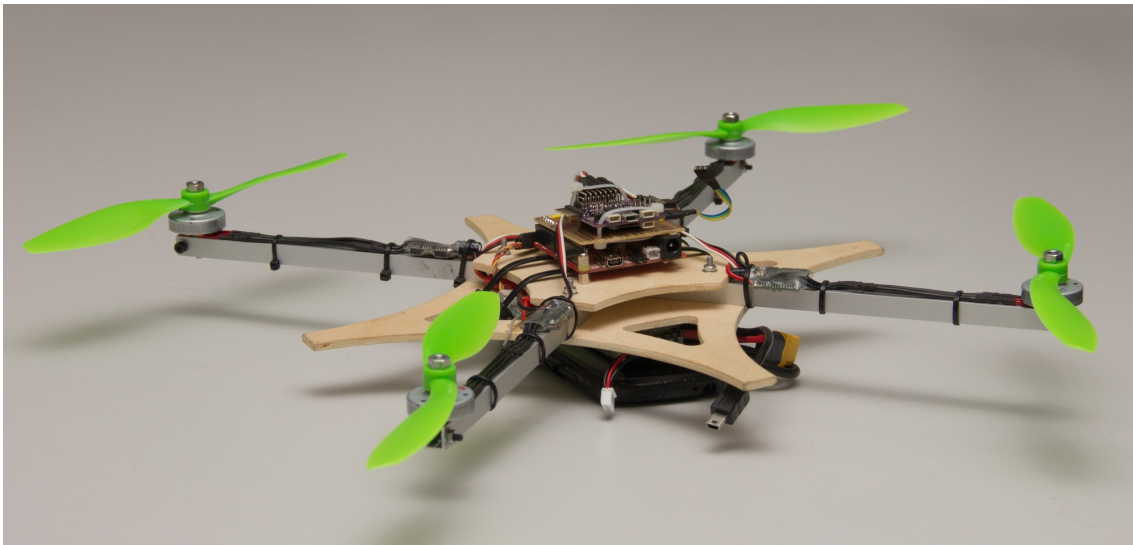


Abbildung 6.3: Ansicht des Versuchsaufbaus im Profil

6.3 Einstellung des Reglers

Um ein sicheres autonomes Flugverhalten während der Landeplatzdetektion realisieren zu können, ist eine korrekte Parametrierung des Reglers notwendig (vgl. Kapitel 5.6). Zu diesem Zweck muss zunächst die Art des betrachteten Systems bestimmt werden. Dies kann beispielsweise durch die Betrachtung der Übergangsfunktion als Antwort auf eine Beaufschlagung des Systems mit einem Sprung des Eingangssignals (in diesem Fall z. B. das Erscheinen eines Kreises am äußeren Rand des betrachteten Bildes) geschehen. Mittels dieser Methode ergibt sich, dass es sich bei dem vorliegenden System, um ein IT1-System handelt. D. h. die Ausgangsgröße des Systems, in diesem Fall der zurückgelegte Weg des Quadropters in x - oder y -Richtung, verhält sich mit einer gewissen Anlaufverzögerung integral zur Eingangsgröße, dem Steuersignal in der entsprechenden Dimension. Dies entspricht dem erwarteten Verhalten, dass das UAV bei einem Sprung des Steuersignals nach einer Beschleunigungsphase (bedingt durch seine Massenträgheit) eine lineare Änderung der Trajektorie aufweist.

Um nun dieses Systemverhalten stabilisieren zu können, muss ein PD-Regler verwendet werden. Somit lässt sich bereits an dieser Stelle schlussfolgern, dass der Integral-Anteil des implementierten PID-Reglers wegfallen muss, da dieser ansonsten zu instabilem Verhalten führen würde. Für die passende Parametrierung des Proportional- und Differential-Anteils sind hingegen Tests notwendig.

Zu diesem Zweck wurde in der Flugsteuerung des in Kapitel 6.1 beschriebene Test-Quadropters der sog. Angle-Mode (automatische Stabilisierung innerhalb der Ebene) aktiviert. Als Landeplatzmarker diente ein auf ebenem Untergrund befestigter schwarzer Kreis im DIN A4-Format auf weißem Papier. Anschließend wurden die Ausgabewerte der Mobilapplikation auf feste Startwerte gesetzt, indem der Versuchsaufbau (zunächst nicht im Betrieb) mittig über dem Ziel positioniert, die Kamera abgedeckt und der Kreis entfernt wurde.

Der konkrete Testablauf gestaltete sich derart, dass das UAV manuell in einer erfahrungsgemäß geeigneten Höhe für eine zuverlässige Erfassung des geometrischen Markers gehalten wurde. Sobald das Ziel im Erfassungsbereich des Quadropters erschien, wurde der autonome Betrieb über den Mischer mittels eines per Potentiometer auf Seiten der Fernsteuerung einstellbaren Faktors hinzugeschaltet. Somit war es möglich, weiterhin in das Flugverhalten einzugreifen, um beispielsweise auf unvorhergesehene Situationen oder unerwünschtes Reglerverhalten zu reagieren.

Der gesamte Testprozess gestaltete sich iterativ, d. h. nach jedem Test wurden die Ergebnisse evaluiert, die Regel-Parameter angepasst und anschließend erneut getestet.

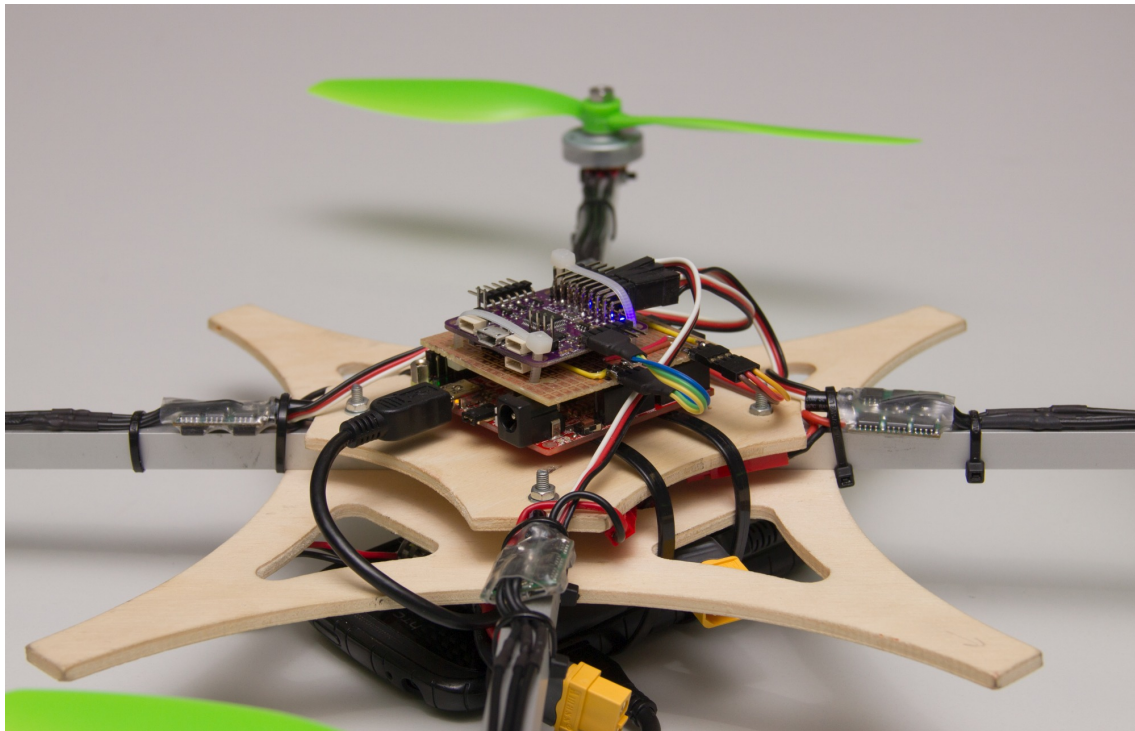


Abbildung 6.4: Nahansicht des Versuchsaufbaus im Profil

Wichtig: Für möglichst präzise Testergebnisse sollte von Anfang an darauf geachtet werden, möglichst alle störenden Umwelteinflüsse während den Tests zu eliminieren. So sollte z. B. als Testumgebung ein möglichst großer geschlossener Raum gewählt werden, um den Einfluss der durch das UAV entstehenden Luftverwirblungen zu reduzieren. Ebenfalls sollten Fenster nicht geöffnet werden, um Störungen durch Windböen vorzubeugen. Weiterhin ist bei Flugbetrieb des UAVs auf die Einhaltung von Sicherheitsmaßnahmen (vgl. beispielsweise [Qua]) zu achten, um Verletzungen und Sachbeschädigungen zu vermeiden. Gerade zu diesem Zweck empfiehlt sich vor Allem für ungeübte Piloten auch die Verwendung des Angle-Modus.

6.4 Ergebnisse

Während des iterativen Testverfahrens ließen sich diverse Faktoren feststellen, welche sich kompromittierend auf die Ergebnisse auswirkten:

- Eine statische Befestigung des Mobilgeräts unter dem UAV resultiert in einer Verschiebung des Erfassungsbereichs bei Roll- und Nickbewegungen. Dies hat zur Folge, dass der Kreis z.T. das Bild verlässt (vor Allem, wenn er sich im Randbereich befindet oder gerade von außen in den Erfassungsbereich kommt und der Ausschlag des Steuerimpulses als Reaktion darauf hoch ist).
Eine mögliche Lösung für dieses Problem bestünde in der Bestigung des Mobilgeräts mittels eines Gimbals (Schwenk-Neige-Vorrichtung). Alternativ könnte die aus dem Neigewinkel des UAVs resultierenden Verschiebung des Erfassungsbereichs einberechnet werden (hierzu wäre eine Erweiterung der bestehende Sensorik um eine absolute Höhenmessung (z. B. per Ultraschall) notwendig; ein Gyroskop bringen sowohl das Mobilgerät als auch der Flugcontroller im Normalfall mit sich).
- Zuweilen kommt es zu Problemen bzgl. der Stromversorgung der Komponenten, was zur Folge hat, dass der Flugcontroller nicht korrekt initialisiert werden kann und es

anschließend im Betrieb zu Fehlern bei der Kalibrierung der Sensorik oder dem Verarbeiten der Eingangsdaten des Mischers kommen kann (z. B. alternierender Wert für Throttle (Gas)). Dieses Fehlerbild begründet sich darin, dass das Mobilgerät bedingt durch den verwendeten USB-Treiber (vgl. Kapitel 5.5) als Client und nicht als Host der USB-Verbindung agiert und daher versucht, sich über diese aufzuladen.

Behoben werden kann dies durch das Einhalten der richtigen Reihenfolge des Zuschaltens der Stromversorgung für die einzelnen Elemente. Zunächst muss der Flugcontroller ohne angeschlossenes Mobilgerät initialisiert werden (per USB-Verbindung oder direkt über die mit dem Akku verbundenen BECs (Abk., engl. für „Battery Eliminator Circuit“) der ESCs (Abk., engl. für „Electronic Speed Control“, regelt die Geschwindigkeit der Motoren des UAVs)). Anschließend kann das Mobilgerät angeschlossen werden. Alternativ könnte diese Problematik behoben werden, indem die unterschiedlichen Funktionalitäten zusammengefasst würden oder der USB-Treiber derartig modifiziert würde, dass das Mobilgerät als Host der USB-Verbindung agiert.

- Teilweise interferieren die standardmäßigen Regler des Flugkontrollers mit den Steuersignalen des Reglers des Mobilgeräts. Gibt die Applikation beispielsweise einen starken Nick-Impuls aus, erzeugen die Regler des Controllers einen entgegengesetzten Impuls, um das UAV in Waage zu halten. Diesem Verhalten könnte Abhilfe geschaffen werden, indem die Regler (möglicherweise im Rahmen einer Zusammenfassung der einzelnen Plattformen) kombiniert werden.
- Schlussendlich ist es sehr kompliziert, den Quadrocopter manuell stabil genug in einer halbwegs stationären Höhe über dem Ziel zu positionieren, um die Effekte der Parametrierung des Reglers evaluieren zu können, solange dieser noch nicht korrekt eingestellt ist. Dieser Prozess erfordert sehr viel Geschick und Übung von Seiten des Piloten und beeinflusst die Güte der Ergebnisse sehr stark, da es gerade bei sanfteren Parametrierungen (geringe Werte für den Proportional-Anteil) schwierig abzuschätzen ist, welche Reaktionen des Fluggeräts dem Regler und welche manuellen Eingriffen des Piloten entstammen.

Unter Berücksichtigung der angeführten beeinflussenden Faktoren war es nicht möglich, eine stabile Regelung für die autonome Zentrierung des UAVs über einem gegebenen Landeplatz zu erreichen. Jedoch konnte mittels der erzielten Ergebnisse ein anschaulicher Machbarkeitsbeweis erbracht und die Funktionsfähigkeit des Gesamtsystems nachgewiesen werden. Für weitere Optimierung im Rahmen von Folgeprojekten können folgende Werte als Grundlage bzw. Orientierung für die Größenordnung des stabilen Bereichs der Regelparameter verwendet werden:

Proportional-Anteil:	0.15
Integral-Anteil:	0.00
Differential-Anteil:	0.02

Diese Parameter-Werte bewirken in Kombination mit dem verwendeten RC-Mischer und hundertprozentiger Beaufschlagung auf die Signale der Fernbedienung relativ geringe Steuereimpulse, die jedoch noch klar von manuellen Eingriffen durch den Piloten zu differenzieren sind.

7 VERGLEICH ZIELSETZUNG-ENDERGEBNIS

Resümierend lässt sich festhalten, dass die Zielsetzung der Erstellung eines frei zugänglichen und einfach zu verwendenden Frameworks zur Entwicklung von OpenCV-gestützten Bildverarbeitungsapplikationen für Android-Plattformen erfüllt wurde. In Kombination mit dem im Rahmen eines Parallelprojekts entwickelten echtzeitfähigen RC-Mischer auf Arduino-Basis ergibt sich ein modularer Aufbau sämtlicher Hardware- und Software-Komponenten. Dies unterstützt die geforderte einfache Verwendung und ermöglicht potentiellen Nutzern die Erweiterung um bzw. Verwendung von bereits bestehenden Komponenten. Darüber hinaus war es möglich, beide Aufgabenstellung auf Anwendungsebene (d. h. zur Erstellung der Applikationen benötigte Bibliotheken, SDKs und Programme, sowie Bootloader (vgl. Kapitel 10.2)) ausschließlich unter Verwendung von Open Source-Software umzusetzen, wodurch die freie Zugänglichkeit zu allen Elementen des Projekts bestmöglichst gewährleistet wird.

Ebenfalls konnte ein Machbarkeitsbeweis hinsichtlich der Funktionalität der einzelnen Komponenten erbracht und potentieller praktischer Anwendungsmöglichkeiten aufgezeigt werden. Im Rahmen des iterativen Testverfahrens zur Parametrierung des Reglers offenbarten sich jedoch die Grenzen des verwendeten Versuchsaufbaus (starre Kamerabefestigung, Regler-Interferenzen, etc.), wodurch sich der Vorgang komplizierter als erwartet gestaltete. In Anbetracht der in Kapitel 6.4 angeführten Herausforderungen besteht daher weiterhin Optimierungspotential hinsichtlich der Realisierung einer stabilen Regelung zur autonome Zentrierung über einem gegebenen Landeplatz.

Zusammenfassend lässt sich sagen, dass die definierten Zielstellungen in weitesten Teilen erfüllt werden konnten. Trotzdem besteht teilweise weiterhin Optimierungspotential (vgl. Kapitel 8). Unter diesem Gesichtspunkt stellt das Projekt eine gute Grundlage für weiterführende oder eigenständige Projekte in Form der Bildverarbeitung mittels OpenCV auf Android-Plattformen, potentiell kombinierbar mit einer Datenausgabe zu angeschlossenen Peripheriegeräten, dar.

8 AUSBLICK

Hinsichtlich der Optimierung der autonomen Landeplatzlokalisierung sind im Verlauf der Arbeiten zusätzlich zu den bereits realisierten Funktionalitäten folgende weitere Aspekte aufgekomen:

1. Grundsätzlich besteht weiterhin Potential hinsichtlich der Parametrierung des Reglers auf Seite der Applikation, um eine stabile Zentrierung über dem Landeplatz mit möglichst geringen Überschwingern zu realisieren. Sollten mechanische Änderungen an dem Versuchsaufbau vorgenommen werden, so ist eine (komplette) Rekalibrierung der Parameter empfehlenswert um ein gutes Ergebnis zu erzielen.
Durch geeignete Modifikationen kann der Anspruch an den Regler weiterhin reduziert werden (beispielsweise durch Verwendung eines Gimbals (s. Punkt drei) oder der Optimierung in den Randbereichen des Erfassungsgebiets (s. Punkt zwei)).
2. Wie bereits in Kapitel 6.4 angeführt, kann zur Optimierung der Erfassung des Landeplatzes im Randbereich des Bilds die bestehende Sensorik um eine absolute Höhenmessung (beispielsweise mittels Ultraschall) ergänzt werden.
Somit könnte in Kombination mit dem bereits auf Seiten der meisten Mobilgeräte und Flugcontroller existierenden Gyroskop aus dem gemessenen Neigungswinkel gegenüber der Ebene die Verschiebung des Erfassungsbereichs bei Neigung des UAVs ermittelt werden. Durch die Verrechnung des so erhaltenen Wertes mit dem von der Bildverarbeitung ausgegebenen Wert kann die Funktionsfähigkeit des Reglers in diesen Randbereichen verbessert und die Störgröße der Neigung weitestmöglich eliminiert werden.
3. Die mechanische Konstruktion des Testgeräts (dargestellt in Kapitel 6.1) ist derzeit noch insofern suboptimal, dass das kapazitive Display des Mobilgeräts zuweilen auf den darüber liegenden Akku reagiert und so beispielsweise die Applikation beendet wird. Weiterhin ist das Display an sich nur sehr begrenzt zugänglich, wodurch es erschwert wird, Statusmeldungen zu quittieren. Die Lösung der Befestigung des Geräts am Versuchsaufbau mittels Kabelbindern ist ebenfalls sicherheitstechnisch suboptimal.
Insofern wäre eine mechanische Umkonstruktion mit dem Fokus der Neupositionierung des Akkus und des Mobilgeräts, so dass der Bildschirm frei zugänglich ist, denkbar. Dies könnte potentiell in Kombination mit der Verwendung eines Gimbals zur Bildstabilisierung geschehen.
4. Derzeit existiert keine Möglichkeit, um den Test-Quadrokopter bei einem potentiellen Versagen des Flugcontrollers oder des RC-Mischers abzuschalten, außer am laufenden System die Stromversorgung zu unterbrechen. Da dies jedoch mit Sicherheits- und Verletzungsrisiken einhergeht, wäre es denkbar, einen drahtlosen Notaus als zusätzliche Sicherheitsinstanz zu implementieren, mittels dem das System „remote“ (aus der Ferne) abgeschaltet werden kann. Wichtig wäre bzgl. der Umsetzung dieser Funktionalität insbesondere der Fokus auf die Echtzeitfähigkeit und die Stabilität.
5. Um eine Interferenz der standardmäßigen Regler des Flugcontrollers und der vorgestellten Applikation zu vermeiden, könnten diese zusammengefasst werden, so dass bspw. die Steuersignale des Mobilgeräts diejenigen des Controllers während des autonomen Betriebs zu einem gewissen Grad überlagern. Somit könnte diese Störgröße eliminiert und eine einfachere Parametrierung des Reglers auf Applikations-Seite ermöglicht werden. Potentiell wäre es möglich, dies mit einer Fusion der verwendeten Hardware- und Software-Komponenten (s. Punkt sechs) zu verbinden.
6. Wie bereits zu Beginn in der Zielsetzung (vgl. Kapitel 1.2) dargestellt, lag der Fokus dieses Projektes darauf, möglichst einfach verwendbare und frei zugängliche Software zu entwickeln. Dies inkludiert bzw. induziert zwangsläufig einen modularen Aufbau des Systems, so dass potentielle Nutzer möglichst einfach auf ihren bestehenden Komponenten aufbauen können.

Zu Optimierungszwecken wäre es jedoch zielführender, die einzelnen Software- und Hardware-Module zusammenzufassen und auf einer einzigen Platine zu bündeln. Somit könnte das Gewicht und der Stromverbrauch deutlich reduziert und die Kommunikation zwischen den einzelnen Komponenten sowie die mechanische Anbringung am UAV stark vereinfacht werden. Ebenfalls könnten teilweise aufgrund des modularen Aufbaus redundante Code-Elemente eliminiert werden. Anbieten würde sich für diese Zwecke beispielsweise ein Einplatinencomputer wie der BeagleBone Black, wobei ein Kern rein für die Flugsteuerung und ein weiterer Kern für die restliche Software verwendet werden könnten, um die Echtzeitfähigkeit zu erhalten.

Die angeführten Aspekte sollen als Anregung für eigenständige oder auf diesem Projekt aufbauende Folgeprojekte dienen und sind nicht als zwangsläufige Voraussetzung zu sehen, um das erstellte Framework oder die Demoapplikation in ihrer bestehenden Funktion nutzen zu können. Vielmehr dienen sie wie zu Beginn der Sektion angeführt der Optimierung.

9 LITERATUR

- [Ale16] Alexander Alekhin. OpenCV Change Logs. Juni 2016. URL: <https://github.com/opencv/opencv/wiki/ChangeLog> (besucht am 05. 11. 2016).
- [All07] Open Handset Alliance. Industry Leaders Announce Open Platform for Mobile Devices. Nov. 2007. URL: http://www.openhandsetalliance.com/press_110507.html (besucht am 03. 11. 2016).
- [Cal15] John Callahan. Google says there are now 1.4 billion active Android devices worldwide. Sep. 2015. URL: <http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide> (besucht am 03. 11. 2016).
- [Coc+15] F. Cocchioni u. a. „Visual Based Landing for an Unmanned Quadrotor“. In: Journal of Intelligent & Robotic Systems (Sep. 2015).
- [Dav16] Doug Davis. Intel Acquires Computer Vision for IOT, Automotive. Mai 2016. URL: <https://newsroom.intel.com/editorials/intel-acquires-computer-vision-for-iot-automotive/> (besucht am 05. 11. 2016).
- [Fre16] Inc. Free Software Foundation. Various Licenses and Comments about Them. Nov. 2016. URL: <https://www.gnu.org/licenses/license-list.en.html> (besucht am 05. 11. 2016).
- [Gar] Willow Garage. OpenCV. URL: <http://www.willowgarage.com/pages/software/opencv> (besucht am 05. 11. 2016).
- [Goo] Google. Android, the world's most popular mobile platform. URL: <https://developer.android.com/about/android.html> (besucht am 03. 11. 2016).
- [Goo15] Google. ADT Plugin Release Notes. Aug. 2015. URL: <https://developer.android.com/studio/tools/sdk/eclipse-adt.html> (besucht am 30. 10. 2016).
- [Goo16a] Google. Activity. 2016. URL: <https://developer.android.com/reference/android/app/Activity.html> (besucht am 03. 11. 2016).
- [Goo16b] Google. Android Studio Release Notes. Okt. 2016. URL: <https://developer.android.com/studio/releases/index.html> (besucht am 03. 11. 2016).
- [Goo16c] Google. App Manifest. 2016. URL: <https://developer.android.com/guide/topics/manifest/manifest-intro.html> (besucht am 17. 11. 2016).
- [Goo16d] Google. Layouts. 2016. URL: <https://developer.android.com/guide/topics/ui/declaring-layout.html> (besucht am 20. 11. 2016).
- [its] Inc. itseez. About. URL: <http://opencv.org/about.html> (besucht am 05. 11. 2016).
- [Lia99] Sheng Liang. „The Java Native Interface: Programmer's Guide and Specification“. In: 1.Edition. Reading, Massachusetts: Addison Wesley Longman Inc., 1999. Kap. Role of the JNI.
- [Man15] Farhad Manjoo. „A Murky Road Ahead for Android, Despite Market Dominance“. In: New York Times (Mai 2015).
- [Mar07] John Markoff. „I Robot The Man Behind the Google Phone“. In: New York Times (Nov. 2007).
- [Qua] Quadrocopter. Safety. URL: http://www.quadrocopter.com/Safety_ep_59-1.html (besucht am 06. 02. 2017).
- [Sch16] Stefan Schwark. „Messen und Visualisieren mit Android - Apps für Elektroniker“. In: 1.Edition. Amersfoort, Netherlands: WILCO, 2016. Kap. Android und USB.
- [Web] Jan Weber. In: ().

10 ANHANG

10.1 Persönliches Fazit

Was haben wir als Entwickler nun persönlich aus dem Projekt mitgenommen? Neben den rein inhaltlichen Aspekten war es uns möglich, unser Wissen im Bereich der Programmierung mit Java, C und C++ zu vertiefen. Ebenfalls lernten wir den Wert qualitativ hochwertiger Dokumentation (was vor Allem im Bereich der Open Source-Software keine Selbstverständlichkeit ist) zu schätzen und einen erhöhten Fokus auf die Lizenzierung der verwendeten Komponenten und Elemente zu legen. Weiterhin haben wir gelernt, dass eine stabile Stromversorgung maßgeblich für die Funktionalität der angeschlossenen Komponenten ist und Instabilitäten zu schwierig nachvollziehbaren Fehlerbildern führen können.

Im Hinblick auf die vermittelten Soft Skills konnten, bedingt durch die Durchführung des Projekts als Gruppenarbeit, die eigenen Kompetenzen in den Bereichen Planung, Teamarbeit und Kommunikation verbessert werden. Vor Allem letztere beiden Aspekten wurden durch die kooperative Arbeit mit Versionskontrollsystemen wie dem über den campuseigenen Server zur Verfügung stehenden Gitlab unterstützt bzw. gefördert.

Der jedoch wahrscheinlich entscheidendste Aspekt, den wir im Verlauf dieser Projektarbeit lernen konnten, ist, wie man im Allgemeinen planungstechnisch an größere Projekte hertritt, diese sinnvoll aufteilt („Top-Down-Ansatz“ der Softwareentwicklung) und anschließend mit einem agilen Entwicklungsansatz bearbeitet.

10.2 Lizenzen

Der im Rahmen dieses Projektes geschaffene Programmcode steht ebenso wie diese Dokumentation unter der Modifizierten BSD Lizenz (<https://gitlab.cvh-server.de/lf.ps/vbbs/blob/master/common/BSD-MODIFIED.txt>).

Eine Weiterverbreitung und Weiterverwendung ist ausdrücklich erwünscht. Die Software ist in der Hoffnung entstanden, nützlich zu sein und wird „wie sie ist“ zur Verfügung gestellt. Es kann keine Gewährleistung auf Fehlerfreiheit oder die Eignung für einen bestimmten Zweck übernommen werden. Darüber hinaus kann keinerlei Haftung für direkt oder indirekt aus der Nutzung dieser Software hervorgegangene Personen- oder Sachschäden übernommen werden.

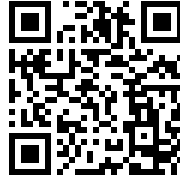
Jeder ist für sein eigenes Handeln verantwortlich. Wir raten zu einem sicheren und besonnenen Umgang mit Fluggerät jedweder Art.

Auflistung verwendeter Software und deren Lizenzen (ohne Anspruch auf Vollständigkeit):

- Atom
MIT License
<https://github.com/atom/atom/blob/master/LICENSE.md>
- PlatformIO
Apache License Version 2.0
<https://github.com/platformio/platformio-docs/blob/develop/LICENSE>
- Eclipse
Eclipse Public License - v1.0
<http://www.eclipse.org/legal/epl-v10.html>
- Android SDK und NDK
Apache License Version 2.0
<https://developer.android.com/license.html>
- Arduino
GNU General Public License Version 2
<https://github.com/arduino/Arduino/blob/master/license.txt>
- avr-gcc
GNU General Public License Version 2
<https://gcc.gnu.org/wiki/avr-gcc>
- AVRDUDE
GNU General Public License Version 2
<http://savannah.nongnu.org/projects/avrdude>
- L^AT_EX
LaTeX project public license
<https://www.latex-project.org/lppl.txt>
- GIMP
GNU General Public License Version 2
<https://www.gimp.org/about/COPYING>
- Adobe Photoshop Lightroom
Proprietär & Kommerziell
<http://labs.adobe.com/technologies/eula/lightroom.html>

10.3 gitlab-Repository

Der im Rahmen dieses Projektes entstandene Quellcode sowie die hier vorliegende Dokumentation können über den gitlab-Server des Campus Velbert-Heiligenhaus der Hochschule Bochum bezogen werden.



<https://gitlab.cvh-server.de/lf.ps/vbls>

10.4 Eidesstattliche Erklärung

Die Autoren versichern durch ihre Unterschriften, dass diese Arbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel verwendet wurden. Alle Zitate und Übernahmen sind im Text der Hausarbeit kenntlich gemacht.

Lukas Friedrichsen
Solingen, den 19. März 2017

Philipp Stenkamp
Langenfeld, den 19. März 2017