**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 1](#_Toc176651610)

[2. Grundlagen 1](#_Toc176651611)

[2.1 Programmiergrundlagen 1](#_Toc176651612)

[2.1.1 Programmiersprache Python 1](#_Toc176651613)

[2.1.2 Grafische Darstellung: PyGame 3](#_Toc176651614)

[2.2 Drohnentechnik 3](#_Toc176651615)

[2.2.2 Arten von Drohnen 3](#_Toc176651616)

[2.3 Bildverarbeitung 3](#_Toc176651617)

[3. Systemkomponenten 7](#_Toc176651618)

[3.1 Hardware 7](#_Toc176651619)

[3.1.1 Drohne 7](#_Toc176651620)

[3.2 Software 8](#_Toc176651621)

[3.2.1 PiCharm 8](#_Toc176651622)

[3.2.2 ORiN2 8](#_Toc176651623)

[3.2.2.1 Grundlagen 8](#_Toc176651624)

[3.2.3 Denso Teach Pendant 8](#_Toc176651625)

[3.2.4 OpenCV 8](#_Toc176651626)

[3.2.4.1 Grundlagen 8](#_Toc176651627)

[3.2.4.3 Wichtige Bildoperationen 10](#_Toc176651628)

[4. Implementation / Ergebnisse 11](#_Toc176651629)

[4.1 Verbindungsaufbau 11](#_Toc176651630)

[4.2 Erstellte Klassenbibliotheken 12](#_Toc176651631)

[4.2.1 Keyboardcontrol 12](#_Toc176651632)

[4.2.2 XBoxControl 12](#_Toc176651633)

[4.2.3 dronecomms 12](#_Toc176651634)

[4.2.4 ImageProcessing 12](#_Toc176651635)

[4.3 Benutzeroberfläche 16](#_Toc176651636)

[4.3.1 Allgemeiner Aufbau 16](#_Toc176651637)

[4.4 Manuelle Drohnensteuerung 16](#_Toc176651638)

[4.4.1 Per Tastatur 16](#_Toc176651639)

[4.4.2 Per Controller 16](#_Toc176651640)

[4.5 Automatischer Betrieb 16](#_Toc176651641)

[4.5.1 Objektverfolgung 16](#_Toc176651642)

[4.5.2 XY 17](#_Toc176651643)

[4.5.2.4 Schwierigkeiten 17](#_Toc176651644)

[5. Zusammenfassung und Ausblick 18](#_Toc176651645)

[Literaturverzeichnis I](#_Toc176651646)

[Anhang III](#_Toc176651647)

# 1. Einleitung / Aufgabenstellung

# 2. Grundlagen

## 2.1 Programmiergrundlagen

### 2.1.1 Programmiersprache Python

Die im Rahmen dieser Bachelorarbeit erstellte Software basiert auf der Programmiersprache C#[[1]](#footnote-1).

Die Sprache wurde erstmalig im Jahre 2002 von Microsoft veröffentlicht und erhielt zuletzt 2020 mit ihrer neunten Version eine umfassende Erweiterung. Obwohl sie plattformübergreifend angewendet werden kann, ist sie vorwiegend für den Einsatz mit den ebenfalls von Microsoft stammenden Software-Plattformen .NET-Framework oder .NET-Core vorgesehen. (Vgl. [[JOS20](#JOS20)]) Besondere Kompatibilität besteht entsprechend auf Computern mit dem Betriebssystem Windows.

Syntaktisch orientiert sich C# am Aufbau von Java. Bei beiden Sprachen handelt es sich um sogenannte objektorientierte Programmiersprachen, die sich in Programmierablauf und Aufbau grundlegend von den prozeduralen Programmiersprachen unterscheiden.

Die Objektorientierung basiert auf dem Konzept, Daten und Variablen innerhalb von Klassen zu abstrahieren und dadurch universell und mehrfach nutzbar zu machen. Eine Klasse ist dabei eine Kapselung von Methoden und Eigenschaften, welche zumeist ein bestimmtes reales oder theoretisches Modell beschreiben und die Logik des selbigen definieren kann.

Die Daten in einer objektorientierten Umgebung gehorchen einer festgelegten Hierarchie, die durch bestimmte Parameter insofern beeinflusst werden kann, dass beispielsweise auserwählte Variablen global festgelegt werden können. Dies geschieht in diesem Fall über das Attribut *public* während der Deklaration[[2]](#footnote-2) der Variable.

Grundsätzlich jedoch gehören einem Objekt alle ihm zugewiesenen Variablen und Methoden und es kann sie selbsttätig verwalten. Dadurch bleibt die „Konsistenz der Daten“ innerhalb der Klasse geschützt. [[BER09](#BER09)]

Diese Klassen lassen sich nach ihrer Definition beliebig oft in anderem Kontext aufrufen und ihre Funktionen in diesem flexibel nutzen. Dabei können aus einer einzigen Klasse unbegrenzt viele Objekte mit jeweils einzigartigen Eigenschaften, aber dennoch gleichem Verhalten erzeugt werden.

