

---

# DIPLOMARBEIT

## Marvin Boardgame Playing Robot

Ausgeführt im Schuljahr 2016/17 von:

Robotersteuerung und Simulation  
Alexander WELLER 5AHIF

Bildverarbeitung und Spiellogik  
Daniel HONIES 5AHIF

**Betreuer / Betreuerin:**

Dipl.-Ing. Harald Haberstroh  
Dr. Michael Stifter

---

Wiener Neustadt, am 4<sup>th</sup> April, 2017

---

Abgabevermerk:

Übernommen von:

# Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 4<sup>th</sup> April, 2017

**Verfasser / Verfasserinnen:**

Alexander WELLER

Daniel HONIES

# Contents

<b>Eidesstattliche Erklärung</b>	<b>i</b>
<b>Acknowledgement</b>	<b>iii</b>
<b>Diplomarbeit Dokumentation</b>	<b>iv</b>
<b>Diploma Thesis Documentation</b>	<b>vi</b>
<b>Kurzfassung</b>	<b>viii</b>
<b>Abstract</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Background . . . . .	1
1.2 Goal . . . . .	1
1.3 Digital Image Processing and Computer Vision (Honies) . . . . .	2
1.3.1 Definition . . . . .	2
1.3.2 History . . . . .	2
1.3.3 Current State . . . . .	3
1.4 Humanoid Robots (Weller) . . . . .	3
1.4.1 General . . . . .	3
1.4.2 History . . . . .	4
1.4.3 Current state . . . . .	4
1.5 Outline . . . . .	4
<b>2 Methodology</b>	<b>5</b>
2.1 Python (Weller) . . . . .	5
2.1.1 General . . . . .	5
2.1.2 Why Python? . . . . .	5
2.2 Computer Vision (Honies) . . . . .	5
2.2.1 OpenCV . . . . .	5
2.2.2 Alternatives . . . . .	6
2.2.3 Camera . . . . .	6
2.3 Checkers (Honies) . . . . .	6
2.3.1 General . . . . .	6
2.3.2 Why Checkers? . . . . .	6
2.3.3 Board and Game Pieces . . . . .	6
2.3.4 How does it work? . . . . .	7
2.3.5 Why English Checkers . . . . .	7
2.4 Checkers engine and GUI (Honies) . . . . .	7

2.4.1	Criteria for selecting the Engine . . . . .	7
2.4.2	Chinook . . . . .	8
2.4.3	Cake . . . . .	8
2.4.4	KingsRow . . . . .	8
2.4.5	GUI Checkers . . . . .	8
2.4.6	Samuel . . . . .	8
2.4.7	Decision . . . . .	9
2.4.8	Others . . . . .	9
2.5	Blender (Weller) . . . . .	10
2.5.1	General . . . . .	10
2.5.2	History . . . . .	10
2.5.3	Development . . . . .	10
2.5.4	Alternatives . . . . .	10
2.6	Inverse Kinematics (Weller) . . . . .	11
2.6.1	General . . . . .	11
2.7	Robot (Weller) . . . . .	12
2.7.1	General . . . . .	12
2.7.2	ASIMO . . . . .	12
2.7.3	Poppy . . . . .	13
2.7.4	Atlas . . . . .	13
2.7.5	Inmoov . . . . .	13
2.7.6	Conclusion . . . . .	14
2.8	Virtualisation (Weller) . . . . .	14
2.8.1	General . . . . .	14
2.8.2	VirtualInmoov . . . . .	15
<b>3</b>	<b>Computer Vision Algorithms (Honies)</b> . . . . .	<b>16</b>
3.1	Grayscale Conversion . . . . .	16
3.2	Downscaling . . . . .	16
3.3	Digital Image Smoothing . . . . .	16
3.3.1	Mean Filter . . . . .	17
3.3.2	Gaussian Blur Filter . . . . .	17
3.4	Thresholding . . . . .	17
3.4.1	Global Thresholding . . . . .	18
3.4.2	Otsu-Thresholding . . . . .	18
3.5	Histogram Equalization . . . . .	19
3.6	Morphological Transformations . . . . .	19
3.6.1	Erosion . . . . .	19
3.7	Corner Detection . . . . .	20
3.7.1	Harris Corner Detector . . . . .	21
3.7.2	Shi-Tomasi Corner Detector . . . . .	21
3.7.3	Good Features to Track Function . . . . .	21
3.8	Blob Detection . . . . .	21
<b>4</b>	<b>Computer Vision Implementation (Honies)</b> . . . . .	<b>23</b>
4.1	Game Flow . . . . .	23
4.2	Detecting the Board - Contours . . . . .	24
4.2.1	Algorithm . . . . .	25
4.2.2	Filtering the Fields . . . . .	25
4.2.3	Arranging the Squares to their Board Position . . . . .	25

4.2.4	Testing . . . . .	25
4.3	Detecting the Board - Corner Detection . . . . .	26
4.3.1	Detection . . . . .	26
4.3.2	Mapping the Corners . . . . .	26
4.3.3	Testing . . . . .	28
4.4	Comparison of Board Detection Methods . . . . .	28
4.5	Detecting Game Pieces . . . . .	28
4.5.1	Process . . . . .	29
4.5.2	Finding the x, y Position . . . . .	30
<b>5</b>	<b>Connecting the engine (Honies)</b>	<b>31</b>
5.1	Connecting Computer Vision and Samuel . . . . .	31
5.2	Board Representation . . . . .	32
5.3	Detecting a Move . . . . .	32
5.4	Generating a Move . . . . .	33
5.5	Passing a Move from the Engine to the Robot (Weller) . . . . .	33
5.5.1	General . . . . .	33
5.5.2	Syntax . . . . .	33
5.5.3	Basic Implementation via a txt File . . . . .	34
5.5.4	Second Implementation via PIPE and Subprocess . . . . .	35
5.5.5	Third Implementation via Sockets . . . . .	36
<b>6</b>	<b>Real Robot (Weller)</b>	<b>38</b>
6.1	General . . . . .	38
6.2	MRL . . . . .	38
6.3	Limitations . . . . .	39
6.4	Inverse Kinematics . . . . .	40
6.4.1	General . . . . .	40
6.4.2	IK3D . . . . .	40
6.4.3	Integrated Movement . . . . .	40
6.4.4	KDL . . . . .	40
6.5	Movement with IK . . . . .	40
6.6	Grabbing stones . . . . .	41
6.7	Human-Robot-Interaction . . . . .	41
<b>7</b>	<b>Simulated Robot</b>	<b>42</b>
7.1	General . . . . .	42
7.2	Limitations . . . . .	42
7.3	Inverse Kinematics . . . . .	42
7.3.1	Standard . . . . .	42
7.3.2	iTaSC . . . . .	43
7.3.3	IK Constraint . . . . .	44
7.3.4	IK Solver . . . . .	44
7.3.5	Bone Configuration . . . . .	44
7.4	Movement with IK . . . . .	44
7.4.1	Basic Steps . . . . .	45
7.4.2	Moving a Stone . . . . .	46
7.4.3	Removing a Stone . . . . .	46
7.4.4	Adding a Stone . . . . .	46
7.5	Grabbing Stones . . . . .	46

7.6 Increasing the Realism . . . . .	47
7.7 Human-Robot-Interaction . . . . .	48
<b>8 Conclusion</b>	<b>49</b>
8.1 Computer Vision (Honies) . . . . .	49
8.1.1 Conclusion . . . . .	49
8.1.2 Discussion . . . . .	49
8.2 Engine (Honies) . . . . .	49
8.2.1 Conclusion . . . . .	49
8.2.2 Future Work . . . . .	50
8.3 Simulation (Weller) . . . . .	50
8.3.1 Conclusion . . . . .	50
8.3.2 Discussion . . . . .	50
8.4 Real Robot (Weller) . . . . .	51
8.4.1 Conclusion . . . . .	51
8.4.2 Discussion . . . . .	51
<b>Glossary</b>	<b>52</b>
<b>Bibliography</b>	<b>53</b>

# Acknowledgement

First we would like to thank everybody who has supported us throughout this thesis.

Most notably we would like to thank our supervisors Dip. Ing. Harald Haberstroh and Dr. Michael Stifter not only for their assistance on this thesis, but for their amazing support in the last four years.

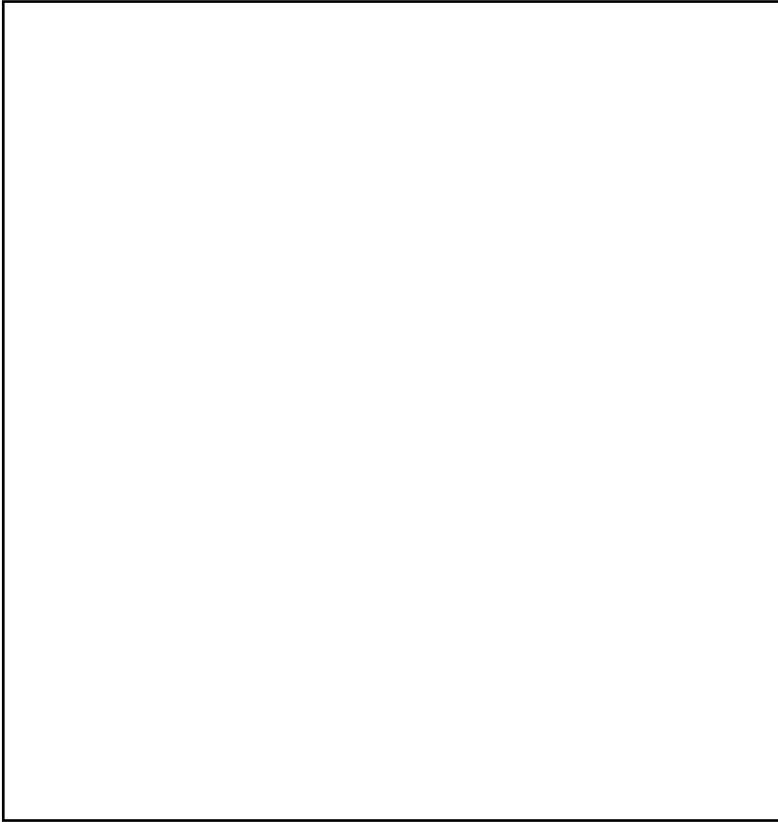
Another notable mention has to be given to Maker Austria and especially Arno Aumayr for allowing us to use the InMoov for this thesis.

We would also like to thank our teachers for the immense amount of knowledge they have taught us and without which we wouldn't have been able to succeed in this project.

Last but not least we would like to thank our families and friends for always believing in us.

## Diplomarbeit Dokumentation

Namen der Verfasser/innen	Alexander Weller Daniel Honies
Jahrgang Schuljahr	5AHIF 2016 / 17
Thema der Diplomarbeit	Marvin: A boardgame playing humanoid robot
Kooperationspartner	F-WUTS; Maker Austria
Aufgabenstellung	Ein humanoider Roboter (Inmoov), oder eine Simulation davon, spielt Dame gegen einen menschlichen Spieler.
Realisierung	Eine Kamera nimmt das Spielbrett und die Spielzüge auf, erkennt diese mittels OpenCV und leitet sie an eine Dame Engine weiter. In dieser Engine wird dann der nächste Zug berechnet und dann per Sockets an ein Python Skript geschickt, welches den Spielzug in Roboterbewegungen umwandelt und diese dann durchführt.
Ergebnisse	Mithilfe einer Simulation in Blender kann der Spieler gegen den Inmoov spielen. Die Spielzüge müssen in der Realität von einem Assistenten nachgespielt werden, da die Simulation auf das echte Spielfeld keinen Einfluss hat.

Typische Grafik, Foto etc. (mit Erläuterung)	Ein Render der Simulation.  
--	---

Teilnahme an Wettbewerben, Auszeichnungen	
---	--

Möglichkeiten der Einsichtnahme in die Arbeit	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt
---	--

Approbation  (Datum, Unterschrift)	Prüfer	Abteilungsvorstand
--	--------	--------------------

## Diploma Thesis Documentation

Authors	Alexander Weller Daniel Honies
---------	-----------------------------------

Form	5AHIF
Academic Year	2016 / 17
Topic	Marvin: A boardgame playing humanoid robot
Co-operation partners	F-WUTS; Maker Austria

Assignment of tasks	A humanoid robot (Inmoov), or simulation thereof, plays checkers against a human player.
---------------------	--

Realization	A camera captures the game field and - turns via OpenCV and sends them to a checkers engine. The engine calculates the best move and transmits them to a Python script via sockets which then converts this move to robot movements.
-------------	---

Results	Through a simulation in Blender the player can challenge the Inmoov and the connected engine. The game turns of the engine have to be replicated by an assistant, because the simulation has no access to the real game board.
---------	---

Illustrative graph, photo  
(incl. explanation)

A render of the simulation.



Participation in  
competitions,  
Awards

Accessibility of diploma  
thesis

HTBLuVA Wiener Neustadt  
Dr.-Eckener-Gasse 2  
A 2700 Wiener Neustadt

Approval  
  
(Date, Sign)

Examiner

Head of Department

# Kurzfassung

Die vorliegende Diplomarbeit beschäftigt sich sowohl mit dem humanoiden Roboter InMoov als auch seiner Simulation und der Möglichkeit ihm Dame beizubringen.

Dafür wurden mehrere Teilbereiche erarbeitet. Dazu gehören Computer Vision, AI, Roboter Steuerung, 3D-Modellierung und Simulationen.

Die erste Aufgabe war die Erkennung von Spielfeld und Spielzügen. Durch Anwendung diverser Filtermethoden und Erkennungsalgorithmen von OpenCV konnten diese erkannt und ihre Eigenschaften festgestellt werden.

Um einen externen Transport der erkannten Spielzüge zur Engine zu vermeiden wurde der CV Code direkt in der Engine implementiert.

Dort wird der erkannte Zug repliziert und der Nächste berechnet. Da der Controller der Simulation bzw. des echten Roboters nicht in der Engine implementierbar war, müssen die berechneten Züge weitergeleitet werden. Dieser Transport erfolgt durch Sockets basierend auf dem multiprocessing Modul von Python, da sowohl die API des echten Inmoov als auch die Simulation des Inmoov in Python implementiert sind.

Nach Empfang der Nachricht wird die Art des Zuges, die betroffenen Steine und sowohl deren alte als auch neue Positionen ausgelesen.

Der Controller wandelt diese Befehle dann in Koordinaten um. Mithilfe von Inverser Kinematik steuert er den Finger beziehungsweise die Hand des Roboters zu dem betroffenen Feld, hebt den Stein auf und bewegt ihn zu dem Ziel. Dies kann ein anderes Spielfeld oder eine Fläche außerhalb des Spielfelds sein um z.B. das entfernen eines Steins darzustellen. Des Weiteren wurden auch Maßnahmen ergriffen um die Realität der Simulation zu erhöhen. Dazu gehören nicht nur Verbesserungen der Render Qualität sondern auch Einstellung der Limitationen des Roboters und Nutzen eines IK Läzers welcher physikalische Limits, zumindest teilweise, in Betracht zieht.

# Abstract

The following thesis is about teaching the humanoid robot InMoov, or a simulation thereof, to play checkers.

To achieve this several parts were developed. This includes Computer Vision, AI, robot controlling, 3D modelling and simulations.

Firstly the game board and turns had to be detected. For this various filters and algorithms from OpenCV were used.

To simplify the transport of information between the engine and OpenCV the CV code was implemented directly in the engine. This allowed the transport of e.g. moves without needing external transmission like sockets.

In the engine the recognised move is replicated and the next move is calculated. Since the controller of the simulation and of the real robot could not be implemented in the engine the calculated moves had to be transported externally via sockets. This transport was achieved by using sockets based on the multiprocessing module from python.

After receiving the messages the type of move, the affected stones and start - and end position are extracted. The controller converts these commands to coordinates and using Inverse Kinematics he is able to move the finger or hand of the robot to the specified field, grab the stone and move it to target coordinate.

This target coordinate can either be another game field or a point outside of the game board to simulate a stone being removed from the game.

Furthermore several measures were adopted to increase the realism of the simulation. These include not only measures to improve the render quality, but also setting limits for the simulation as well as the real robot and using IK Solver that, at least in some part, take physical limits into account.

# Chapter 1

## Introduction

### 1.1 Motivation and Background

With significant advances in computer vision and artificial intelligence in recent years, more and more consumer grade household robots have been released. Humanoid concepts were used in industrial applications for centuries, but now the technology is on a level to recreate and resemble almost all parts of a human body. Simultaneously the emergence of low budget 3D Printers made it possible for artists and engineers alike to design and build low-cost open source humanoid robots. This combination of developments led to the creation of the robot used in this diploma thesis, the Inmoov-Robot. An open-source, community-driven humanoid robot by French designer Gael Langevin. It is now used around the globe thanks to the ability to print it anywhere using a 3D-printer.

### 1.2 Goal

This diploma thesis will take a look at one application showcasing future interactions with humanoid robots in our day to day live.

