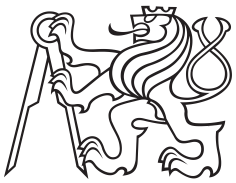
Bachelor thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Human-robot collaboration for playing checkers

Elizaveta Isianova

Supervisor: Pavel Burget
May 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Isianova Elizaveta** Personal ID number: **492326**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Human-robot collaboration for playing checkers

Bachelor's thesis title in Czech:

Kolaborativní robotické pracoviště pro hru dáma

Guidelines:

The purpose of this thesis is to design a robotic workplace for a human-robot collaboration that would be able to play checkers with a human opponent. The robot would be fully autonomous in recognizing the environment, capturing the state of the game, and manipulating the checkers.
1) Get acquainted with KUKA LBR iiwa collaborative robot and its operating system KUKA Sunrise.
2) Suggest a method how to implement recognition of the checker with a camera.
3) Design and implement the system's overall architecture (robot, camera, PLC).
4) Design user scenarios of how the checkers game will be played and implement the game algorithm. The robot will be able to play autonomously.
5) Implement the robotic workplace and test its functionality.

Bibliography / sources:

[1] KUKA Roboter GmbH. KUKA Sunrise.OS 1.11. Operating and Programming Instructions for System Integrators. Augsburg, Germany, 2016.
[2] John, K. & Tiegelkamp, M. IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools, Springer, 2013.
[3] SCHAEFFER, J., BURCH, N., BJÖRNSSON, Y., KISHIMOTO, A., MÜLLER, M., LAKE, R., LU, P. & SUTPHEN, S., "Checkers Is Solved", Science (American Association for the Advancement of Science), vol. 317, no. 5844, pp. 1518-1522, 2007.

Name and workplace of bachelor's thesis supervisor:

Ing. Pavel Burget, Ph.D. Testbed CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **31.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until:
by the end of summer semester 2022/2023

Ing. Pavel Burget, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank Ing. Pavel Burget Ph.D for supervising this work. I thank Ing. Tomáš Jochman for consultations on PLC and KUKA programming. I thank Serhii Voronov for the help in assembling the robotic workplace and 3D printing of parts. I would also like to thank the Czech Institute of Informatics, Robotics and Cybernetics for providing an opportunity to work on this project.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 20. května 2022

Abstract

This thesis aims to create a collaborative robotic workplace for playing the game checkers with a human opponent. The architecture of the system, consisting of a PLC, collaborative robot, and a computer vision system, allows the robot to be fully autonomous in perception of the checkerboard and in playing the game. The game algorithm is extended with artificial intelligence that contributes to fast decision making and the robot's high win rate.

Keywords: collaborative robots, Industry 4.0, KUKA, checkers game, industrial robot

Supervisor: Pavel Burget
Czech Institute of Informatics, Robotics and Cybernetics,
Testbed for Industry 4.0
Praha 6

Abstrakt

Tato práce si klade za cíl vytvořit kolaborativní robotické pracoviště, na kterém má člověk možnost zahrát si hru dáma proti robotu. Architektura systému, který se skládá z PLC, kolaborativního robota a systému počítačového vidění, umožňuje robotovi být plně autonomní ve vnímání šachovnice a při hraní hry. Algoritmus hry je rozšířen o umělou inteligenci, která přispívá k rychlému rozhodování a zaručuje robotu vysokou pravděpodobnost výhry.

Klíčová slova: kolaborativní robot, průmysl 4.0, KUKA, hra dáma, průmyslový robot

Překlad názvu: Kolaborativní robotické pracoviště pro hru dáma

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Human-Robot Collaboration

The relevance of the Human-Robot Collaboration has been rising exponentially with the rapidly growing demands for robotic applications in households and industrial workspaces. Today, the industry development trend is towards Industry 4.0, which is based on fully automated production processes with the human contribution set to a minimum[3]. Despite the reduced prominence of humans at the site, their presence is not entirely eliminated. The humans' role is now to instruct and mentor their robotic colleagues that took over the manual work. Modern design approaches of industrial robots enable robots not only to help employees throughout production but also to protect their health.

However, the scope of collaborative robots application is not limited to industrial environments. Innovative robotic design allows cobots to interact effectively and securely with humans in other spheres. There are numerous examples of robots taking human jobs, such as serving as a barman at a Robobar[4] or assisting laboratory workers in taking COVID-19 tests at Bulovka Hospital in Prague[5]. Such tasks require high cognitive abilities, so the robot would be able to collect data about its surroundings and properly react to a change in its environment.

The task of creating a collaborative robotic workplace for playing a game includes adjusting the input signals from the outside world using all available devices, such as a camera, a touch screen, and sensors of the robot itself. Furthermore, it is necessary to take into account the psychological perception of the game process by a human. When competing with a supposedly stronger robotic opponent, one would expect fast decision-making, conciseness of movements, and the impossibility of deception. Therefore, an implementation of the game algorithm should be robust enough to take little time while calculating an optimal move and complying with defined game rules.

1.2 Objectives

This Bachelor thesis aims to create a collaborative robotic workplace for the checkers game that meets industrial standards and has a convenient user interface. This project demonstrates the interaction of collaborative robots, also called cobots, with humans on the site.

The work on this project has been carried out as part of an internship at the Testbed for Industry 4.0 at Czech Institute of Informatics, Robotics and Cybernetics (CIIRC CTU). The purpose of this work is to create a plug-and-play robotic cell that would act as a show-case of collaborative robots for visitors of Testbed. The game of checkers was chosen because it is a game widely played all over the world. Thus, a robotic workplace would provide an opportunity for the general public to get acquainted with modern industrial technologies, collaborative robots in particular, through participation in the well-known game.
The work on this project can be outlined into four stages.

1. The first stage consists of getting acquainted with the new software tools, such as KUKA Sunrise.Workbench and KUKA.WorkVisual for KUKA iiwa robot programming, Keyence CV-X Series for computer vision system configuration, TIA Portal v16 for PLC programming, hardware and network configuration, as well as learning a new programming language - Java.

2. The second project phase includes designing system's components arrangement and implementation of the overall architecture, subsequent configuration of hardware and network communication.

3. The third stage of the project includes software development, such as implementation of state machines, creation of the game core in Java and integrating artificial intelligence algorithms to a robot player, computer vision setup and robot movements handling.

4. In the final work phase the system testing takes place. The performance and robustness must be tested.

This work does not aim to develop an invincible artificial intelligence player. The checkers game is already solved, and a program capable of this challenge exists[6].

Chapter 2

Checkers

2.1 Checkers rules

Checkers is a two-player game that is widely spread around the world. Its first mention dates from 3000 BC, when a board resembling a checkerboard was found in ancient Mesopotamia[7]. Over the centuries, the rules of the game have evolved, supplementing the old ones and creating the new ones. To date, there are dozens of different versions of the game. For this project, I chose the American version of checkers[8]. The fundamental rules of the game are well known to the general public; however, let me recall the basis required for a deeper understanding of this paper.

Each player starts the game with 12 pieces of their color: black or white. Pieces can only be placed on the dark squares of the playing board. A regular piece can only move diagonally forward, either by landing on the nearest square if it is empty or by jumping over the opponent piece. In the second case, the nearest square must be occupied by the opponent and the square behind it must be empty. When a piece reaches the edge row on the opposite side of the board, known as the *king row*, it becomes a king. In the Czech and American version of the checkers the coronation of a piece occurs with the help of an additional checker, which is placed on top of the first one, thus increasing its height twice. Due to the impossibility of distinguishing the height of a checker by the camera, in my version of the rules, a checker of a different color is used to distinguish the king: the white queen is orange, the black king is blue. Unlike the regular piece, a king can move both forwards and backwards.

A game can end in four ways:

1. a player wins the game if all opponent pieces were captured;
2. a player wins the game if the opponent still has pieces, but all are blocked and none can make a move;
3. a game ends in draw if there have been 50 moves from both sides without any piece captured;
4. a game ends in draw if the same layout of the gameboard was repeated three times in a row.

Additional rules specifications:

- jumping over a piece, i.e., capturing the piece, is obligatory;
- if a player captures a piece and lands on a square with a further possibility to jump over another opponent piece, he is obliged to do so;
- in some game interpretations, kings can "fly" - move any distance along unoccupied paths, thereby being able to capture pieces e.g. 3 empty squares away from them. The so-called *flying kings* are not used in American, and thus my version of the game;

2.2 AI algorithm

The classic game of checkers is a two-player finite zero-sum game [9] due to its closed system structure. It represents a situation where the gain of one player is equal to the loss of another player. The addition of the total gains and subtraction of the total losses will sum up to zero.

Decision-making in such games is visualized with game trees shown in Figure 2.1. Every tree node represents a possible state of the board and the score that the corresponding player could obtain. The current state of the board is seen as a root node with the tree expansion depth level equal to zero. The number of its child nodes at depth level 1 corresponds to the number of possible moves for the current player. At the maximum depth level, whether it is a predefined number or the end of the game has been reached, we calculate the state score as an output of the heuristic function.

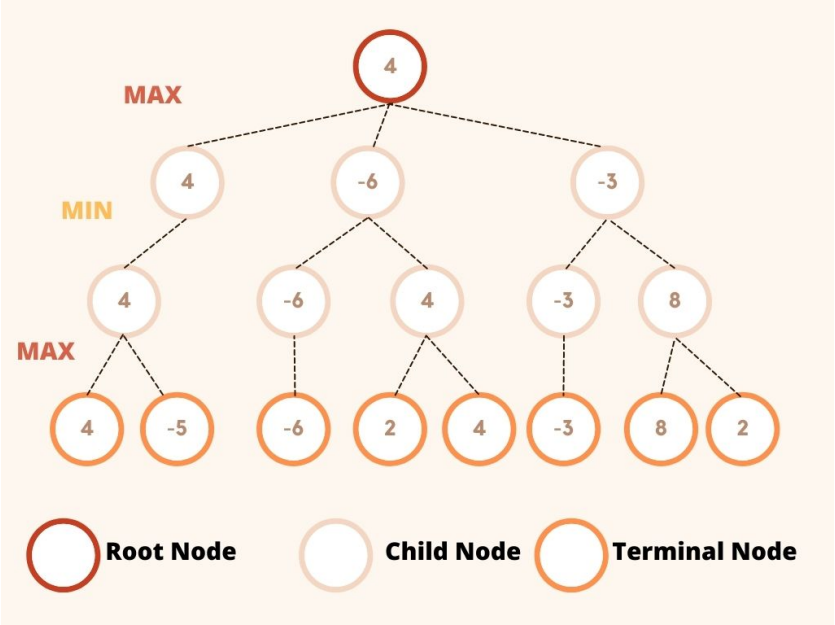


Figure 2.1: Game tree representation for Minimax algorithm.