In C# müssen sämtliche Klassen in einem sogenannten *namespace* deklariert werden, sodass sich für eine einfache „Hallo-Welt“-Applikation[[3]](#footnote-3) folgende Struktur ergibt:

using System;

namespace HelloWorldConsole

{

class HelloWorld

{

static void Main(string[] args)

{

Console.WriteLine("Hallo, Welt");

}

}

}

using System;

Macht die Klassen des *namespace* für den nachfolgenden Quelltext aufrufbar.

namespace HelloWorldConsole

Legt den Namensbereich fest, unter welchem die Klasse *HelloWorld* in wiederum anderen Projekten ansprechbar sein wird. Dabei können mehrere Klassen einem *namespace* hinzugefügt werden und in einem Codeprojekt mehrere *namespaces* existieren.

class HelloWorld

Erstellt eine Klasse nach der eingangs beschriebenen Daten- und Methodenhierarchie. In ihr können beliebig viele Methoden und Parameter festgelegt werden. Sie kann nach Aufruf des *namespace* von anderen Projekten genutzt werden.

static void Main(string[] args)

Deklariert eine Methode in der Klasse *HelloWorld*. Im regulären Fall muss immer der Datentyp des Rückgabewerts angegeben werden, der in diesem Fall *void*, also „nichts“, ist. Sollte ein anderer Datenwert gewählt werden (bspw. *int*[[4]](#footnote-4)), so muss die Methode zwingend ein *return*-Statement besitzen, welches einen entsprechenden Wert zurückliefert.

Während *Main* als Schlüsselwort für jene Methode gilt, die bei Start eines Programms ausgeführt wird, kann dieser Bezeichner für weitere Methoden beliebig verändert werden, darf sich jedoch nicht doppeln.

Console.WriteLine("Hallo, Welt");

Nutzt die aus *System* geladene *Console*-Klasse, um die in ihr enthaltene *WriteLine*-Methode mit dem Übergabeparameter „Hallo, Welt“ auszuführen. Diese schreibt jene Worte in die Ausgabekonsole der Entwicklungsumgebung.

(Beispielcode vgl. [TIM11, S.4])

### 2.1.2 Grafische Darstellung: PyGame

## 2.2 Drohnentechnik

### 2.2.2 Arten von Drohnen

## 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[[5]](#footnote-5)-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[[6]](#footnote-6) 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[[7]](#footnote-7) 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.

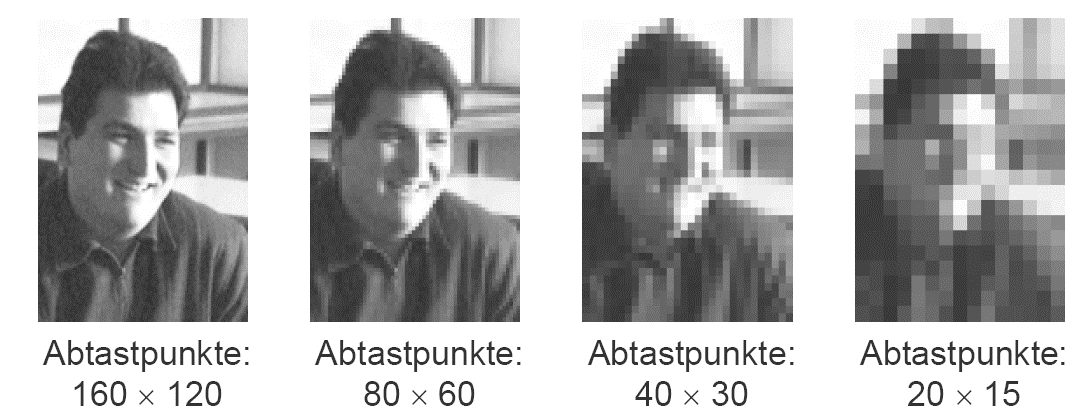
Das vom Prozessor generierte Bild besitzt eine native[[8]](#footnote-8) maximale Auflösung, die von der Anzahl der Fotodioden auf dem Sensor bestimmt wird. Die Pixel sind dabei in einem rechteckigen Raster angeordnet, welches so die Bildhöhe- und breite definiert. Für jedes Pixel wird der finale Farbwert 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“[[9]](#footnote-9).

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

## 3.2 Software

### 3.2.1 PiCharm

### 3.2.2 ORiN2

#### 3.2.2.1 Grundlagen

### 3.2.3 Denso Teach Pendant

### 3.2.4 OpenCV

#### 3.2.4.1 Grundlagen

Das Softwarepaket EmguCV ist ein sogenannter .NET-Wrapper[[10]](#footnote-10) für die ursprünglich in Python geschriebene Bildverarbeitungssoftware OpenCV. [[EMG22](#EMG22)]

Die Klassenbibliothek ist kostenlos erhältlich und wird in Visual Studio über die in Kapitel 3.2.1 erläuterte Verwaltungssoftware „NuGet“ eingebunden und bereitgestellt. Es stehen verschiedene Versionen zur Verfügung, die jeweils auf einer bestimmten Version vom .NET-Framework lauffähig sind. In Kombination mit der in Kapitel 3.2.1 genannten Framework-Version erwies sich die Version 4.4.0.4099 von EmguCV als kompatibel.

Die Software ist dabei in mehreren Teilpaketen erhältlich, welche verschiedene Funktionalitäten hinzufügen. Für diese Arbeit wurden folgende Pakete verwendet:

* Emgu.CV
* Emgu.CV.Bitmap
* Emgu.CV.runtime.windows
* Emgu.CV.UI

**Emgu.CV** stellt dabei alle Grundfunktionen zur Verfügung, während **Emgu.CV.runtime.windows** diese für Windows lauffähig macht. **Emgu.CV.UI** und **Emgu.CV.Bitmap** fügen weitere essenzielle Funktionen hinzu, beispielsweise das Konvertieren eines mit EmguCV bearbeiteten Bildes in ein von C# direkt unterstütztes Bildformat.