At the end of the project a robot, specifically the open source humanoid Inmoov Robot, or a simulation thereof is capable of playing a game of English Checkers against a human player. It is intended that minor nontypical behaviour is needed to play against the robot. Nontypical behaviour means i.e. that it is necessary to interact with the computer controlling the robot, for example telling him when the game starts or that a move was made. All of the interaction after starting the program should be done by visually interacting with the machine. The program is able to detect the game start and the game end and will react accordingly. To make this possible the following parts have to be achieved:

- Detecting following things using Computer Vision:
  - Checkerboard
  - Positions of the opponents game pieces
  - Start and end of a game
  - Moves done by the human player
- Linking the visual recognition to a checkers solving program
- Calculating the movement of the robot's body
- Carrying out the calculated move

The game recognition program passes the detected move to an open source checkers GUI and engine. Moves done by the human player will be passed to the open source

engine and then displayed by the GUI. The GUI will not be necessary to play against the robot but for testing purposes should be able to display the next move of the robot and the detected moves of the human player. A new move will be calculated by the engine and then be passed to the robot.

After receiving the next move from the engine the robot will calculate the necessary movement. Not only the movement to the fields has to be consistent, but also the grabbing mechanisms.

## 1.3 Digital Image Processing and Computer Vision (Honies)

### 1.3.1 Definition

Digital image processing refers to when an input image is processed so that an output image is generated. It can also refer to processes where attributes (e.g. edges, contours) are extracted from the input image.[1, pp. 1-3]

Computer Vision overlaps in the case of recognising objects but goes way further than that. It tries to analyse all the detected objects so the computer can understand the whole scene.[1, pp. 2,3]

### 1.3.2 History

Digital image processing began as soon as modern computers were developed. First applications date back to 1964 when the Jet Propulsion Laboratory of NASA was using early techniques to improve pictures of the moon. The late 1960s and early 1970s saw the techniques adopted by the medical industry to be used for computerised tomography. Since then digital image processing was applied to almost all technical endeavours. Examples include electron microscopy, industry monitoring and automatic processing of fingerprints. [1, pp. 3-7]

The beginning of computer vision is often told based on a story that happened in 1966 at MIT. Marvin Minsky asked a few undergraduate students to link a camera to a computer and getting it to describe what it saw, for a summer project. Over 50 years of research later we are still not at that point. While at that time image processing already existed, the field of computer vision was working to recover the 3D structure of a 2D image, which could then be used to fully understand the scene. First edge detectors and 3D modeling algorithms were developed in the 1970s. The 1980s have seen research in all fields paired with more advanced mathematical techniques. Edge and contour detectors were improved. In the late 1990s and early 2000s, many of the nowadays heavily used computational photography algorithms were being developed. Examples include HDR pictures through exposure bracketing and image stitching for panorama photography. Global optimization problems and feature-based techniques for object recognition are appearing since 2000. In recent years most problems have been combined with advanced machine learning algorithms, a prime example being driverless cars with all major brands working on them at the time of this thesis.[2, pp. 10-19]

As digital image processing and computer vision often overlap the thesis will repeatedly refer to computer vision as a generic term for both computer vision and digital image processing.

### 1.3.3 Current State

The current state of computer vision is highly advanced and driven by the development of new autonomous systems, mainly in the automotive sector. A few examples of industrial and consumer-level applications include the following.

#### Applications

- Text recognition

Text recognition is used to read handwritten postal codes and number plates in radar systems. Also, scanners use it to make scanned documents searchable.

- Face detection

Face detection can be found in many different systems including cameras and surveillance systems. Autofocusing cameras can focus faster and more accurate after detecting a face. The United States and other countries apply it to identify persons at airports.

- Vision-based biometrics

Mobile phones and computers have fingerprint readers to control access while more sophisticated security systems are also applying iris detection to identify persons.

- Motion Tracking

By attaching reference points to humans and filming complex movements, those can then be applied to artificial objects, resulting in more natural and human-like behaviour.

- Driverless cars

Autonomous driving systems are developed by all brands. While at the time of this thesis they are not yet fully developed and safe, statistics show that in known environments, they are already performing safer than human drivers. Computer vision is combined with several other fields of computer science including location tracking and machine learning.

## 1.4 Humanoid Robots (Weller)

### 1.4.1 General

The definition of a humanoid robot can easily be explained by first examining the name itself. Humanoid is a combination of the English word “human” and “-oid”, which is adapted from Ancient Greek and means resembling, and a robot is a machine that performs, often complex, tasks. Concluding from these definitions a humanoid robot is a human appearing machine used for performing tasks.[3][4][5]

This subcategory of machines can be further differentiated for example by the way they are controlled i.e. if it is an autonomous or a teleoperated (telepresence) robot, by their purpose i.e. enjoyment or service and by their legal concept i.e. open source or closed source. [6, p. 34] [7, p. 1]

A fully autonomous robot acts and reacts on its own, and does not need supervision[7, pp. 16]. Current examples include, but are not limited to, Boston Dynamics Atlas[6, p. 45] and Hondas ASIMO[6, pp. 40-41].

Telepresence robots are controlled by a human located at the same or a different site. [7] An example for a telepresence humanoid robot is the robonaut, which can be controlled by a human on earth, through the help of virtual reality, while it is on a mission in space[7, p. 6] [8].

### 1.4.2 History

The idea of humanoid servants can be traced back to the ancient Egypt, Greece and China (e.g. golems or the golden maids of Iliad)[7, p. 3].

During this time frame, this concept went through several stages.

From fiction, like the golden maids, statues behaving like human maids [7, p. 3], to basic humanoid machines like "Leonardo's robot"[9], to today's humanoid robots like Inmoov and ASIMO. While at the beginning machines only had certain gestures and couldn't react to their surrounding, modern humanoid robots can, thanks to modern motors and sensors, move in complex ways and interact with the real world [10, p. 280].

### 1.4.3 Current state

This change throughout the last decades now allows them to not be restricted to entertainment and factories anymore, but instead, they are on their way to the military and the service sector[7, p. 27]. Current examples for this development are the therapy of children with autism using humanoids and service or personal assistant robots such as PEARL[7, p. 36][11, p. 229].

## 1.5 Outline

Starting with the Methodology Section the thesis gives an overview of the hardware and software used in the implementation. Furthermore, alternatives and the decision process is presented. Following the Methodology, it is described how the project was implemented in the fields of computer vision, robot control, simulation and game logic. Experiments showcasing and testing the implementations are demonstrated for each subsystem. Finally, a conclusion will give a summary of the work done and a preview of possible future implementations, their challenges and applications.

# Chapter 2

## Methodology

### 2.1 Python (Weller)

#### 2.1.1 General

Python is a high-level, general purpose, object-oriented programming language. It was created by Guido van Rossum as a successor to ABC. The main reason for the development was the lack of exception handling and the difficulty to modify or extend it.[\[12, p. 3\]](#) Another big part in its history was the 2.0 update when it introduced, amongst other things, functionalities like list comprehensions and a garbage collector.

These were not the biggest changes with the 2.0 version. After the release Python went from being developed by one person (Guido van Rossum) to a community-driven, open source project. [\[12, pp. 2-5\]](#)

Eight years later update 3.0 was released with some minor update such as removing the long-datatype and views instead of lists. The big controversy about the switch from 2.7 to 3.0 was the lack of backward compatibility. This meant that most programs written in 2.7 had to be rewritten to support 3.0. That change was necessary because many redundant modules had to be removed or changed to fit the new motto that there should be, preferably, only one obvious way to do something.[\[12, pp. 2-5\]](#)[\[13\]](#)[\[14\]](#)

#### 2.1.2 Why Python?

Python was chosen because most alternatives were either not supported by the libraries or the APIs used in this thesis. These libraries include e.g. OpenCV, iTaSC and the API is the Blender API. [\[15\]](#) [\[16\]](#)

Another advantage of Python is that both parts of the thesis used it. This simplified the connection and communication between the robot, computer vision and checkers engine.

### 2.2 Computer Vision (Honies)

#### 2.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library launched in 1999 by Intel Corporation employee Gary Bradsky. Having received lots of support from big companies, it is currently under development by Itseez (part of Intel) and Arraiy. The library is written in C and C++ and interfaces are available for many common languages like Python and Java. Designed for computational efficiency it provides over 5000 functions and implements more than 500

computer vision algorithms. Used for this project was version 3.1.0 while the current stable version is 3.2.0.[17]

### 2.2.2 Alternatives

Several other computer vision libraries are available and were considered for the project. Amongst them were SimpleCV, BoofCV and libccv. The decision to use OpenCV was made because all of them either lacked important features, their development was stopped or they were not written in python and no python interface was provided. Also, OpenCV is the best documented computer vision library available.

### 2.2.3 Camera

In the project's scope, it was decided that the program should work with a standard low-cost USB webcam. The team, therefore, used an old Logitech C310 HD Webcam with a resolution of 1280 by 720 pixels. It has been used for all computer vision tests and all images in this thesis describing those have been taken with it.

## 2.3 Checkers (Honies)

### 2.3.1 General

Draughts or Checkers is a group of board games popular around the globe with the English version of Checkers being the most popular one. 2007 it was announced that English Checkers is solved, with the result that a perfect play by both sides leads to a draw. It was the most complex game solved at that time [18].

### 2.3.2 Why Checkers?

When searching for a game for the robot to play, Chess and Checkers were considered. It was decided to go for Checkers for two reasons:

- Game Pieces

Checkers game pieces are universal. All have the same shape and height and the opponent's pieces only differ in color. Thus it is easier for the robot to grab them and to not crash into other pieces. For the same reason, they can also be detected easier with a single webcam.

- Complexity

Checkers can be explained easier as there is only one possible movement and therefore even people or kids who are not familiar with checkers can try to play against the robot on fairs. On the engine side, less computation power is needed as checkers moves are calculated with simpler logic than chess moves.

### 2.3.3 Board and Game Pieces

#### Board

The board used for the project is an 8 by 8 black and white checkerboard printed out on paper. Edge length for the squares is 35mm resulting in an overall board dimension of 280mm by 280mm. It was necessary to keep it small so the robot can reach every field.

## Pieces

The pieces used for testing are black and white, have a diameter of 23mm and a height of 5mm.

### 2.3.4 How does it work?

English draughts is a game for 2 persons where every player starts with 12 pieces. It is played on an 8 by 8 checkerboard where the pieces are placed on the dark squares of the three rows closest to a player. Usually, the player with the dark pieces starts the game. Different movement and capturing rules apply based on the piece being either a man or a king. While man can only move diagonally forward onto free squares a king can also move diagonally backwards. A man will be promoted to a king when it reaches the row furthest away from its player. If a man meets an opponent's piece, diagonally, behind which a square is free, the player has to "jump" over the piece and therefore capture it, meaning that the opponent's piece has to be removed from the board. A player then needs to capture the opponent's pieces as long as the same piece has an opportunity to jump over a piece. While men can only capture forward, kings can also capture backwards. A player wins if the opponent has no pieces left to play with.

### 2.3.5 Why English Checkers

The decision on which version of checkers to use encompassed several factors. The English version was selected because of the following reasons:

- This version uses an 8x8 checkerboard and 24 pieces, which means that less computing power is needed compared to the international version with 40 pieces.
- English Checkers neither allows flying kings nor men being able to capture backwards, both resulting in less computing power needed to calculate moves for the robot.
- The 8x8 board is used for many games and thus other games can be adapted more easily.

## 2.4 Checkers engine and GUI (Honies)

The scope of the project included connecting the computer vision code to an already available checkers engine. Therefore several criteria were determined and available engines were researched.

### 2.4.1 Criteria for selecting the Engine

When selecting the engine these criteria had to be obeyed:

- Graphical Interface

The engine has a graphical interface for easier testing ability and to be able to monitor what the robot recognizes. Writing a graphical interface would exceed the scope of the project.

- Programming Language

As all the robot code and all the computer vision code is written in python, the engine code is written in python or has a python wrapper.

- Game Strength

Game Strength is an important factor when selecting the engine. The criterium for this project is that the strength can be selected and an advanced human player should be able to win at least against the easiest stage.

- Complexity

As the engine and GUI need to be integrated with the mechanical and computer vision implementation it is of importance that they are well written and easy to comprehend.

- License

As the code being written for the thesis is being released open source it is important that the Checkers code is also released open source and that all license rules are being followed.

#### 2.4.2 Chinook

Chinook is a checkers program developed by scientists at the university of Alberta. Its development started in 1989 with the goal to defeat the human World Checkers Champion. It did so in 1994 against Marion Tinsley, who is considered the best checkers player of all time. In 2007 Chinooks developers announced that checkers has been solved and that a perfect game results in a draw. It took several computers running almost two decades to solve the game. Only end game databases containing perfect information for 8 or fewer pieces are available storing 443,748,401,247 different positions. Those are very big though with a size of 5.6 GB. The code isn't available publicly.[18]

#### 2.4.3 Cake

Cake is an engine developed by Martin Fiertz and it's one of the strongest engines available. It's the successor of Cake Manchester which will be mentioned further down. He also developed CheckerBoard, a windows application specifically designed to work with different engines. Like most other engines it is written in C++.[19]

#### 2.4.4 KingsRow

KingsRow is another checkers engine, written by Ed Gilbert. It is, as Cake is, one of the strongest engines. What sets it apart from Cake is that there are versions for several checkers variants available, for example, Italian Checkers and international draughts. As for performance, it's the only one being able to benefit from multi-core systems.[20]

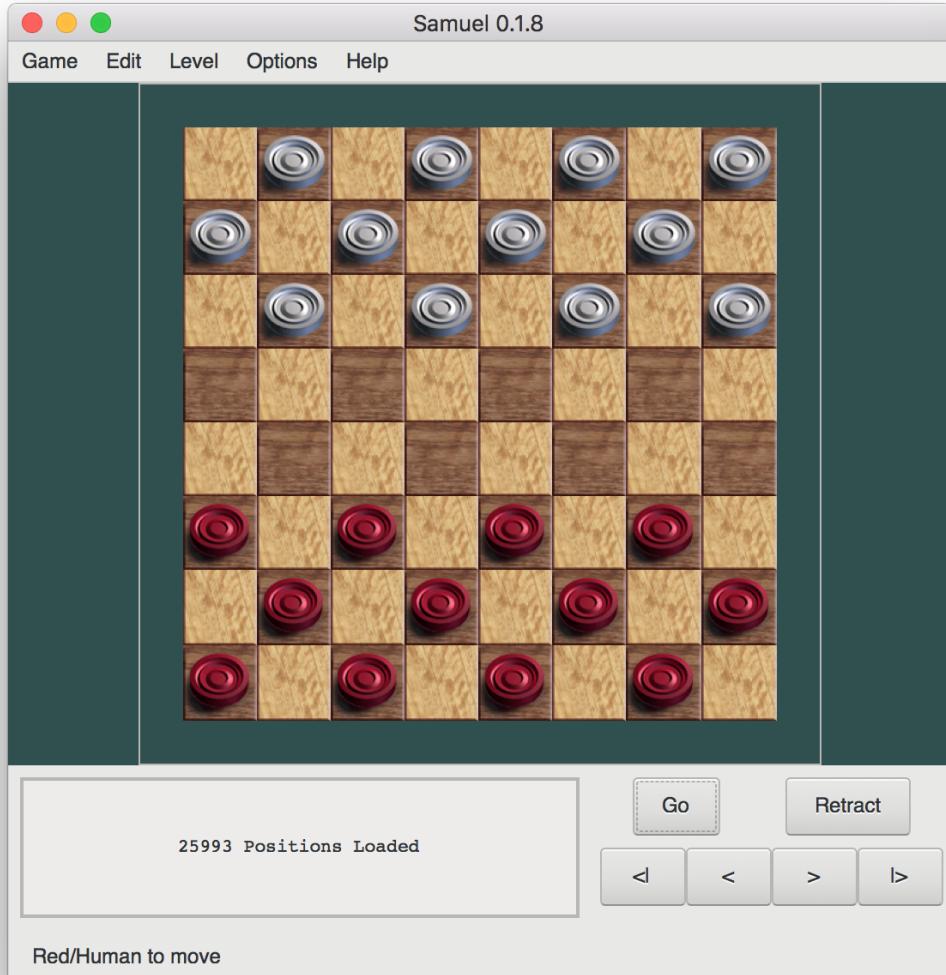
#### 2.4.5 GUI Checkers

GUI Checkers is an entire checkers engine and graphical user interface written and released in March 2006. It is entirely written in C++ and was originally designed for Windows XP. As tested by the author its game strength isn't the best. Tests show that it is beaten by more advanced engines, namely KingsRow and Cake Manchester. It does beat the SimpleCheckers engine almost every time, though. Its play strength can be set and it gets a lot weaker when set to Beginner. Against casual humans, it usually wins or draws.[21]

#### 2.4.6 Samuel

Samuel is an open source project written by John Cheetham in 2009. It is available under the GNU 3 License on GitHub. Samuel uses the engine from the GUI Checkers program

mentioned above, but entirely re-implements the GUI for Linux. The graphical user interface is written in python and uses the pyGTK and GTK+ framework. It also uses the same graphics as GUI Checkers. The code base is relatively small and easy to follow.[22]



**Figure 2.1:** Samuel on macOS

#### 2.4.7 Decision

Based on the above-described criteria to select the engine, the samuel engine was chosen to be used for the project. Especially it's low complex and well-structured python codebase and the graphical user interface did heavily contribute to the decision towards it.

