**Projektarbeit Robotik**

Titel der Arbeit // Title of Thesis

**Entwicklung einer Robotersteuerung zur bildverarbeitungsgestützten Objekterkennung auf Basis neuronaler Netze**

Akademischer Abschlussgrad: Grad, Fachrichtung (Abkürzung) // Degree

Bachelor of Engineering (B.Eng)

Autorenname, Geburtsort // Name, Place of Birth

Tobias Uhle, Datteln

Studiengang // Course of Study

Maschinenbau

Fachbereich // Department

Maschinenbau und Maschinenbau, Umwelt und Gebäudetechnik

Erstprüferin/Erstprüfer // First Examiner

Prof. Dr.-Ing. Jürgen Dunker

Zweitprüferin/Zweitprüfer // Second Examiner

Prof. Dr.-Ing. Dirk Fröhling

Abgabedatum // Date of Submission

Klicken Sie hier, um Text einzugeben.

**Eidesstattliche Versicherung**

Uhle, Tobias

Name, Vorname // Name, First Name

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel

„Entwicklung einer Robotersteuerung zur bildverarbeitungsgestützten Objekterkennung auf Basis neuronaler Netze“

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Gelsenkirchen,

Ort, Datum, Unterschrift // Place, Date, Signature

**Inhaltsverzeichnis**

[1. Einleitung / Aufgabenstellung I](#_Toc177419984)

[2. Grundlagen I](#_Toc177419985)

[2.1 Programmiergrundlagen I](#_Toc177419986)

[2.1.1 Programmiersprache Python I](#_Toc177419987)

[2.1.2 Grafische Darstellung: PyGame II](#_Toc177419988)

[2.2 Drohnentechnik II](#_Toc177419989)

[2.3 Bildverarbeitung II](#_Toc177419990)

[3. Systemkomponenten V](#_Toc177419991)

[3.1 Hardware V](#_Toc177419992)

[3.1.1 Drohne V](#_Toc177419993)

[3.2 Software VI](#_Toc177419994)

[3.2.1 PiCharm VI](#_Toc177419995)

[3.2.2 djitellopy VI](#_Toc177419996)

[3.2.3 OpenCV VI](#_Toc177419997)

[3.2.3.1 Grundlagen VI](#_Toc177419998)

[3.2.4.3 Wichtige Bildoperationen VIII](#_Toc177419999)

[4. Implementation / Ergebnisse IX](#_Toc177420000)

[4.1 Verbindungsaufbau IX](#_Toc177420001)

[4.2 Erstellte Klassenbibliotheken X](#_Toc177420002)

[4.2.1 Keyboardcontrol X](#_Toc177420003)

[4.2.2 XBoxControl X](#_Toc177420004)

[4.2.3 dronecomms X](#_Toc177420005)

[4.2.4 ImageProcessing X](#_Toc177420006)

[4.3 Benutzeroberfläche XIV](#_Toc177420007)

[4.3.1 Allgemeiner Aufbau XIV](#_Toc177420008)

[4.4 Manuelle Drohnensteuerung XIV](#_Toc177420009)

[4.4.1 Per Tastatur XIV](#_Toc177420010)

[4.4.2 Per Controller XIV](#_Toc177420011)

[4.5 Automatischer Betrieb XIV](#_Toc177420012)

[4.5.1 Objektverfolgung XIV](#_Toc177420013)

[4.5.2 XY XV](#_Toc177420014)

[4.5.2.4 Schwierigkeiten XV](#_Toc177420015)

[5. Zusammenfassung und Ausblick XVI](#_Toc177420016)

[Anhang XVIII](#_Toc177420017)

# 1. Einleitung / Aufgabenstellung

# 2. Grundlagen

## 2.1 Programmiergrundlagen

### 2.1.1 Programmiersprache Python

Die für dieses Projekt wird die Scriptsprache Python in der Version 3.8 verwendet.

Die Sprache wurde im Februar 1991 erstmals veröffentlicht und seitdem bis in die Gegenwart weiterentwickelt. Sie kann auf unterschiedlichsten Betriebssystemen ausgeführt werden und bietet entsprechend eine sehr große Kompatibilität und Verwendbarkeit von in Python geschriebenen Skripten. Zur Ausführung eines Skriptes ist stets ein bereits installierter Interpreter notwendig. Die Skripte werden also nicht kompiliert.