Wie eingangs erwähnt, basiert EmguCV auf der für die Programmiersprache Python verfügbaren Software OpenCV zur dortigen Bildverarbeitung. Sie wurde erstmals 2002 veröffentlicht und seitdem stetig weiterentwickelt. [[SHE22](#SHE22)] Kurz darauf wurde sie mittels EmguCV für in Visual Studio geschriebene Codeprojekte nutzbar gemacht.

Der Aufbau von Klassenhierarchie und Syntax in OpenCV respektive EmguCV unterscheidet sich in Python und C# je nach Anwendungsfall enorm.

Am auffälligsten ist die Art der Objektrückgabe nach Ausführung einer Operation. Während in Python die Rückgabe der aufgerufenen Methode als Zuweisung abgefangen werden muss, basieren die meisten korrespondierenden Funktionen von EmguCV auf der Angabe des Zielobjekts als Argument.

Veranschaulichen lässt sich dies am besten am nachstehenden Vergleich. Es soll ein Bild mit einem Gauß’schen Glättungsfilter bearbeitet werden. Ausgangslage ist das Vorliegen eines Bildes *img* und die importierte Bibliothek *cv2*(Python) beziehungsweise *Emgu.CV* (C#):

**Python:**

blurredimg = cv2.GaussianBlur(img, (7,7), 0)

**C#:**

Mat blurredimg = new Mat();

Emgu.CV.CVInvoke.GaussianBlur(img, blurredimg, new Size(7, 7), 0);

Beide Codebeispiele führen exakt die gleiche Operation aus. Jeder Bildpunkt des eingegebenen Bildes *img* wird mit allen benachbarten Pixeln im vertikalen und horizontalen Abstand von sieben Pixeln nach dem Gauß’schen Verfahren weichgezeichnet und dann auf die Variable *blurredimg* geschrieben. Allerdings ist die Angabe dieses Ergebnisses wie angesprochen grundlegend verschieden.

Im den nächsten Kapiteln werden nun die wichtigsten Methoden der Bibliothek EmguCV erläutert.

#### 3.2.4.3 Wichtige Bildoperationen

Die wichtigste Methode zur Bildmanipulation ist die bereits beispielhaft genutzte Klasse *CVInvoke*. Wenn auch syntaktisch unterschiedlich, verfügt sie über fast alle auch in OpenCV vorhandenen Funktionen. Die nächsten Tabellen stellen die für dieses Codeprojekt wichtigen Methoden vor. Der Aufbau der Tabellen ähnelt selbigem in Kapitel 3.2.2.

*Hinweis:* Die im vorigen Kapitel beschriebene Matrix wird bei der *CVInvoke*-Methode oftmals als *IInputArray* (Eingabebild) respektive *IOutputArray* (Rückgabebild) verarbeitet. Die Struktur ist jedoch identisch mit jener Matrix.

|  |  |
| --- | --- |
| **CVInvoke::GaussianBlur** | |
| Übergabeparameter: 1 | IInputArray src, IOutputArray dest, Size size, double sigmaX |
| Rückgabeparameter: 0 |  |
| 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[[11]](#footnote-11) oder Bildrauschen[[12]](#footnote-12) reduziert. | |
| CVInvoke.GaussianBlur(inimg, outimg, new Size(7,7),0); | |

# 4. Implementation / Ergebnisse

## 4.1 Verbindungsaufbau

## 4.2 Erstellte Klassenbibliotheken

### 4.2.1 Keyboardcontrol

### 4.2.2 XBoxControl

### 4.2.3 dronecomms

### 4.2.4 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*[[13]](#footnote-13) 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[[14]](#footnote-14) 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 Per Tastatur