#### 2.4.8 Others

Several other open source projects were considered to be used for the project. Those include Raven Checkers and Checkers-Overlord.

## 2.5 Blender (Weller)

### 2.5.1 General

Blender is an open source project aiming to provide a free computer graphics software alternative. It supports amongst other things modelling, animating, compositing and game development. The software is published under the GNU General Public License, allowing everybody to read and modify the source code as long as the changed version is, again, under the GNU General Public License.[23][24]

### 2.5.2 History

Development first started in 1995 by NeoGeo as an in-house animation tool, but after three years its development was shifted to NaN. At the beginning, Blender was designed to only be free for interactive content and for everything else a commercial version was available.[24]

This business plan lasted until 2002 when the support by investors was stopped and a new course for the company was needed. A successful crowdfunding campaign allowed the release of the new official open source version under the GNU General Public License. [24] To showcase the possibility of Blender and improve the development several Open Movies e.g. Elephants Dream and Big Buck Bunny. These movies not only made it possible to market Blender but also to find possible improvements or necessary features.[24]

### 2.5.3 Development

For the development of Blender C, C++ and Python were used. E.g. scripts inside Blender for user interface and object manipulation etc. are written in Python. These scripts can be changed by the user and it's also possible to add individual scripts capable of not only affecting the scene but also changing the UI and features.[25, p. 2]

Because the source code is available for everyone to download it's is also possible to circumvent the Python development of feature and to directly change the behaviour of Blender from the ground up.[15]

### 2.5.4 Alternatives

The biggest competitor of Blender is Maya. Originally planned as a combination of Alias Sketch! and The Advanced Visualizer by Wavefront Technologies, after both were bought by Silicon Graphics in 1995, it later was developed to be a next generation animation tool. [26, p. 315]

It is written in C++, C#, Python and Mayas own, on Sophia based, scripting language "MEL" (Maya Embedded Language). [26, p. 317]

Maya is used for television, film, game development, architecture and art, and has been honoured several times by the Academy of Motion Picture Arts and Sciences with the Award for Technical Achievement for scientific and technical achievement.[26, p. 315]

An exemplary company and project, which, partly, used Maya is Industrial Light & Magic for Star Wars [26, p. 315]

A major drawback of Maya is the licensing. As a student it is possible to access a free trial, but this trial does not include the possibility for commercial use. When using Maya for commercial projects a subscription based license needs to be acquired, costing about 238,- € / month.[27]

## 2.6 Inverse Kinematics (Weller)

### 2.6.1 General

Kinematics can be divided into two categories: Forward- and Inverse Kinematics. These concepts can be applied to any robot and the selection depends on the available input and the wanted output.[28]

For further explanations some basic terms and concepts that apply to both have to be described beforehand. In Kinematics a certain segment of a body is called the end effector. This end effector is the last piece of a chain of revolute or prismatic joints.[28]

Each segment and joint is described using the DH Convention, which is named after its creators Jacques Dennavit and Richard S. Hartenberg. This convention simplifies the description by reducing the number of needed variables. Instead of the normally needed six variables of a rotation matrix, only four link variables are needed:

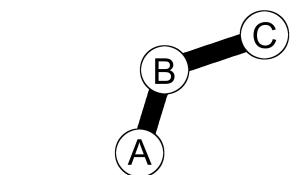
- $\alpha_i$  Link twist of link i. The angle from  $z_{i-1}$  to  $z_i$  measured about  $x_i$ .
- $a_i$  Link length of link i. Describes the distance from  $z_{i-1}$  to  $z_i$  along  $x_i$ .
- $d_i$  Link offset of link i ( prismatic variable ). The distance from  $x_{i-1}$  to  $x_i$  measured along  $z_{i-1}$ .
- $\theta_i$  Joint angle of joint i ( revolute variable ). The angle from  $x_{i-1}$  to  $x_i$  measured about  $z_{i-1}$ .

[28]

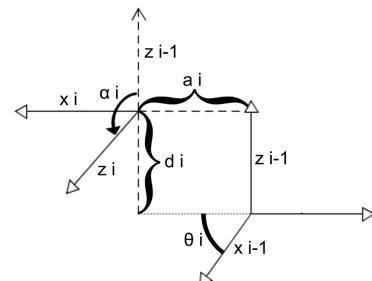
For this notation to work certain rules have to be followed:

- $z_{i-1}$  is the axis of actuation (revolution or translation) of joint i
- $x_i$  is set so it is perpendicular to and intersects  $z_{i-1}$
- Derive  $y_i$  from  $x_i$  and  $z_i$

[28]



(a) Chain with joints A, B and C



(b) DH notation of a chain [28]

**Figure 2.2: IK**

Using these parameters and rules the Forward- and Inverse-Kinematics can be calculated. They both deal with the same concepts, namely the joints and angles of a robots segments, but are used in different situations. Forward is used when the angles between the segments are known but the position of the end effector is unknown.[28]

The Inverse Kinematics, commonly referred to as IK, is used when the opposite situation applies. This means the joint angles are unknown. Instead, the end effector position is

given as input and through various calculation, explained later, the joint angles are calculated and can be applied to the robot segments thus moving the end effector to a desired position.[28]

Forward Kinematics will not be further discussed in this thesis, instead, the focus will lay on IK because it was used for the real - and simulated robot to calculate the movement and rotations necessary to move the hand (end effector) to a certain point in space i.e. a game piece or field.

Commonly in IK the construct, i.e. the robot's arm, is called a chain of links. This chain consists of multiple bodies called links ( each represented by the letter A ). These links are connected in a way that allows  $A_n$  to be translated or rotated in respect to  $A_{n-1}$ .[28]

Using the aforementioned notation the DH matrix can be created. This matrix is used e.g. in the Jacobian transpose, a method to calculate the necessary movement. [28]

Other algorithms are e.g. the DLS, SDLS and FABRIK[29, pp. 244 - 245]

While DLS and SDLS are based on the same concepts but try to improve them by decreasing jitter and random movement, FABRIK tries to solve the problem from a different perspective, namely adapting algorithms from rope simulations. [29, p. 244]

## 2.7 Robot (Weller)

### 2.7.1 General

As discussed before a humanoid robot is a human-like robot. This commonly also includes e.g. assembly line robots, that do not resemble humans, but have a design inspired by - or based on a feature of the human body. In an assembly robots case, this could be the human arm or elbow which is then adapted to a mechanical design.

This approach of humanoid robotic design has certain benefits but can also lead to problems. One of the biggest problems of humanoids is walking, because it encompasses precise measurement of the current movement and complex algorithms to calculate the future necessary movement to balance out or continue walking while also needing accurate motors with enough power and speed to react fast enough to possible loss of balance that can be encountered when for example walking on uneven ground. [6, p. 63]

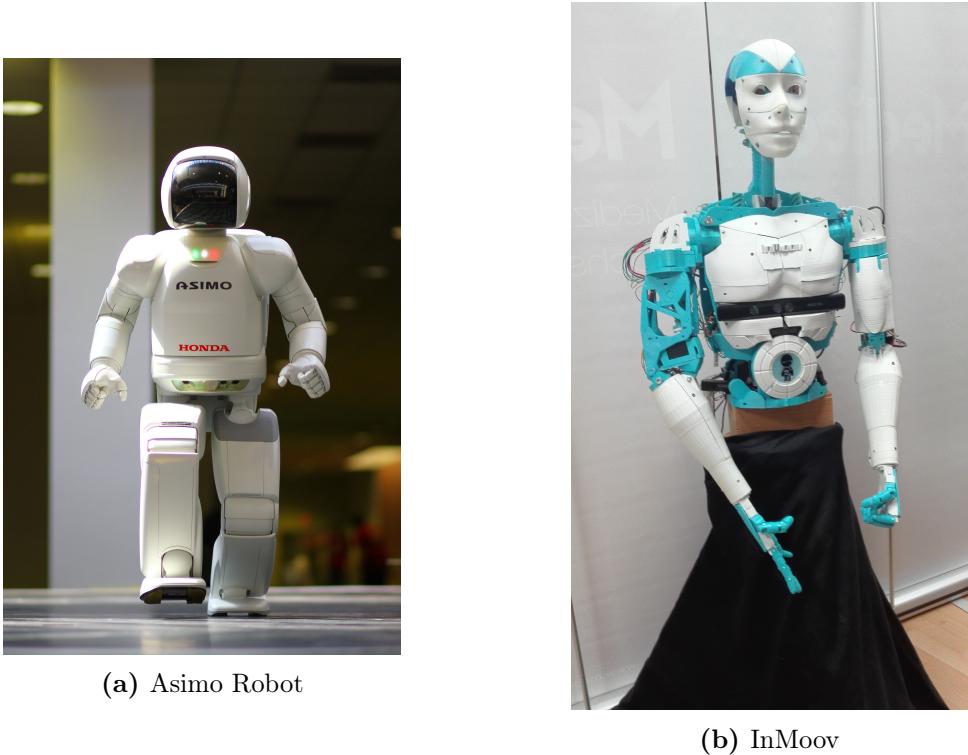
These complex algorithms do not only include the calculation of the movement and balancing but can also include computer vision to assess the environment and approximations for the best possible way to overcome obstacles. [7, p. 54]

For this thesis, a robot design that only has basic features such as an elbow wasn't enough, since the HRI with a human like robot should be as high as possible while simultaneously trying to circumvent the uncanny valley. In preparation for the thesis and the decision of the appropriate robot several humanoid robots were researched and evaluated, and some of them, each representing a certain group of humanoid robots, will be compared in the following paragraphs.

### 2.7.2 ASIMO

ASIMO is a high-end, consumer humanoid robot. Developed by Honda in 2011 he is one of the most advanced consumer robots available. Not only is he able to move through crowded places, climb and descend stairs, but he is also able to run with up to 7 km/h. [30] [31] He uses a camera, multiple gyroscope-, acceleration- and 6 axis force-sensors to compute his walk and possible obstacles.[30]

The big disadvantage of ASIMO is that it is not for sale and only for some people available for leasing. According to some sources e.g. Forbes, the leasing price is US \$ 150.000,- per



**Figure 2.3:** Humanoid robots

month. [32]

### 2.7.3 Poppy

On the other spectrum of consumer humanoid robots is Poppy. It's open source and has 3D printable parts. Poppy can either be, partly, printed at home (€ 6.459) or bought complete online (€ 9.039). He is 83 cm big, has 25 actuators and uses a camera for input.[33] [34] Compared to the Inmoov he lacks a humanoid face and can not move as many body parts, e.g. his hands are rigid and thus hinder him from playing draughts.

#### 2.7.4 Atlas

A big part of humanoid robots designs come from research facilities e.g. Boston Dynamics. Representing this group is Atlas, which was built and designed by Boston Dynamics for DARPA in 2013. He weighs 150 kg, is about 1.88 m big and has two vision systems.[35] A laser rangefinder (LIDAR) and stereo cameras, with which he took part in the DARPA challenge this includes, amongst other things, driving a vehicle, opening a door, using a tool to break through a concrete panel or operating valves. [36, pp. 5-6]

Atlas is neither open source nor 3D printable, can not be bought and hardly resembles a human, which made him unsuitable for this thesis.

### 2.7.5 Inmoov

Another section of humanoid robots are open source, community driven robots. Illustrating this area is Inmoov. He is mostly designed by French artist Gael Langevin. The project

started as the first open source prosthetic hand and continued to grow into the first open source 3D printed human-sized robot. [37]

Inmoov is open source and 3D printable, resembles the upper body of a human and can move most of his body parts.

### 2.7.6 Conclusion

A table of the before listed facts to compare the robots:

**Table 2.1: Robots**

Name	Price	Advantage	Disadvantage
ASIMO	US \$ 150.000 per month	Advanced sensors Advanced motors Object recognition Object avoidance	Price
Poppy	built at home: € 6.459 bought online: € 9.039	Average sensors Average motors	Price Rigid body parts
Atlas	Unknown	Advanced sensors Advanced motors	Not available for consumers
Inmoov	Maker Austria's built est. at 1500 € - 2000 €	Open Source 3D printable Community	Inaccurate movement Undocumented libraries

## 2.8 Virtualisation (Weller)

### 2.8.1 General

A virtualisation or simulation is a, often computer generated, representation of a process.[38]

In this thesis virtualisation stands for the computer generated representation of the humanoid robot. This model needs to be as accurate and realistic as possible, to proof that this concept can also be recreated with a real robot in a nonsimulated environment e.g. against human players.

Accuracy and realism do not only mean that the render, a computer generated graphic, should look realistic but also encompasses restrictions on movement i.e. rotation and translation of joints, and physical rules i.e. physical constraints. These conditions are necessary to allow an accurate assessment of the applicability of the algorithms and concepts on a real world robot and to further allow a detailed run down of possible limits and boundaries of the software. These boundaries can e.g. be that certain movements can not be done by the robot.

A list of the examined boundaries and possible solutions can be found in the implementation chapter.

For the virtualisation of the Inmoov two options were available. The *VirtualInmoov* and creating a new model with realistic proportions and restrictions. Because of the time that would be necessary to create a life-like model the *VirtualInmoov* was chosen, but slightly adapted.

### 2.8.2 VirtualInmoov

The **VirtualInmoov** (aka **VinMoov**) was designed by a user on the MRL forum named **GroG** and is available as a Blender model. It not only has the appearance of the robot but also the servos and an attached armature. [39]

He can be moved through MRL, the Blender API and the Blender user interface. All of these have certain benefits and drawbacks. MRL has already implemented the communication with the robot controlling simulated servos but does not run reliably and also lacks a proper documentation for the implemented functions. The Blender API, on the other hand, is capable of accessing the robot controlling armature and everything in the scene, including the game board, pieces and everything of the Blender user interface. Problems start arising when trying to be as realistic as possible since the armature on its own does not have the limits of the body or servos and they have to be added. From the beginning, the Blender user interface was out of question, since the robot can be only controlled via mouse and keyboard when using this method, which would add unnecessary complexity to the program.

In the end, the Blender API was chosen because it had the best integration. It did not need any additional startup or configuration since it is included in and starts with Blender. [40] The limits were added via joint rotation restrictions at the IK level, which will be further discussed in the implementation chapter in the section “Simulated Robot”.

# Chapter 3

## Computer Vision Algorithms (Honies)

Various computer vision algorithms are used for the project. To understand how the robot detects the different states of the game, which will be described in the next chapter, it is necessary to describe which algorithms were used or looked into. This chapter will compare them to alternatives and show examples of processed images to point out their effectiveness.

### 3.1 Grayscale Conversion

As the human's pieces and the board are either black or white, no other color is of any significance. Also, most OpenCV algorithms used for the thesis will only accept grayscale or binary images and those which accept coloured ones will also run faster as there is only one value to process instead of 3. Every Image taken is therefore converted into a grayscale image. It can be done by passing the original image and the format it should be converted to, to a simple OpenCV function:

```
1 grayscaleImg = cv2.cvtColor(rgbImg, cv2.COLOR_BGR2GRAY)
```

### 3.2 Downscaling

Downscaling is another very popular method to decrease the computation power needed by a computer vision application. As there aren't a lot of applications which need pixel perfect precision, downscaling the image before passing it to other more performance heavy algorithms is a way to let a program run more performant.

### 3.3 Digital Image Smoothing

Digital Image is needed to filter out noise. Noise is problematic as the webcam isn't of great quality and lightning isn't always the same, so the webcam automatically sets a high ISO which results in lots of noise. It is done by the convolution of 2 arrays. One 2D array representing the grayscale values of an image and another 2D array representing a kernel with weighted values. Convolution works by shifting the kernel over every pixel of the image, multiplying every intensity value of the neighbouring pixels with the overlaid value of the kernel and then building the sum of all those values. The sum then needs to be divided by the sum of all weights in the kernel to get the new value of the pixel.[41]

### 3.3.1 Mean Filter

A mean filter is the most basic version of an image filter. Its kernel is evenly distributed, resulting in every pixel having the average value of its neighbours pixels in the x, y size of the kernel. A kernel for a mean filter with an x and y of 5 is shown in 3.1.