Python kann dabei sowohl objektorientiert als auch prozedural verwendet werden.

Es zeichnet sich vor allem durch mittels Einrückungen definierten Syntax aus. Dabei werden abhängige Befehlsanweisungen, beispielsweise ein „if-else-Statement“ nicht – wie bei den meisten anderen Programmiersprachen üblich – mittels einer Klammer begonnen und beendet.

Klar wird dies an einem typischen „Hello-World“-Beispiel. Hierbei wurde zunächst die Funktion *main* deklariert und danach aufgerufen, um eine dadurch nötige Einrückung darzustellen.

def main():

print(„Hello World“)

main()

def main():

Deklariert die Methode *main* . In der Klammer können Argumente angegeben werden. Der Doppelpunkt zeigt an, dass daraufhin ein entsprechend eine Ebene weiter eingerückter Anweisungsblock folgt. Dieser wird beendet, sobald diese Einrückung endet.

print("Hello World")

Gibt den als Argument mitgegebenen Text in der Interpreter-Konsole aus.

### 2.1.2 Grafische Darstellung: PyGame

TEXT VON DOME

## 2.2 Drohnentechnik

## 2.3 Bildverarbeitung

Die Bildverarbeitung dient dazu, optisch aufgenommene oder digital generierte Bilder zu analysieren und daraus verschiedene Arten von Informationen zu gewinnen. Sie ist zentrale Komponente der in dieser Bachelorarbeit entwickelten Steuerung des Roboters.

Während in Kapitel 3.2.5 näher auf die Möglichkeiten der in diesem Kontext verwendeten Bildverarbeitungs-Software eingegangen wird, soll an dieser Stelle die grundlegende Idee der Bildverarbeitung und der Weg vom optischen Bild zur digitalen Matrix dargestellt werden.

Moderne digitale Kameras besitzen üblicherweise einen CCD[[1]](#footnote-1)-Sensor zur Aufnahme von Fotografien. Diese enthalten rasterförmig angeordnete, auf Licht reagierende Fotodioden, welche je eine Größe zwischen 1,4 µm und 20 µm haben. (Vgl. [[HAN22](#HAN22)])

Der Sensor erzeugt anhand der an den Fotodioden anliegenden induzierten[[2]](#footnote-2) Spannungen für jede Fotodiode, die sogenannten Pixel, eine elektrische Information über die Lichtintensität an dieser Stelle. Da Fotodioden lediglich die Intensität des Lichts, nicht aber die Farbe messen können, werden sie bei Farbkameras in einem bestimmten Muster[[3]](#footnote-3) mit roten, grünen und blauen Farbfiltern überzogen. Der Bildprozessor innerhalb der Kamera errechnet dann mit Hilfe des Wissens, welches Muster verwendet wurde, das eigentliche farbige Bild.

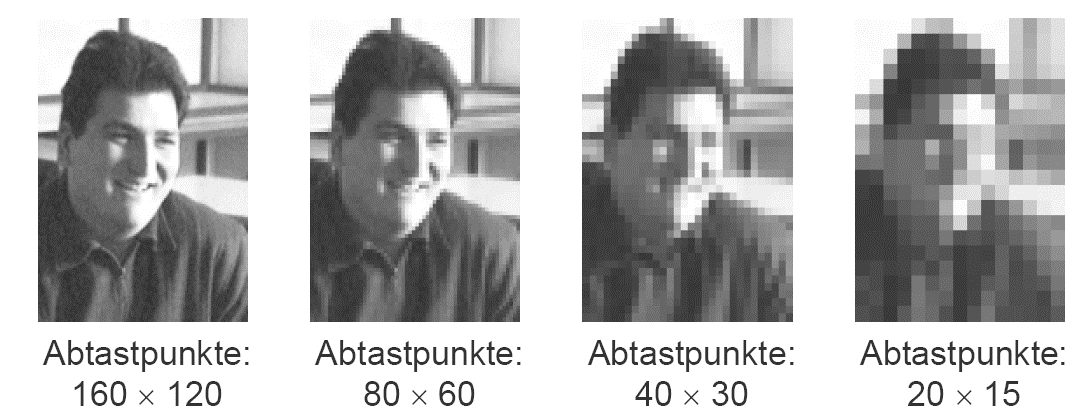
Das vom Prozessor generierte Bild besitzt eine native[[4]](#footnote-4) maximale Auflösung, die von der Anzahl der Fotodioden auf dem Sensor bestimmt wird. Die Pixel sind dabei in einem rechteckigen Raster angeordnet, welches so die Bildhöhe- und breite definiert. Für jedes Pixel wird der finale Farbwert in der Regel im RGB-Farbraum bestimmt. RGB steht dabei für die drei Grundfarben **R**ot, **G**rün und **B**lau im sogenannten „additiven Farbraum“[[5]](#footnote-5).