### 4.4.2 Per Controller

## 4.5 Automatischer Betrieb

### 4.5.1 Objektverfolgung

### 4.5.2 XY

#### 4.5.2.4 Schwierigkeiten

# 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.

# Literaturverzeichnis

|  |  |
| --- | --- |
| [IFR22] | IFR Statistical Department, "Top 5 Robot Trends 2022," Februar 2022. |
| [JOS20] | Joseph Albahari & Ben Albahari, *C# 8.0 kurz & gut*. Heidelberg: O'Reilly, 2020. |
| [BER09] | Gregor Raýman, Stefan Strich Bernhard Lahres, *Objektorientierte Programmierung*.: Rheinwerk-Verlag, 2009. [Online]. <https://openbook.rheinwerk-verlag.de/oop/oop_kapitel_02_002.htm#mjd081ab77f1ee1d3bf3f15> |
| [HAN22] | Hans-Peter Willig. CCD-Sensor - Physik-Schule. [Online]. <https://physik.cosmos-indirekt.de/Physik-Schule/CCD-Sensor> |
| [MIC15] | Michael A. Nielsen, *Neural Networks and Deep Learning*.: Determination Press, 2015. [Online]. <http://neuralnetworksanddeeplearning.com/> |
| [CAN18] | Canon Inc. (2018) N10-W02. Handbuch. |
| [JUN22] | Benjamin Jung. (2022) Das große Computer ABC. [Online]. <https://www.das-grosse-computer-abc.de/CSharp/Erweitert/Klassenbibliotheken> |
| [MIC22] | Microsoft Corporation. (2022) Was ist eine DLL? [Online]. <https://docs.microsoft.com/de-de/troubleshoot/windows-client/deployment/dynamic-link-library> |
| [GUS22] | Daniel Gustorff. Was bedeutet Debuggen?. [Online]. <https://www.lernmoment.de/alle/was-bedeutet-debuggen/> |
| [SCH22] | Dr. Holger Schwichtenberg. IT-Visions - Was ist.NET Framework? [Online]. <https://www.it-visions.de/glossar/alle/284/Was_ist_das_NET_Framework.aspx> |
| [RCP19] | o.V, Denso Wave Inc. (2019, August) RC8 Provider for DENSO Robot RC8. Handbuch. |
| [EMG22] | o.V., EmguCV. (2022, Januar) EmguCV - Main Page. [Online]. <https://www.emgu.com> |
| [SHE22] | David Millan Escriva Roy Shilkrot. History of OpenCV from v1 to v4. Buch "Mastering OpenCV 4". [Online]. <https://www.oreilly.com/library/view/mastering-opencv-4/9781789533576/2de0893d-e450-417d-b336-a4143799b43d.xhtml> |
| [KER22] | Keras.NET. GitHub - SciSharp/Keras.NET. [Online]. <https://github.com/SciSharp/Keras.NET> |
| [RAS17] | Tariq Rashid, *Neuronale Netze selbst programmieren*. Heidelberg: O'Reilly, 2017. |
| [TIM11] | Timo Riedel, "Denso-Roboter," Westfälische Hochschule, Gelsenkirchen, Bachelorarbeit 2011. |
| [DEN19] | o.V, Denso Wave Inc. (2019) COBOTTA-BENUTZERHANDBÜCHER. |
| [KON21] | Prof. Dr. Wolfgang Konen. (2021) Bildverarbeitung und Algorithmen. Vorlegung. [Online]. <http://www.gm.fh-koeln.de/~konen/WPF-BV/BV03.pdf> |
| [KAI18] | Richard Kaiser, *C++ mit Visual Studio 2017 und Windows Forms-Anwendungen*. Tübingen, Deutschland: Springer Vieweg, 2018. |
| [DEN22] | o.V, Denso Wave Inc. WINCAPSⅢ. [Online]. <https://www.denso-wave.com/en/robot/product/software/wincaps3.html> |
| [CON22] | o.V. Convolutional Neuronal Network - Wikipedia. [Online]. <https://de.wikipedia.org/wiki/Convolutional_Neural_Network> |

# Anhang

**Anhangsverzeichnis**

[A1 Methoden der COBOTTA\_Lib IV](#_Toc101351471)

[A2 Menüseiten des COBOTTA Control Center XI](#_Toc101351472)

[A2.1 Seite „Configure“ XI](#_Toc101351473)

[A2.2 Seite „Move manually“ XI](#_Toc101351474)

[A2.2.1 Unterseite „Move Relative“ XI](#_Toc101351475)

[A2.2.2 Unterseite „Move Absoulte“ XII](#_Toc101351476)

[A2.2.3 Unterseite „Depart“ XII](#_Toc101351477)

[A2.2.4 Unterseite „Hand“ XIII](#_Toc101351478)

[A2.3 Seite „Object sorting“ XIII](#_Toc101351479)

[A2.3.1 Unterseite „Normal sorting“ XIII](#_Toc101351480)

[A2.3.2 Unterseite „Intelligent sorting“ XIV](#_Toc101351481)

[A2.4 Seite „Video“ XIV](#_Toc101351482)

[A2.5 Seite „Errors“ XV](#_Toc101351483)

[A3 Detailansichten des Testaufbaus XVI](#_Toc101351484)

[A3.1 Ablagebereich für zu sortierende Objekte XVI](#_Toc101351485)

[A3.2 Register zum Einsortieren XVII](#_Toc101351486)

[A4 Übersicht über verwendete LEGO-Steine XVIII](#_Toc101351487)

[A5 Für das neuronale Netz vorbereitete Bilddaten XVIII](#_Toc101351488)

[A6 Code zur Erzeugung des neuronalen Netzes XIX](#_Toc101351489)