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (3.1)$$

#### Code

```
1 meanFilteredImage = cv2.blur(gray, (5,5))
```

### 3.3.2 Gaussian Blur Filter

The gaussian blur filter is using a discrete approximation of a gaussian 2-D distribution for the kernel. A kernel for a standard deviation of 1 and a mean of 0 is shown in 3.2.

$$K = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (3.2)$$

#### Advantages

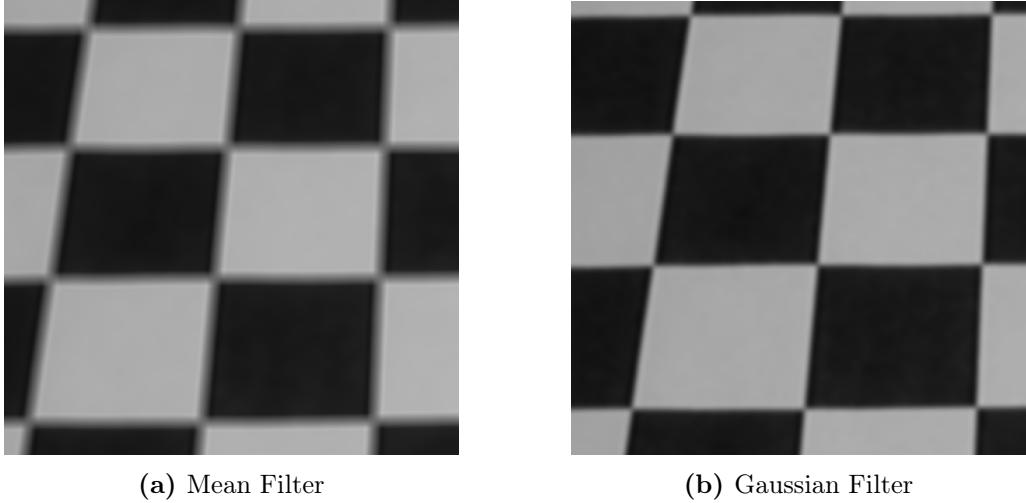
As the distribution of the kernel is weighted towards the pixel which is being calculated, noise will be filtered out as good as with the mean filter but edges will stay sharper. Figure 3.1 shows both filters with a kernel size of 11x11.

#### Code

```
1 gaussianFilteredImage = cv2.GaussianBlur(gray, (5, 5), 0)
```

## 3.4 Thresholding

Thresholding is used to separate the objects in an image from the background. For the images taken in this project, only grayscale images are passed to a thresholding function and only binary images are expected to be returned. Binary images are images whose pixels have only two possible intensity values. They are usually displayed in black and white, white being the object, numerically 0 for black and 1 or 255 for white. This simple form of thresholding works by comparing every pixel's intensity value to a threshold value and if it is higher it will get a value of 1 or else 0. Dozens of different methods exist to determine that threshold value and 2 which were tested for the project will be described.[41]



**Figure 3.1:** Comparison of mean and gaussian filter

### 3.4.1 Global Thresholding

Global Thresholding means setting the threshold value to one constant intensity value which every pixel of the image will be compared to.

$$f(r) = \begin{cases} 0, & \text{if } r \leq T \\ 1, & \text{if } r > T \end{cases} \quad (3.3)$$

It is the least complex and least resource intensive of all thresholding methods and therefore comes with a few disadvantages. The biggest being a bad performance if the lightning isn't even over the whole image which is shown in Figure 3.2.[41] As the robot needs to work well in different lighting conditions when being presented at fairs, another method had to be tested.

#### Code

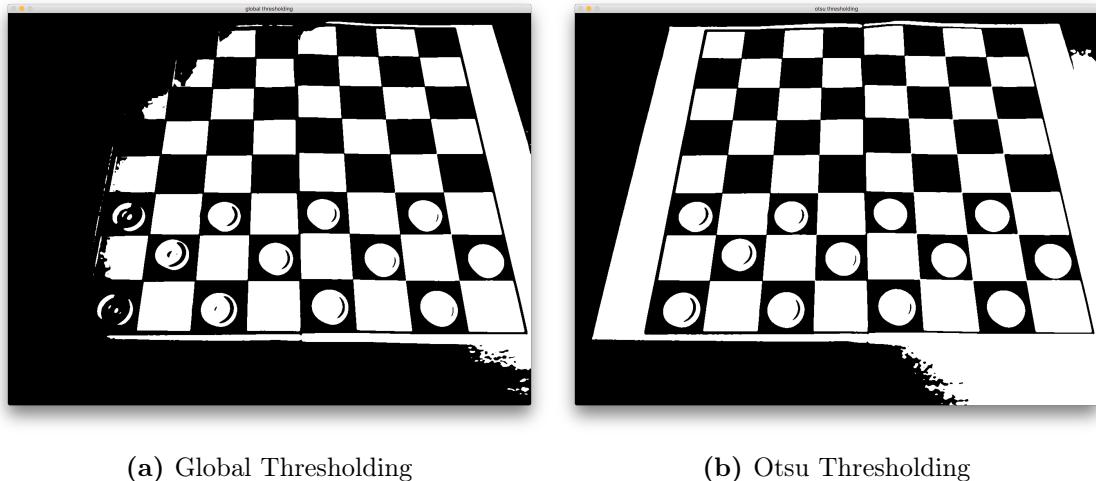
```
1 ret,thresholdImg = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

### 3.4.2 Otsu-Thresholding

While in global thresholding the optimal thresholding value can only be found by trial and error, otsu-thresholding will generate the optimal value for bimodal images. Bimodal images are images whose histograms have two peaks, i.e. images with a lot of foreground pixels with similar intensity values and a lot of background pixels with again similar intensity values. It will automatically find an optimal value between those two peaks. All of the images being processed for the thesis include mostly the black and white checkerboard, therefor otsu-thresholding works excellent.[42]

#### Code

```
1 ret, thresholdImg = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```



**Figure 3.2:** Comparison of otsu and global thresholding at difficult lightning

After calling the function the `ret` variable will store the calculated threshold value and `thresholdImg` the generated image with this threshold value.

### 3.5 Histogram Equalization

Performing histogram equalization onto an image is a way to increase its contrast. By using the cumulative distribution function of the histogram we can remap the original distribution to an equalised distribution.[17, pp. 328-332] The general histogram equalization formula is[43]:

$$h(v) = \text{round} \left( \frac{cdf(v) - cdf_{min}}{(M \times N) - 1} \times (L - 1) \right) \quad (3.4)$$

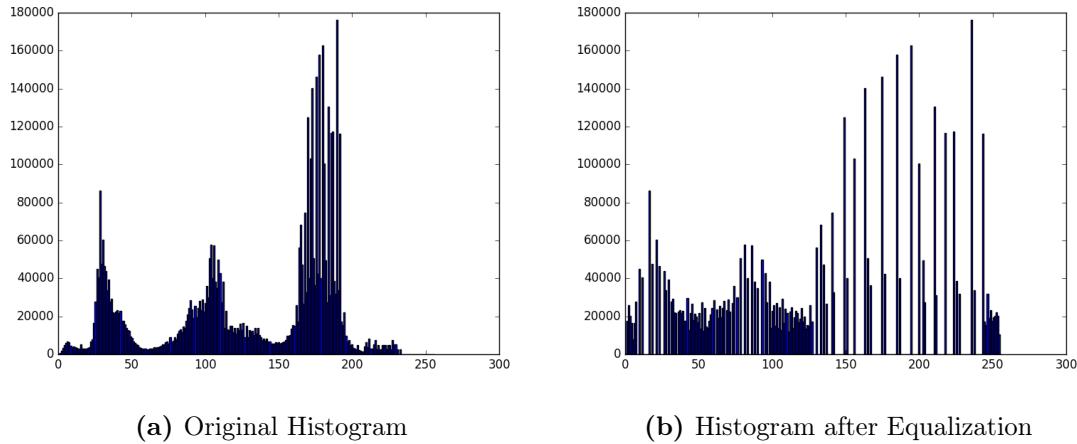
$h(v)$  is the new intensity value,  $v$  the old one.  $cdf(v)$  describes the cumulative distribution function at  $v$  and  $cdf_{min}$  the minimum  $cdf > 0$ .  $M$  is the width,  $N$  the height and  $L$  the highest possible intensity value. An image with low contrast is an image with most of its pixels being in a relatively small range of available intensities. Histogram equalization "stretches" the distribution over the whole available range resulting in an image with a lot more contrast.

### 3.6 Morphological Transformations

For the project, it was only looked into one of the 2 basic operators of mathematical morphology, erosion, the other one being dilation. The operators usually take a  $3 \times 3$  structuring element and the image as an input. They work similar to already described image filters by shifting the structuring element over the image with the pixel to be calculated being in the center of the structuring element. Instead of building the average, all the pixels of a binary image being overlaid by the structuring element will be compared to the value at its position in the structural and if one or more are false, the center pixel will get a value of 0.[41]

#### 3.6.1 Erosion

Erosion, as used in the project for binary images, uses a structuring element with just 1's.



**Figure 3.3:** Demonstration of Histogram Equalization applied to image of a checkerboard

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3.5)$$

Therefore white foreground areas will get smaller. This has a positive effect on the blob detection which is later ran in the process. Erosion can also work with grayscale images. Then the structuring isn't compared to the image but the center pixel will get the minimum value of all pixel being overlaid by the structuring element[44]. This smoothens out very bright sections like reflections on the stone and decreases the size of bright foreground areas, namely the human's game pieces, making it easier for them to be detected by the blob detector.

### Code

```
1 kernel = np.ones((6,6),np.uint8)
2 erImg = cv2.erode(gray, kernel, iterations = 2)
```

The first line uses the numpy library to generate a structuring element with a size of 6 by 6. The second line then passes a grayscale, the structuring element and an iteration count to the OpenCV function. The iterations parameter describes how often the image should be eroded. A new grayscale image is returned.

### 3.7 Corner Detection

Both the Harris and the Shi-Tomasi Corner Detector use the same basic concepts to detect corners. While an edge is a point in the image with a strong derivative of its neighbourhood intensities, a corner needs a strong derivation in at least two orthogonal directions. To prevent detecting linear gradients, always the second derivative is used. To calculate how strong the derivatives at a point are, the autocorrelation matrix over a small window around every point is used. If the autocorrelation matrix has two large eigenvalues at a certain point, the point is considered a corner.[45, pp. 317-318]

### 3.7.1 Harris Corner Detector

3.6 shows how Harris calculates a value for each point which is then compared to a predefined threshold value, to determine if a corner is given or not.  $\lambda_1, \lambda_2$  are the eigenvalues, k is a weighting coefficient which can be set by the developer.[45, p. 318][46]

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (3.6)$$

### 3.7.2 Shi-Tomasi Corner Detector

Shi and Tomasi proposed in 1994 that good corners can already be found by comparing the smaller of the two eigenvalues with a predetermined threshold.[45, p. 318][46]

$$R = \min(\lambda_1, \lambda_2) \quad (3.7)$$

### 3.7.3 Good Features to Track Function

OpenCV provides a highly advanced function for corner detection: `cv2.goodFeaturesToTrack` `cv2.goodFeaturesToTrack`. To control the behaviour of the method several parameters can be set with the most important being:

- maxCorners
- qualityLevel
- minDistance
- useHarrisDetector

If the `useHarrisDetector` parameter is not set to true, the function will use the Shi-Tomasi corner detector. The function returns an array of tuples with the coordinates of the corners.

## 3.8 Blob Detection

Blob Detection is a way to detect regions of interest. Many different algorithms exist for blob detection but for the project just the one implemented by the OpenCV class `cv2.SimpleBlobDetector` was looked into. The algorithm takes a grayscale image and generates several thresholded images based on the parameters minimum threshold, maximum threshold and threshold step. Centers of connected components, called contours, are then extracted. Those are called candidate blob centers. Candidate blob centers with a similar position and from similar thresholded images are grouped together. A radius and center are computed for each group and a new object is being generated. Those objects are the blobs. The `cv2.SimpleBlobDetector` class also provides built-in filtering features. The detected blobs can be filtered by size, color, circularity, inertia ratio and their convexity.[17, pp. 534-536]

### Code

```

1 params = cv2.SimpleBlobDetector_Params()
2 params.filterByColor = True
3 params.blobColor = 255
4
5 # Change thresholds
6 params.minThreshold = 0;
7 params.maxThreshold = 255;
8
9 # Filter by Area.

```

```
10 params.filterByArea = True
11 params.minArea = 100
12
13 # Filter by Circularity
14 params.filterByCircularity = True
15 params.minCircularity = 0.8
16
17 # Filter by Convexity
18 params.filterByConvexity = False
19 params.minConvexity = 0.7
20
21 detector = cv2.SimpleBlobDetector_create(params)
22 # Detect blobs.
23 keypoints = detector.detect(im)
```

The code shows how to use the cv::SimpleBlobDetector class in OpenCV. First a parameter object has to be created, then every available filter has to be activated to work. Also, parameters for each filter have to be set. A SimpleBlobDetector Object can then be initialised with those parameters and its detect method can be called. Returned keypoints store the coordinates of the center of the blob and its radius.

## Chapter 4

# Computer Vision Implementation (Honies)

Computer Vision is used widely in the project. While it is mandatory to implement basic functions like detecting the board, game pieces on the board and moves, it is also needed to implement simple functions for controlling the game flow to prevent the human player from having to use nonnatural behaviour while playing against the robot. Simple game flow control functions include detecting the game start, a game pause and a restart. Every part of the computer vision module is programmed in python and uses the OpenCV python framework.

### 4.1 Game Flow

Figure 4.1 shows the general process of the whole implementation. QR Codes are used to communicate to the robot that the game has to be started or has to be stopped.

#### Game Start

After starting the program and setting up the board a QR Code of the encoded string "start" has to be shown to the robot. The machine will then detect the board and all pieces of its opponent and will assume its own game pieces are set up correctly. It will then start with the first move.

#### Game Abort

If a QR Code of an encoded "abort" string is shown to the robot while the player has the right to move, the robot will detect it and stop the game. The engine will be reset and the robot will move to its initial position. The program then awaits a new start signal as described above.

#### Game End

If the game ends the robot will reset the engine move to its initial position and wait for a new start signal.

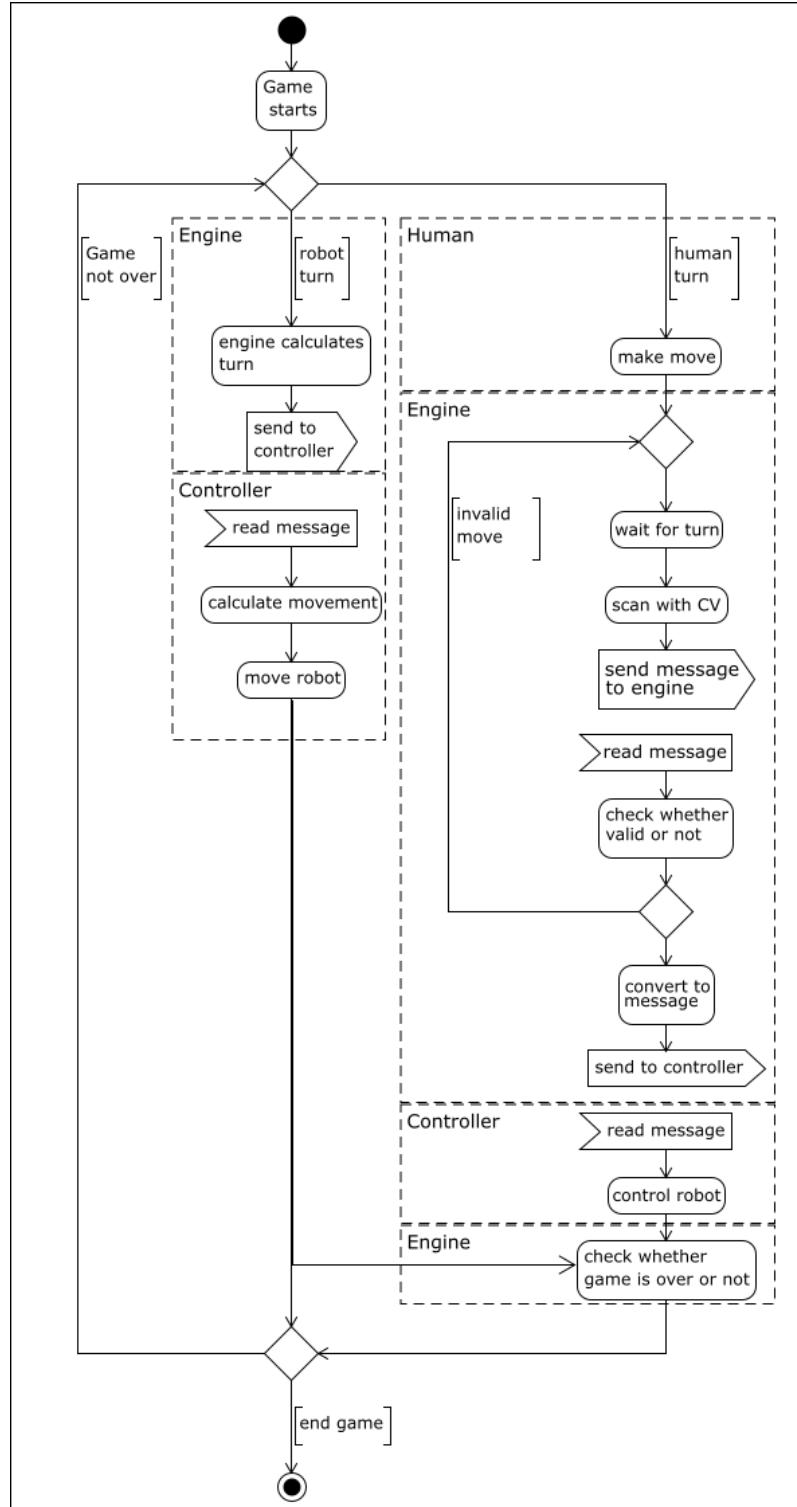


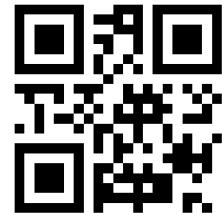
Figure 4.1: Game flow

## 4.2 Detecting the Board - Contours

For the first implementation of board detection, the OpenCV concept of contours is used. It is tried to detect all squares of the board and sort them according to their position.