The heuristic function was inspired from [10], where it is calculated as a linear combination of the following board parameters:

$$H(\mathbf{s}) = w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4 + w_5 s_5 + w_6 s_6, \tag{2.1}$$

where

s_1: number of the current player's regular pieces;
s_2: number of the opponent player's regular pieces;
s_3: number of the current player's kings;
s_4: number of the opponent player's kings;
s_5: number of pieces of the current player threatened by its opponent;
s_6: number of pieces of the opponent player threatened by the current player;
$w_1, ..., w_6$: corresponding numerical coefficients.

After investigating strategies for winning the game, I discovered that it is more desirable for a piece to be placed in the center of the checkerboard, rather than on the edge[11]. In this way, a checker has a greater degree of freedom in the next move. So two elements were added to a heuristic function:

s_7: number of current player's pieces placed on the edge of a checkerboard;
s_8: number of opponent player's pieces placed on the edge of a checkerboard;

The heuristic function was also extended with a parameter c that is assigned to 100 in case of the end of the game and the victory of the current player, to -100 in case of the victory of its opponent, and is set to zero if the end of the game has not been reached.

A modified heuristic function is as follows:

$$H(\mathbf{s}) = w_1 \cdot (s_1 - s_7) + w_2 \cdot (s_2 - s_8) + w_3 s_3 + w_4 s_4 + w_5 s_5 + w_6 s_6 + w_7 s_7 + w_8 s_8 + c \tag{2.2}$$

The values of coefficients $w_1, ..., w_6$ were empirically established in a series of experiments. The corresponding values are shown in the table 2.1.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
value	10	-10	20	-20	-10	10	6	-6

Table 2.1: The values of the heuristic function coefficients.

In order to win, the current player aims to maximize the function output, while their opponent aims to minimize it. So, whether its the current player's or their opponent's turn, the function output approaches its maximum, respectively, its minimum. From the perspective of the game tree, the odd depth level value corresponds to the current player's move, and the even value to their opponent's move. This principle underlies the **Minimax** (also called **Minmax**) algorithm.

```
function  minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
```

```
    if maximizingPlayer then
        value := -∞
        for each child of node do
            value := max(value, minimax(child, depth - 1, FALSE))
        return value
    else (* minimizing player *)
        value := ∞
        for each child of node do
            value := min(value, minimax(child, depth - 1, TRUE))
        return value
```

With the initial call:

```
    minimax(origin, depth, TRUE)
```

Algorithm 2.1: Minimax[12]

The downwards of this algorithm is that its computational complexity increases exponentially with every tree expansion into a new depth level. The most common solution how to improve Minimax is to supplement it with **Alpha-Beta pruning** algorithm described bellow.

In the process of evaluating the playing field at different depth levels of the game tree, the player operates with two newly introduced dynamically changing coefficients - **alpha** (the minimum value of the heuristic function encountered in the branch - i.e., more favorable for the minimizing player) and **beta** (the maximum value of the heuristic function encountered in the branch - i.e. more favorable for the maximizing player). At each depth level, the comparison of the heuristic function value of the current position with the alpha and beta coefficients makes it possible to reject (without calculating them completely) branches that are less beneficial for the current player evaluating the position and/or more beneficial for his opponent. The player discards this branch and does not waste time and resources on considering sub-options from this obviously worst branch for him, thereby reducing the computation time.

The Minimax with Alpha-Beta pruning underlies the implementation of the AI checkers player. The depth of the searching tree expansion represents the difficulty of competing with the AI player.

```
    function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := -∞
        for each child of node do
            value := max(value, alphabeta(child, depth - 1, α, β,
    FALSE))
            α := max(α, value)
            if value ≥ β then
                break (* β cutoff *)
```

```
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth - 1, α, β,
    TRUE))
            β := min(β, value)
            if value ≤ α then
                break (* α cutoff *)
        return value
```

With the initial call:

```
    alphabeta(origin, depth, -∞, +∞, TRUE)
```

Algorithm 2.2: Alpha-Beta pruning[13]

Chapter 3

Robotic workplace architecture

3.1 Hardware configuration

The first prototype of the system had the following architecture. The main core connecting the rest of the elements was the PLC. A KUKA iiwa LBR robot, an HMI panel, and a gripper were connected to it through the Profinet industrial communication protocol. The part responsible for computer vision consisted of an industrial camera connected to a computing unit, a desktop computer.

Figure 3.1: First prototype of the system architecture

One of the important conditions of the project is that this robotic station should always be ready to work and would not require an operator even after a power restart.

Experience has shown that the PC would not contribute to this. With

the possibility of trying out the Keyence computer vision (CV) system, it was decided to eliminate the PC from the architecture scheme and replace the initial computer vision unit with an independent solution, where CV functions are already built into the camera controller. The other parts remained unchanged and the final version of the system architecture is shown in Figure 3.2.

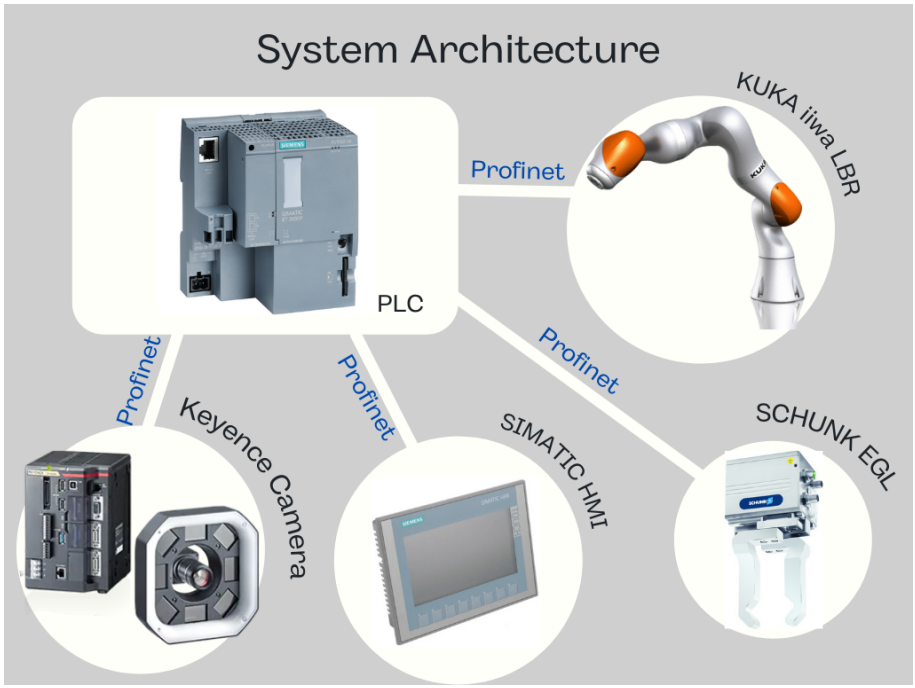


Figure 3.2: Final version of the system architecture.

The physical representation of the robot cell is shown in Figure 3.3. The robotic manipulator is mounted vertically on a solid welding table, which provides high stability against mechanical vibrations and ensures overall stability of the robotic station.

The game board is placed at a distance of 40 to 60 cm from the KUKA iiwa, so it is located in the robot's workspace. The checkerboard is illuminated with an LED strip whose brightness can be manually set by its remote controller. However, the amount of light is adjusted solely for the convenience of the player, as the industrial camera is set up in that way, that it is not highly dependent on lighting conditions and is able to perform tasks with daylight.

The gripper, used as an end effector, is covered in a soft case and is equipped with the grasping parts. Both elements were printed out of a plastic containing rubber. The edges of the objects were smoothed. This material and the design of the elements contribute to a higher safety for the human player. In addition to safety regulations, the user is given a red emergency stop button for an immediate shutdown of the robot's processes.

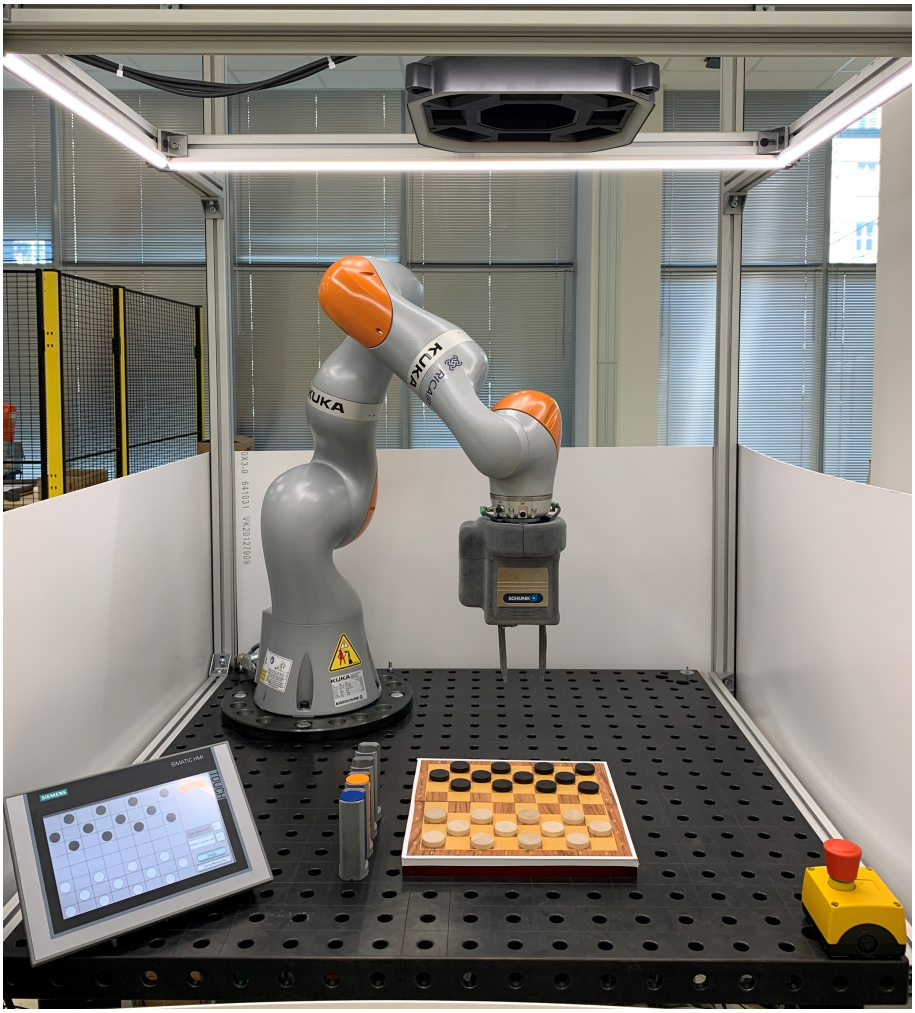


Figure 3.3: Physical representation of the final version of the system architecture