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 | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::disconnect** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | bool success |
| Trennt die Verbindung zum Roboter | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getPVariable** | |
| Übergabeparameter: 1 | string var |
| Rückgabeparameter: 1 | float[] pos |
| Gibt die Werte der Punktvariable *var* (P0-P20) als Array zurück. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::setPVariable** | |
| Übergabeparameter: 3 | string var, double x, double y, double z, double rx, double ry, double rz, [double fig] |
| Rückgabeparameter: 1 | bool success |
| Setzt den Wert der Variable *var* (P0-P20) auf die mitgegebenen Werte für Position und Rotation. Die Figur *fig* muss nicht zwingend angegeben werden. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::TakeArm** | |
| Übergabeparameter: 2 | [int armgroup], [int keep] |
| Rückgabeparameter: 1 | bool success |
| Übernimmt die Armgruppe.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::GiveArm** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | bool success |
| Gibt den Arm frei.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::MotorOn** | |
| Übergabeparameter: 1 | int wait |
| Rückgabeparameter: 1 | bool success |
| Schaltet die Motoren ein.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::MotorOff** | |
| Übergabeparameter: 1 | int wait |
| Rückgabeparameter: 1 | bool success |
| Schaltet die Motoren aus.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::curPos** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | double[] pos |
| Gibt die Werte der aktuellen Position als Koordinatenwerte als Array zurück. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::curJnt** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | double[] jnt |
| Gibt die aktuellen Werte der Armgelenke als Koordinatenwerte als Array zurück. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getBusyValue** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | bool busy |
| Gibt den Wert der internen Variable „BUSY\_VALUE“ als bool zurück. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getCurrentErrorCount** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | int errorcnt |
| Gibt die Anzahl der aktuell vorhandenen Fehlermeldungen zurück.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getErrorInfo** | |
| Übergabeparameter: 1 | int index |
| Rückgabeparameter: 1 | object[] info |
| Gibt die Daten des Fehlers an der Stelle *index* in der Fehlerliste als Array zurück.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::ClearErrors** | |
| Übergabeparameter: 0 | int index |
| Rückgabeparameter: 1 | bool success |
| Löscht alle vorhandenen Fehler.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getMovementDestination** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | double[] success |
| Gibt die Werte der Variable „@DEST\_POSITION“ als Array zurück. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::MotionStatus** | |
| Übergabeparameter: 2 | int armgroup, int mode |
| Rückgabeparameter: 1 | bool status |
| Stellt fest, ob die Bewegung des Roboters beendet ist.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen Funktion *MotionComplete* für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::SetSpeed** | |
| Übergabeparameter: 1 | int speed, int acceleration, int deceleration |
| Rückgabeparameter: 1 | bool success |
| Setzt die Bewegungsgeschwindigkeit fest.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.5 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::Halt** | |
| Übergabeparameter: 1 | int index |
| Rückgabeparameter: 1 | bool success |
| Hält die Bewegung des Roboters direkt an.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.4 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::MoveToPos** | |
| Übergabeparameter: 6 | int movemode, string atmode, string destination, [string options], [string WaitMoveEndPicDest], [PictureBox pic] |
| Rückgabeparameter: 1 | bool success |
| Bewegt den Roboter an die angegebene Position *destination*. Die Bewegung wird über den Befehl *CaoRobot::Move* durchgeführt. *Movemode* gibt die Interpolationsart an, *atmode* die Art des Zielpunktes (Anzugeben bspw. als „@P“ ohne Leerzeichen).  Über die *WaitMoveEndPicDest* und die PictureBox *pic* kann angegeben werden, dass während des Fahrens der Bewegung kontunierlich Bilder durch die Kamera aufgenommen und in dem entsprechenden Ordner gespeichert werden sollen. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::waitMoveEnd** | |
| Übergabeparameter: 2 | [string filedest], [PictureBox pic] |
| Rückgabeparameter: |  |
| Wartet darauf, dass der *Execute*-Befehl *MotionComplete* ein „false“ zurückgibt. Währenddessen können unter Angabe des Ordnerpfades *filedest* und einer PictureBox *pic* durchgehend Bilder von der Kamera aufgenommen und ebendort gespeichert beziehungsweise angezeigt werden. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::MoveCurve** | |
| Übergabeparameter: 7 | int movemode, string atmode, string destination, string curvepoint, [string options], [string WaitMoveEndPicDest], [PictureBox pic] |
| Rückgabeparameter: 1 | bool success |
| Bewegt den Roboter an die angegebene Position *destination*. Die Bewegung wird über den Befehl *CaoRobot::Move* durchgeführt. *Movemode* gibt die Interpolationsart an, *atmode* die Art des Zielpunktes (Anzugeben bspw. als „@P“ ohne Leerzeichen), der *curvepoint* den zu überfahrenden Punkt.  Über die *WaitMoveEndPicDest* und die PictureBox *pic* kann angegeben werden, dass während des Fahrens der Bewegung kontunierlich Bilder durch die Kamera aufgenommen und in dem entsprechenden Ordner gespeichert werden sollen. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::Depart** | |
| Übergabeparameter: 5 | int movemode, double distance, string atmode, [string options], [string WaitMoveEndPicDest], [PictureBox pic] |
| Rückgabeparameter: 1 | bool success |
| Gibt die Daten des Fehlers an der Stelle *index* in der Fehlerliste als Array zurück.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::HandChuck** | |
| Übergabeparameter: 1 | bool waitfordone |
| Rückgabeparameter: 1 | bool success |
| Lässt den Greifer schließen.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::HandUnChuck** | |
| Übergabeparameter: 1 | bool waitfordone |
| Rückgabeparameter: 1 | bool success |
| Lässt den Greifer öffnen.  Die Funktion ist gleichbedeutend mit der in Kapitel 3.2.2.3 beschriebenen für ORiN2. | |