Der Wert jeder Farbkomponente umfasst üblicherweise eine Farbtiefe von 8 Bit, bietet also 28 = 256 Abstufungen (Vgl. [TIM11, S. 20]) und gibt die Helligkeit selbiger an. Die Kombination der drei Komponenten bestimmt das Erscheinungsbild des jeweiligen Pixels. Ein RGB-Wert von (255, 255, 255) ergibt beispielsweise weiß, (255, 255, 0) resultiert in gelber Farbe.

Mit Hilfe der Bildverarbeitung können diese digitalen Bilder nun beliebig genutzt und manipuliert werden.

Ein grundlegendes Beispiel ist die Veränderung der Bildgröße, bei der mittels verschiedener Methoden die eingehende Pixel-Matrix auf die vorgegebenen Zielmaße umgerechnet wird. Neben der Verkleinerung und Vergrößerung (Abbildung 4) ist auch die Änderung des Seitenverhältnisses möglich.

Weiterführend bietet die Bildverarbeitung Methoden zur Änderung des Farbraumes, etwa um ein Farbbild in ein Monochrombild zu wandeln, sowie einen großen Umfang an sogenannten Filtern, die sich verschiedenste Gesichtspunkte eines digitalen Bildes zunutze machen und dieses somit faktisch unbegrenzt verändern können.



**Abbildung 4 - Darstellung unterschiedlicher Bildauflösungen (Quelle: [KON21, S. 4])**

# 3. Systemkomponenten

## 3.1 Hardware

### 3.1.1 Drohne

Als Flugobjekt findet die Drohne „Tello“ des Herstellers DJI Verwendung. Hierbei handelt es sich um ein Modell, das hauptsächlich in der Lehre genutzt und verglichen mit professionellen Drohnen entsprechend günstig angeboten wird.

Wie zumeist üblich besitzt die Drohne vier starre Rotoren, welche mittels flexiblen Armen mit der Hauptplatine verbunden sind. Als Zubehör lassen sich Schutzgitter anbringen, die sowohl den Anwender als auch die Drohne vor Schädigungen schützen.

Die Drohne wiegt etwa 80 Gramm, erreicht eine maximale Geschwindigkeit von 28,8 km/h, besitzt eine Reichweite von 100 Metern und kann bis zu 13 Minuten fliegen.

Der Akku ist austauschbar, um einen mehrfachen Einsatz innerhalb kurzer Zeit zu ermöglichen.

In der Front der Drohne ist eine Kamera verbaut, die mittels eines fünf Megapixel messenden Sensors ein Live-Video von maximal 720p Auflösung und 30 Bildern pro Sekunde liefert.

Die Drohne kann entweder mittels einer App des Herstellers über das Handy oder über eine entsprechende Programmbibliothek vom PC aus gesteuert werden. Dafür wird sie jeweils mittels eines eigens erzeugten WLAN-Netzwerks verbunden.

## 3.2 Software

### 3.2.1 PiCharm

PiCharm ist eine Entwicklungsumgebung des Entwicklers JetBrains, die speziell für die Softwareentwicklung mittels Python gedacht ist. Sie bietet Funktionalitäten wie Projekt- und Versionsverwaltung sowie die Verwaltung von in Python installierten Bibliotheken.

In PiCharm können Projekte direkt ausgeführt werden, sodass sich erzeugte Skripte schnell und einfach testen lassen.

### 3.2.2 djitellopy

Die Klassenbibliothek djitellopy für Python wird von Damià Fuentes Escoté entwickelt und enthält Methoden, um mittels eines Python-Skriptes mit der Tello zu kommunizieren. Sie implementiert alle verfügbaren Tello-Befehle und bietet entsprechend sämtliche Funktionalitäten, die zur Steuerung und Überwachung der Drohne benötigt werden.

Des Weiteren überwacht die Bibliothek die Verbindung zur Drohne, reagiert auf Programmabstürze, indem sie die Drohne sicher landet und eignet sich sogar dazu, mehrere Drohnen auf einmal zu steuern.

Um die Funktionen nutzen zu können, muss zunächst die Bibliothek mittels

from djitellopy import tello

importiert werden. Danach ist es zwingend notwendig, ein Objekt der Klasse *tello* zu instanzieren.

mytello = tello.tello()

Wichtige im Kontext dieser Projektarbeit genutzte Funktionen sind:

* mytello.connect()
* mytello.takeoff()
* mytello.land()
* mytello.getBattery()
* mytello.streamon()
* mytello.streamoff()
* mytello.getImage()
* mytello.send\_rc\_control()
* mytello.move\_up()
* mytello.move\_down()
* mytello.rotate\_clockwise()
* mytello.rotate\_counter\_clockwise()
* mytello.move\_forward()
* mytello.move\_back()
* mytello.move\_right()
* mytello.move\_left()

### 3.2.3 OpenCV

#### 3.2.3.1 Grundlagen

Das Softwarepaket OpenCV ist eine quelloffene Klassenbibliothek für Python und bietet zahlreiche Algorithmen, um Bilder zur Programmlaufzeit zu manipulieren. Es wurde erstmals 2002 veröffentlicht und seitdem stetig weiterentwickelt.

Beispielsweise lassen sich mittels OpenCV Farbwerte einzelner Pixel auslesen, Bilder wie bei der klassischen Bildbearbeitung editieren (weichzeichnen oder schärfen), umfärben und Farbräume konvertieren. Zudem enthält die Bibliothek Methoden, um etwa Kreise, Quadrate oder Text in Bilder einzufügen.

#### 3.2.4.3 Wichtige Bildoperationen

Im Folgenden werden die in diesem Projekt genutzten Befehle aus OpenCV aufgelistet.

|  |  |
| --- | --- |
| **cv2.GaussianBlur** | |
| Übergabeparameter: 3 | src, ksize, sigmaX |
| Rückgabeparameter: 1 | dest |
| Belegt ein Bild mit einem Gauß‘schen Weichzeichnungsfilter. Dabei wird jedes Pixel basierend auf seiner Umgebung der Fläche *size* mit Hilfe der Gauß-Verteilung neu berechnet. Das Bild wird geglättet und mögliche Störpixel[[6]](#footnote-6) oder Bildrauschen[[7]](#footnote-7) reduziert. | |
| dest = cv2.GaussianBlur(img, (5, 5), 5) | |

|  |  |
| --- | --- |
| **cv2.CvtColor** | |
| Übergabeparameter: 2 | src, converttype |
| Rückgabeparameter: 1 | dest |
| Konvertiert ein Bild in einen anderen Farbraum. Essenziell sind hierbei vor allem der monochrome und HSV-Farbraum. Letzterer stellt die Pixelwerte als „**H**ue, **S**aturation, **V**alue“ dar, also Farbton, Sättigung und Helligkeit. Der *code* bestimmt die Art der Konvertierung und folgt dem Schema „Vorher-zu-Nachher“. | |
| dest = cv2.cvtColor(img, cv2.COLOR\_RGB2HSV) | |

|  |  |
| --- | --- |
| **cv2.rectangle** | |
| Übergabeparameter: 5 | src, start, end, color, thickness |
| Rückgabeparameter: 1 | dest |
| Zeichnet ein Rechteck zwischen den Punkten *start* und *end* in das angegebene Bild. Die Liniendicke wird mittels *thickness* angegeben, die Farbe mittels *color*. | |
| dest = cv2.rectangle(img, (10, 10), (50, 20), (255,0,0), 2) | |

|  |  |
| --- | --- |
| **cv2.circle** | |
| Übergabeparameter: 5 | src, center, radius, color, thickness |
| Rückgabeparameter: 1 | dest |
| Zeichnet einen Kreis mit dem Mittelpunkt an den Koordinaten *center* und dem Radius *radius* in das angegebene Bild. Die Liniendicke wird mittels *thickness* angegeben, die Farbe mittels *color*. | |
| dest = cv2.circle(img, (20, 50), 10, (255,0,0), 2) | |

|  |  |
| --- | --- |
| **cv2.fitEllipse** | |
| Übergabeparameter: 1 | contour |
| Rückgabeparameter: 1 | rect |
| Zeichnet eine Ellipse um einen Datensatz von 2D-Positionsdaten, wie beispielsweise eine Kontur. Wichtigster Parameter der Ellipse ist in diesem Kontext ihr Winkel, der Aufschluss über die Ausrichtung der längsten Ausdehnung einer Kontur gibt. | |
| rect = cv2.fitEllipse(contour) | |

|  |  |
| --- | --- |
| **cv2.Ellipse** | |
| Übergabeparameter: 1 | src, rct, color, thickness |
| Rückgabeparameter: 1 | dest |
| Zeichnet eine Ellipse mit angegebenen Maßen in ein Bild. Die Farbe *color* und Liniendicke *thickness* können bestimmt werden. | |
| dest = cv2.ellipse(img,rct,(0,0,255),3) | |

|  |  |
| --- | --- |
| **cv2.putText** | |
| Übergabeparameter: 8 | src, text, org, fontFace, fontScale, color, thickness, lineType |
| Rückgabeparameter: 1 | dest |
| Fügt den angegebenen *text* an der Stelle *org* in das Bild ein. Die Schriftart wird dabei mittels *fontFace* festgelegt, die Schriftgröße über *fontScale*. Farbe (*color*), Liniendicke (*thickness*) und Linientyp (*lineType*) lassen sich ebenfalls definieren. | |
| cv2.putText(img, "Hallo", (50,50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2, cv2.LINE\_AA) | |

|  |  |
| --- | --- |
| **cv2.inRange** | |
| Übergabeparameter: 3 | src, lower, upper |
| Rückgabeparameter: 1 | mask |
| Findet Pixel, deren Werte zwischen *lower* und *upper* liegen und markiert diese nach demselben Schema binär.  Die Tupel für die untere und obere Grenze bedienen dabei je nach Aufbau des eingegebenen Bildes (RGB, BGR, HSV) die im jeweiligen Farbraum gültigen Parameter. | |
| mask = cv2.inRange(img, (20,20,40), (55,255,230)) | |

|  |  |
| --- | --- |
| **cv2.findContours** | |
| Übergabeparameter: 3 | mask, mode, method |
| Rückgabeparameter: 2 | contours, hierarchy |
| Detektiert sämtliche Konturen auf einem Bild. Diese Methode funktioniert am besten mit einem per *inRange* aufgeteilten Bild, in welchem die Konturen durch den Farbwechsel klar abgegrenzt sind. | |
| contours, hierarchy = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE) | |

# 4. Implementation / Ergebnisse

## 4.1 Verbindungsaufbau

## 4.2 Erstellte Python-Skripte

### 4.2.1 main

### 4.2.2 Keyboardcontrol

Um die Drohne zunächst per Tastatur zu steuern und damit eine immer verfügbare Möglichkeit zur direkten Kontrolle zu haben, wurde das Skript „keyboardControl“ entworfen. Dieses verwendet die Bibliothek PyGame, um Tastatureingaben zu erkennen und eine der in Kapitel „4.4.1 Steuerungschema“ festgelegten Richtline entsprechende Rückgabe an die Hauptroutine zu liefern.

### 4.2.3 XBoxControl

### 4.2.4 dronecomms

### 4.2.5 ImageProcessing

Um die Handhabung mit dem neuen Roboter auch für weiterführende Projekte einfach und adaptiv zu gestalten, bot es sich an, zunächst eine Klassenbibliothek zu entwickeln, welche grundlegende Funktionen des COBOTTA bereitstellt.

Das COBOTTA\_Lib genannte Code-Projekt nutzt die in Kapitel 3.2.2 vorgestellte Bibliothek ORiN2 und vereinfacht die von ihr bereitgestellten Befehle. Besonderes Augenmerk liegt hierbei darauf, die in ORiN2 gegebene kombinierte Schreibweise aus C# und PacScript in eine einheitliche Methodenliste mit entsprechenden Übergabeparametern umzuwandeln und somit zu vereinfachen. Der Nutzer muss dadurch nicht die Syntax von PacScript kennen, sondern kann die COBOTTA\_Lib wie viele andere Klassenbibliotheken als einfach aufgebaute Methodengruppe verwenden.