(a) Start QR Code



(b) Abort QR Code

**Figure 4.2:** QR Codes for Program Flow

### 4.2.1 Algorithm

First, the image is converted to grayscale and then a gaussian blur filter with a kernel size of 9 is applied. Then a for loop will iterate through threshold values with a step size of 26 and the image will be thresholded with that value every time. `cv2.findContours` is run to identify contours. The algorithm will then loop through the detected contours and filter them according to the characteristics of squares:

- Convex
- 4 edges
- Minimum and maximum size
- 4 almost 90° angles between their edges (if looked on the board from above)

To reduce noise and disturbances at edges the contour is converted to an approximated polygon of itself. The convexity of a contour can be checked by using the OpenCV function `cv2.isContourConvex`, the count of edges by using `len` and minimum and maximum by using `cv2.ContourArea`. 800 and 5000 have shown to be good values for min and max size. To check for the angles, the max cosine of all connected edges of the contour is calculated. 0.4 has been tested out to be a good threshold value here.

### 4.2.2 Filtering the Fields

Detecting the squares is not enough as there will be about 10 times more squares detected than there should be. The problem is that several squares are detected on one field. Fortunately the unnecessary squares can be filtered out fairly easy by comparing their midpoints and removing the superfluous squares from the vector. This then results in a vector with 32 squares which can be identified by their x, y coordinates and their width.

### 4.2.3 Arranging the Squares to their Board Position

After detecting all fields they are sorted into a 2-dimensional array with the indexes of those representing the x and y position of the field on an 8x8 board.

### 4.2.4 Testing

Testing the Board Detection implemented with Contours made clear that this was not the right approach. Different problems occurred which made it necessary to develop another, more stable, version of the board detection.

## Lightning

The implementation just worked with perfect Lightning. Lightning had to not come from above but from the side of the board. Also too bright lightning resulted in too much noise and not all squares could be detected. As fairs almost always have bright lightning from above it is not possible to use the implementation for the purpose of the project.

## Perspective

As the board's fields only represent squares when looked onto from above, the implementation made it necessary to mount the camera almost directly over the board. While testing the robot's ability to reach all areas of the board, it was found out that the board needed to be placed a little bit off the robot, resulting in a too low angle of view for the implementation to work.

## Perspective

The implementation is everything else than performant and as it is not detecting all fields reliably it needs several seconds for initializing the board. The next section will describe a more performant and reliable approach.

## 4.3 Detecting the Board - Corner Detection

The last section describes why a new approach was needed. By researching different methods, it was decided to try out corner detection next. 2 corner detection algorithms were tested and are described in the methodology section:

- Harris Corner Detector
- Shi-Tomasi Corner Detector

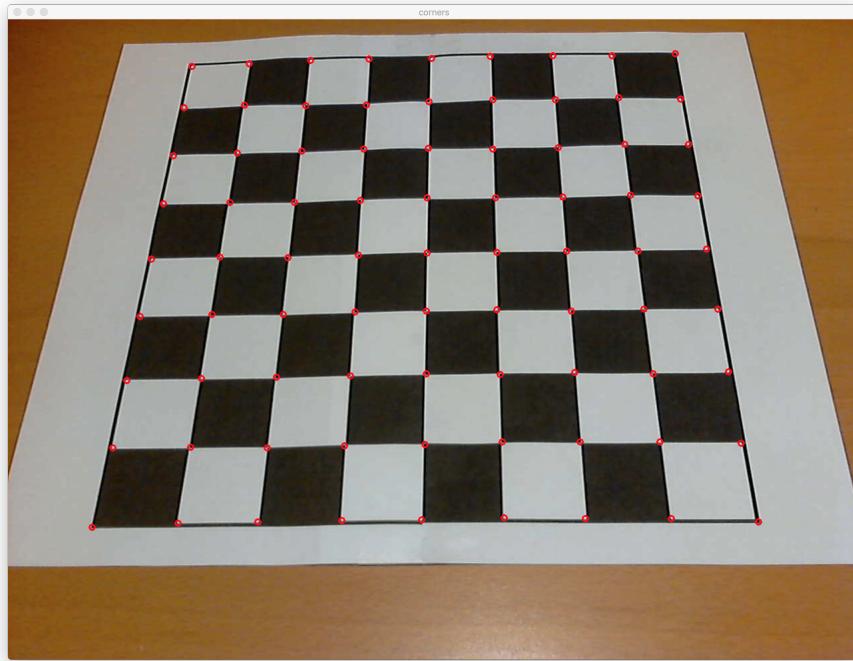
While both perform well, tests have shown that the Shi-Tomasi algorithm runs just a little bit better.

### 4.3.1 Detection

To detect the fields OpenCV's `cv2.goodFeaturesToTrack` method is used. How it works in detail is already described in the methodology section, for the future sections, it's just important to know that it returns an array of tuples of the x, y coordinates of the detected corners. While preprocessing the image was tested, it was found out that the function actually works best if a raw picture is provided. Therefore the image was just converted to grayscale and then the function was called. Figure 4.3 shows what corners would be detected on a sample image of the project.

### 4.3.2 Mapping the Corners

The `cv::goodFeaturesToTrack` function returns the corners as an unordered array of tuples with the representation (x-coordinate, y-coordinate). As the blob detection of the pieces will return coordinates we need to know the x, y position of every field and its width and height. Therefore it is needed to map the returned corners, so these attributes can be extracted. The code section shows how to first sort the corners into which row of fields they belong. It then sorts every row, so a complete map of the corners is generated with their indices representing their x, y position on the board. After that it generates a 2 dimensional



**Figure 4.3:** Returned corners from Good Features to Track Function circled in red

array of tuples for the fields with the representation (x-coordinate, y-coordinate, width, height).

```

1 fields = [] []
2 #sort detected corners after y-position
3 cornerarr = sorted(cornerarr, key=operator.itemgetter(1))
4 #matrix for sorted corners
5 cornerm = [[0 for x in range(9)] for y in range(9)]
6 for i in range(9):
7     for ind in range(9):
8         cornerm[i][ind] = cornerarr[i*9+ind]
9 #sort corners in a row after their x-position
10 #so indices in matrix are equivalent to their indices on the board
11 for row in range(9):
12     cornerm[row] = sorted(cornerm[row], key=operator.itemgetter(0))
13
14 for i in range(8):
15     fields.append([]);
16     for ind in range(8):
17         #calculate height and width of field
18         x = cornerm[i][ind][0]
19         y = cornerm[i][ind][1]
20         w = cornerm[i][ind + 1][0] - x
21         h = cornerm[i+1][ind][1] - y
22         #add field to field matrix with indices equivalent to real world board
23         fields[i].append([x,y,w,h])

```

### 4.3.3 Testing

#### Lightning

The new algorithm works way better under difficult lightning than the first implementation. While in the early stages of testing a cardboard with a shiny print was used, the final version uses a matt paper print. That's because the algorithm sometimes detected corners at reflections of light, which can't happen with a matt surface.

#### Perspective

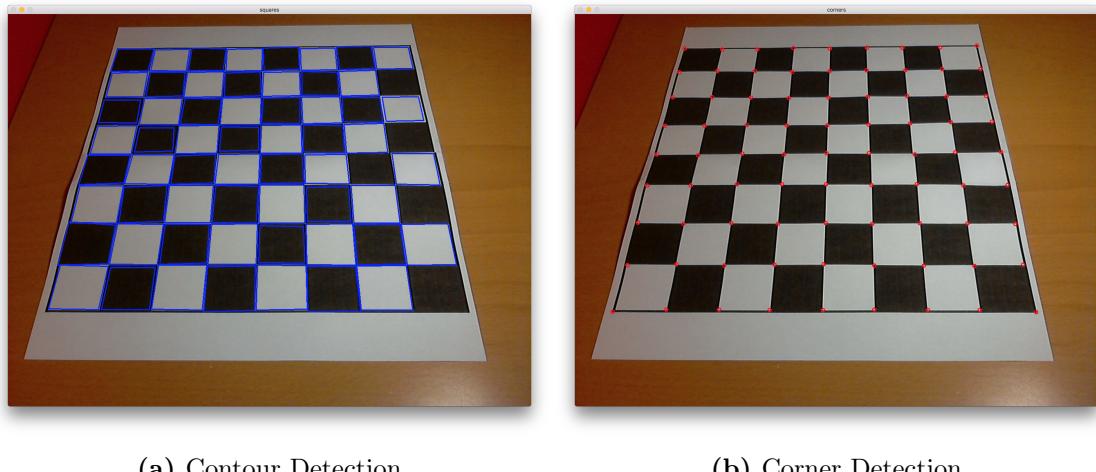
Perspective isn't an issue anymore with the new implementation. Corners are corners, no matter if they are looked onto from above or at a 45°angle.

#### Performance

It outperforms the old implementation by far. While the old one took several seconds under difficult conditions, the new one often only needs to analyse a single frame.

## 4.4 Comparison of Board Detection Methods

Figure 4.4 shows a comparison of both Methods under the same conditions. While are corners are getting detected correctly, the contour detection is having problems especially on the right edge of the board. Only 59 of 64 squares were detected.

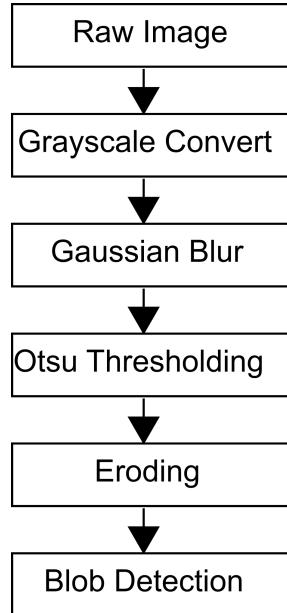


**Figure 4.4:** Comparison of Board Detection Methods

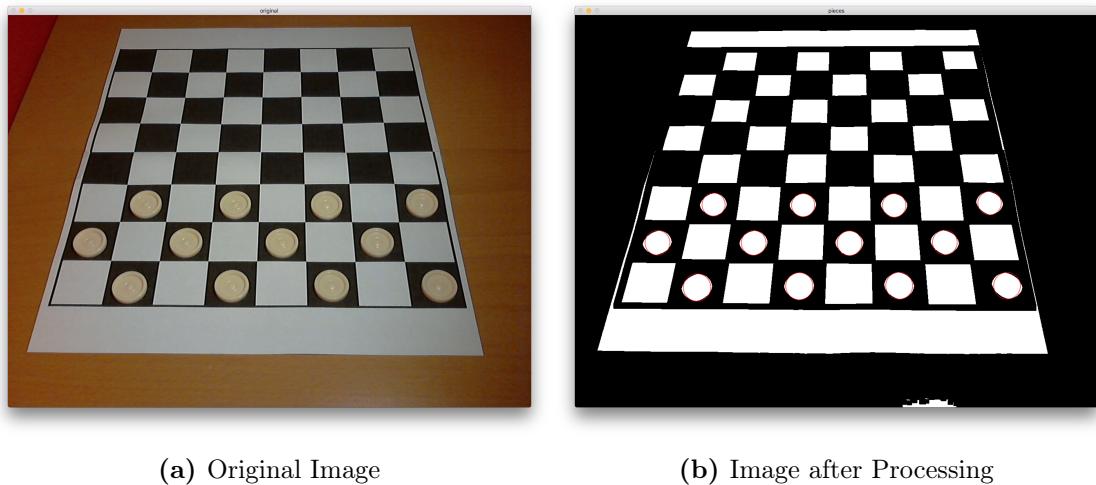
## 4.5 Detecting Game Pieces

Detecting the human's game pieces is implemented by using OpenCV's built in "Simple Blob Detector". Although the Blob Detector accepts a raw image from the camera and detects pieces sometimes, the image needs to be heavily preprocessed for the Detector to work consistently. After testing various combinations of algorithms, the combination shown in figure[insert number] has shown to be the most consistent one. Figure 4.6 compares a

raw image with its processed one after all algorithms were applied and blob detection was run. Detected blobs are circled in red.



**Figure 4.5:** Processing of the image for piece detection



**Figure 4.6:** Before / After Processing and Blob Detection

#### 4.5.1 Process

The following will describe why those algorithms were chosen, what parameters were used for them and why they were called in that order. The goal of preprocessing the image for Blob Detection is to remove all unnecessary colours, surfaces and disturbances to end up with white circles clearly separated from the board, the human's game pieces. First, the image is converted to grayscale to remove the superfluous colours and disturbing noise

is removed by applying a gaussian blur filter with a kernel size of 5 by 5. Then Otsu-Thresholding is applied to get a binary image because tests demonstrated that this has an enormous impact on the later run Blob Detection, where lots of pieces were not detected due to light reflections. Thresholding to a binary image removes any light reflection as there will only be white and black pixels in the output image. After those steps, Blob Detection already ran well, but pieces which are laying almost at the edge of the field weren't consistently detected. Therefore the image is eroded with a kernel size of 6 by 6 and an iteration of 2. This will decrease the size of the pieces and increase the black surface of its field, resulting in a more consistent Detection. After all that preprocessing just a few parameters are set for the cv2.SimpleBlobDetector class, namely blobColor to 255 for white blobs, minArea to 50, minCircularity to 0.8 to only detect white pieces and not squares, and minCovexity to 0.87 because the pieces usually don't have any notches.

#### 4.5.2 Finding the x, y Position

As the Blob Detector returns keypoints with x, y coordinates and a radius it is needed to find out on which field the piece lays. How the attributes (coordinates, width, height) of the fields are stored is already explained above. To detect on which square a piece is located, the function simply goes over the array of fields and compares it with the detected coordinates. For maximum consistency and as checkers isn't a fast sport, 5 images will be taken and only pieces which are detected at least 3 times out of 5 will be sent to the engine. How the engine works, how detected pieces are transferred to it and how a move is detected will be explained in the following chapter.

# Chapter 5

## Connecting the engine (Honies)

As already described in the methodology section, it was decided to use Samuel an open source checkers program, written by John Cheetham, which is based on the also open-source Gui Checkers program, written by Jon Kreuzer, for the project. John Cheetham basically took the C++ engine code of Gui Checkers program and wrote a new GUI in python by using the GUI Framework pyGTK. As this is not an engine with a fully documented API, it was needed to explore the code so instead of a human input, moves can be generated by the computer vision code and calculated moves for the robot can be extracted in a convenient format.

### 5.1 Connecting Computer Vision and Samuel

Different methods are available for process communication. While pipes and sockets can be used to communicate between processes, it was decided to use the most simplistic form of communication. Merging both parts to start as one process so the computer vision has immediate access to all attributes and settings of the game. The computer vision code for detecting the board and pieces is organised as a class which starts by waiting for the start signal (QR Code), then detecting the board and afterwards going into a loop, where it will always monitor if a new move has been done by the human. The main thread of samuel creates an object of the class Game where everything that has nothing to do with the GUI is handled. After this creation, a new object of the vision class is created and the game object is passed as a parameter. Then the main method of the vision class is started as a new thread. The main method of samuel, therefore, looks like this:

```
1 def run():
2     g = Game()  #create game object
3     v = Vision(g)#create vision object
4     thread.start_new_thread(v.main, () ) #start main method of vision class in new
5         thread
6
7     #start GUI
8     Gdk.threads_init()
9     GObject.threads_init()
10    Gtk.main()
11
12    return 0
```

## 5.2 Board Representation

For performance advantages, especially move calculating, the Gui Checkers engine doesn't use a two-dimensional array to store positions, but a one-dimensional array with the values shown in the table being the indices in the array.