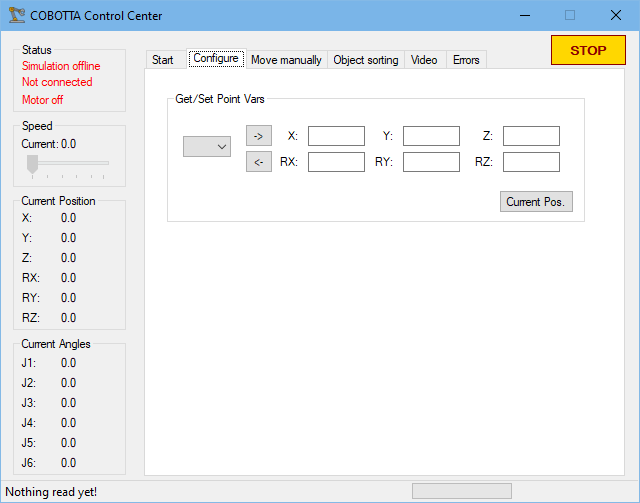
|  |  |
| --- | --- |
| **COBOTTA\_Lib::OneShotFocus** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | bool success |
| Fokussiert die Kamera. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::OneShotWhiteBalance** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | bool success |
| Stellt den Weißabgleich der Kamera neu ein. | |

|  |  |
| --- | --- |
| **COBOTTA\_Lib::getCamImage** | |
| Übergabeparameter: 0 |  |
| Rückgabeparameter: 1 | Image camimage |
| Gibt ein Bild der Kamera zurück. | |

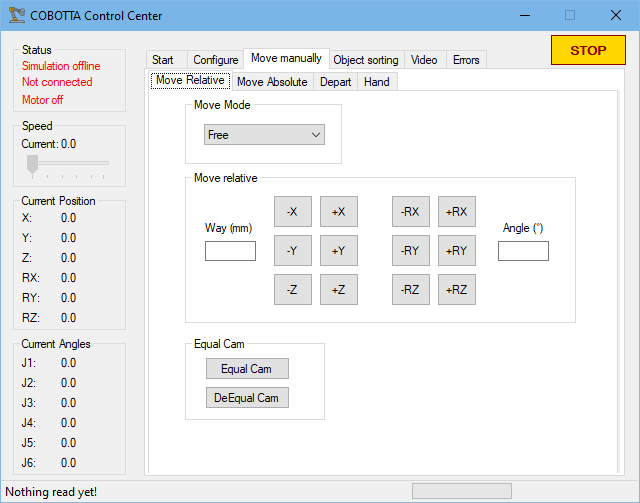
A2 Menüseiten des COBOTTA Control Center