To the left of the playing board, there are four holders that are used to arrange the locations of the unused checkers. They were printed on a powder 3D printer. These holders are located in fixed positions, so when the robot picks up the user's pieces, it can sort them and place them in the suitable compartment.

The playing board itself is a standard 8 by 8 checkerboard. It has a standard color scheme, where all squares have contrasting colors to facilitate recognition of checkers on them. The colors of the board were chosen so that they did not match the colors of checkers, to eliminate errors in computer vision recognition.

In order to manage the state of the game by the human player, the robotic cell is equipped with an HMI (Human-machine interface) screen, a touch panel for user interface. It is a standard solution for displaying and monitoring information in industrial environments.

3.2 Network configuration

The communication network is configured in **STEP 7 TIA Portal**. The Figure 3.4 shows the connections between the devices used in this project.

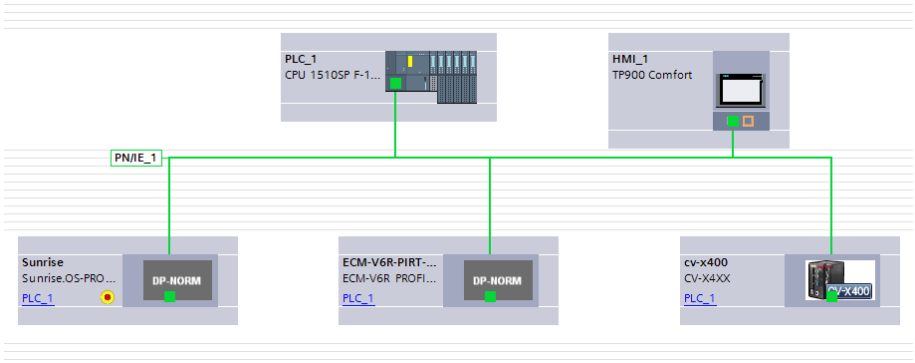


Figure 3.4: Network topology in the project

All devices in the network are connected through the Profinet industrial communication protocol. The devices mentioned are as follows:

- **PLC_1** is the main and only PLC of this project. It is a Siemens CPU 1510SP F-1 PN. The letter "F" stands for *Fail-safe*, which means that the safety and standard programs can be processed on the same processor, which allowed me to implement a safety program to stop the system processes running by pressing the emergency button. For reading an input from the button, the PLC was extended with a digital input module DI8 x DC24V HF;
- **HMI_1** is the Siemens TP900 comfort HMI screen, which enables user communication with the system;
- **Sunrise** represents the robot KUKA LBR iiwa, which is described in Chapter 4;
- **ECM-V6R PROFINET IRT** - represents the SCHUNK EGL gripper. Although the gripper is physically attached to the robot's flange, the gripper is connected to the local Profinet network and controlled by signals from the PLC, not from the KUKA iiwa. It is a servo-electric parallel gripper with two fingers; its gripping force is controlled by current;
- **sv-x400** is a controller of the Keyence computer vision system. It is responsible for receiving an image from the camera, processing the CV tasks and transmitting its output to PLC.

Chapter 4

Robot specifications

4.1 Robot description

KUKA LBR iiwa 14 R820, chosen as a manipulator in the project, is a light-weight collaborative robot. It is equipped with position, joint torque, and temperature sensors in all seven axes, allowing precise measurement of robot position deviation, making KUKA iiwa safely HRC compatible. The robot's payload capacity of 14 kg is half the weight of the manipulator itself.

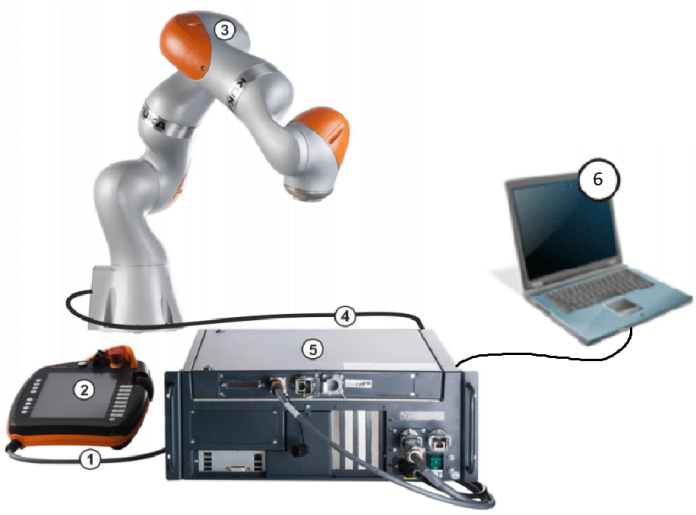
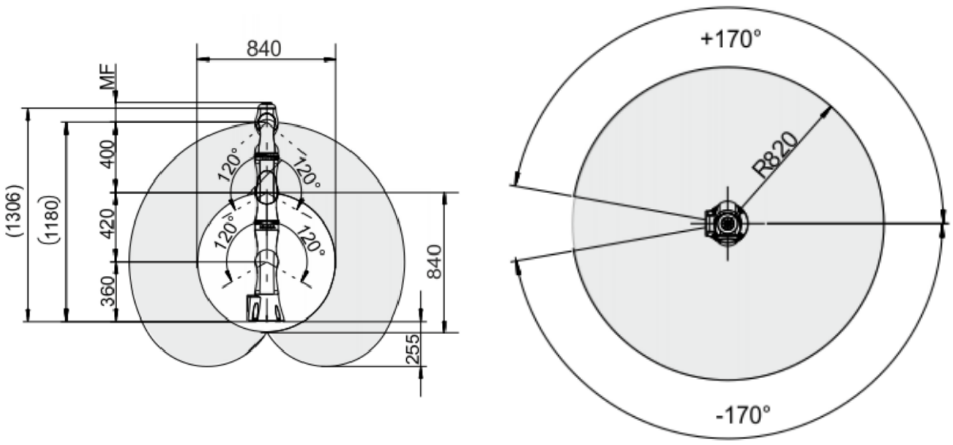


Figure 4.1: Overview of the KUKA LBR iiwa system: [1]
1. Connecting cable to smartPAD,
2. KUKA smartPAD control panel,
3. Manipulator,
4. Connecting cable to KUKA Sunrise Cabinet robot controller,
5. KUKA Sunrise Cabinet robot controller,
6. Development computer with KUKA Sunrise.Workbench.

The robot uses its own operating system **KUKA Sunrise.OS** of version 1.16. In this software package, the programming and operator control tasks are separated and managed by different components.

- **KUKA Sunrise.Workbench** is a Software Development Kit (SDK) with Java programming language, used for application programming, station and safety settings configuration, software installation, and remote debugging.
- **KUKA WorkVisual** 5.0 is a software tool for bus mapping, configuration and diagnosis. In this project WorkVisual is used for configuring and mapping control signals of the Profinet communication with PLC.
- **KUKA smartPAD** is a control panel used by the operator e.g. for calibrating tools, axis mastering, jogging, teaching frames. Neither the station and safety configuration, nor the application content can be changed by the operator. The smartPAD was used during the development of this project to start applications on the station. After project completion, the application start-up was switched to external control via PLC and HMI.
- **KUKA Sunrise Cabinet** is a robot controller based on operating system Windows 7 Embedded that manages safety, logic and process control of the entire system.



(a) : Workspace section in the XZ plane. Dimensions are in mm.

(b) : Workspace section in the XY plane at a height of 360 mm above the base frame.

Figure 4.2: KUKA LBR iiwa 14 workspace representation[1].

4.2 Robot control

There are five categories of the robot motions used in the project:

1. returning the robot to its home position,
2. picking a checker from the checkerboard,
3. picking a checker from the stack for captured checkers,
4. placing the checker onto the game board,
5. placing the checker onto the stack for captured checkers.

The home position is chosen in such a way that the robot would not disturb neither the opponent from making a move, nor the camera from capturing a picture.

Actions (3) and (5) differ from actions (2) and (4) in need to dynamically change the TCP position in Z axis according to the amount of captured checkers.

Actions (2) and (4) require from the camera the coordinates of a TCP position in XY plane.

The rotational parameters along all axes remain unchanged, so the gripper is always placed perpendicularly to the XY plane and does not rotate along the Z axis.

4.2.1 Homography

A homography matrix is used to obtain a transformation of the camera coordinate system to the robot coordinate system. Methods of a planar geometry were applied to establish it.

Since the board's position remains unchanged over time, it was decided to set the coordinate system (CS) of the checkerboard plane from the camera perspective in the following way. The origin of coordinates is in the upper left corner of the board. The squares on the board represent the units of the CS. That is, the upper left square is located at [0, 0], the adjacent square to its right is located at [0, 1], the adjacent square to its bottom is at [1,0], and so on. This representation of the checkerboard position simplifies the communication set up between the computer vision unit and the PLC.