	37	38	39	40
32	33	34	25	
	28	29	30	31
23	24	25	26	
	19	20	21	22
14	15	16	17	
	10	11	12	13
5	6	7	8	

The status of a field is stored using integers:

- 0 if the field is not occupied
  - 1 for a red/black piece
  - 2 for a white piece
  - 5 for a red/black king
  - 6 for a white king
  - 8 if the index isn't a field

Therefore the initial starting position of the pieces looks like this:

### 5.3 Detecting a Move

It is only possible to pass a move to the engine when having the start and end point of the move. To determine if the human made a move the implementation needs to compare the new board position with the current engine position. To do that, a new board position is generated by copying the engine position and setting all entries of the human's color to 0. The positions of the detected pieces are converted to the engine's board representation and a 0 at that position is overwritten by an integer of the human's color. Then the arrays will be compared for an occupied field by the human on the current engine position and an unoccupied field on the new board position. This is the starting position of the move. If such a field isn't found, the process will go back to taking pictures again. After the starting position is found the end position will be found by comparing the arrays for an occupied field which was unoccupied in the old position. The following engine method is called to check if the move is legal if so it will return a new board position following the executed move.

```
1 board_position = engine.hmove(self.src, self.dst)
```

The engine also sets an attribute of samuel called legalmove, which can have 3 values:

- 0 - the detected move is illegal
  - 2000 - the detected move is legal, but the human can move again as he "jumped" one of the robot's pieces
  - everything else - the detected move is legal and it's now the robot's turn

## 5.4 Generating a Move

After submitting a legal move to samuel, it will be displayed and the process of calculating the next move for the robot is started by samuel. For the GUI to be still usable, this is started in a new thread. To get the next move the following function is called.

```
1 board_position = engine.cmove(str(self.side_to_move))
```

However as this code shows, the C++ engine just returns a new board position instead of a source and a destination. To avoid changing the engine code, simply the same algorithm as used for detecting human moves is run with the board position before the calculation and the new one. However it is also necessary to detect which game pieces are "jumped" by the robot's move and therefore need to be removed from the board. To do so a new algorithm was developed which compares the old board representation with the newly generated one and detects human pieces which were removed. It then sends one command per removed stone to the robot to remove it.

## 5.5 Passing a Move from the Engine to the Robot (Weller)

### 5.5.1 General

The Arduino is significantly less powerful than a normal laptop not only in the sense of computation power but also in the sense of space available.

Since `samuel` is written for Windows and has requirements above the capabilities of an Arduino the engine was installed and used on the laptop that is connected to the robot. Because of this it is necessary to use methods for passing the calculated moves to the Arduino or Blender.

For this a syntax was developed to describe the maneuvers in a simple and easy to parse way.

As mentioned before the moves are either first captured by CV and then parsed into the engine or calculated in the engine. These moves are then converted to a character sequence with a predefined syntax and sent to the Arduino or Blender.

It has to be noted that, after being controlled by the engine, the commands are never checked for validity. The reason for this is, that it is expected of the engine to register illegal moves and prohibit them from being sent.

### 5.5.2 Syntax

The syntax for the messages consists of multiple variable separated by commas “,”.

At the beginning is a character describing the player i.e. `h` for human player or `r` for the robot.

Following this is the character determining the type of move. Possibilities are a normal move i.e. moving a stone from one field to a different one, removing a stone and adding a stone.

The message is concluded by the x- and y- coordinates of the start- and end field also separated by commas.

```
1 move = '[player],[type of move],[start row],[start column],[end row],[end column]'

[player],[type of move],[start row],[start column],[end row],[end column]
```

Example lines:

```
1 h,m,3,1,4,2
2 r,a,0,0,5,3
```

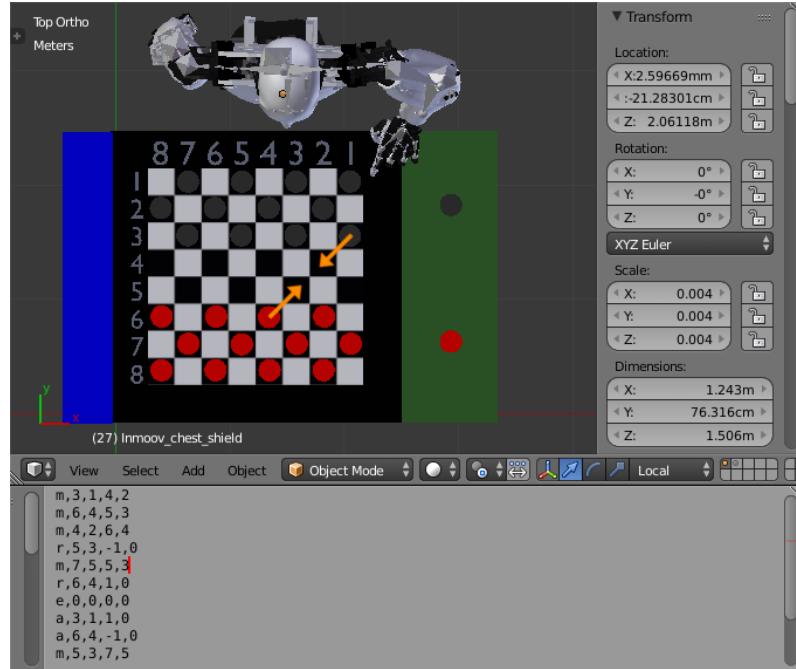


Figure 5.1: Visualization of example moves

In special cases, i.e. adding and removing stones, some fields are discarded since they do not affect the maneuver. For example when removing a stone there is no end row or column since the stone will be placed on a predetermined player specific pile. The opposite, i.e. no starting row and column, is applicable when adding stones since they are taken from the team pile.

Remove:

```
[player],r,[start row],[start column],0,0
```

Add:

```
[player],a,0,0,[end row],[end column]
```

### 5.5.3 Basic Implementation via a txt File

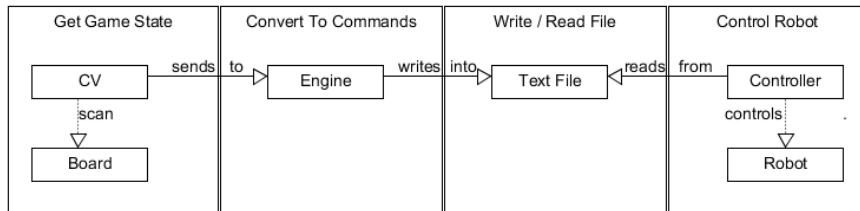


Figure 5.2: Text file implementation

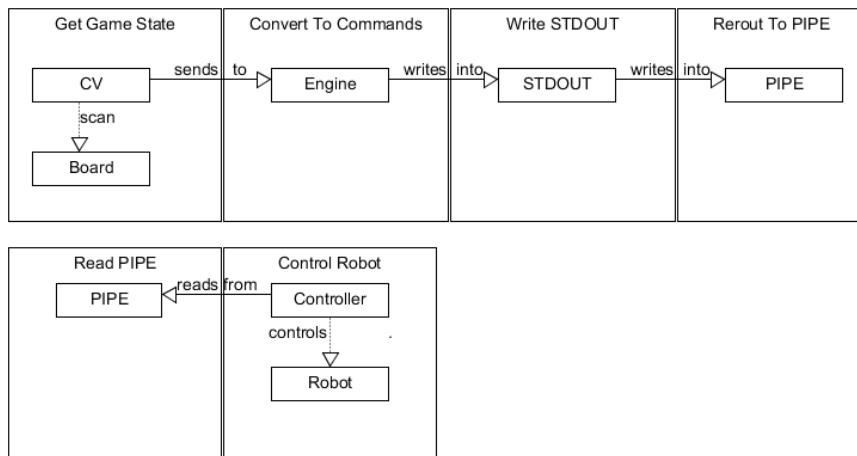
The first version of the communication model for passing moves from the engine to the controller uses a simple .txt file. This file contains every move passed by the engine and gets expanded with every new command by the engine. The controller then repeatedly checks for changes in the file and if any are found he reads the added commands from it.

Through this file the robot can not only be told to move -, remove - and add stones but also to end a turn, meaning that he moves his hand to the default position and waits for file changes.

The changes are parsed line by line, split by commas and then saved into separate variables. These variables are then used to call the according functions with the parameters. This approach was tested with the Blender simulated robot. Instead of direct input from the engine a testing file already containing several moves was used. The python script read it and executed the commands successfully but nonetheless different functions were implemented and tested, and used in the final product.

Several factors lead to this decision. Big factors against this approach were the overhead and inefficiency problems. Opening a file, writing to it, closing it, opening it again from a different script and then reading it adds significant overhead to a simple transportation.

#### 5.5.4 Second Implementation via PIPE and Subprocess



**Figure 5.3:** Subprocess implementation

These disadvantages can be solved or reduced by using the Python module `subprocess`. This module allows, amongst other things, to start other Python scripts as child subprocesses of the current process and connecting their in- and outputs via various operations. [47, p. 358]

For this implementation the in- and outputs have been connected using `pipes`. When a subprocess tries to write to its standard out, e.g. the shell, it will instead be rerouted to the standard input of the parent process.

```

1 child = subprocess.Popen(["python", "./exampleFile.py"], stdout=PIPE, stderr=PIPE)
2 output, errors = child.communicate()

```

This is a simplified version of the code used for the communication.

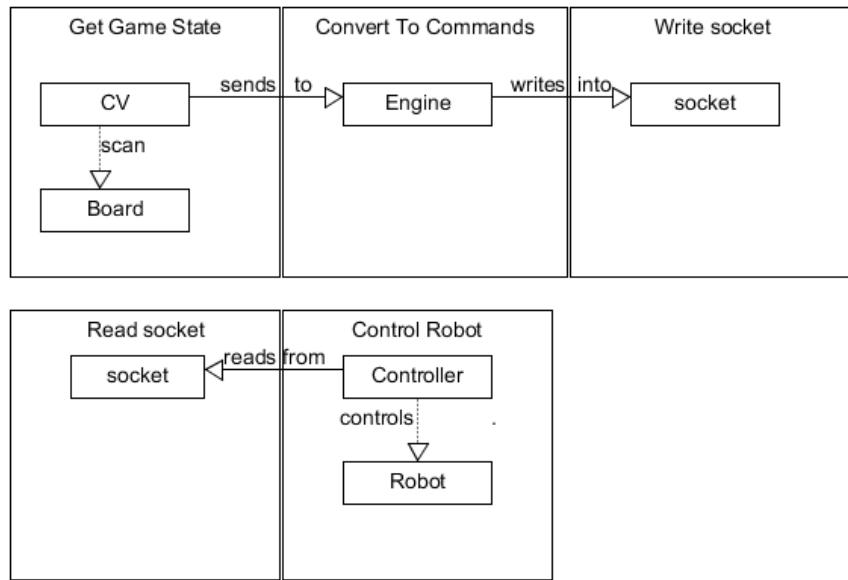
It starts `exampleFile.py` as a subprocess and reroutes its error- and standard output via `pipe` to the parent process (`subprocessRead`).

The function `communicate()` reads everything from the pipe after the child has terminated and saves the standard output and error output to two variables. [47, p. 360]

Using the before shown files the communication was tested and several reasons for another implementation became apparent.

Even though this approach is more efficient, there were problems with the communication. Subprocess only allows to read the output after the child process has been terminated. Simultaneous execution and communication was needed for this thesis, since the engine sends the moves and then waits for the turn of the human. It would be necessary to always start the engine from anew with the current game field to be able to use subprocess. Again this would produce unnecessary overhead. [47, p. 360]

### 5.5.5 Third Implementation via Sockets



**Figure 5.4:** Socket implementation

The third and final implementation replaced the `pipe` communication with a server and client.

This removed the unnecessary overhead of starting the engine or accessing a file every turn. Instead a local connection via `sockets` is created which allows simultaneous and instant messaging.

On the server side a listener was created that listens on a specific port for a connection request. After a successful authentication messages can be sent to it. A big advantage of the used implementation of the listener class is that it allows the transport of any python object through its connection. [48]

This means that the client, after he connects to the server using a connection request to a specific port, can send messages via that connection containing e.g. strings.[48]

These messages are saved on the listener side and are not lost even after the client has disconnected. This and the fact that messages are saved in the order of their arrival allows the two scripts to run parallel, because even if the robot takes too long and a new move has already been sent to it, it can complete its current turn and then read the next one from the buffered listener when it's ready.[48]

The communication was tested by implementing a local server in the Python script that controls the simulated robot and a simple client script that connects to the local server and sends example moves. In this client script not only normal moves, i.e. robot move a

game piece from field 2:7 to field 3:6, were sent but also multiple moves in a short amount of time and moves after a pause (30 seconds).

# Chapter 6

## Real Robot (Weller)

### 6.1 General

The Inmoov used for this thesis was built by Maker Austria and is based on an older version. In comparison to the newest version it lacks the lower body, including feet, and only has one eye.

Two Arduinos are used to control the robots 23 motors. They are connected to the computer via USB cables and can be addressed separately, which simplifies e.g. movement of the left and right hand simultaneously.

### 6.2 MRL

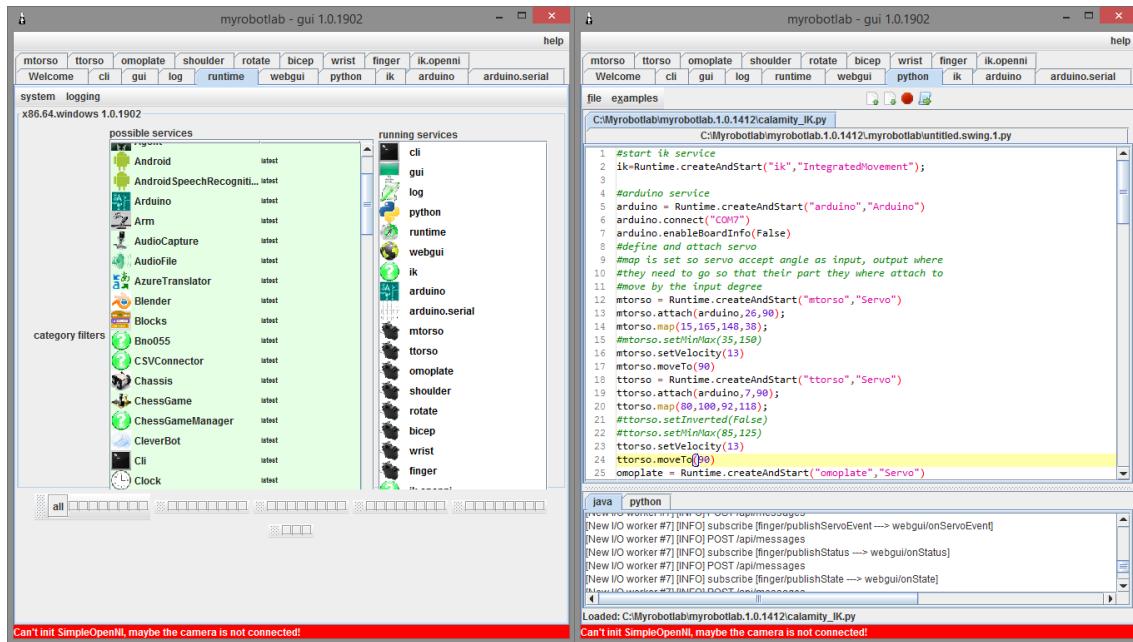


Figure 6.1: MRL GUI

To program the robot, MRL was used instead of the default Arduino IDE. MyRobotLab

includes certain features and libraries specifically tailored for Inmoov. These include, but are not limited to, libraries for servos, OpenCV, Inverse Kinematics and gesture creation. MRL also has an Python Interpreter capable of translating the Python code to Arduino compatible code.

It can either be used via a Java Swing GUI or a WebGUI, both having disadvantages. The GUI tends to crash when starting or when activating features. The WebGUI on the other hand does not support most features.

For this thesis the default Java Swing GUI was used. Inside this user interface a Python IDE, with limited features, is available. When a Python script inside this IDE is written it is capable of accessing every library of MRL. This is possible via the Runtime class, which allows to start certain services and then attach them to the program.

When using MRL in combination with an Inmoov, and control of the robots sensors or motors is necessary, the Arduino service has to be connected to the current Python script, as explained before:

```
1 comPort = "COM7"
2 arduino = Runtime.start("arduino", "Arduino")
3 arduino.connect(comPort)
```

Furthermore for every servo used, a new service has to be created and started. This service is then attached to the arduino service via its pin and optionally the range of the servo can be set or mapped to different values:

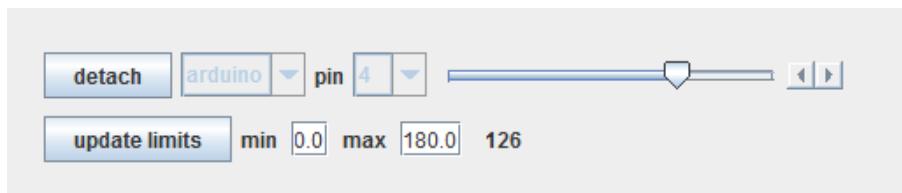
```
1 mtorso = Runtime.createAndStart("mtorso", "Servo")
2 mtorso.attach(arduino, 26);
3 mtorso.map(15,165,148,38);
```

In this example the map means, that the values 15 and 165 are changed to 148 and 38, so if the script moves the servo to 38 in reality the servo moves to 148.

This servo can then be controlled from inside the Python script:

```
1 mtorso.moveTo(50)
2 sleep(0.5)
```

Furthermore it is possible to create and attach the servos in a script and then control them through the UI:



**Figure 6.2:** Controlling a Servo from the UI

### 6.3 Limitations

The biggest limitation of a non-virtual robot is the influence of physics. When raising its hand through the gravity and change of angle it will slow down and thus an error of "virtual" position input and "real" position output is created. This error can also be seen when moving towards certain positions from different sides. The settings of certain servos might work from one direction but can be completely wrong from another.