A2.1 Seite „Configure“

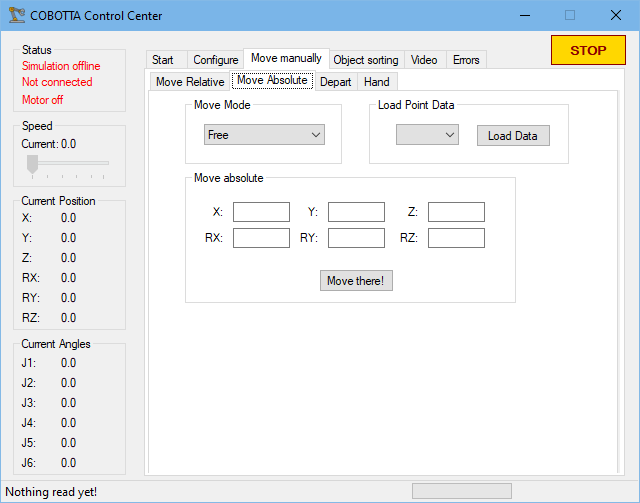
****

A2.2 Seite „Move manually“

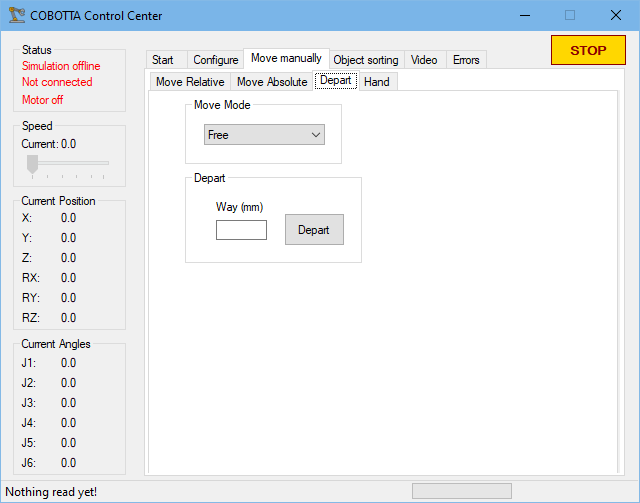
A2.2.1 Unterseite „Move Relative“

****

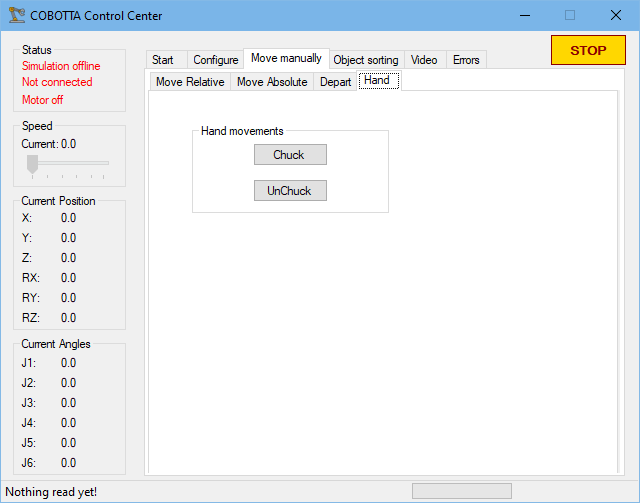
A2.2.2 Unterseite „Move Absoulte“

****

A2.2.3 Unterseite „Depart“

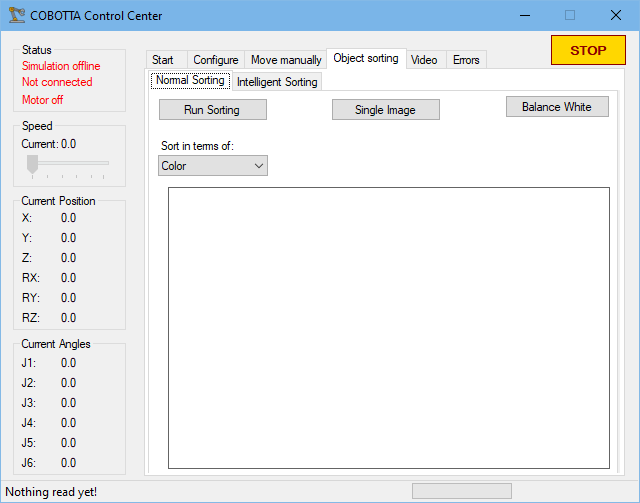
****

A2.2.4 Unterseite „Hand“

****

A2.3 Seite „Object sorting“

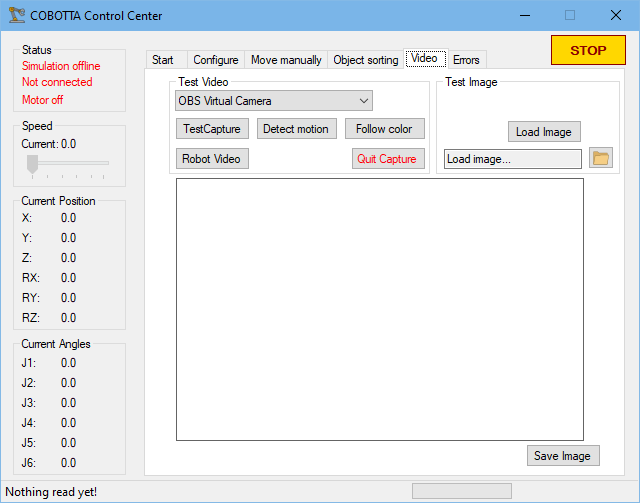
A2.3.1 Unterseite „Normal sorting“

****

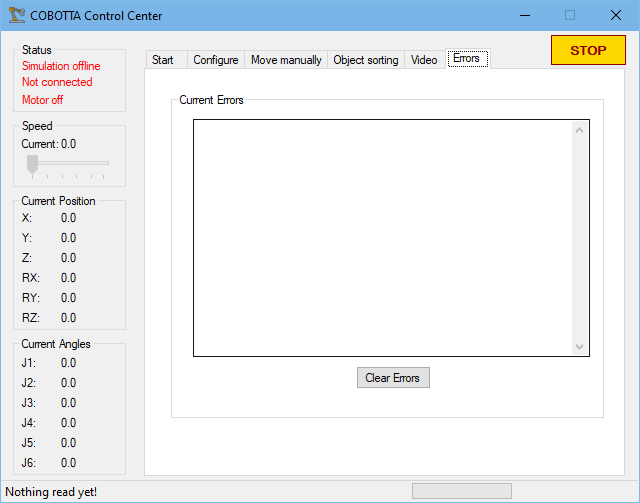
A2.3.2 Unterseite „Intelligent sorting“

****

A2.4 Seite „Video“

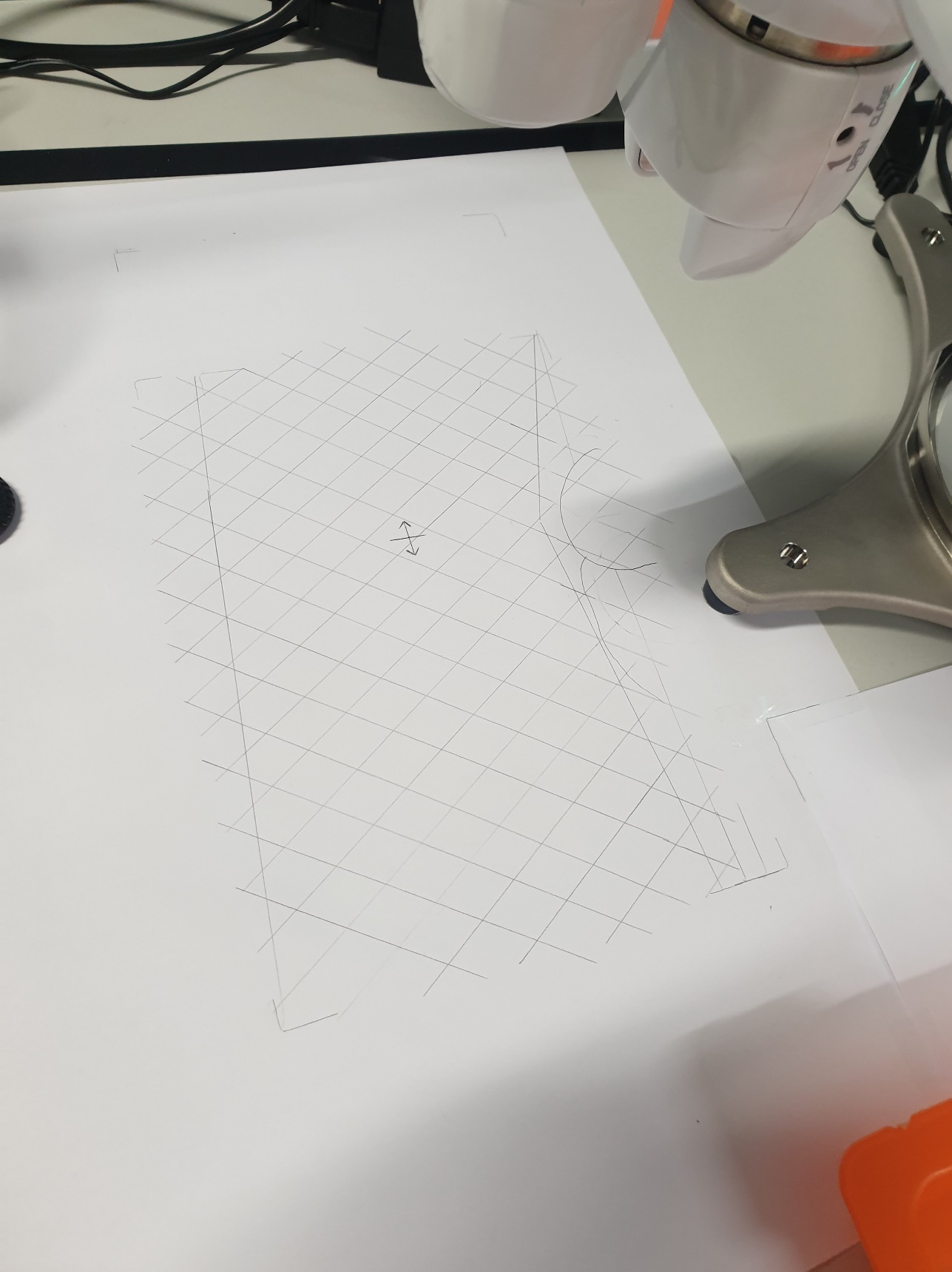
****

A2.5 Seite „Errors“

****

A3 Detailansichten des Testaufbaus

A3.1 Ablagebereich für zu sortierende Objekte

****

A3.2 Register zum Einsortieren

****

A4 Übersicht über verwendete LEGO-Steine

****

A5 Für das neuronale Netz vorbereitete Bilddaten

**    **

A6 Code zur Erzeugung des neuronalen Netzes

var model = new Sequential();

5

10

15

20

25

30

// Layer hinzufuegen

model.Add(new Conv2D(32, kernel\_size: (3, 3).ToTuple(),

activation: "relu",

input\_shape: input\_shape));

model.Add(new MaxPooling2D(pool\_size: (2, 2).ToTuple()));

model.Add(new Conv2D(64, kernel\_size: (3, 3).ToTuple(), activation: "relu"));

model.Add(new MaxPooling2D(pool\_size: (2, 2).ToTuple()));

model.Add(new Dropout(0.35));

model.Add(new Flatten());

model.Add(new Dense(128, activation: "relu"));

model.Add(new Dropout(0.25));

model.Add(new Dense(num\_classes, activation: "softmax"));

//MessageBox.Show("Eingestellt");

// Modell kompilieren

model.Compile(loss: "categorical\_crossentropy",

optimizer: new Adam(), metrics: new string[] { "accuracy" });

//MessageBox.Show("Compiliert");#

MessageBox.Show("CNN-Model successfully loaded");

Anhangsverzeichnis

Links entbläuen

Platzhalter Datum

Fragen:

Anhangsverzeichnis?

1. Gesprochen: C-Sharp [↑](#footnote-ref-1)
2. Definierung, Erstellung [↑](#footnote-ref-2)
3. Anwendung [↑](#footnote-ref-3)
4. Abk. für „Integer“, Ganzzahlwert [↑](#footnote-ref-4)
5. *charge-coupled device* (dt. ‚ladungsgekoppeltes Bauteil‘) [↑](#footnote-ref-5)
6. Eine Fotodiode erzeugt eine dem Lichteinfall entsprechende Spannung [↑](#footnote-ref-6)
7. Üblicherweise die sog. Bayer-Matrix [↑](#footnote-ref-7)
8. unveränderte [↑](#footnote-ref-8)
9. Addiert bzw. mischt die Farbanteile der Grundfarben zu einem finalen Farbwert [↑](#footnote-ref-9)
10. Von engl. „wrapping“, das Einpacken [↑](#footnote-ref-10)
11. Fotodioden, die beispielsweise aufgrund einer Fehlfunktion zu helle oder dunkele Farbwerte ausgeben [↑](#footnote-ref-11)
12. Körnige, unruhige Struktur, entsteht meist durch variierende Empfindlichkeit einzelner Sensordioden, nicht zu verwechseln mit Rauschen innerhalb einer Datenmenge (vgl. Kapitel 2.4) [↑](#footnote-ref-12)
13. Sinnbildlich etwa „Speicherdatenstrom“ [↑](#footnote-ref-13)
14. durchläuft [↑](#footnote-ref-14)