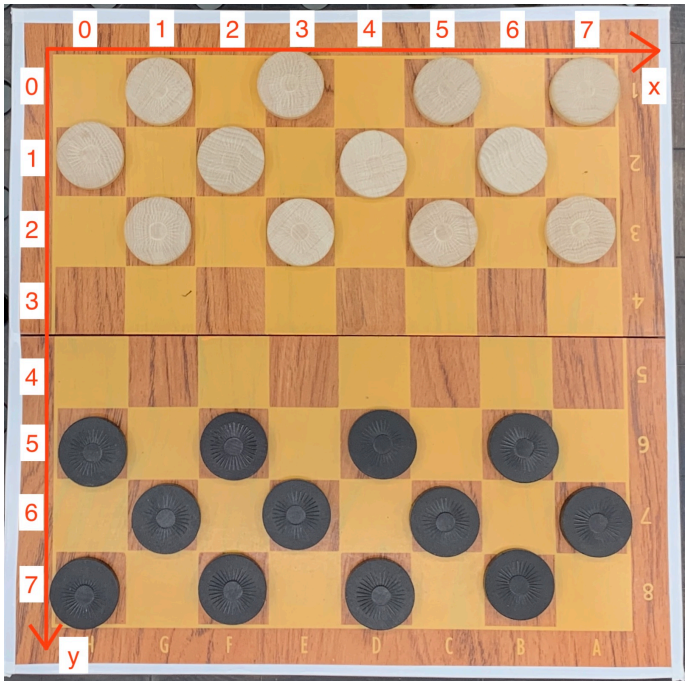


Figure 4.3: Checkerboard coordinate system from the camera perspective.

A projective planar transformation of a vector $\vec{\mathbf{x}}$ in the checkerboard CS to its representation in the robot world $\vec{\mathbf{x}}'$ is a linear transformation defined by a non-singular 3 × 3 matrix $\mathbf{H}$[14]:

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \Leftrightarrow \quad \vec{\mathbf{x}}' = \mathbf{H} \cdot \vec{\mathbf{x}} \tag{4.1}$$

Let us consider a 2D plane as a vector space in $\mathbb{R}^2$. Thus a point in a plane is represented by a vector identified by the coordinate pair (x,y) in $\mathbb{R}^2$. As we only aim to find a transformation matrix that maps between two planes at a fixed position on the third axis in $\mathbb{R}^3$, we will adhere to the homogeneous vector notation. Hereby we can simplify the desired matrix $\mathbf{H}$ and look for its six entries instead of nine.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.2}$$

The unknown elements of the homography matrix $\mathbf{H}$ can be fitted in a 6 × 1 vector. Hence, the expression (4.2) can be rewritten with (4.3), where the coordinate pairs (x_i, y_i) and (x_i', y_i') represent the positions of the i-th point (i ∈ { 1, 2, ..., n}) in the checkerboard coordinate system and the robot coordinate system, respectively[15]. To estimate the components $h_{11}, ..., h_{23}$ we solve the system of equations (4.3) for $\vec{h}$ in the sense of least squares. The

Linear Least Squares problem is solved using the pseudoinverse or inverse of **A**.

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \\ y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & x_n & y_n & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{21} \\ h_{22} \\ h_{13} \\ h_{23} \end{bmatrix} \quad \Leftrightarrow \quad \vec{b} = \mathbf{A} \cdot \vec{h} \tag{4.3}$$

I collected positions of six independent points in both coordinate systems to fill in the expression (4.3) with numerical values. The desired components $h_{11}, ..., h_{23}$ of the homography matrix **H** were found using Matlab expression `h = A \ b`.

$$\mathbf{H} = \begin{bmatrix} 23.5042 & 20.6189 & 435.3667 \\ -21.0579 & 23.4253 & -50.9534 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

The KUKA LBR iiwa 14 pose repeatability is ±0.15 mm[1], so it only makes sense to round up to two decimal places when assigning movement coordinates in mm.

```
private void transfCoords(int x, int y){
    x_kuka =  23.50*x + 20.62*y + 435.37;
    y_kuka = -21.06*x + 23.43*y - 50.95;
}
```

Figure 4.4: Implementation of the coordinates transformation.

4.3 Safe collaboration

External control

In the final version of the robotic workplace, the station would not require an operator for proper operation. An untrained user must be able to turn on, restart or resume the application via HMI. Therefore, the robot station is configured for external control. It allows the PLC to start the robot application with the `App_Start` signal.[1]

Emergency stop

The robotic cell is also equipped with an emergency button that enables the user to stop the running processes immediately in case of abnormal robot

behavior. The robot's reaction to a falling edge on the emergency stop input signal is of type "Stop 1 (on-path)", meaning that the power shortcut only takes place as soon as the robot stands still, allowing it to remain on the programmed path.

Compliance

To ensure higher safety for a human-robot interaction, the KUKA iiwa's motions are made compliant and sensitive to external influences. This is accomplished by establishing impedance control over the robot's motors. The Cartesian impedance controller represents a model of virtual springs and dampers.[1] By assigning the model properties (stiffness and damping for each Cartesian degree of freedom), we can define the robot's response to encountering an obstacle. The force applied by the motors is calculated on the basis of Hooke's law:

$$F = k \cdot \Delta x \tag{4.5}$$

An external force applied to the system causes the deviation of the robot's actual position from the planned path. The lower value of stiffness k leads to greater deviations Δx.

Chapter 5

Computer vision

5.1 Test of two CV systems

The machine vision task is to recognize the state of the playing board, inspect the presence and positions of all checkers, and determine whether the robot's opponent completed a move. Two types of computer vision systems were tested during the project development.

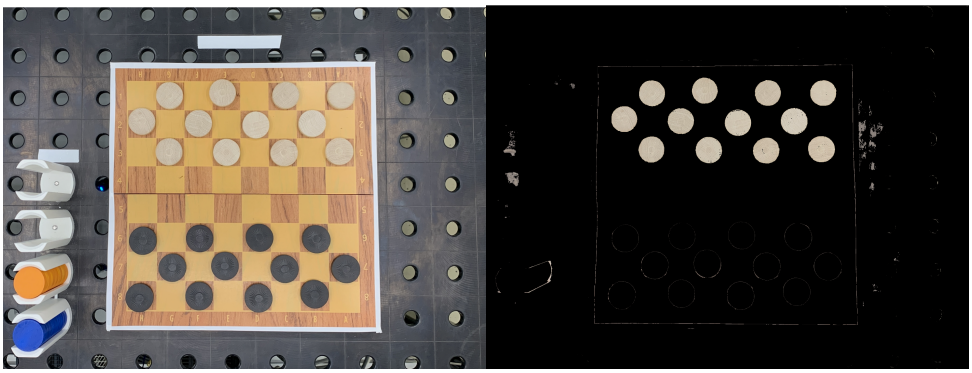
5.1.1 Industrial camera and PC

As the first prototype of the computer vision system, I used an industrial camera Basler ace A4024 to capture the image and a computer for subsequent image processing. Computer vision tasks were performed using the Python programming language and the OpenCV library.

Checkers detection

The challenge of detecting checkers lies in finding circles of a certain diameter and distinguishing their color. The algorithm I used for this task is based on detecting the edges of the objects found in the image.[16]

The first step of the image processing is to convert an image from the RGB model to a binary representation, where the objects of a desired color (e. g. white checkers) are white and the rest of the objects are black. As there are two colors of checkers, two corresponding binary images are needed for both of them. To do so, the HSV filter is applied to the image. In the HSV representation, the picture is divided into three channels: hue, saturation, and brightness (value). The filter specifies the tolerance for each of these channels. In the binary image retrieved after the filter application, all pixels that fall within this tolerance are highlighted in white, while the rest of the pixels appear black.



(a) : Original raw image. **(b) :** Image with applied HSV filter.

Figure 5.1: Demonstration of a HSV filter application for a white color extraction.

For a convenient filter creation, I used a Python script that generates a GUI window with six sliderbars to adjust the mentioned parameters. The desired HSV values for a specific color are tuned only during the development stage and are then stored in the main image processing program. To consider possible lighting conditions in the robotic cell environment, I used a number of images taken under different lighting conditions while establishing filter values. The results are shown in Table 5.1.

Color	H	S	V
White	0 - 23	23 - 59	0 - 220
Black	0 - 134	0 - 33	0 - 116

Table 5.1: Tolerance value ranges for all three channels of HSV filter for two colors used in the game: white and black.

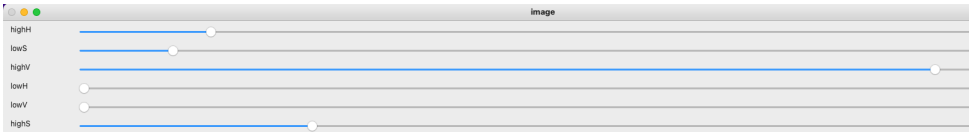


Figure 5.2: Slider bar for configuring filters in HSV domain.

The obtained binary image typically contains some noise that contaminates the picture and should be reduced for further image processing. A typical noise reduction approach is provided with morphology transformations that take into account the spatial structure of objects in a picture[16]. There are a number of various morphology operations with distinct functionalities, but each of them examines an area around each pixel defined by the structuring element and adjusts it according to the corresponding operation. The structuring element can be compared to an image kernel sliding through an image, and the concept of morphology transformations is similar to applying a convolution to an image, with the difference that only simple tests on the pixels are performed.

The morphology transformations are based on two fundamental operations with directly opposite functionalities: Erosion and Dilation[17]. While the idea of Erosion resembles soil erosion and diminishes the foreground of a picture, the Dilation does the opposite, increasing the size of foreground objects, thus expanding the white regions. These two basic operations form the two types of morphology procedures used for the current problem of noise reduction: Closing, which is a consistent application of dilation and erosion, and Opening, which is an erosion followed by dilation. In other words, the Closing removes black noise from the white objects, and the Opening removes white noise on the black objects. The corresponding OpenCV function is `cv2.morphologyEx(src, operation, kernel)`. The parameter `src` contains the binary image, `operation` is assigned to `cv2.MORPH_CLOSE` or `cv2.MORPH_OPEN` respectively to the application of the function, `kernel` defines the structuring element, which is set to 5x5 size matrix filled with ones.

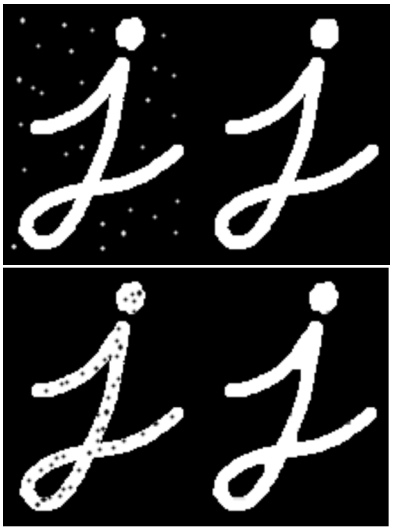


Figure 5.3: Comparison between two morphology operations: Opening (the upper image) and the Closing (the bottom image). [2]

(a) : Original picture.

(b) : Erosion application.

(c) : Dilation application.

Figure 5.4: Comparison between two basic morphology operations: Erosion and Dilation. [2]

The next step is to apply the function `cv2.findContours(image, mode, method)` to the enhanced image. As the name proposes, the function detects objects' contours, i. e. curves that connect all continuous points along the border of the same hue or intensity. The `mode` of the contour retrieval algorithm is set to `RETR_LIST`, thus the contours are retrieved without any hierarchical relationships being established. The contour approximation `method` is set to `CHAIN_APPROX_SIMPLE` for memory optimization. By compressing horizontal,

vertical, and diagonal segments, the function using this method removes all redundant points leaving just edges of the segments. The function outputs the detected contours, each being represented as an array of (x,y) coordinates of the object's edge points. Among the contours obtained, I look for those with the desired area. As the relative position of a checkerboard and a camera does not change over time, the diameter of a checker remains within the constant range of values. In order to be considered as a circle, a detected contour must contain no less than twenty edges.

To find the center of the detected circle, the edges of the corresponding contour are first fitted into a rectangle with the function **`cv2.boundingRect( contour_points)`**, that returns the rectangle's width w, height h, and coordinates of its upper left corner x_l, y_l. Then the coordinates of the circle center x_o, y_o are calculated with the help of basic geometry:

$$x_o = x_l + \left|\frac{w}{2}\right|, \quad y_o = y_l + \left|\frac{h}{2}\right| \tag{5.1}$$

To determine whether a checker belongs to a certain position on the checkerboard, the position of the circle center is compared with respect to 14 lines dividing the plane into 64 regions. Visualization of the image processing result is shown in Figure 5.5.

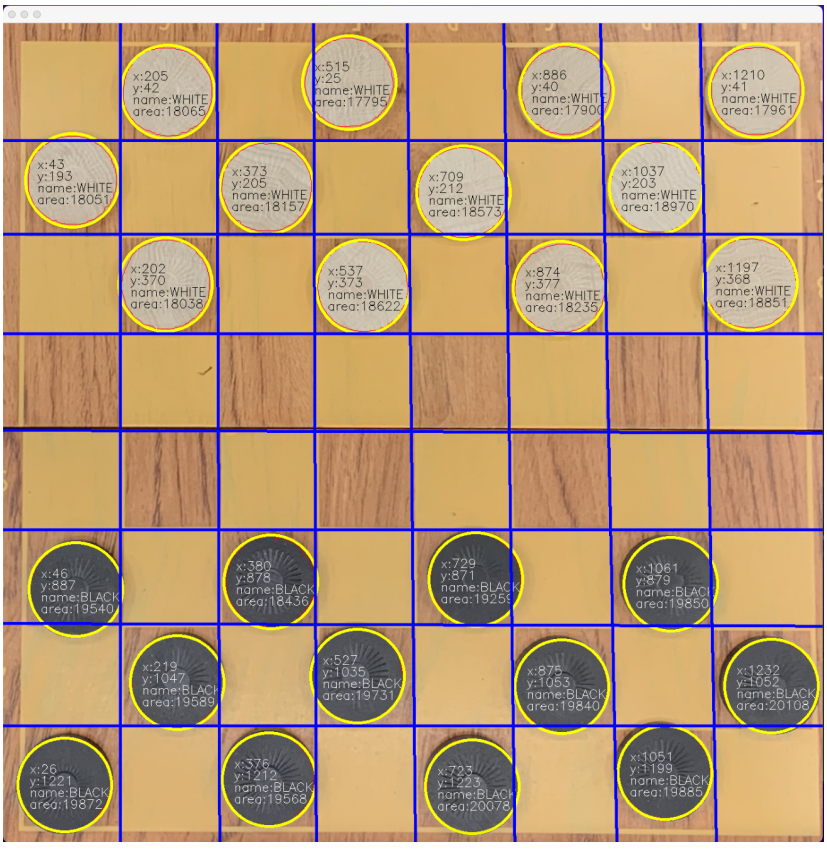


Figure 5.5: Circle recognition outcome. Detected circles are highlighted in yellow and contain information about the piece's color, area and (x,y) coordinates in respect to pixels.

Depth estimation

According to the Czech version of checkers, when a regular piece becomes a king, its height is doubled with an additional checker. This fact extends the functionality of the CV program with the task of determining objects' height.

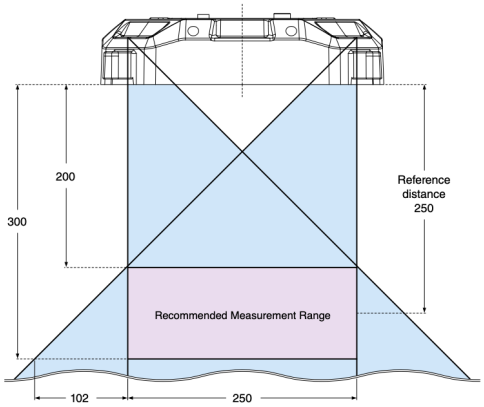
There are many deep learning algorithms to estimate the depth of a single monocular image. Most of them are based on modern convolutional neural networks (CNNs), which are fast enough for real-time image processing while achieving high-accuracy results. [18] However, these methods are only applicable to images where a human is able to recognize the objects' geometric perspective or correlate the type of an object with its distance from the camera, e.g., to determine that clouds in the sky are far away but the grass underfoot is close. On a photograph of a checkerboard plane taken perpendicularly, it is impossible to visually distinguish the height of the checkers. This task requires different solutions, e. g. a 3D reconstruction with structured light. In the absence of a device enabling 3D reconstruction[19], the kings in the game can be distinguished from regular piece in a different way, e.g. by an additional color.

5.1.2 Keyence Vision System

After the test of the first computer vision system, I got the opportunity to test another setup - newly arrived Keyence CV-450 Vision System (KVS). This type of computer vision system is a typical industrial solution. It includes a high-resolution industrial camera, a lighting unit, and a computing unit with a built-in CV system. Initially, this machine vision option had several advantages over the first one. Not only does the KVS have more advanced CV capabilities and a more reliable setup, but it also offers a 3D shooting mode. The possibility of measuring objects' heights would allow me to adhere to the standard way of a checker coronation by increasing its height with an additional checker.

Depth estimation

According to the user manual[20], in the 3D capture mode, the recommended distance to an object to measure its height is ranging from 200 to 300 mm from the vision system. However, due to the workplace layout, it would not be possible to place the camera at a distance of 30 cm above the game board; otherwise, the lighting panel will block the view for a human player. Nevertheless, I examined the detecting capabilities of KVS at a longer distance to check whether the camera combined with the structured light is able to differentiate the ordinary checkers from the kings.

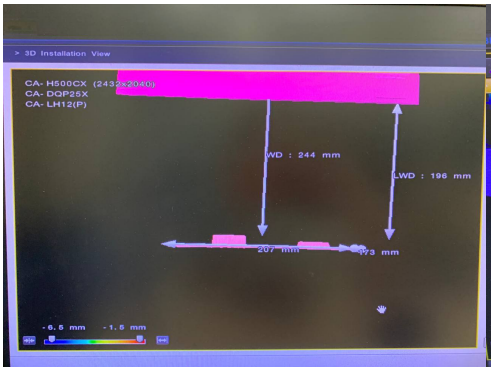


(a) : Recommended arrangement according to a manual.

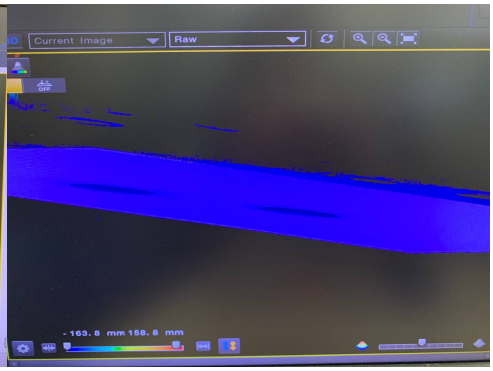
(b) : Actual arrangement.

Figure 5.6: Experiment setup for estimating the heights of two objects with CV system.

The experiment was carried out at two testing points: precisely in the recommended area at a distance of 25 cm from the camera and in a desired area at a distance of 100 cm from the camera. The experiment setup included two elements: one checker and two checkers placed on each other, representing a regular piece and a king, respectively. During the experiment, I attempted to impact the sensitivity of the system by applying different lighting and recalibrating the camera under various conditions. However, the results shown in Figure N demonstrate that the accuracy of the depth measurement was not sufficient to distinguish between the compositions of one or two checkers at the second test point.



(a) : Camera is placed 25 cm from the surface, where the objects are placed. View of a XZ plane.



(b) : Camera is placed 100 cm from the surface, where the objects are placed. View of a XZ plane.

Figure 5.7: Depth measurement in 3D capturing mode.

Circle detection

However, even in the two-dimensional capturing mode alone, the Keyence system includes an extensive range of functionalities for detecting objects or their properties. The principle of most of them is based on taking a sample of the desired result, the so-called ground truth, when initializing the function and then comparing the subsequent samples with the desired one. When applying the "Pattern Detection" function, for example, two regions in the image are selected: the one having the desired pattern and the one where the pattern must be subsequently searched. The function's output is then the current photo's resemblance to the ground truth expressed as a percentage.

The KVS has a built-in function for a circle detection. It's operation principle is similar to the algorithms I implemented using the OpenCV library, which are based on the edge information after detecting multiple edges in the inspection region. As the checkerboard is fixed on a platform, the coordinates of the dark squares, where the pieces can be placed, are known in advance. Therefore, the circle detecting function examines 32 regions - dark squares - and outputs 32 boolean values representing the presence of a circle in a particular area. If the circle exists, then the color detection function checks the region's dominant color by comparing the sample with four defined colors used in the game: white, black, orange and blue.

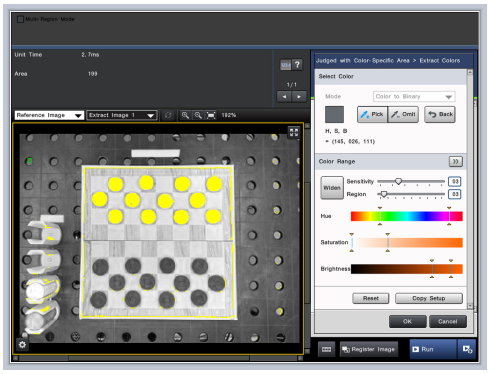


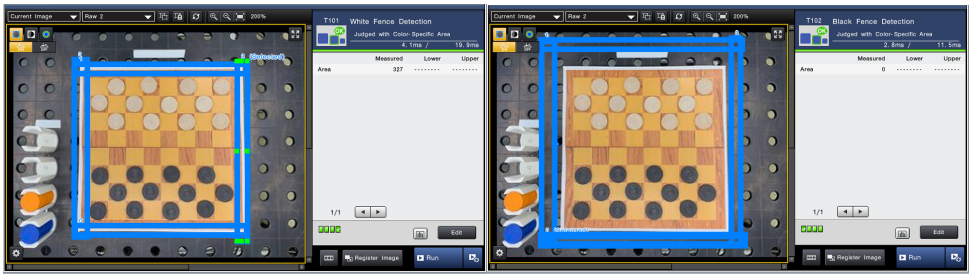
Figure 5.8: Visualisation of the white color detection. The color is not defined by a precise value, but rather with a range of values in HSV color model. The regions with desired hue are automatically highlighted on a gray-scale image.

By processing the image directly on the camera's computing unit, the need for a separate computer is eliminated and communication is simplified. The PLC receives only information about the presence of checkers on the defined positions and information about their colors. The communication between KSV and PLC is carried out via Profinet.

Move completion detection

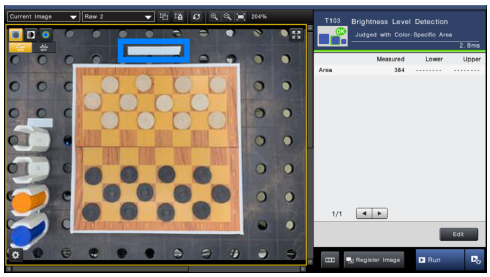
The camera captures a photo only when there is nothing blocking the view of a game board. Detection of the foreign objects above the board is based on the principle of a laser fence. When a foreign object intersects the restricted area, it interrupts the laser ray on its way from the source to a detector, thus triggering the laser fence. This project has no physical laser, but strips on the table surface simulate its principle. Two highlighted frames surround the studied area of the game board; each of them is used to measure the

concentration of pixels of a specific color within the frame borders. Each frame has a rectangular shape and consists of 4 separate rectangles. The inner frame measures the amount of white color in itself, and the outer one measures the amount of black color. If the amount of pixels in the highlighted area is not sufficient to reach the defined threshold, then there is something blocking the view from the camera. The use of two fences, rather than one, is necessary to define contrasting colors and avoid the situation where, for example, a person playing checkers will wear clothes with sleeves in the color of the fence.



(a) : The inner frame of white color.

(b) : The outter frame of black color.



(c) : A region of white color used as a reference of the amount of light present in the environment.

Figure 5.9: Definition of areas used for detecting a hand above the board.

To eliminate false negative scenarios, when there are no foreign objects between the camera and the board but the number of pixels is still insufficient due to dim lighting, I integrated a dynamic threshold definition. The threshold value that determines the absence of foreign objects depends on the lighting conditions. The number of pixels of one color is directly proportional to the illumination intensity of the environment and can be expressed as a linear relation.

To represent this dependency with a function, I performed ten measurements in ten different lighting settings and noted the number of pixels in each highlighted area. The level of intensity of illumination is represented by the number of pixels in the highlighted area in Figure 5.9c The required function was then found using linear regression.

```
#THOLD_Left_White   := 4.51 * "Brightness_Level" - 60031.0;
#THOLD_Right_White  := 2.27 * "Brightness_Level" - 21279.0;
#THOLD_Bottom_White := 4.34 * "Brightness_Level" - 60555.0;
#THOLD_Top_White    := 3.25 * "Brightness_Level" - 35021.0;

#THOLD_Left_Black   := 0.61 * "Brightness_Level" + 50047.0;
#THOLD_Right_Black  := 0.11 * "Brightness_Level" + 51034.0;
#THOLD_Bottom_Black := 0.58 * "Brightness_Level" + 58084.0;
#THOLD_Top_Black    := 1.29 * "Brightness_Level" + 38150.0;
```

Figure 5.10: Threshold value definition for every side of two rectangles - inner white one and black outer one.

If the measured amount of pixels in the studied area is bellow the calculated threshold value for the current lighting conditions, then there is something blocking the view of the checkerboard in that area.

5.2 Final choice of a CV system

After comparing two computer vision systems, I can conclude that despite the impossibility of using 3D shooting mode in the current workplace layout, the Keyence unit still has a number of advantages over the first option. With the Keyence system, I achieved better results in object detection in a shorter amount of time than with the first mentioned setup. This system is more resistant to changes in the surrounding environment and less sensitive to changes in lighting. For a project that is not placed in a laboratory with constant conditions but in a real dynamically changing environment, it is a rather critical parameter. Another undeniable advantage is the ability to exclude one of the system components, a computer, thereby optimizing the system architecture. In addition, the Keyence Vision System is a typical tool for performing computer vision tasks in industry.

Chapter 6

Application structure

The software application of the robotic workplace consists of four components running on four devices which have their own tasks:

- PLC: manages the state machine of the application,
- HMI screen: provides the user interface,
- KUKA iiwa controller: handles game core and robot's motions,
- Keyence CV system controller: conducts image processing.

6.1 User interface

One of the points of a human-robot interaction is the user interface (UI). In this project, the UI is implemented in HMI and is designed to be straightforward and consistent. The HMI is used to start a new game or to end the one that is ongoing, to keep track of the state of the board, to receive notifications of wrong moves, and to monitor the conditions of the safety system. The sequence of screens is shown in Figure 6.1. Each screen intentionally contains only one piece of information and avoids unnecessary elements, so its content is convenient to interact with for the general public. The user experience starts with the choice of their pieces' color. The double arrows represent the reversibility of actions. The content and the sequence of the screens were programmed in STEP 7 Tia Portal.

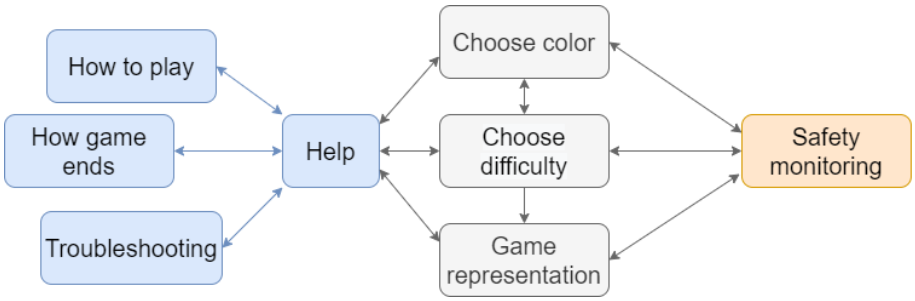


Figure 6.1: User interface diagram, where every block represents a screen on HMI.

6.2 State machine

Once the human player declares the beginning of the game through HMI, the system is set to the initial state. The simplified version of the state machine is shown in Figure 6.2. The whole process is handled by a PLC that keeps track of the current state and sends command requests to other devices.

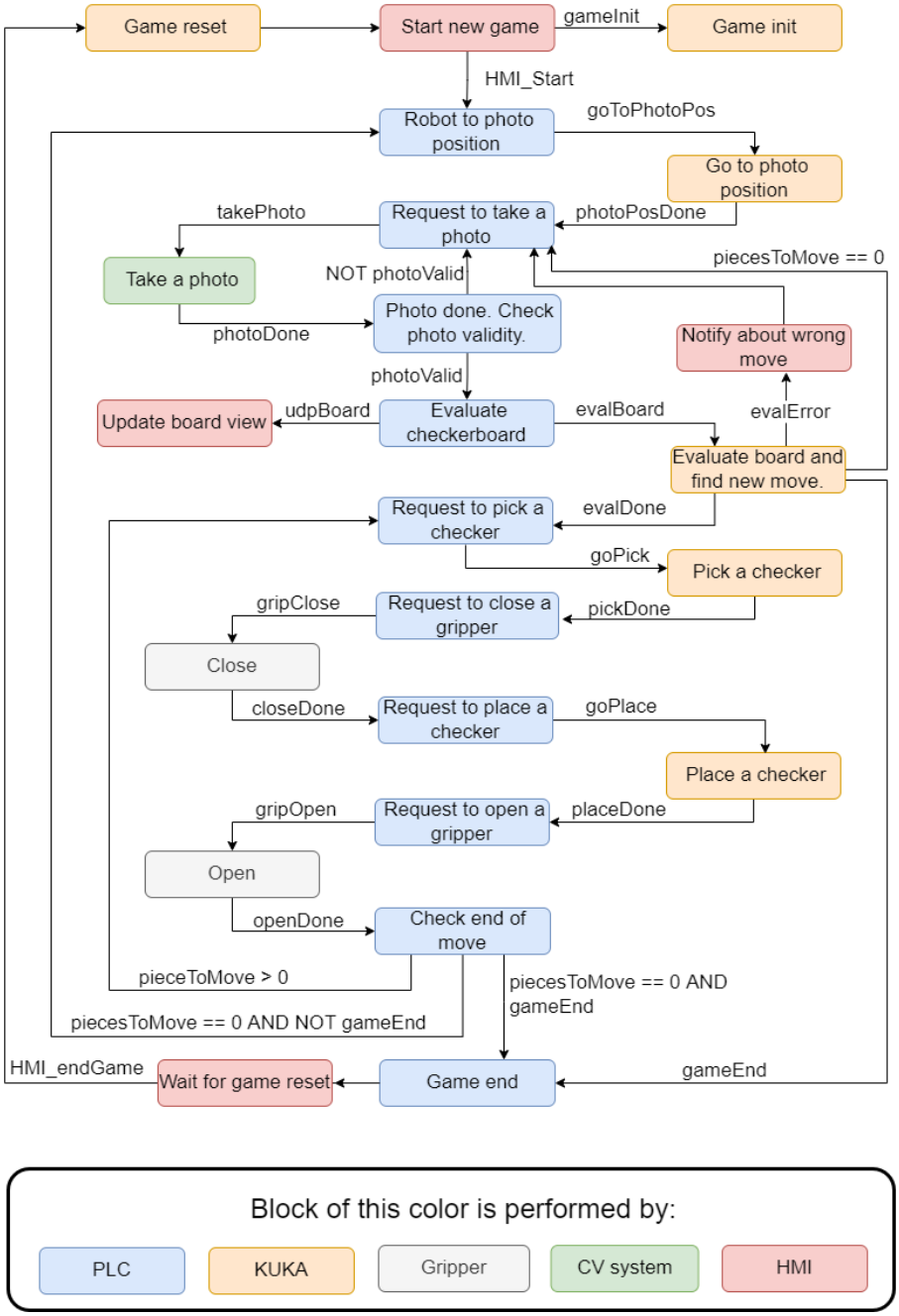


Figure 6.2: State machine diagram.

6.3 PLC program

The main organization block of the PLC program consists of four function blocks:

- `State_Machine`. As the name suggests, this function block keeps track of the current state of the system according to Figure 6.2, assigning output signals and monitoring input signals from other devices and other function blocks.
- `EGL_Control` is an implementation of the state machine for the gripper control that includes states "reset", "initialize", "grip" and "release". `Force_parameter` defines the desired gripping force, `Position_parameter` corresponds to the distance between the fingers, `Command` defines the gripper's state. The code of this block was provided by my colleague and more detailed description on it can be found in his work [4].
- `Camera_Control` After receiving a positive value of `TakePhoto`, the request for an image update is sent to KVS. After receiving the output of the image processing, the function checks if the image is valid (the laser fence was not triggered by foreign objects) and acknowledges the completion and validity of the image.
- `HMI_Control` monitors the state of the HMI screen, resets the game (by setting the state of the system to the initial value), and initializes a new game (by assigning the default values to the internal variables) as a response to a corresponding input signal from the user.

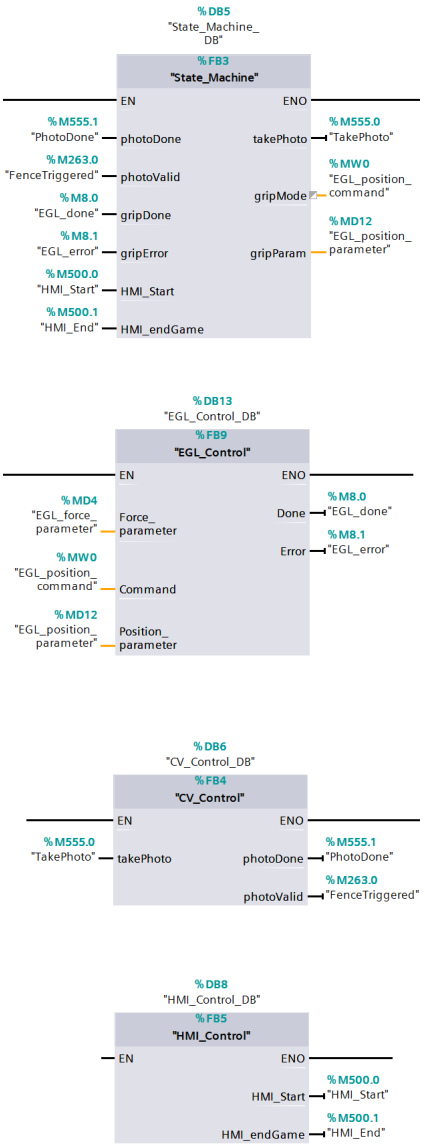


Figure 6.3: Function blocks of the PLC program.

6.4 Java application

The handling of the checkers game is implemented in the robot application. It was developed in KUKA Sunrise.OS, and it runs by KUKA Sunrise.Cabinet and KUKA iiwa itself. Initially, when the project's architecture included a computer, the idea was to implement the game application in Python programming language and run it together with the CV program. However, after the replacement of the CV system, the need for an external computer was eliminated. The tests of running the game demonstrated that the computational power of the robot controller is sufficient for a real-time decision making. Furthermore, the KUKA Sunrise.OS enables object-oriented programming using the Java language, which is a typical and convenient tool for game development.

The robot application consists of six Java classes: **`Checkers`**, **`Game`**, **`Board`**, **`Piece`**, **`AI`** and **`Moves`**.

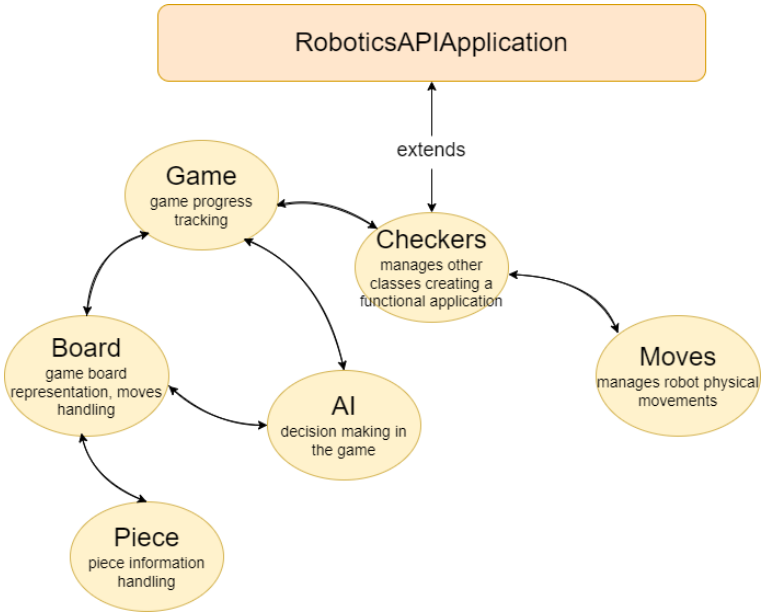


Figure 6.4: Java application diagram representing relation between the classes.

- Class: Checkers

  This class extends the KUKA **`RoboticAPIApplication`** and inherits its features. This is the main class of the Java application and the link between the rest of its components. Its methods provide communication with the PLC at the application layer, receiving information about the current state of the system, executing a corresponding task, and sending back acknowledgments of the action taken.

- Class: Piece

  As the name suggests, this class stores essential information about a

checker. The main methods provided in this class enable turning a piece into a king, changing the piece's characteristics, finding valid moves for this piece in a given board setup, and calculating a score for the found valid move. The score value is directly proportional to the number of opponent pieces captured during the robot's best move.

- Class: Board

  The game board is represented with a 2D array sized 8 by 8 of the Piece data type. The instance of this class provides methods for obtaining possible moves for a specified player; making an abstract move while finding the optimal one with AI; examining if one of the player cannot move or has lost the game.

- Class: Game

  Each new game launch creates a new instance of the **`Game`** class. This object keeps track of the number of both black and white pieces and kings, contains current and previous game-board states, and provides coordinates of the next robot's move. This class contains six main methods:

  - `initGame()` initializes a new game by setting all variables to their default values.
  - `fillCurBoard()` updates an instance of the Board class *`curBoard`* that represents the current layout of the game. The method converts the array received from the PLC to the current application convention. The checkerboard representation is stored in the PLC program as an array of 32 values (although the board consists of 64 squares, a checker can be placed on only 32 squares) ranging from 1 to 4, where 1 corresponds to a regular white checker, 2 to a black checker, and 3 and 4 to white and black kings, respectively. This method creates a corresponding checker for each cell on the board and adds it to *`curBoard`*.
  - `analizeBoard()` receives a current state of the game board from and evaluates possible moves for a robot.
  - `validPlayersMove()` inspects the completion and correctness of the opponent's move.
  - `findMoveAI()` sets the difficulty of the game and initializes the AI function for searching for a best move.
  - `makeMove()` when an appropriate move is found, fills the queues with the sequence of robot actions needed to make this move. Queues are arrays that contain the coordinates of the positions from which the robot picks up checkers and where it places them. The arrays are stored in the static ArrayLists, so they are accessible from other classes.

Other methods are auxiliary components of the main methods and are separated from them for a clear code arrangement, therefore are not worth mentioning here.

- Class: AI

  The AI class contains an implementation of the Minimax algorithm with the Alpha-Beta Pruning described in the previous section. The *`alphaBeta()`* method calculates the best move for the current state of the game by analyzing all the possible states of the game after N moves. The number of moves N is defined by the depth of search that corresponds to the chosen difficulty of the game.

  The "Easy" level corresponds to the depth equal to 3.

  The "Medium" level to the depth equal to 5.

  The "Hard" to the depth equals 7.

  The method returns the score of the current game state and stores coordinates of the position. After finding the optimal move, the method stores coordinates of the positions from where the robot's piece should be picked and where it should be placed in static *`bestMoveStart`* and *`bestMoveEnd`* respectively.

- Class: Moves

  This class provides methods for performing moves of the robot mentioned in section 4.2. The movements consist of sequential linear motions. The class also sets the Cartesian impedance control over the robot with functions provided from the `roboticsAPI` from KUKA.

Chapter 7

System tests

7.1 Final tests

The purpose of this robotic cell is to be actively used by the general public. Therefore, its functionality and performance should be adequately tested before launching the product to be available to the general public.

Although most developers face such challenges while testing their robotic systems, as the lack of confidence in the accuracy of simulation tests[21], the problems that I have encountered are slightly different. The specificity of this project does not contribute to the automation of the testing process, which requires human participation. Therefore, practices such as automated software tests could not be integrated. This aspect makes testing the same situation numerous times in a row time-consuming. For testing purposes, I invited a group of subjects. The group consisted of ten people aged 20 to 55 years with different backgrounds. While most of them were familiar with robotic systems, there were 4 people who had never interacted with robots before.

Type of test	Description
Algorithm performance	Examines the robot's winning rate
Robustness testing	Determines how the system responds to unpredicted scenarios.
User interface clarity	Demonstrates the convenience of interaction with the robotic cell
Compliance with the game rules	Inspects if the game core of the robotic application contains errors

Table 7.1: Categories of the performed system tests.

The overall testing process began in the development stage of the project and lasted several months. However, this section provides data only from the final testing.

Compliance with the game rules

At the time of the final testing, the robot did not perform any illegal moves that did not comply with the game's rules. On the contrary, the robot recognized all of its opponent's incorrect moves.

Algorithm performance

In order to study this category, the group of subjects played 30 games. None of them was a professional checkers player, but everyone was familiar with the rules and had played the game before. Each tester was asked to play three games in all three difficulty levels. Results are shown in Figure 7.1. As anticipated, an inexperienced player cannot predict all possible states of the game 5 or 7 moves ahead and therefore can only win in "easy" difficulty mode.

However, the draw rate turned out to be higher than was alleged. The heuristic function of the Minimax algorithm does not take into account a draw situation that takes place when the board state remains unchanged three moves in a row. After discovering this feature some players kept this strategy while playing in a harder mode, so they would at least aim for a draw, thus increasing the draw rate.

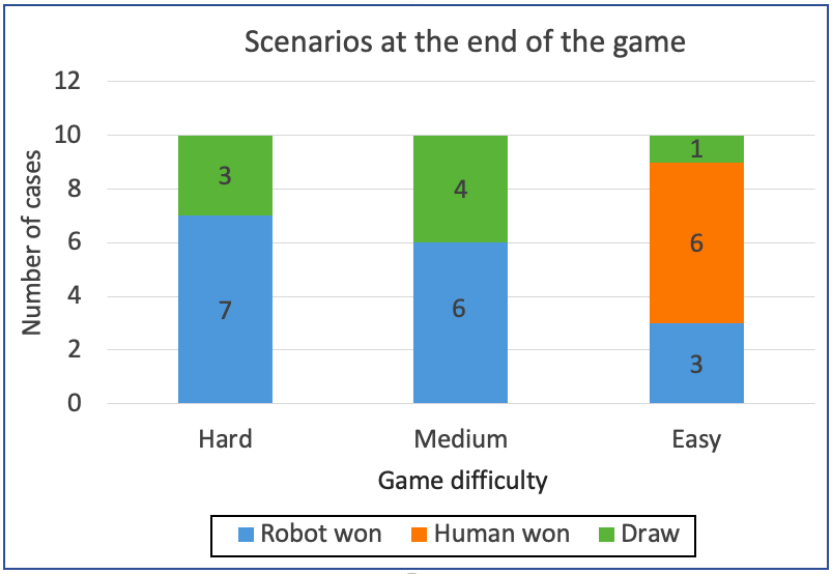


Figure 7.1: Statistics on winning rate from the 30 game outcomes.

Robustness testing

Each participant was asked to perform unpredictable actions that could possibly entail the system malfunction, but without causing serious damage to the robotic cell. I outlined three of the most common "damaging" strategies that were applied:

- **Illegal checker placement**. Every tester tried to confound the robot by removing some checkers from the board, placing extra pieces in empty slots, or moving a piece to a white square where it cannot be placed. In these cases, the game application identified wrongly placed checkers as invalid moves and notified about the mistake.
- **Disruption of normal CV system function**. Some testers tried to block regions of the checkerboard from the camera view with their hand. These attempts did not succeed, as the presence of hand triggered the fence around the board, so the board layout was rejected by PLC and was not even sent for evauation to KUKA. The other participants attempted to distract the CV system by placing foreign objects on the game board. If the objects were of one of the colors used in the game, the CV system occasionally identified them as checkers, but in most cases the game application discerned the invalid move.
- **Changes in lighting conditions**. The robotic cell showed the worst performance with significant changes in lighting conditions. Four participants placed a flashlight close to the checkerboard, causing an image overexposure. The CV system was unable to detect any objects in the over-illuminated region. In the opposite scenario, with extremely insufficient lighting, the system also performed erratically and struggled to detect checkers.

User interface clarity

According to the survey participants, the setup and start of the game with HMI were intuitive and convenient. However, some have encountered the problem of returning the robot to working condition after releasing the red safety button. Initially, a player had to open a safety conditions monitoring screen and resume the game. The additional information on the state of the robot safety system was overwhelming and irrelevant. After receiving the feedback, I substituted the process of confirming the application resumption with the need to press only one button on the main game screen.

7.2 Field testing

The robotic cell was displayed at the opening of the newly renovated RICAIP Testbed for Industry 4.0 at the Czech Institute of Informatics, Robotics, and Cybernetics Czech Technical University in Prague (CIIRC CTU). Around twenty people interacted with the robot during the three-day event and played at least one game. In some cases, visitors did not have enough time to finish the game and left the site after completing a few moves. The next version of the project could be supplemented with a smaller version of the board, e.g., sized five by five, which would be used for demonstrations at such events. The statistics of the game outcomes were similar to those obtained earlier. Most games at the easy level ended with the victory of a human; at the other

levels, with his defeat. Unlike previous tests, the group of subjects in this case included a professional chess player who managed to win at the average level of difficulty in a relatively short time. Among the average players, there were two children under the age of six whose behavior was quite challenging to predict. Despite their attempts to deceive the robot or disrupt the system's stability, the game cycle proceeded as intended.

Chapter 8

Conclusion

The goal of this Bachelor thesis was to design and develop a robotic workplace for human-robot collaboration for playing checkers.

The created robotic workplace meets all the requirements. The overall architecture of the designed system enables the robot to play the game autonomously. The concept of this robotic workplace was kept to the idea of a "plug and play" system, whose functionality does not require any manual setups or any operator involvement. The launch of the game takes only two button presses on the touch panel. A player can choose between two checker colors and three difficulty levels. The robot is set in collaborative mode, so the risks of accidents caused by a collision of a robot with a human are negligible. In case of an unexpected behavior, the robot can be stopped immediately with an emergency button.

The implemented game algorithm allows the robot to be fully independent in fast decision making while looking for an optimal next move. The robot workplace tests have shown great performance in interacting with the general public. The choice of parameters for the heuristic function of a Minimax algorithm was proper, and thus the AI player was almost invincible in difficulty modes higher than the easiest possible.

The principle of the Human-Robot collaboration implemented on the example of the checkers game can later be integrated into any other application of industrial robots.

Appendix A

Bibliography

[1] KUKA Roboter GmbH, *KUKA Sunrise.OS 1.14, Operating and Programming Instructions for System Integrators*, 2017.

[2] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[3] B. S. Sergi, E. G. Popkova, A. V. Bogoviz, T. N. Litvinova, and E. Insight, *Understanding industry 4.0: AI, the Internet of things, and the future of work*. Bingley, UK: Emerald Publishing Limited, first publish ed., 2019.

[4] T. Jochman, "Design of a workplace with two collaborative robots," 2020. Master's Thesis.

[5] "Lbr iiwa in action combating coronavirus." https://www.kuka.com/en-de/company/press/news/2020/06/robot-helps-with-coronavirus-tests. Accessed on 2022-04-04.

[6] J. SCHAEFFER, N. BURCH, Y. BJÖRNSSON, A. KISHIMOTO, M. MÜLLER, R. LAKE, P. LU, and S. SUTPHEN, "Checkers is solved," *Science (American Association for the Advancement of Science)*, vol. 317, no. 5844, pp. 1518–1522, 2007.

[7] K. Oxland, *Gameplay and design*. Addison-Wesley, 2004.

[8] "Rules of draughts (checkers)." https://www.wcdf.net/rules.htm. Accessed on 2022-04-04.

[9] G. Owen, *Game Theory*. Emerald Group Publishing Limited, 2013.

[10] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[11] E. Lasker, *Chess and Checkers: The Way to Mastership*. Blackmask Online, 2002.

[12] Wikipedia contributors, "Minimax — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Minimax&oldid=1076761456, 2022. [Online; accessed 1-March-2022].

[13] Wikipedia contributors, "Alpha–beta pruning — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Alpha%E2%80%93beta_pruning&oldid=1075297799, 2022. [Online; accessed 1-March-2022].

[14] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 ed., 2004.

[15] T. Pajdla, *Elements of Geometry for Computer Vision and Computer Graphics*. 2021.

[16] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[17] N. Jamil, T. Sembok, and Z. Bakar, "Noise removal and enhancement of binary images using morphological operations," vol. 3, pp. 1 – 6, 09 2008.

[18] H. Ibrahem, A. Salem, and H.-S. Kang, "Dts-depth: Real-time single-image depth estimation using depth-to-space image construction," *Sensors*, vol. 22, no. 5, 2022.

[19] Z. Ma and S. Liu, "A review of 3d reconstruction techniques in civil engineering and their applications," *Advanced Engineering Informatics*, vol. 37, pp. 163–174, 08 2018.

[20] *Keyence Intuitive Vision System CV-X Series. User's Manual*, 2021.

[21] A. Afzal, C. Goues, M. Hilton, and C. Timperley, "A study on challenges of testing robotic systems," pp. 96–107, 10 2020.