A limitation specific to the Inmoov lies in the grabbing of objects. The hand was not

designed for grabbing small objects. He can only rotate his thumb towards his palm but not towards his fingers, which would be needed when grabbing e.g. draught pieces.

## 6.4 Inverse Kinematics

### 6.4.1 General

The aforementioned uncertainty can lead to wrong positioning through IK even though the calculation were correct. Several IK algorithms were tried to circumvent this problem but none could reliable move to every game field from every field reliably.

We tested the following libraries included in MRL:

- Inverse Kinematics 3D
- Integrated Movement

### 6.4.2 IK3D

MRLs first IK library was used at the beginning. It was first only developed as a 2D IK solver but got expanded into a 3D solver.

Amongst the problems of IK3D are the tendency to crash and lack of documentation. These factors paired with a missing community support lead to the decision of moving to a different library.

### 6.4.3 Integrated Movement

After IK3D a newer implementation called **Integrated Movement** was tested. It is currently work in progress and has limited functionality.

### 6.4.4 KDL

The last library tested was the **Kinematics and Dynamics Library**(KDL). This is also the library underlying the Blender iTASC IK Solver.

Problems with KDL are the missing integration with the Inmoov. For different robots there are already presets available but for the Inmoov everything would be needed to be calculated from the ground up.

## 6.5 Movement with IK

As aforementioned when using IK to calculate and carry out movement on a real robot there always exists a certain error margin that needs to be considered. There are ways to reduce or completely eradicate this margin e.g. by using cameras to check if a correction factor needs to be applied or by simply building the robot with more accurate motors.

Neither of these were available for the Inmoov. Even though it would be possible to create such a system that uses the eyes or external cameras to monitor and correct the movement, this system would have been too complex for this thesis, instead the movement errors have been minimised by always moving to a fixed position in-between every turn.

The robot is controlled from inside MRL with a Python script. This caused problems, because at the time of development, MRL had severe problems e.g. crashing when started or not being able to connect to the Arduinos. Because of this after several weeks of development the focus of the thesis was switched to a virtual version of the Inmoov.

## 6.6 Grabbing stones

Since the hand of the Inmoov makes grabbing game pieces problematic a small electromagnet is used. When grabbing game pieces the robot moves its index finger above the game piece and then activates the magnet. While moving the game piece sticks to the finger and after reaching the target the robot releases the game piece by deactivating the magnet.

## 6.7 Human-Robot-Interaction

The human interacts with the robot through a game of draughts. This allows a less serious interaction with a humanoid robot than it would occur in most real life situation e.g. in a car producing facility. Because of the appearance of the robot a more "human" enemy is created.

# Chapter 7

## Simulated Robot

### 7.1 General

To simulate the robot a 3D representation is used inside of Blender. This model is rigged using an armature, which represents the skeleton of the model and defines movement and limits of the robot. The armature is necessary for the Inverse Kinematics and the realistic movement.

In addition a target armature consisting of a single bone was created because the Inverse Kinematic Solvers implemented in Blender need an end point for their calculations.

### 7.2 Limitations

The biggest, intertwined limitations are the human robot interaction and the realism of the robot. By using the real robot the human user has the robot in front of him and he can directly interact with it. The virtual robot on the other hand has an additional stage, namely a laptop etc. that displays the robot. The impact of this stage can be minimized with virtual reality i. e. a VR headset. The realism limitation is not only limited to how real the robot looks but also includes the movement, since simulated and real physics can differ.

### 7.3 Inverse Kinematics

Inverse Kinematic Solvers can be applied to any model in Blender on the condition that it has an armature rigged to it, including an end bone that can be used as an end effector, and a target object exists.

Blender provides multiple Inverse Kinematic Solvers:

- Standard
- iTaSC
  - Damped Least Square
  - Selectively Damped Least Square

#### 7.3.1 Standard

The standard IK function first calculates the **Jacobian Matrix** and inverts it. Because of this the standard IK solver is not a dynamic solver because neither masses, forces nor accelerations are taken into account and thus the realism of movements may vary. [49]

### 7.3.2 iTaSC

iTaSC, short for Instantaneous Task Specification using Constraints, consists of multiple parts:

- Solver including parts of KDL which is an open source kinematic library
- Plugin system to support multiple IK solvers
- Template library for matrix operations

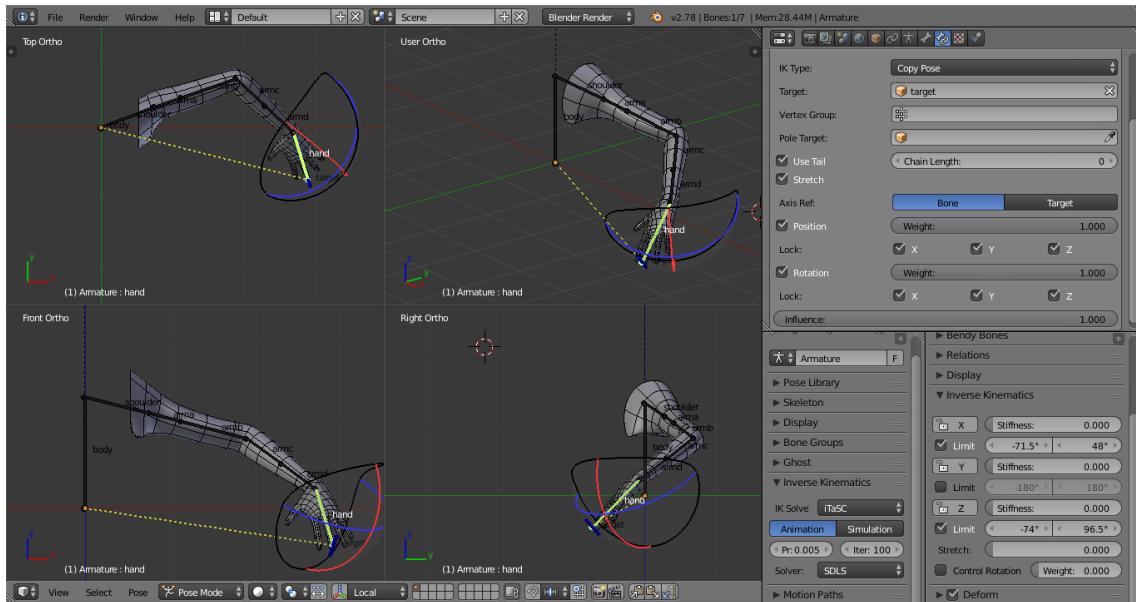
Like the standard IK solver iTaSC first calculates the **Jacobian Matrix** and then inverts it, but it uses a different method for the calculations which allows it to factor in different constraints. iTaSC also allows multiple weighted constraints and a stateful / stateless mode is included. The stateful mode is referred to as **Simulation mode** and factors in time in the calculations. In a standard calculation joint velocity is ignored but in **Simulation mode** the joint velocity between two frames is taken into account allowing it to be up to 40 times as fast, compute more natural movement, hindering pose flipping and handling bigger armatures.[49]

The **stateless mode**, called **Animation mode**, on the other hand does not take joint velocity into account and thus can lead to pose flipping and performance loss. The name was chosen because it is the preferred mode for animators who don't want to be restricted in their movement by previous frames.

Animation mode still benefits from multiple constraint types and multiple constraints per bone.[49]

## Configuration

The easiest way to configure iTaSC is via the Blender GUI, since the plugin is fully integrated into the UI.



**Figure 7.1:** Configuration of an armature for IK

To simulate the IK at a realistic standard multiple components and constraints need to be set up. The default values of these are set to be as open as possible, meaning that they do not restrict translation or rotation, instead they allow the user to set the restrictions

according to his needs.[49]

One of these constraints is the restraint of rotation of a bone compared to his parent bone. This rotation constraint describes the maximum allowed X,Y and Z rotation along the local axis of the bone compared to the parent. The default value for these parameters is unrestricted. This allows the child to rotate on every axis without limits. When a user wants to simulate realistic movement these unrestricted rotations can be problematic since a real object e.g. a human has certain limits. These parameters have to be known and configured properly to allow reasonable movement and inhibit unrealistic situations.[49] To better showcase the configuration of these parameters in this thesis images and explanation of their effects on the armature will be presented in the following paragraphs.

### 7.3.3 IK Constraint

The first necessary step is to add the **IK Constraint** must be added to the end effector bone. Without this step the IK can not be used with the armature since the default bone has no IK and thus no end effector exists to control the remaining rig. [49]

In this constraint it is also possible to determine how much weight to give to each parameter of the target object. For example if the location weight is set to zero the location is not important, but it will still affect the rig. These parameters can also be set according to axis i.e. how the axis location and rotation of the target will affect the end effector.[49]

### 7.3.4 IK Solver

In the armature tab it is possible to choose the IK Solver used for the calculations. The options for this parameter i.e. **Standard** and **iTaSC**, have already been discussed in [chapter][section]. Additionally it is also possible to set the precision, the maximum amount of calculation iterations and the mode of the IK Solver.[49]

**SDLS** stands for **s**electedively **d**amped **l**east **s**quares and **DLS** for **d**amped **l**east **s**quare respectively. In comparison to the **Jacobian Method** these algorithms reduce the jitter and shaking that may emerge when the end effector is trying to extend to something out of its reach. **SDLS** is an extension of **DLS** and is able to converge in fewer iterations and not needing damping constants.[49]

### 7.3.5 Bone Configuration

As mentioned before a realistic simulation needs to adhere to certain physical limits e.g. rotation. Each bone in the human body has certain rotation limits in comparison to each connected bone. These limits are controlled by their joints which need to be simulated to maximise realism.[49]

Blender allows the configuration of each axis rotation limit separately and even as a section or completely lock it. It also draws these limits in UI to simplify the process. In this example the X- and Y-axis rotation has been limited but the bone can rotate freely on the y axis.[49] The values used for the limits of this thesis are only approximates of real world human limits and may diverge from real life limits to some degree.

## 7.4 Movement with IK

Blender has an built in Python IDE for development and a Python module called **bpy** that is capable of accessing every functionality of the program including everything a user could access using the UI.

There are several modules inside `bpy` but for this thesis mostly `bpy.data` was used. The module allows the program to access the scene data e.g. model position. Every object in a scene is saved inside `bpy.data.objects` and can be either be accessed by using the name of the object and a get function

```
1 bpy.data.objects.get("name")
```

or directly using the name as a key

```
1 bpy.data.objects["name"]
```

This feature was taken advantage of by naming every stone in correspondence to the field it is situated at the beginning of the game and renaming it after every move to the updated location.

This allows an easy way to search for a stone by simply creating a string that matches the wanted coordinates:

```
1 name = 'stone'+str(ROW)+str(COLUMN)
```

Because Blender needs a target for an IK Solver an invisible object was created named “target”. When the simulated robot needs to move its hand the target models coordinates are repeatedly changed towards the target to create a smooth movement.

```
1
2     def makeMove(self,startRow,startCol,endRow,endCol):
3         self.moveToField(startRow,startCol)
4
5         nameS = 'stone'+str(startRow)+str(startCol)
6         nameE = 'stone'+str(endRow)+str(endCol)
7         bpy.data.objects[nameS].name = nameE
8
9         parent = bpy.data.objects.get("target")
10        child = bpy.data.objects.get(nameE)
11
12        self.makeParent(parent,child)
13
14        endX,endY = self.getFieldCoord(endRow,endCol)
15        targX,targY,targZ = self.getTargetLoc()
16        self.moveTargetSize(targX,targY,targZ,0,0,0.10,10)
17        targZ += 0.10
18
19        self.moveTarget(targX,targY,targZ,endX,endY,targZ,10)
20
21        targX,targY,targZ = self.getTargetLoc()
22        self.moveTargetSize(targX,targY,targZ,0,0,-0.10,10)
23        targZ -= 0.10
24
25        child.parent = None
26        bpy.data.objects[nameE].location.xyz = (endX,endY,targZ)
27        self.moveToField(0,0)
```

Several types of moves are possible during a game:

- Moving a stone
- Removing a stone
- Adding a stone

#### 7.4.1 Basic Steps

Each maneuver starts with the same steps, namely finding the stone affected by the move and parenting the target object to the stone.

### 7.4.2 Moving a Stone

When moving a stone from one field to another these basics steps are follow by a name generation for the moved stone. Subsequently the coordinates for the goal field is calculated and the end effector bone is moved to these coordinates. After reaching the destination the parent child relationship of the game piece and the bone is broken up. To allow the other player to make his move without interruption or collision the arm is then positioned at a resting position outside the game field.

For adding and removing stones fixed points are defined at the start of the program where the out of game stone are stored. These points are called **graveyard**.and they are represented by one stone for each team, coloured according to the team colour.

### 7.4.3 Removing a Stone

When stones are removed from the game they go through the default move and then they are moved to the proper **graveyard**. There their parent child relationship is reset and they are unlinked from the scene and then deleted. The unlinking from the scene is a blender specific necessity to be able to remove models.

### 7.4.4 Adding a Stone

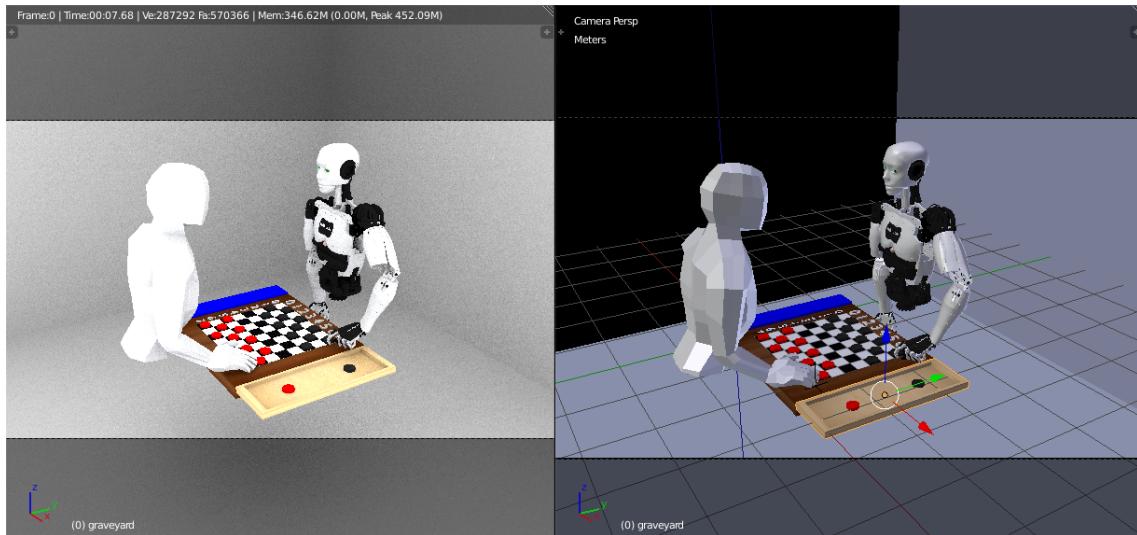
Before adding stones the team of the current player is first checked and then the arm of the robot is moved to the according **graveyard**. As explained before deleted stones are completely removed from the game. This means that when the player wants to add a stone it is necessary to create a new object. This is accomplished by copying the model and the data of the **graveyard** stone and then renaming it according to the naming conventions i.e. stone[Destination Row][Destination Column]. Subsequently the steps of a normal move are followed.

## 7.5 Grabbing Stones

The grabbing of stones is only simulated by a small animation, since there is no physics in blender and because of this the fingers of the robot would have no collision with the pieces and they wouldn't move them. For the movement the current game piece is parented to the target bone of the IK Solver and the bone is moved to the target.

Because the armature of the robot is also linked to the target bone it now looks as if the robot is grabbing the game piece, moving its arm to reach the new location and holding the game piece during this movement. After reaching the destination the robot releases the stone, meaning the parenting of the game piece is reset, and the robot moves its arm back to the default position leaving the stone behind.

## 7.6 Increasing the Realism



**Figure 7.2:** Comparison Blender Render and Blender Camera Persp

A big problem of a simulation is not only unrealistic movement but also how realistic it appears. To reduce this several approaches were tested.

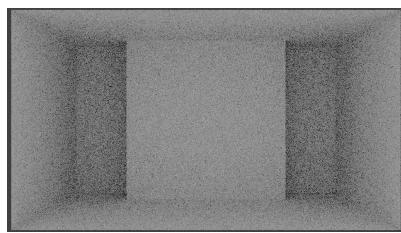
Firstly a model for the robot had to be chosen. As discussed before `VirtualInmoov` was chosen and the armature was restricted to realistic limits.

As a representation of the player a minimalistic human model was inserted and slightly adapted by increasing the smoothness and adding a basic texture.

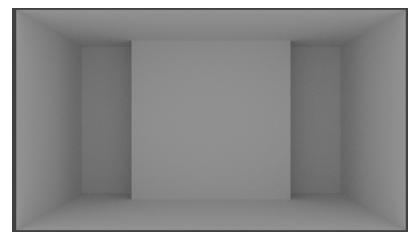
At the beginning the game board was a simple black cube, but later on a wood texture was added and UV unwrapped. Additionally a dish was placed next to the board for the graveyards of the teams. This bowl was modelled out of a cube and also includes a wood texture.