Für das Herstellen der Verbindung steht die Methode *COBOTTA\_Lib::connect* zur Verfügung, welche ähnlich dem Codebeispiel in Kapitel 3.2.2.1 die benötigten Objekte für Engine, Workspace und Controller erzeugt, optional die Kamera aktiviert und eine sogenannte „Motion preparation“ durchführt, die den Roboter nach dem Einschalten in einen betriebsbereiten Zustand versetzt.

Viele Funktionen sind dabei einfache Verbindungsglieder zu deren in Kapitel 3.2.2.2ff aufgelisteten Pendanten innerhalb von ORiN2 und bedürfen keiner weiteren Erläuterung. Dazu zählen folgende Methoden:

* COBOTTA\_Lib::TakeArm
* COBOTTA\_Lib::GiveArm
* COBOTTA\_Lib::MotorOn
* COBOTTA\_Lib::MotorOff
* COBOTTA\_Lib::curPos
* COBOTTA\_Lib::curJnt
* COBOTTA\_Lib::getCurrentErrorCount
* COBOTTA\_Lib::getErrorInfo
* COBOTTA\_Lib::ClearErrors
* COBOTTA\_Lib::MotionStatus
* COBOTTA\_Lib::SetSpeed
* COBOTTA\_Lib::Halt
* COBOTTA\_Lib::HandChuck
* COBOTTA\_Lib::HandUnChuck

Für die Kommunikation mit der Kamera werden Methoden zur Verfügung gestellt, die nur Code ausführen, wenn in der *connect*-Methode die Kamera aktiviert wurde:

* COBOTTA\_Lib::OneShotFocus
* COBOTTA\_Lib::OneShotWhiteBalance
* COBOTTA\_Lib::getCamImage

Um mit der letztgenannten Methode ein Kamerabild zu bekommen, muss dem beim Verbindungsaufbau erzeugten *CaoController* zunächst die Variable „IMAGE“ hinzugefügt, deren Wert als sogenannter *MemoryStream*[[8]](#footnote-8) ausgelesen und in ein Bildobjekt übergeben werden. Daraufhin wird die Variable wieder vom *CaoController* entbunden.

Dasselbe Verfahren kommt bei dem Auslesen verschiedenster Variablen des Roboters zum Einsatz, welche per *CaoController::AddVariable* angesprochen werden können. Die dafür zuständigen Methoden lauten:

* COBOTTA\_Lib::getPVariable
* COBOTTA\_Lib::setPVariable
* COBOTTA\_Lib::getBusyValue
* COBOTTA\_Lib::getMovementDestination

Zum Ausführen von Bewegungen stehen folgende Methoden zur Verfügung:

* COBOTTA\_Lib::MoveToPos
* COBOTTA\_Lib::MoveCurve
* COBOTTA\_Lib::Depart

Innerhalb dieser Methoden wird zunächst die übergebene Zielposition per *Execute*-Option *OutRange* auf Erreichbarkeit geprüft. Ist dies erfolgreich, wird der Roboter über den Befehl *CaoRobot::Move* dorthin bewegt.

Um sicherzustellen, dass die Bewegung vor dem nächsten Befehl abgeschlossen wird, folgt der Aufruf der Methode *COBOTTA\_Lib::waitMoveEnd* direkt auf den Bewegungsbefehl. Innerhalb dieser iteriert[[9]](#footnote-9) eine *while*-Schleife so lange, bis der *Execute*-Befehl *MotionComplete* den Wert *false* zurückgibt.

Danach wird nochmals 700 Millisekunden gewartet, bevor die Methode wieder verlassen wird.

Ist der Benutzer mit seiner Arbeit fertig, kann die Roboterverbindung per *COBOTTA\_Lib:disconnect* wieder getrennt werden, was alle in der *connect*-Methode erzeugten Objekte entkoppelt und als Nullwert definiert.

Sämtliche Methoden besitzen eine try-catch-Anweisung, welche dafür sorgt, mögliche Fehler abzufangen und dem Benutzer als Benachrichtigung anzuzeigen. Somit wird verhindert, dass das Programm unkontrolliert abstürzt und die Roboterverbindung womöglich in einem kritischen Moment abbricht.

Eine detaillierte Auflistung aller genannten Methoden samt Übergabe- und Rückgabeparameter ist in Anhang A1 enthalten.

## 4.3 Benutzeroberfläche

### 4.3.1 Allgemeiner Aufbau

## 4.4 Manuelle Drohnensteuerung

### 4.4.1 Steuerungschema

HIER DAS STEUERUNGSSCHEMA BESCHREIBEN. WICHTIG!!!!

### 4.4.1 Per Tastatur

### 4.4.2 Per Controller

## 4.5 Automatischer Betrieb

### 4.5.1 Objektverfolgung

### 4.5.2 XY

#### 4.5.2.4 Schwierigkeiten

- Schlechte Ansteuerbarkeit

- Langsame Reaktionen

- keine Steuerung im Dunklen per cm

- per PC langsamere Reaktion als per Surface (WLAN-Speed)

- keine wirkliche Echtzeit

- spontane Fehler seitens der Bibliothek

# 5. Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine leicht handhabbare Software zur Steuerung des neuen Robotermodells COBOTTA zu entwickeln, die zu Lehrzwecken verschiedene LEGO-Steine anhand von Form und Farbe sortieren kann.

Die Sortieralgorithmen sollten sich dabei einerseits auf herkömmliche Bildverarbeitung und andererseits auf das intelligente Lernen mittels eines neuronalen Netzes stützen. Die Benutzerschnittstelle wurde in der Programmiersprache C# entwickelt und von den bewährten Klassenbibliotheken EmguCV und Keras.NET unterstützt.

Für diese wurde zunächst eine grundlegende Evaluation durchgeführt, welche zu der genannten Auswahl führte.

Zur Kommunikation mit dem Roboter wurde eine separate Klassenbibliothek entworfen, welche die kombinierte Befehlsverwaltung der mit dem Roboter gelieferten Schnittstelle ORiN2 in ein einheitliches Erscheinungsbild verpackt. Sie bietet somit eine flexible Basis für zukünftige Projekte, in denen der COBOTTA ebenfalls Anwendung findet.

Die Steuerung der Objekterkennung und die eigentliche Roboterbewegung finden hingegen innerhalb der Benutzeroberfläche statt. Diese bietet eine Auswahl an manuellen und programmgesteuerten Bewegungsmöglichkeiten.

Das Kernstück des Projekts bildet dabei der Reiter „Object sorting“, unter dem zum einen nur mittels der Bildverarbeitungssoftware EmguCV die LEGO-Steine grob nach Farbe und Form sortiert werden können, zum anderen aber auch die Möglichkeit des maschinellen Lernens geboten wird, welche präzise auch unkonventionellere Steine erkennt.

Bedingt durch nur wenige Ablagemöglichkeiten und optische Fehler wie falschem Weißabgleich und problematischem Schattenwurf kann die konventionelle Objekterkennung zwar nur recht grobe Unterschiede erkennen, sie bietet allerdings den Vorteil, auch nach Farben sortieren zu können.

Dagegen ist die intelligente Objekterkennung aufgrund des Aufbaus ihres neuronalen Netzes und zugunsten einer einheitlichen und genauen Kantenerkennung zunächst auf schwarze LEGO-Steine begrenzt, kann im Gegenzug aber mit einer beinahe fehlerfreien Klassifizierung komplexer Formen überzeugen, die nur bei sehr ähnlichen Objekten sporadisch versagt.

Das Anfahren und Aufnehmen der Testobjekte erfolgt nach dem jeweils gleichen Schema und stützt sich auf die Berechnung der Position auf dem jeweils aufgenommenen Bild, kombiniert mit dem Wissen über den Abstand der Kamera zur Ablagefläche. Das Verfahren würde mit nur einem weiteren Korrekturbild auskommen, wurde aus Sicherheitsgründen aber mit zwei Prüfvorgängen versehen, wodurch es sehr zuverlässig agiert.

Neben den bereits umgesetzten Bewegungsbefehlen bietet die Klassenbibliothek ORiN2 weitere Möglichkeiten, den Roboter zu manipulieren. Da sie für diese Bachelorarbeit nicht von Relevanz waren, wurden sie bisher nicht implementiert, könnten aber in Zukunft zu der Klassenbibliothek COBOTTA\_Lib hinzugefügt werden.

Darüber hinaus existieren viele weitere Möglichkeiten, die Bilderkennung in beiden Methoden zu verbessern respektive zu verfeinern.

Denkbar wäre eine Kombination beider Verfahren, die sich sowohl der genauen Klassifizierung eines LEGO-Steins mittels neuronalem Netz bedient, zusätzlich aber auch die Farbe über die konventionelle Bildverarbeitung zur Einordnung heranzieht.

Weiterhin könnten tiefergreifende Versuche mit verschiedenen Konfigurationen des neuronalen Netzes durchgeführt werden, um beispielsweise die beschriebenen Fehleinschätzungen bei ähnlichen Objekten zu vermeiden und die Genauigkeit noch weiter zu steigern.

Ebenfalls zu erwähnen ist, dass der Einfluss des beschriebenen „Overfitting“ im Rahmen dieser Arbeit nicht hinreichend untersucht werden konnte. Durch weitere Forschungen in diesem Gesichtspunkt könnte ebenfalls eine Verbesserung der Klassifizierung erreicht werden.

# Anhang

**Anhangsverzeichnis**

[A1 Methoden der COBOTTA\_Lib IV](#_Toc177419945)

[A2 Menüseiten des COBOTTA Control Center IV](#_Toc177419946)

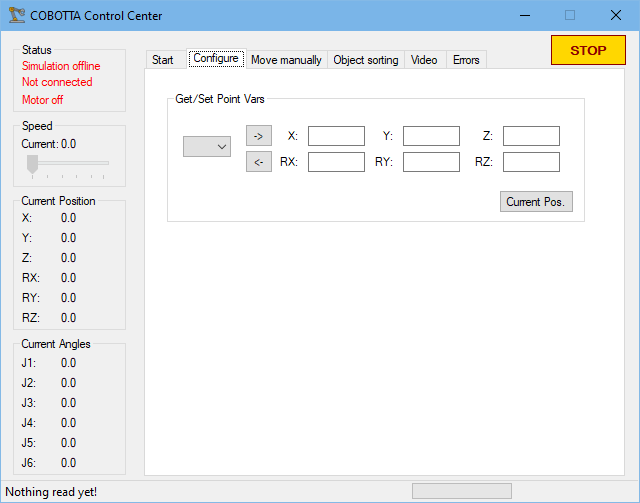
[A2.1 Seite „Configure“ IV](#_Toc177419947)

A1 Methoden der COBOTTA\_Lib

|  |  |
| --- | --- |
| **COBOTTA\_Lib::connect** | |
| Übergabeparameter: 3 | string RC8\_ip, [bool add\_cam], [string cam\_ip] |
| Rückgabeparameter: 1 | bool success |
| Stellt die Verbindung zum Roboter her. *RC8\_ip* ist dabei die IP-Adresse des Roboters, *add\_cam* definiert, ob die Kamera aktiviert werden soll und *cam\_ip* definiert die IP der Kamera | |

A2 Menüseiten des COBOTTA Control Center

A2.1 Seite „Configure“

****

1. *charge-coupled device* (dt. ‚ladungsgekoppeltes Bauteil‘) [↑](#footnote-ref-1)
2. Eine Fotodiode erzeugt eine dem Lichteinfall entsprechende Spannung [↑](#footnote-ref-2)
3. Üblicherweise die sog. Bayer-Matrix [↑](#footnote-ref-3)
4. unveränderte [↑](#footnote-ref-4)
5. Addiert bzw. mischt die Farbanteile der Grundfarben zu einem finalen Farbwert [↑](#footnote-ref-5)
6. Fotodioden, die beispielsweise aufgrund einer Fehlfunktion zu helle oder dunkele Farbwerte ausgeben [↑](#footnote-ref-6)
7. Körnige, unruhige Struktur, entsteht meist durch variierende Empfindlichkeit einzelner Sensordioden, nicht zu verwechseln mit Rauschen innerhalb einer Datenmenge (vgl. Kapitel 2.4) [↑](#footnote-ref-7)
8. Sinnbildlich etwa „Speicherdatenstrom“ [↑](#footnote-ref-8)
9. durchläuft [↑](#footnote-ref-9)