Lastly the render module was chosen. Blender allows the user to choose between Blender Render, Cycles Render and Blender Game.

Cycles was chosen because its advanced material creation and render options, especially the possibility to set the samples per pixel in a render, which reduces so called **fireflies**. To further decrease **fireflies** in the render additional light sources were added.

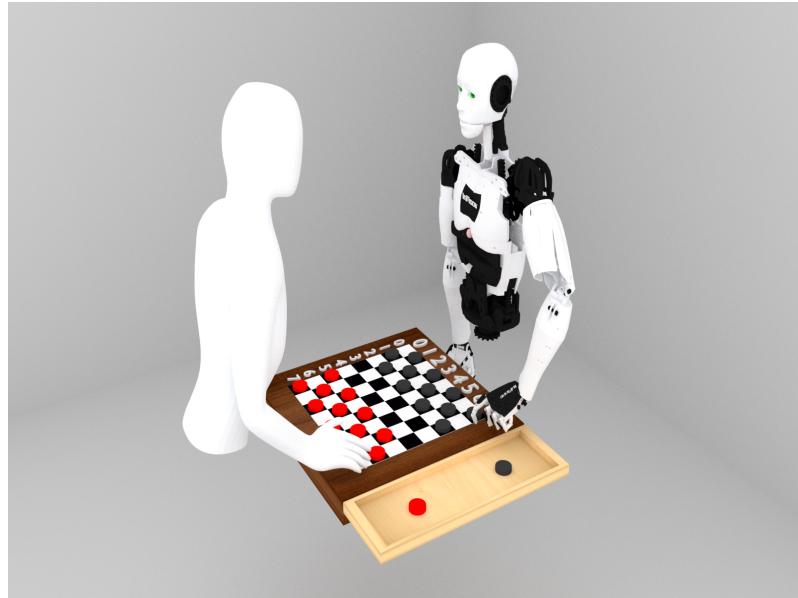


(a) Three samples per pixel



(b) 150 samples per pixel

**Figure 7.3:** Comparison fireflies depending on samples



**Figure 7.4:** Final Render using cycles, 150 samples per pixel and one additional light source

## 7.7 Human-Robot-Interaction

In the simulation there is less HRI than in a real world solution, since the user can not directly interact with the robot. In a real world solution the user is able to touch and examine the robot directly whereas in the simulation he can not touch the robot and needs to use the Blender interface and controls to examine the robot.

Another problem is the HRI on the game board. The real world game board is scanned using CV, parsed to the engine and then sent to a Python script inside Blender where a simulated game board is changed to replicate the real game board.

This copying mechanism can only be done in one way, since the simulation can not directly affect the real world and thus can't change the position of the game pieces. Because of this a human assistant or real robot is needed to redo the simulations game moves.

# Chapter 8

## Conclusion

### 8.1 Computer Vision (Honies)

#### 8.1.1 Conclusion

As stated in the introduction the goal for the computer vision part was to reliably detect a board and game pieces under different and difficult conditions. It has been shown that corner detection works fast and stable for board detection. To prepare an image for piece detection several image processing algorithms are run. Especially Otsu-Thresholding and Erosion are important steps for a predictable outcome. As a final step blob detection is applied to detect the pieces. The implementation was tested heavily under different light conditions to ensure the work can be shown not only in a laboratory but also in real-world examples.

#### 8.1.2 Discussion

While the given implementation works solid, one of its advantages is definitely that it can be adapted easily. Through its clean code base everyone can use it as a playground for OpenCV algorithms to see how they would impact the solution. For example, by adding positive/negative machine learning algorithms for the game piece and corner detection, the program could be improved massively.

#### Future Work

Another goal of the project was to develop a solution which cannot only be used for checkers but can easily be adapted for different games which are also played on a checkerboard. While the board detection could be used without any modification for future projects like chess, the piece detection needs modifications for every sort of game piece that needs to be detected. For chess pieces color-based algorithms could be used as a relatively easy way to detect tall pieces from an angle. However this would mean that instead of the standard black and white pieces, flashy colours like yellow and red have to be used.

### 8.2 Engine (Honies)

#### 8.2.1 Conclusion

To ensure a fully functioning project without breaking the scope of it, an easily adaptable engine and GUI had to be found. Samuel meets this criteria and is now fully integrated. The

computer vision part is connected by running a new thread directly in samuel while the robot is connected using TCP Sockets.

### 8.2.2 Future Work

For future projects the amount of time needed to integrate or develop a engine heavily depends on the game. Chess could be integrated more easily than checkers as there are lots of standardised and well documented engines, for example Stockfish, available. However engines for other games, especially less famous ones, may require more modification or there might even be none available. This would impact development time massively.

## 8.3 Simulation (Weller)

### 8.3.1 Conclusion

The goal of the simulation was to create a realistic representation of the robot, as well as the game and interaction. To achieve this goal a IK Solver, that allowed parameters to control the realism of movement, and the simulation environment was adapted by e.g. adding textures to the objects.

Realistic movement was possible, because the IK Solver iTAsC is able to compare every movement with the previous and thus prevent impossible or illogical positions.

Via the connection of the engine and the Python script controlling the robot, which was implemented using sockets, moves are transferred and performed.

The script can differentiate between adding, moving and removing game pieces. Allowing not only a normal game flow but also a realistic resetting of the game board.

Furthermore configurations were made to allow realistic visualisations. The computers used for this thesis were not able to render these in real-time, instead the camera perspective was used for testing.

### 8.3.2 Discussion

As expected and discussed in section 7.7 the biggest problem of the simulation is the Human Robot Interaction. The user can only indirectly interact with the robot and the real game board has to be set by a human assistant or the opponent, because the simulation cannot access or control it.

The biggest advantage of the simulation were the simplicity of testing and implementation, since physical limitations, i.e. weight of the arms, did not play a role in the movement and thus the IK had no error margin that had to be taken into account.

### Future

With the advance of hardware faster computers or current supercomputers and improved IK algorithms the speed and realism of IK calculations can be improved and the simulation can be rendered in real time instead of the view through the camera perspective.

This would allow a more realistic experience, while at the same time reducing the risks and circumventing the limitations of a real robot.

## 8.4 Real Robot (Weller)

### 8.4.1 Conclusion

The two Arduinos built into the Inmoov were controlled via a Python script inside MRL. This script uses the built in services Servo and IntegratedMovement to control the robot and move the arm to certain positions.

Due to the design of the hand, especially his thumb, he is not able to grab game pieces.

### 8.4.2 Discussion

Though the advance of 3D printers and the open source community allowed the creation of Inmoov, there are still several problems that need to be solved before he can play checkers in real life.

Firstly the servos, in the maker austria model, need to be replaced with more accurate and powerful servos to allow exact movement.

Additionally a different material for the robot has to be used to reduce the amount of bending it undergoes when moving to extreme positions e.g. raising his hand.

The design of the hand has to be redone to allow a grabbing motion capable of holding small objects i.e. game pieces.

Nevertheless it is possible to create a checkers playing Inmoov. To achieve this without implementing the aforementioned changes some measures have to be adopted.

A system to accurately assess the real position of the hand or finger has to be created. This can be implemented e.g. via a camera system capable of comparing the wanted and reached position.

Secondly the hand and game pieces have to be slightly adapted with e.g. electromagnets or modelling them in a special way to circumvent the grabbing problem.

### Future

More accurate servos and different materials combined with working and documented IK libraries can allow the Inmoov to not only play checkers, but instead be used in many different applications e.g. as an assistant physiotherapist.

# Glossary

**OpenCV** Open Source Computer Vision Library

**HRI** Human Robot Interaction

# Bibliography

- [1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN: 013168728X.
- [2] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 1848829345, 9781848829343.
- [3] *-oid meaning*. [Online; accessed 3-April-2017]. URL: <https://en.oxforddictionaries.com/definition/us/-oid>.
- [4] *humanoid meaning*. [Online; accessed 3-April-2017]. URL: <https://en.oxforddictionaries.com/definition/us/humanoid>.
- [5] *robot meaning*. [Online; accessed 3-April-2017]. URL: <https://en.oxforddictionaries.com/definition/us/robot>.
- [6] Malachy Eaton. *Evolutionary Humanoid Robotics*. Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-44599-0.
- [7] M. A. Goodrich and A. C. Schultz. *Human–Robot Interaction: A Survey, Foundation and Trends in Human–Computer Interaction*. now Publishers Inc., 2008. ISBN: 978-1-60198-092-2.
- [8] *Telepresence*. [Online; accessed 3-April-2017]. URL: <https://robonaut.jsc.nasa.gov/r1/sub/telepresence.asp>.
- [9] M.E. Rosheim. *Leonardo's Lost Robot. 2006*. Berlin: Springer Verlag, 2006.
- [10] Boris Duran and Serge Thill. *Rob's Robot: Current and Future Challenges for Humanoid Robots*. InTech, 2012. DOI: [10.5772/27197](https://doi.org/10.5772/27197).
- [11] Md. Hasanuzzaman and H. Ueno. *The Future of Humanoid Robots - Research and Applications; Chapter 12: User, Gesture and Robot Behaviour Adaptation for Human-Robot Interaction*. InTech. ISBN: 978-953-307-951-6.
- [12] M. Telles. *Python Power! The comprehensive Guide*. Power!, 2006. ISBN: 978-1598631586.
- [13] *History of Python*. [Online; accessed 3-April-2017]. URL: [http://www.python-course.eu/python3\\_history\\_and\\_philosophy.php](http://www.python-course.eu/python3_history_and_philosophy.php).
- [14] S. Wood. *Brief History of Python*. [Online; accessed 3-April-2017]. URL: <https://www.packtpub.com/books/content/brief-history-python>.
- [15] *Programming languages inside Blender*. URL: <https://www.blender.org/get-involved/>.
- [16] *OpenCV supported Platforms*. URL: <http://opencv.org/platforms.html>.
- [17] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. 2nd. O'Reilly Media, Inc., 2013. ISBN: 1449314651, 9781449314651.
- [18] Jonathan Schaeffer et al. “Checkers is solved”. In: *Science* (2007).

- [19] Martin Fierz. *Cake*. [Online; accessed 3-April-2017]. URL: <http://www.fierz.ch/cake.php>.
- [20] Ed Gilbert. *KingsRow*. [Online; accessed 3-April-2017]. URL: <http://edgilbert.org/EnglishCheckers/KingsRowEnglish.htm>.
- [21] Jon Kreuzer. *GUI Checkers*. [Online; accessed 3-April-2017]. URL: <http://www.3dkingdoms.com/checkers.htm>.
- [22] Samuel. [Online; accessed 3-April-2017]. URL: <https://github.com/johncheetham/samuel>.
- [23] *License of Blender*. [Online; accessed 3-April-2017]. URL: <https://www.blender.org/about/license/>.
- [24] *History of Blender and the Blender Foundation*. [Online; accessed 3-April-2017]. URL: <https://www.blender.org/foundation/history/>.
- [25] T. Larsson. *Introduction to Python scripting for Blender 2.5x*. 2011.
- [26] D. Ryan. *History of Computer Graphics: Dlr Associates Series*. 2011. ISBN: 978-1456751173.
- [27] *Maya Licenssing*. [Online; accessed 3-April-2017]. URL: <http://www.autodesk.de/store/products/maya?term=1month&support=basic>.
- [28] Michael Milford. *Introduction to Robotics (ENB339); Lecture 3 - Forward and Inverse Kinematics*. 2011.
- [29] Joan Lasenby Andreas Aristidou. *FABRIK: A fast, iterative solver for the Inverse Kinematics problems*. Department of Engineering, University of Cambridge, 2011.
- [30] *ASIMO - Specifications*. URL: <http://asimo.honda.com/asimo-specs/>.
- [31] *ASIMO - History*. URL: <http://asimo.honda.com/asimo-history/>.
- [32] *ASIMO Price*. URL: <https://www.forbes.com/2002/02/21/0221tentech.html>.
- [33] *Poppy Specification*. URL: <https://www.poppy-project.org/en/robots/poppy-humanoidl>.
- [34] *Poppy Price*. URL: <https://www.generationrobots.com/en/312-poppy-humanoid-robot1>.
- [35] *ATLAS Specification*. URL: [http://archive.darpa.mil/roboticschallengetrialsarchive/files/ATLAS-Datasheet\\_v15\\_DARPA.PDF1](http://archive.darpa.mil/roboticschallengetrialsarchive/files/ATLAS-Datasheet_v15_DARPA.PDF1).
- [36] *DARPA BAA-12-39*. 2012.
- [37] *Inmoov - History*. URL: <http://inmoov.fr/project/>.
- [38] *Simulation meaning*. [Online; accessed 3-April-2017]. URL: <https://en.oxforddictionaries.com/definition/simulation>.
- [39] *VinMoov*. [Online; accessed 3-April-2017]. URL: <http://myrobotlab.org/content/virtual-inmoov>.
- [40] *Blender API*. [Online; accessed 3-April-2017]. URL: [https://docs.blender.org/api/blender\\_python\\_api\\_2\\_77\\_0/info\\_overview.html](https://docs.blender.org/api/blender_python_api_2_77_0/info_overview.html).
- [41] Robert Fisher et al. *71 - Hypermedia Image Processing Reference (HIPR)*. [Online; accessed 3-April-2017]. 1997.
- [42] *Thresholding*. [Online; accessed 3-April-2017]. URL: [http://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_thresholding/py\\_thresholding.html](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html).

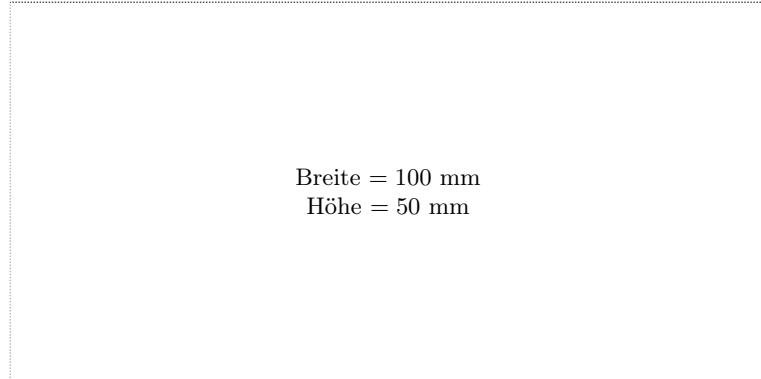
- [43] Robert Krutsch and David Tenorio. *Histogram Equalization*. [Online; accessed 3-April-2017]. URL: [http://cache.freescale.com/files/dsp/doc/app\\_note/AN4318.pdf](http://cache.freescale.com/files/dsp/doc/app_note/AN4318.pdf).
- [44] *Eroding and Dilating*. [Online; accessed 3-April-2017]. URL: [http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion\\_dilatation/erosion\\_dilatation.html](http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html).
- [45] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning OpenCV, 1st Edition*. First. O'Reilly Media, Inc., 2008. ISBN: 9780596516130.
- [46] *OpenCV Corner Detection*. [Online; accessed 3-April-2017]. URL: [http://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py\\_shi\\_tomasi/py\\_shi\\_tomasi.html#shi-tomasi](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi).
- [47] Alex Martelli. *Python in a nutshell*. O'REILLY, 2006. ISBN: 978-0-596-10046-9.
- [48] *Python multiprocessing module*. URL: <https://docs.python.org/2/library/multiprocessing.html>.
- [49] *Blender iTaSC Documentation*. URL: <https://wiki.blender.org/index.php/Dev:Source/GameEngine/RobotIKSolver>.

# List of Figures

2.1	Samuel on macOS . . . . .	9
2.2	IK . . . . .	11
2.3	Humanoid robots . . . . .	13
3.1	Comparison of mean and gaussian filter . . . . .	18
3.2	Comparison of otsu and global thresholding at difficult lightning . . . . .	19
3.3	Demonstration of Histogram Equalization applied to image of a checkerboard	20
4.1	Game flow . . . . .	24
4.2	QR Codes for Program Flow . . . . .	25
4.3	Returned corners from Good Features to Track Function circled in red . . . . .	27
4.4	Comparison of Board Detection Methods . . . . .	28
4.5	Processing of the image for piece detection . . . . .	29
4.6	Before / After Processing and Blob Detection . . . . .	29
5.1	Visualization of example moves . . . . .	34
5.2	Text file implementation . . . . .	34
5.3	Subprocess implementation . . . . .	35
5.4	Socket implementation . . . . .	36
6.1	MRL GUI . . . . .	38
6.2	Controlling a Servo from the UI . . . . .	39
7.1	Configuration of an armature for IK . . . . .	43
7.2	Comparison Blender Render and Blender Camera Persp . . . . .	47
7.3	Comparison fireflies depending on samples . . . . .	47
7.4	Final Render using cycles, 150 samples per pixel and one additional light source . . . . .	48

# Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



Breite = 100 mm  
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —