NTNU – Trondheim
Norwegian University of
Science and Technology

Magnus the Chess Robot

Tobias Helstad Unger

Master of Science in Cybernetics
Submission date: July 2014
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Board games can provide a good testing environment for robotics as it often requires high accuracy and human-robot interaction. With chess gaining massive popularity in recent years, there is as interest in investigating the possibility of creating a low-cost robotic chess player. The task will be focused on creating a cheap and reliable conceptual prototype for entertainment and educational purposes. The system will primarily be focused on playing chess, with possible adaption to other board games. The task will consist of:

- Get familiarized with relevant background theory
- Evaluate existing solutions if some exist
- Propose solutions and design of system
- Implement proposed system if time permits
- Test and evaluate implemented system

Summary

The goal of this thesis was to create a low-cost chess-playing robot to help progress the field of small-scale robotics and make it more available for personal use. To achieve this, methods for moving pieces on a chessboard has been evaluated, resulting in the design of a four positional degrees-of-freedom elbow manipulator arm. Of-the-shelf servo motors were chosen as the arm's actuators, providing closed-loop control contained within each motor. Control of the motors was interfaced through a dedicated servo controller, consisting of both a microcontroller and interfacing software written in C++. An inverse kinematic based open-loop robot controller was made to control the overall movements of the arm and ensure that the arm avoided any potential obstacles in its workspace.

To help the robotic arm with perceiving its surroundings, a wide selection of sensory configurations and computer vision algorithms were evaluated. The sensory unit was implemented by capturing images with a Microsoft LifeCam Studio camera and using OpenCV to process the captured images. The chess engine was not implemented as part of this thesis, therefore the work done on this part has mainly been to find a suitable chess engine and incorporating it into the implemented system.

Based on the results obtained, a complete chess-playing robot was created. The robot performed with a relatively good accuracy and repeatability, but will require some improvements to increase the stability and smoothness of motion. The computer vision results exhibited a good degree of robustness, allowing the robotic arm to act fully autonomously. Some obstacles were not overcome, specifically the sensory unit did not provide the robot controller with the chessboard coordinates necessary to allow manipulation on chessboards of arbitrary size and orientation.

The goal of proving the possibility of making small-scale robotics more available for personal use is believed to be achieved. With some further work on the chess robot, it is also believed that the robot could be used for entertainment and to inspire future student into taking up a study in robotics.

Sammendrag

Målet med denne avhandlingen var å lage en lavkostnads sjakkspillende robot for å fremme hobbyrobotikk og gjøre det mer tilgjengelig for personlig bruk. Målet er forsøkt oppnådd ved å evaluere forskjellige metoder for å flytte sjakkbrikker, som resulterte i konstruksjonen av en manipulatorarm med fire frihetsgrader for posisjonsstyring. Butikkjøpte servomotorer ble brukt for bevege de forskjellige leddene i armen. Dette sørger for lukket kontroll av hvert ledd da servomotorene kommer med innebygget posisjonsstyring. Styring av servomotorene ble utført av en dedikert servokontroller som inkluderer både en mikrokontroller og programvare for å sørge for at styringen kan utføres gjennom C++. Styring av robotarmen som en helhet ble utført av en robotkontroller basert på inverskinematikk. Robotkontrolleren sørger også for at armen beveges så den unngår å krasje med potensielle uønskede objekter i omgivelsene.

Forskjellige varianter av sensorer ble vurdert for å avlese nødvendig sjakkinformasjon mens spillet pågår. Sensorenheten ble implementert ved å ta bilder med et Microsoft LifeCam Studio-kamera og prosessere bildene med datasynsbiblioteket OpenCV. Sjakkmotoren har ikke blitt implementert som en del av arbeidet i denne avhandlingen, dermed har arbeidet rundt dette vært fokusert på å finne en passende sjakkmotor og tilpasse den for å passe det implementerte systemet.

En automatisk sjakkspillende robot ble laget basert på de oppnådde resultatene i de forskjellige modulene. Robotarmen viste seg å ha gode egenskaper når det kom til både presisjon og gjentakelsen av tidligere utførte bevegelser. Det gjenstår enda potensielle forbedringer for å øke stabiliteten og få armen til å utføre jevnere bevegelser. Det ble oppnådd relativt robuste resultater med de implementerte datasynmetodene, som resulterte i at sjakkroboten fikk spilt flere runder sjakk uten menneskelig innblanding. Det største gjenstående problemet er å implementerte en metode så datasynenheten gir robotkontrolleren de nødvendig sjakk-kordinatene som er nødvendig for å spille på sjakkbrett av vilkårlig størrelse og plassering.

Muligheten for å bringe hobbyrobotikk inn i hverdagen og gjøre det mer tilgjengelig for personlig bruk anses som bevist ved denne avhandlingen. Med videre arbeid på den sjakkspillende roboten vil det også være mulig å benytte den direkte både som underholdning og inspirasjon til potensielle kybernetikkstudenter.

Preface

This project constitutes my master thesis that was completed during the spring semester of 2014 at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology.

I would like to thank my supervisor Amund Skavhaug for providing a large degree of flexibility when choosing the focus of my thesis, as well as providing important guidance during the course of this project.

I would also like to thank my family and significant other for providing both help and moral support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Robotics is a relatively immature field that has seen an immense amount of progress these past few decades. Initial advances in robotics were mostly motivated by industries that benefited from replacing humans in repetitive tasks and hazardous environments. With advances in computer processing power and the affordability of advance motors and sensory equipment, the use of autonomous robots can expand from purely industrial to performing household tasks and for entertainment purposes. With robots becoming more available for the general public, inventions such as robot toys and robot vacuum cleaners have seen increasing use over the years.

Robotics paves way for a new form of entertainment that allows humans to interact or play with artificial intelligent units without the use of computer screens. One such use saw its first inspiration with The Turk [1], a chess playing machine that proved to be not so autonomous after all. While The Turk ended up being nothing more than an elaborate hoax, the concept proved to be one of great interest to people. This interest reached a new high point when the chess playing computer Deep Blue [2] managed to beat the world champion chess player Garry Kasparov. Recent interest has been focused on bringing the power of chess computers into everyday life by the use of robotics.

Research into robotic chess players is one of many steps that can result in robotics being used in our everyday lives. The research is also partially motivated for studying the human-robot interaction [3]. This was the main goal of Gambit [4], a mid-cost chess-playing robot that was made to further study the human-robot collaboration. While the results obtained by Gambit are interesting with respect to its targeted study, the cost of the arm (ca $18K) makes it too expensive for a small-scale robotic chess player. Most affordable solutions require either specialized electronic chessboards, or require disjunctively colored chessboards and are constrained by the reach of the arm [5]. This thesis is motivated by an interest in making robotics more affordable and available to the general public. Specifically,

this is hoped to be achieved by studying small-scale robotics in a chess-playing environment.

1.2 Goal and Method

The goal of this thesis is to investigate the possibility of creating a robust and low-cost autonomous chess-playing robot. As few similar solutions exist to provide insight into low-scale chess automation, the main focus will be on developing a complete system based on known methods in robotics and computer vision. Specifically, the goal is to:

- Research and implement a chess recognition scheme using cheap and available cameras
- Design a robotic arm well suited for chess-playing and general small-scale robotics
- Research potential control schemes that is suitable for chess-playing and small-scale robotics

To accomplish these goals, extensive research into robotics and computer vision is necessary, as well as how to turn theory into a feasible practical solution. The various fields will initially be studied separately to ensure that progress is made on every aspect involved in the creation of a fully autonomous chess robot. This thesis will not cover the development of a chess engine as extensive research already exists on the field of chess computers. A suitable open-source chess engine will be located and integrated into the total system. Later parts of the project will be focused on making the various modules collaborate and hopefully providing results stable enough to test the chess-playing robot arm as a whole.

1.3 Outline of Report

This report is divided into two main parts. The first part is covered by chapter 2 and introduces the relevant background information and tools for creating a chess robot. The fundamentals of each theoretical field is included, therefore some of the information provided may seem too basic, however, it is included to provide the necessary information regardless of the reader's expertise. Sections 2.1-2.3 serve as a quick introduction into robotics and motors. Section 2.4 covers some basic computer vision methods that are relevant for a chess detection unit. Section 2.5 presents the core concepts of a chess-game while section 2.6 presents an overview of the tools and libraries used in the development of the chess robot.

The implemented system is described in chapter 3 with each module being described in its own section with section 3.1 serving as an overview of the implemented system. Section 3.2 introduces the design of a robotic arm, specifically made for this thesis. Section 3.3 and 3.4 cover the individual control of each motor and a control scheme for the entire robotic arm. The extensive vision based sensory unit

for the chess system is presented in section 3.5 and the main controller that handles communication and coordination between the various sub-systems is presented in section 3.6. Some discussion and testing is included in the various sections of chapter 3. This is to ease the reading of the thesis and present a reason as to why some choices were made.

Chapter 4 describes the results obtained with the proposed solution as well as some discussion of specific results and how different choices could have impacted the results. As with chapter 3, the discussion in chapter 4 is included to simplify the reading because of the large scope of the thesis. A general discussion of the main result and work method is presented in chapter 5, which also includes some suggestions for future improvements. Chapter 6 provides a quick recap of the results and serves as the conclusion of this thesis.

Chapter 2

Background and Research

2.1 Robotic Systems

This section provides a short introduction to robotic manipulators and how they can be made to act completely autonomously.

2.1.1 Robot Manipulators

A robotic manipulator is a mechanical construction that uses motors to perform some functionality similar to that of a human arm. It has seen wide spread use as a remote controlled arm to aid or replace human arms in tasks where the human arm is infeasible or insufficient. In recent decades a lot of the focus on robotic arms has turned to making the entire process automated, with a computer analyzing sensory data from the arm's surroundings and controlling the movements.

The mechanical construction of a robotic arm is in itself a complex field. Robotic arms are generally designed to be highly task specific, with limited use outside of its intended purpose. There will usually be some tradeoff between rigidness and weight of an arm, which will affect how smooth and accurate movements the arm will be capable of. Designing an arm with the concept of "one-size fits all" will be costly and redundant in most applications.

When designing a robotic arm, the first thing one must consider is the *workspace*. A robot manipulator's workspace is defined as the collection of points in space that are reachable by the end effector. Any points outside the workspace cannot be manipulated by the arm. To achieve the desired workspace, the robot's rotational or translational joints and links need to be chosen appropriately. Combining a series of joints through appropriately chosen link lengths one can achieve a workspace of any size or shape.

The workspace shows the positions the end effector is capable of in "free space", however most robotic environments will have some obstacles within that workspace. To help analyze the entirety of the arms geometry one must also consider its *configuration space*. The configuration space is the set of all configurations that specify the location of every point on the manipulator arm. Considering the configuration

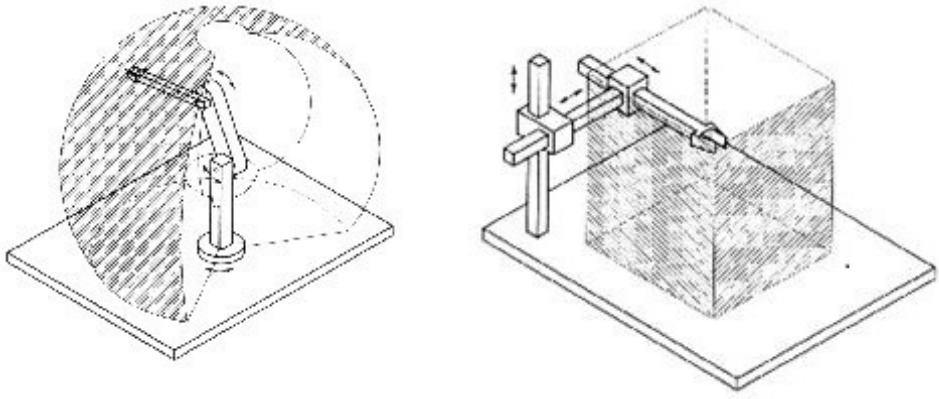


Figure 2.1: Workspace of two manipulators from [6]

space when controlling the robotic arm can help in ensuring that no parts of the arm will be obstructed by known obstacles in the environment.

The robotic arm is just one part of the overall robotic system. For a fully autonomous robotic system the arm must be integrated with a controller, power supply, sensors and some decision unit that replaces the need for human instructions.

2.1.2 Robot Control

Any electrically powered robotic arm will move by applying voltage to of a series of electrical motors. The translation from the natural language of humans to movement of motors is not necessarily intuitively easy, therefore there is a need for an interface controller that translates natural language or motion based instructions into stable actuation of each separate motor. Section 2.2.2 will explain the inputs of servo motors and how it is translated into actuation. Section 2.3 will go deeper into robot control and how a complete configuration of the robotic arm can be translated into separate instructions for each joint actuator.

2.1.3 Sensing

A robotic system's purpose is to interact with its surroundings, which is impossible without some real time information about its environment. In the same way humans have their senses, a robotic system uses input from sensors to analyze and react on its surroundings. Data from sensors generally serve two purposes in a robotic system:

- Input to the system with information needed to make a decision on which actions it should take.
- Input to the controller that incorporates it as feedback information to control the motion of the robot.

The latter kind are often in the form of positional feedback devises for the motors, such as potentiometers. Vision based feedback information has also become more prevalent in recent years, the technique of incorporating vision data into a motion control is known as *visual servoing*. Sensory information as input in the decision making process could take any form that is relevant for the task at hand. In robotics this information often correlates to some human sense, such as vision or sound, but also comes in the form of for example distance or temperature measurements.

2.1.4 Computer Control

To make a robotic system fully autonomous there is a need for a computer controller that decides the actions of the arm based on available information. The unit could typically instruct the motor controller to move the arm into a given position, pick up some object and place it somewhere else. The computer controller's task is twofold:

- Decide which actions best serve its purpose
- Translate these actions into motion based instructions for the robot controller

The computer controller will often also consist of a module for processing sensory information. Decision making is often closely related to artificial intelligence and serves as a context specific replacement for human interaction.

2.2 Electrical motors

Electrical motors are the most common actuators for small-scale robotic systems. An electric motor is a machine that converts electricity to kinetic energy. Electrical motors can be separated into two main categories, alternating current (AC) powered motors and direct current (DC) powered motors. This chapter will serve as a quick introduction to the general information that is necessary to understand the subsequent chapters. Section 2.2.1 serves as an overview on DC motors, while section 2.2.2 covers the basic functionality of a servo motor.

2.2.1 DC Motors

When not considering the internal parts, the concept behind the DC motor is quite simple. Applying a voltage will make the output shaft spin. A high input voltage will result in a large angular velocity; consequently low input voltage will result in a smaller angular velocity. Reversing the direction of the spin is done by reversing the input voltage polarity. DC motors have an operating voltage range; voltages above this limit can result in an overheated and potentially broken motor, while voltages bellow the lowest limit might not run the motor at all.

DC motors tend to spin with a high angular velocity; this is only possible as the output shaft is spinning with no significant external force stopping the movement. A motors capability of moving an object will depend on its operational torque, which is usually low at high rotational speed. In robotics a high motor torque is often more interesting than a high rotational velocity, in fact a high rotational velocity might have unwanted effects as long stiff objects tend to break when rotated too quickly. To increase motor torque and therefore decrease speed, motors are equipped with gears.

The most common gears are toothed circular disks, mounting the gears so the teeth engage into each other results in a gear train. A gear train allows for transmission of torque, with the possibility of altering the torque, speed and direction of the rotation. A small rotating gear attached to a larger gear will result in an increased torque at the expense of rotational velocity. With a substantial amount of gearing it is possible for a motor rotating at a high velocity to exert high torque at the final output shaft. The drawback with gearing is that it usually leads to a reduction in motor efficiency.

A motors operational torque is the torque it is designed to provide. When the torque exerted by some load exceeds what the motor can produce, the motor is said to be stalling, which means the necessary torque for rotation is more than the motors maximum torque. This torque is also known as the motors stall torque. A DC motor will draw a certain amount of current depending on what torque is necessary for the desired rotation. The stall current is the current the motor will draw when stalling, which is also the highest amount of current a normal DC motor will draw.

2.2.2 Servo Motors

A servo motor is a small device that incorporates an electric motor with appropriate gear train and an integrated closed-loop control scheme. The servo motor includes a motor, gearing mechanism, a potentiometer and an integrated control circuit. The motor turns the output shaft through a series of gears to supply the necessary torque and speed. The potentiometer is used as a positional feedback device that connects to the control circuit on one end, and rotates with the output shaft on the other end. Positional information is fed to the control circuit through the potentiometer and allows for closed-loop control of the output shaft.

Servo motors are commonly rated by their operating voltage and the maximum torque the motor is able to supply at a given input voltage (stall torque). A lot of medium sized servos have an operating voltage of 4.8-6 V. If the servos are given a voltage bellow 4.8V, they might stop working, while feeding the servos anything above 6V can result in overheating and a damaged motor. A servo motor supplies the highest torque at its maximum rated operating voltage, reducing the input voltage will result in a reduced potential torque. Keeping the supply voltage high might not seem like a big problem at first, however, when using multiple servo motors at once, the current drawn can become very high, which might result in a voltage drop on the power supply.

The built in gearing and control circuitry makes servo motors ideal for simple and precise positioning. A servo motor requires a power supply and an input signal to adjust the angle. Most servo motors are controlled by sending the integrated circuit a Pulse Width Modulated (PWM) signal. The pulse width represents a given servo angular position, increasing or decreasing the pulse width will drive the servo in a given direction. As long as the PWM signal is present on the servo motors input line, the servo will maintain the angular position. In most hobby-scale projects the PWM signal is generated by a microcontroller or dedicated servo controller. The PWM signals are usually configured by some internal controller scripts or externally, for example by interfacing the controller to a more powerful computer

2.3 Robot kinematics

A robotic arm consists of a base connected to an end effector through a series of joints and stiff links. The end effector is the tool with which we want to manipulate our surroundings, similar to a hand on a human arm. The joints connect each link in the series to the next and are characterized by their potential movement. Each joint represents a single movement, rotational movement for revolute joints and translational movement for prismatic joints. An arm is said to have a certain amount of degrees of freedom (DoF) pertaining to how many parameters that needs to be specified for the arms configuration. Under the assumption that each joint has a single degree of freedom, an angle for revolute joints and a distance for prismatic joints, the complete configuration and position of the robotic arm can be described by the complete set of joint variables. A core concept of robotic arm control is to translate positional instructions into individual joint variables. To understand how this is done, a quick introduction into rigid motions, forward- and inverse kinematics is necessary.

2.3.1 Rigid motions

In robotics a point in space is usually represented by its Cartesian coordinates with respect to a specific coordinate frame. However, when introducing multiple coordinate frames we also introduce multiple ways of representing the same point. Consider figure 2.2, the point p_1 can be represented in the 0-th and 1-th coordinate frame as respectively:

$$p_1^0 = \begin{bmatrix} x_0^* \\ y_0^* \end{bmatrix} \tag{2.1}$$

$$p_1^1 = \begin{bmatrix} x_1^* \\ y_1^* \end{bmatrix} \tag{2.2}$$

When introducing multiple coordinate frames it becomes useful to be able to represent one coordinate frames position and orientation with respect to another. The rotation from coordinate frame 0 to coordinate frame 1 can be represented by the unit vectors of frame 1 with respect to frame 0 as:

$$R_1^0 = \begin{bmatrix} x_1^0 & y_1^0 \end{bmatrix} \tag{2.3}$$

In the two-dimensional case represented in figure 2.3, the rotation can be represented as a rotation θ about the z-axis, resulting in

$$x_1^0 = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, y_1^0 = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \tag{2.4}$$

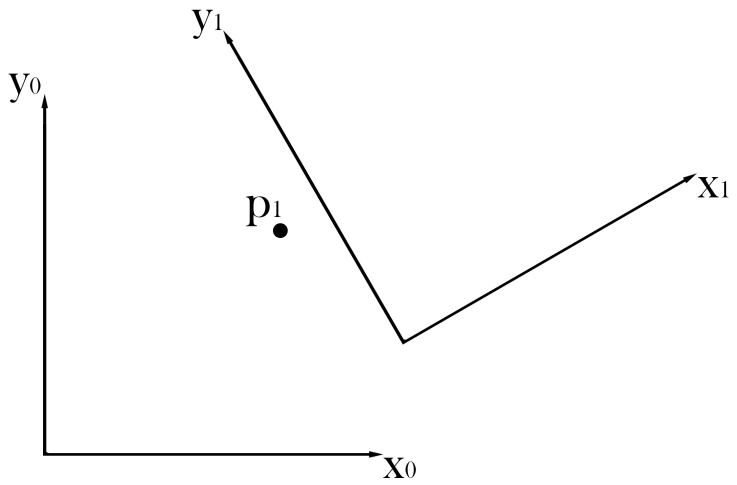


Figure 2.2: Point in two coordinate frames

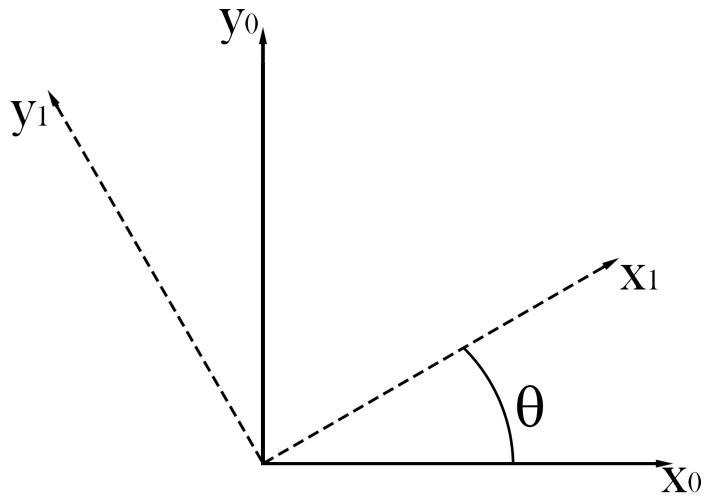


Figure 2.3: Rotation of Coordinate Frames

The concept is the same when representing the rotation in three dimensions, the axis of rotation is kept static while we use trigonometry to calculate the new coordinate frame orientation. This results in the three rotational matrices:

$$R_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.5}$$

$$R_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \sin\theta \end{bmatrix} \tag{2.6}$$

$$R_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Where x,y,z represent the axis of rotation and theta the angle. For proof on how these rotational matrices are found, see [6, Ch.2.2].

Using these rotational matrices in combination with a distance vector d_1^0 any point represented with respect to one coordinate frame can be represented in another coordinate frame by simple rotation and translation. Assuming we have a point p represented in the 1-th frame, and the distance vector o_1^0 from the origins of frame 0 and 1, we can represent the point p_1 in the 0-th frame as:

$$p_0 = R_1^0 p_1 + o_1^0 \tag{2.8}$$

These types of rotations and translations can be combined into a general homogeneous transformation matrix between coordinate frame i-1 and i as:

$$A_i(q_i) = \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{2.9}$$

Performing multiple rotations and translations can be done by considering each transformation between subsequent coordinate frames to be independent. The result is that a rotation from coordinate frame 0 to 3 can be represented as three independent rotations:

$$R_3^0 = R_1^0 R_2^1 R_3^2 \tag{2.10}$$

The translation from coordinate frame 0 to 3 can be represented as a series of translations and rotations, as each distance vector is represented in its own reference frame. Hence the distance from coordinate frame 0 to 3 can be represented as the distance between each subsequent coordinate frame combined with rotation to the base coordinate frame 0:

$$o_3^0 = o_1^0 + R_1^0 o_2^1 + R_1^0 R_2^1 o_3^2 \tag{2.11}$$

Using the homogeneous transformation matrix, the movements and rotations form a kinetic chain, which can be represented as a product of the individual transformations:

$$T_n^0 = A_1(q_1) A_2(q_2) \dots A_n(q_n) \tag{2.12}$$

2.3.2 Forward kinematics

Kinematics is the process of describing the motion and position of a robot manipulator without considering actuating or external forces. Forward kinematics uses the principles of rigid motion to describe the end effectors position and orientation using join variables and link constants. In a kinematic chain, each joint is represented by a coordinate frame. The Denavit-Hartenberg (DH) convention was a concept introduced by Jacques Denavit and Richard Hartenberg [7] to standardize the selection of reference coordinate frames in robotics. Using this convention, the coordinate frames are chosen to satisfy the DH-conditions:

1. The axis x_i is perpendicular to the axis z_{i-1}
2. The axis x_i intersects the axis z_{i-1}

The coordinate frames pose and origin for each joint does not need to relate to any physical position or orientation on the manipulator, as long as it satisfies these two conditions. For a kinematic chain that satisfies the DH conditions, the homogeneous transformation between coordinate frames can be represented as a product of four basic transformations:

$$A_i = Rot_{z,theta} * Trans_{z,d} * Trans_{x,a} * Rot_{x,alpha} \tag{2.13}$$

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

With:
θ_i = *joint angle, the angle between* x_{i-1} *and* x_i
d_i = *link offset, the distance from* o_i *to* o_{i-1} *along* z_{i-1}
a_i = *link length, the distance from* o_i *to* o_{i-1} *along* x_i
α_i = *link twist, the angle between* z_i *and* z_{i-1}

The matrix A_i is single-variable function, depending only on the joint variable. The joint variable is represented by θ for revolute joints and d for prismatic joints, while the three remaining parameters are constant. Using the DH convention we can describe an end effectors orientation and position in the base reference frame by knowing the relevant variables θ_i, d_i, a_i and α_i for each joint.

Forward kinematics is quite useful when analyzing a manipulator, however, when controlling a robotic arm, the end effectors pose and orientation is often known, while the join variables need to be calculated.

2.3.3 Inverse kinematics

Inverse kinematics is the process of using a robots kinematic equations to calculate the joint variables that correspond to a given end effector position and orientation.

The general problem can be stated as finding the joint variables q_1,. . . .,q_n that solves the equation

$$T_n^0(q_1, ..., q_n) = A_1(q_1)...A_n(q_n) = H \tag{2.15}$$

Where H represents the end effectors position and orientation with respect to the base coordinate system. The general inverse kinematic problem can be quite complex, with multiple nonlinear equations. Inverse kinematics also gives the possibility of finding multiple solutions, or in fact no useable solutions, as the manipulator will have some inherent geometrical restrictions. While it is computationally complex, most inverse kinematic problems can be solved numerically.

The inverse kinematic problem for most simple robotic manipulators can be decoupled into a geometric and an algebraic problem. By combining some of the relevant kinematic equations with a geometric approach the problem of solving equation 2.15 can often be reduced to solving a few geometric equations. The geometric approach is used to find the inverse position kinematics, while the remaining variables can be found by using the algebraic approach. Using the geometric approach, the joint variables are projected onto a two-dimensional plane and solved using trigonometry. The variables that can't be solved using the geometric approach can often be solved by combining the relevant kinematic equations with the results from the geometric approach. Inverse kinematics provides the angular position of each separate motor so that the robotic arm achieves its desired end-effector position. With an accurate model of the arm and correct calculations an inverse kinematic robot controller can provide accurate and robust control.

2.3.4 Vision-based Control

Vision-based robot control, also known as visual servoing, is a method for controlling robot movements by analyzing vision based sensory data. The general method is using image data to calculate some error function between the robot manipulator and target object. This error is then fed into the robot controller as feedback and results in a closed-loop positional control. Vision-based sensors are useful for robot control as it allows the robot to mimic the human vision sense and often contains an abundance of relevant data. Visual servoing techniques are commonly separated into two categories:

Position-Based Visual Servoing (PBVS)
: Information extracted from image features is combined with a known camera model to reconstruct a geometric model of the objects pose and position. This reconstruction is used as feedback input to calculate the error between the manipulator and the target and subsequently drive the manipulator arm to the target location. PBVS can be seen as visual servoing in three dimensional Cartesian space where an error is calculated based on Cartesian coordinates.

Image-Based Visual Servoing (IBVS)
: The error is calculated directly in image space based on current and desired image features. This error is then used as feedback to the manipulator arm.

This approach requires no reconstruction of three dimensional pose as the arm is controlled to bring the current image features to their desired values.

There are also some hybrid approaches that try to combine the PBVS and IBVS to reduce the impact of problems with either approach. These approaches are also separated into two sub-categories based on camera configuration:

End-point closed-loop control
The camera is attached to the arm and observes the scene relative to the manipulator arm. This approach allows the controller to shift the camera to inspect different areas.

End-point open-loop control
Fixed camera position that observes both the target scene and manipulator motion.

Position-based visual servoing provides intuitively easy control as it has the advantage of defining the error in the Cartesian frame of the robot manipulator. A drawback with PVBS is its need for an exact task space and camera model which makes it sensitive to intrinsic camera parameters and camera calibration. This property is not shared by image-based visual servoing controllers as they are relatively insensitive to camera parameters and calibration. IBVS approaches require online computation of the control law with respect to image features which can lead to some problems when such a computation is not possible. Chaumette [8] takes a more in depth analysis of IBVS and notes the relation between local minima in image features and unrealizable singularities in robot manipulators. There is also the challenge of finding adequate visual feature representation for any objects from any camera position. IBVS approaches in general are highly sensitive to camera position[9] and often requires multiple cameras to provide adequate stereo vision when used with the end-point open-loop control approach. Camera placement is not a challenge unique to IBVS as the position-based approach relies heavily on being able to acquire and process all relevant image data to ensure a robust model of the object target. A more in depth introduction to visual servoing can be seen in [10] and a comparison between image-based and position-based control can be read in [11].

2.4 Computer Vision

2.4.1 The image and its properties

A digitized image is represented as a two-dimensional matrix of pixels. The two dimensions represent the amount of pixels in either direction, also known as the pixel resolution. Pixel resolution is often represented as width $\times$ height, representing how many pixels the image contains in the two dimensions. An image pixel is the smallest element in an image and is addressed using its physical location in the image. In gray scale images the pixel contains a single value representing the range from black to white, while in color images it usually holds either three or four values, depending on the color model used. As an image is represented only by height, width and color, there is a lack of depth perception which is crucial to humans' visual perception. The lack of depth in images is compensated by the human brain with domain knowledge, however, this is knowledge a computer does not have, thereby making the already complex field of computer vision even harder.

Color images

Color images are usually captured so that each pixel contains an array representing the Red-Green-Blue(RGB) intensity values. The RGB color model represents a wide range of colors by adding up the values of red, green and blue. While the RGB-model is the common color-model in image acquisition, there are other advantageous models used in image processing. One such color-model is Hue-Saturation-Value(HSV). The HSV-models is popular in many image processing applications as it separates the pure color component (hue) from the brightness (value) and colorfulness (saturation). This property is useful in several image enhancement techniques as well as recognizing color in an environment prone to variation in brightness. Converting between RGB and HSV is a quick and easy process which allows the use of both the HSV- and RGB-models without adding too much computational complexity.

2.4.2 Pre-processing

Captured images will usually have some unwanted properties like existence of noise or low image contrast. To increase the effectiveness of high level image processing some pre-processing is often needed. Pre-processing usually involves enhancing or suppressing certain parts of the image to ease further processing.

Histogram equalization

Histogram equalization is a popular method of image enhancement which aims to increase image contrast by distributing the pixel intensity levels more equally. The result will be an image with enhanced contrast for pixel values close to the histogram maxima and reduced contrast near the histogram minima. Histogram equalization is commonly performed on gray scale images to increase the robustness to various levels of lighting and contrast. It is also applicable to color images,

without dramatically changing the color balance, by using a suitable color model and performing the equalization separately on the channel that does not affect the color component. In the case of the Hue-Saturation-Value color model, color histogram equalization could be performed on the value-channel without drastically changing the image color balance.

Image smoothing

Image smoothing is a pre-processing technique that aims to ease further processing by suppressing noise in an image. The smoothing is usually performed by assigning each pixel the value of some weighted average of its neighborhood and results in a more blurred image. This approach usually entails a loss of information, but using the right smoothing filters one can reduce the loss around important features. A popular filter for image smoothing is based on a two-dimensional Gaussian function:

$$G(x,y) = \frac{1}{2\pi\rho^2} e^{-\frac{x^2+y^2}{2\rho^2}} \tag{2.16}$$

Where x and y represent the distance from the center of the filter and ρ the standard deviation. A Gaussian filter approximates a discretized Gaussian function, resulting in a weighted image smoothing approach with highest weights closest to the center pixel. An example of a $(3x3)$ spatial Gaussian filter can be expressed as:

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.17}$$

Edge detection

Edge detectors are a collection of methods that attempt to locate edges by finding pixel areas with sharp changes in intensity values. The aim is to identify the interesting regions of an image, which in general are located near or contained within edges. As edge detectors typically enhance image noise, it is common to apply image smoothing before edge detection. There are many different approaches to edge detection. A common method is computing the gradient magnitude and direction, and using these to find the edge strength and orientation. A simple way of calculating the gradients is by first applying a one-dimensional derivative filter in both the horizontal and vertical direction, then calculating the magnitude and direction as

$$L_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * I, L_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * I \tag{2.18}$$

$$|Grad(L)| = \sqrt{(L_x)^2 + (L_y)^2}, Direction = atan2(L_y, L_x) \tag{2.19}$$

Where I is the image and L_x the derivative filter in x-direction convolved with the image. By applying some threshold to the gradient values one can identify edges in the locations with a significant change in intensity values. From these edges on could extract useful information such as object contour and region.

Image computational complexity

The amount of pixels in images nowadays typically range anywhere between a few hundred thousand to tens of millions. This leads to image processing being a very computationally expensive branch of programming, simply because of the sheer amount of data that needs to be processed. To speed up the processing of images and maintain some real-time viability for video processing it is common, in the case of large images, to reduce the image resolution before performing other image processing tasks. Reducing the image resolution results in fewer image pixels at the expense of fine image details. The choice between image resolution and detail depends on the computational resources available and details needed for further processing.

A common approach to image downsampling is to smooth the image by some chosen method, then removing a chosen amount of evenly distributed rows and columns. For instance reducing the image size by four entails removing every other row and column after smoothing. Upsampling is performed when there is a need for higher image resolution, although the fine details lost by downsampling will not be recovered. A simple approach would be inserting evenly distributed rows and columns, depending on the amount of upsampling, then assigning each of the new pixels some weighted average of the neighboring pixels.

Mathematical Morphology

Mathematical morphology is a technique for processing geometric structures using point sets, connectivity and shape. The process is commonly used for geometric enhancement, object description and various image pre and post-processing. The concepts behind mathematical morphology can be quite complex, therefore only the basic parts of geometric shape enhancement will be presented here. For more details on mathematical morphology see [12, chap. 14].

A morphological transformation is performed by moving a structuring element across the entire image. The structuring element is placed with its reference point on the location of the current pixel. The resulting image contained in the element is then processed according to the chosen morphological operation. The two basic morphological operations, *dilation* and *erosion*, can be used independently, or combined, to perform various geometric enhancements. Dilation is a morphological transformation that combines two point sets by vector addition and is usually performed to expand an object, either by filling holes or expanding boundaries. The dilation $X \oplus B$ is defined as the result after all possible vector additions of the two vectors:

$$X \oplus B = [p \in E^2, p = x + b, x \in X, b \in B] \tag{2.20}$$

With E^2 representing the two-dimensional Eucledian space.

The other basic morphological operation erosion, which has functionality opposite of dilation, is usually performed to reduce or shrink a geometric structure. However, erosion and dilation are not inverse transformations as performing erosion followed by dilation, or opposite, will not result in the original image. Erosion combines two point sets by vector subtraction, meaning the result is given by the points for which all possible combinations of the image and structuring element are present.

$$X \ominus B = [p \in E^2, (p = x + b) \in X \ \forall b \in B] \tag{2.21}$$

Dilation and erosion by themselves serve no function if one wishes to preserve as much of the shape and size of an object as possible. However, combining them yields some useful results. Opening and Closing combine erosion and dilation to eliminate specific image details smaller than the structuring element while still keeping the general shape of the object. Opening performs erosion followed by dilation and its main functionality is removing small objects from the background. Similarly closing applies dilation followed by erosion and is used to fill up small holes within or around an object. Mathematical morphology is commonly used in the post-processing stage of segmentation to remove noise and merge adjacent shapes.

2.4.3 Segmentation

Image segmentation is the process of dividing an image into labeled parts that correlate with some real life objects or areas. This process is often impossible without global domain knowledge; therefore partial segmentation is often the result. In partial segmentation every pixel in a segment shares some common property, usually brightness or color. These partial segments do not necessarily represent some real life objects as they only rely on local pixel value. The main goal of segmentation is to simplify the representation of an image for further analysis. A good segmentation is often necessary before performing further processing such as object description, recognition and attaining general image understanding. In an application that seeks to process a chessboard, the segmentation result will seek to extract the entire chessboard region. As a chessboard cannot be robustly located based on partial segmentation results, some global domain knowledge and use of higher level processing is necessary.

2.4.4 Feature Detection

Feature detection is the process of searching an image for a specified type of feature. There is no clear definition on what an image feature is, but in general it refers to

interesting regions in an image that helps describe some aspects of the image. Such features are often edges or objects or even the result of a general feature extraction method. The choice of object features has a large impact on object detection as they serve as the general descriptors of the object.

Canny Edge Detection

The Canny edge detector was first proposed by Canny [13] with the aim of finding the optimal edge detection approach. Canny cited three criteria that any good edge detector must fulfill:

1. *Low error rate*: Edges in the image should not be missed, and no edges should be detected where there are none in the image.

2. *Good localization*: The detected edges should be as close to the center of the image edge as possible.

3. *Single edge detection*: An image edge should only be detected once and avoid detection of noisy step edges.

The core concept of the Canny edge detector can be explained as a four step process:

1. *Noise reduction*: As with normal edge detection the image needs to be smoothed over to reduce the amount of noisy edges in subsequent stages. Typically this is done by applying a Gaussian filter as in section 2.4.2, image smoothing.

2. *Gradient Calculation*: This step is done in a similar manner to the edge detection in section 2.4.2, but a more advanced derivative filter that retains gradient size and direction in both vertical, horizontal and diagonal direction is applied. A common derivative filter used with the Canny edge detector is a Sobel filter [14]. The gradient strength and direction for each pixel is stored for the following steps.

3. *Non-maxima Suppression*: The calculated gradients in the previous step are analyzed using their gradient directions. The gradient pixel is removed if the gradient value is not larger than that of its' two neighboring pixels in the gradient direction. This is a popular method of edge thinning that ensures an edge is represented by the thinnest line possible.

4. *Hysteresis thresholding*: This step starts by defining two thresholds, one defining strong gradients and another defining weak gradients. The concept of hysteresis thresholding is that a gradient is deemed classified as an edge if it passes the initial upper threshold. Any gradient that passes the lower threshold but not the upper threshold is classified as an edge if it is connected to a previously located edge. All other gradients are then discarded.

The advantage of using the Canny edge detector is its robustness to noise and spurious edge responses while still retaining most real image edges. These properties are useful to ensure that no important edges are missing and the chessboard can be accurately located.

Hough Transform

Hough transform is a technique for detecting and extracting image features. The classic Hough transform requires the feature to be described by some parametric equation, such as a line or a circle. The generalized Hough transform can detect features of arbitrary shape and size, however, it suffers from increased computational complexity compared to the classical approach. This section will cover the basics behind the classical Hough transform, henceforth referred to as the Hough transform. In an ideal world, the edge detection would locate the complete edge, and thus the feature could be found by simply traversing along the edges. In computer vision, the detection of edges often results in noisy edges with partial occlusions or missing segments. The Hough transform seeks to locate and connect these edges by searching the image for known shapes of arbitrary size and orientation. The concept behind the Hough transform can be seen from detecting a straight line in a binarized edge image. A straight line can be described by its slope a and intercept b as:

$$y = ax + b \tag{2.22}$$

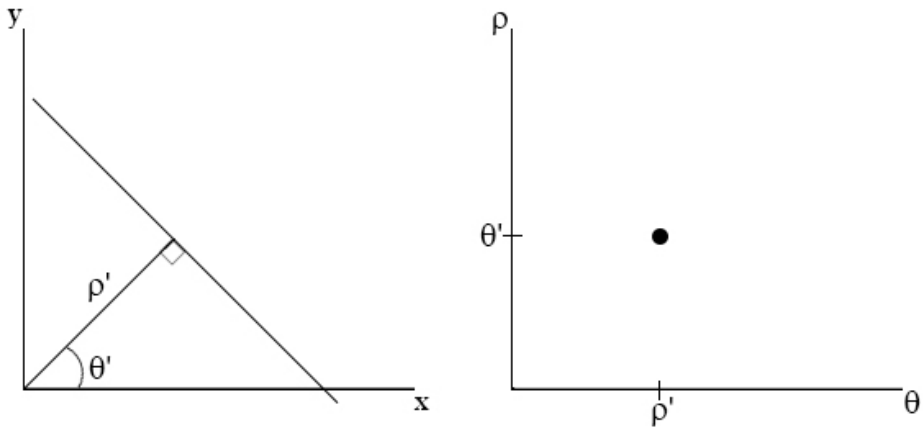
By discretizing the slope and intercept we can represent every possible line in an image by forming an accumulator array containing all combinations of the discretized slope and intercept. This accumulator array will represent not only the lines present in the image, but any conceivable line slope and position that could exist. For each edge pixel all possible lines that could intersect the point is found and the accumulator cells of the corresponding lines slope and intersections are incremented. For any lines actually present in the image, the accumulator cells will see as many increments as they have pixels along the line. The problem of finding image lines has now been reduced to locating high-valued accumulator cells.

Using the parametric equation 2.22 will cause problems when vertical lines are present, as $a \to \infty$. The solution to this problem was presented by Duda and Hart [15] and involved representing the lines by their polar coordinates as:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.23}$$

The concept remains the same, with each line being represented by a point in theta-rho space as in figure 2.4. The discretization of the parameters are commonly confined to $0 \leq \theta \leq \pi$, and $-\rho^* \leq \rho \leq \rho^*$, however, one could also confine θ as $0 \leq \theta \leq 2\pi$ and use purely positive values for ρ.

The strength of the Hough transform is its insensitivity to noise, partially missing lines or other non-line structures in the image. To better understand its



(a) Line parametrization in x-y space (b) Line in ρ-θ parameter space

Figure 2.4: Line parametrization

strength, a few example images and the corresponding Hough line transform parameter space can be seen in figure 2.5. The noisy image details result in sporadic responses in the accumulator array, nevertheless the lines can be easily detected by the four local maxima. The added circle results in increased accumulator values in a larger area without altering the existing lines which are still distinguishable in the Hough transformed image.The last figures represents a typical edge detection result with partially occluded lines, while the four lines now provide less impact on the accumulator array, the lines can still be found as local maxima.

A weakness of the presented Hough transform approach is its need to either threshold the resulting accumulator array or search for local maxima. Either approach, or even a combination of the approaches, could result in some lines not being detected or the detection of lines as a result from random objects. Some a priori knowledge such as expected number of lines or line directions will often ensure a more robust result.

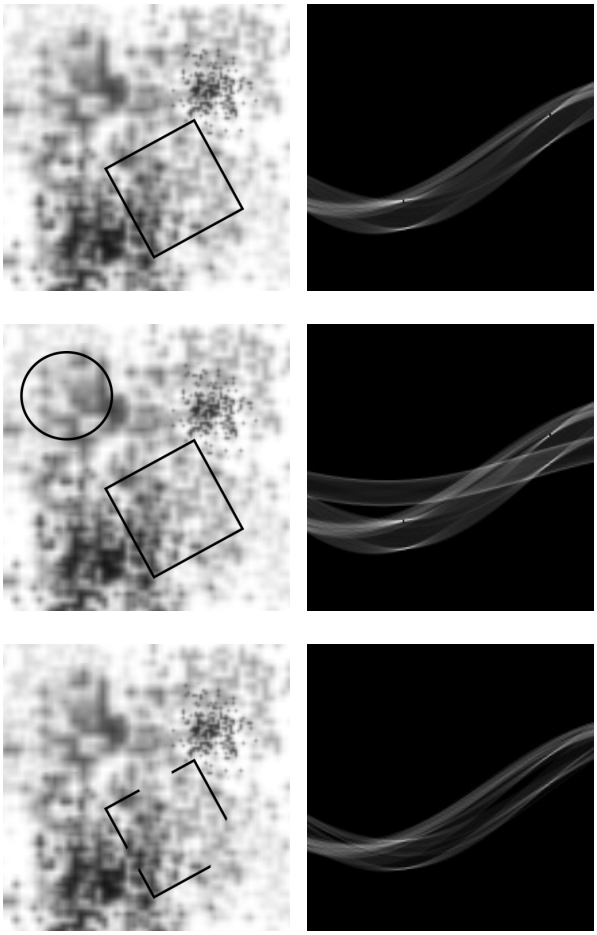


Figure 2.5: Hough transform with noise, occlusion and foreign objects

2.5 Chess

Chess is a two-player turn based strategic board game and is one of the oldest and most popular board games to date. What makes chess unique is that there is no random factor in the game, every game starts the same and is impacted solely by the choices of the players.

Chess is played on a square board that is separated evenly into an 8x8 grid of squares of alternating light and dark color. A square on the grid is referred to by its chess board coordinates, spanning from a-h on one axis and 1-8 on the other. The square in the lower left-most corner is thus named a1, and a chess move can be recorded by supplying the source and destination of a move, such as $a1 \rightarrow a2$. The players are referred to as "white" and "black" which usually reflect the colors of their chess pieces. Both players start the game with 16 pieces: eight pawns, two rooks, two bishops, two knights, one queen and one king. The placement of each piece can be seen in figure 2.6.

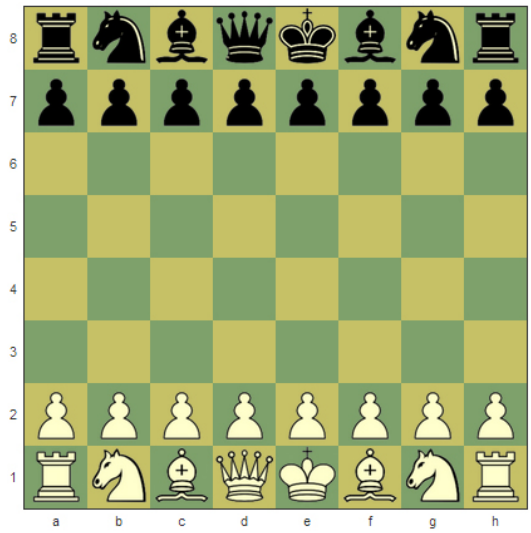


Figure 2.6: Initial Chess Game Position

For the white player the rules go as follows:

- Eight pawns placed on the second row, between a2-h2
- Two rooks placed in the two nearest corners, a1 and h1
- Knights placed on the left and right side of their respective rooks: b1 and g1
- Bishops: Similar to knight placement, one further step inwards: c1 and f1
- Queen and King are placed on the two remaining positions, Queen: d1, King:e1

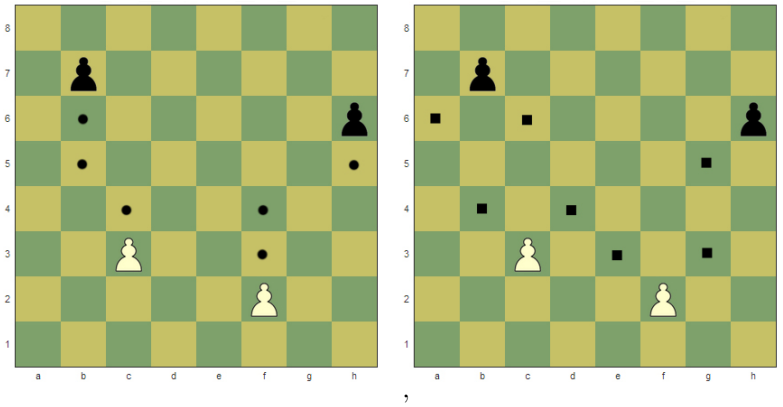
The black player places the pieces in a similar manner on his side of the table. A chess game starts with the white players move followed by alternating black and white moves. The various types of chess pieces can move according to the rules stated later in this chapter. A piece is captured, and subsequently removed from the game, when a piece of the opposite color is moved onto its square. Note that at any time only one piece may occupy a given square. The game ends when the king of either side is captured or one of the players concedes the game.

2.5.1 Regular Movement

The various movements of each piece will be explained, as well as certain "special" moves that apply in unique situations. Note that any subsequent use of the word "forward" will mean movement in the direction towards the other player's side. Unless explicitly stated otherwise, a movement to an occupied square without capturing or movement through an occupied square is not possible. Each turn a player must perform one legal move, if no moves are legal the turn is passed to the other player.

Pawn

A pawn can move one square forward as long as this square is not occupied by any other piece. If the pawn is in its initial position, meaning it has not been moved so far, the player may move the pawn two squares forwards. A pawn may only capture another piece situated on an adjacent square in a diagonally forward direction, shown in figure 2.7 as black squares. Pawns are also governed by the special moves "en passant" and "pawn promotion" as explained later in this section.



(a) Regular Moves (b) Capturing Moves

Figure 2.7: Pawn Movements

Knight

A knight can move to a square situated two squares in either direction and one to the side, so the complete move forms an "L" shape as seen in figure 2.8. The knight is the only piece that can move through a square occupied by either player.

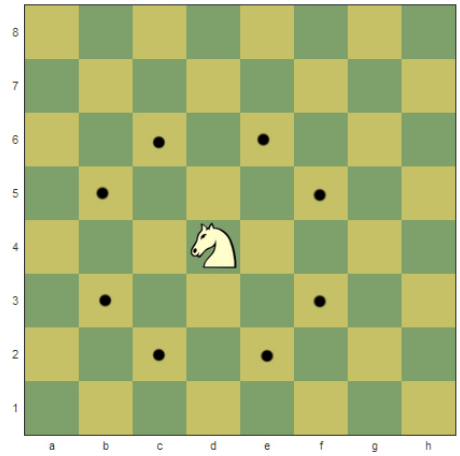


Figure 2.8: Knight Movements

Bishop

A bishop can move any number of squares in a diagonal direction as long as the path is not blocked by another piece.

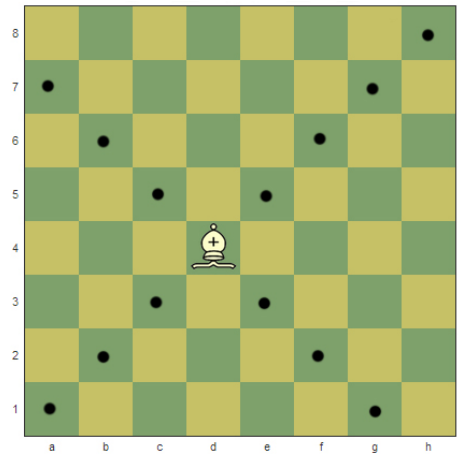


Figure 2.9: Bishop Movements

Rook

A rook can move any number of squares in a vertical or horizontal direction as long as the path is not blocked by another piece.

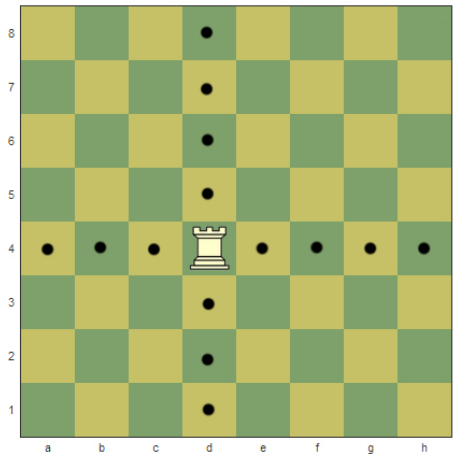


Figure 2.10: Rook Movements

Queen

The queen can be seen as a combination of the bishop and rook with moves being allowed both horizontally, vertically and diagonally as long as the path is not obstructed.

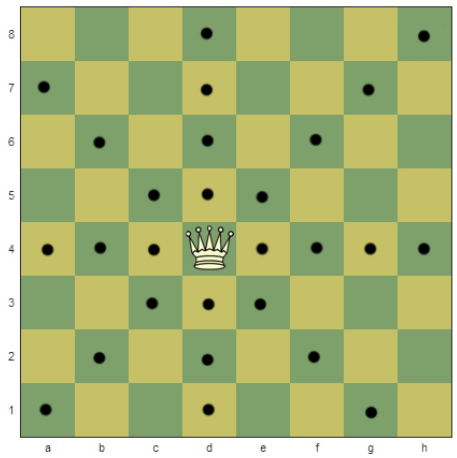


Figure 2.11: Queen Movements

King

The king may move to any adjacent square, both vertically, horizontally and diagonally. As the game ends with the kings' capture, a king may not move into a square that would result in his capture.

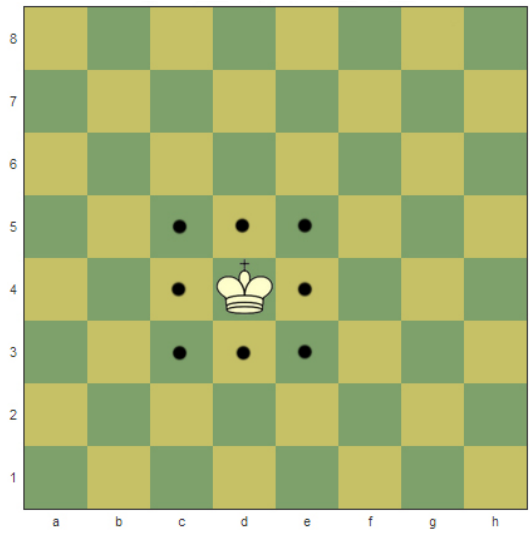


Figure 2.12: King Movements

Check

When a move puts the king as risk of being captured in the king is said to be in check. A player whose king is in check must either move the king to a legal position or stop the checking piece by capturing it or obstructing its path to the king. If no legal moves can be done to get the king out of check it is known as checkmate and the game is won by the checking player.

2.5.2 Special Moves

En passant

When a pawn moves two squares forwards, standing on either side of an opposing paw, the opposing pawn may capture the pawn "en passant", meaning the capture occurs at the position the first pawn moved past. This is best illustrated by figure 2.13. En passant is only possible if done the turn right after the initial pawn moves two squares.

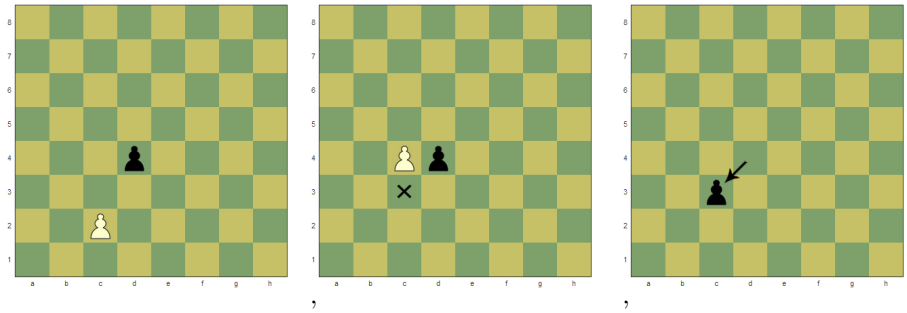


Figure 2.13: En Passant

Castling

Castling is a special move that allows the player to move the king two squares in horizontal direction and set the rook in the chosen direction to the square passed by the king. Castling is the only chess move that allows a player to move two pieces in one turn. Castling is only possible if neither of the pieces involved have previously been moved and there are no pieces in the path between them. The king may not be in check when castling is done and the squares moved to or through may not be under attack by enemy pieces.

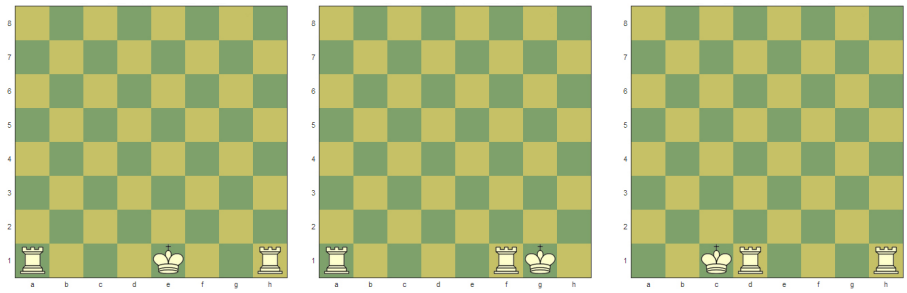


Figure 2.14: Castling

Pawn Promotion

Pawn promotion occurs when either player gets a pawn to the opposite side of the table. The pawn is promoted to a piece of the players own choosing, usually a queen. A king is the only piece that a pawn may not be promoted to, but the player may promote a pawn to a queen even though he already has one in play. Multiple pawn promotions may occur during any game.

2.5.3 Chess Computers

Chess poses an interesting problem in the field of artificial intelligence as it exhibits a property known as perfect information. A decision-making unit having perfect

information means it has all the relevant information necessary to make an informed decision. Perfect information is not common in most AI applications as most situations exhibit some randomness or simplifying models. In theory it should be possible to solve the game of chess, meaning to calculate the perfect moves at any given time. The problem with solving chess arises when considering the number of variations that need to be calculated from an initial position. Shannon [16] estimated that for a 40-move chess game the number of variations would be 10^{120}, a number too large for any current technology to process.

While no chess engine to date has solved chess, the engines have exhibited increasing capability of making chess move decisions. A lot of chess engines use some variety of a decision tree as a basis for its decision. The rood node typically specifies the current game state with branches representing legal moves to other game states. These nodes branch further out with all possible moves of the opponent. The depth of the tree represents how many future moves the computer will process and is usually limited by the amount of time the process is given. Representing a chess game as a decision tree allows the engine to predict the consequences of each possible move and chose the move that maximizes its reward. Typically the reward is some function of chosen chess piece values and their positions on the board.

2.6 Tools and Libraries

2.6.1 OpenCV

OpenCV [17] (Open Source Computer Vision Library) is a multi-platform programming library aimed at handling and processing image and video data. OpenCV was initially developed by Intel with the aim of creating an open, common and optimized computer vision infrastructure that would allow developers to more easily share and transfer applications. From its initial development, with the support of Willow Garage and Itseez, the library has grown from containing methods for basic image capturing and processing to complex object detection and classification. The uses for OpenCV is many, but most importantly it allows developers to focus on high-level image processing without the need for programming basic functionality. The highly optimized OpenCV algorithms and data structures also facilitate real-time computer vision programming.

2.6.2 Pololu Servo Controller

The Pololu servo controller is a microcontroller that can control servo motors either through internal scripting or directly through USB. The servo controller features individual speed and acceleration control of each servo motor and a resolution of $0.25\mu s$, less than what most servo motors can resolve. The usage of a dedicated servo controller takes the development stage one step up, allowing developers to focus on general system functionality instead of basic servo control methods. The Pololu micro maestro servo controller, seen in figure 2.15, has six PWM output channels that can send a pulse width in the range of $64 \rightarrow 3280$ microseconds. For more details on pololu servo controllers see [18].

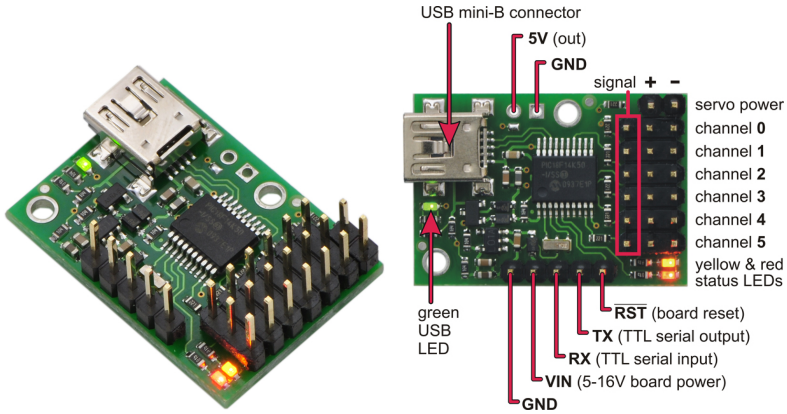


Figure 2.15: Pololu Micro Maestro [18]

2.6.3 libusb

Libusb [19] is a programming library that provides functions for controlling data transfer to and from USB devices. Libusb is popular because of its simplicity as it requires no special privilege and is portable across multiple platforms, USB versions and programming languages.

2.6.4 Chenard the Chess Engine

Chenard [20] is an open source chess program developed by Don Cross. Chenard was chosen for this system mainly because of its simplicity, as it requires no additional chess software and features an easily readable and modifiable source code. To evaluate moves, Chenard uses a min-max algorithm that seeks to minimize the potential loss of a worst case scenario. Chenard seeks to analyze each move and countermove by each player and thus chose the move that result in a minimal probable loss. With this approach one can alter the difficulty of the chess engine by limiting the allowed processing time before a move has to be decided upon.

Chapter 3

Design and Implementation

3.1 System overview

This section aims to give a general overview of the chess-playing robot as well as the interaction between the systems main components. Sections 3.2-3.6 will provide a more detailed description of each module.

The complete system was made as a fully autonomous chess playing robot with the goal of playing on chessboards of arbitrary color, size and orientation. There are some requirements that must be fulfilled for a robot to operate without human interaction. The robot must be able to:

1. Move and manipulate objects within its environment
2. Avoid movement that is harmful to people or its surroundings
3. Gain information about the environment
4. Decide on actions without human intervention

To fulfill the first requirement, the system uses a robotic arm whose main task is to accurately pick up and place pieces at any location on the chessboard. The robotic arm has four degrees of freedom to control position and an additional degree of freedom for the gripper. These degrees of freedom are implemented as a series of servo motors located at each joint on the arm. Control of each individual servo motor is done through a Pololu micro maestro servo controller that also connects the power supply to the motors. Each servo motor has its own closed loop control, however, in this paper the motor controller is defined as the system that includes both the Pololu servo controller and a software interface that stores individual motor variables. For the individual motors to provide smooth motion there is need for a robot controller that regards the robotic arm as a whole. The robot controller ensures that the individual motors collaborate to provide the desired movement. The robot controller is also tasked with finding an obstacle free path as well as executing the various instructions in a timely manner to ensure the

path is followed. The architecture of the robot arm and controllers can be seen in figure 3.1.

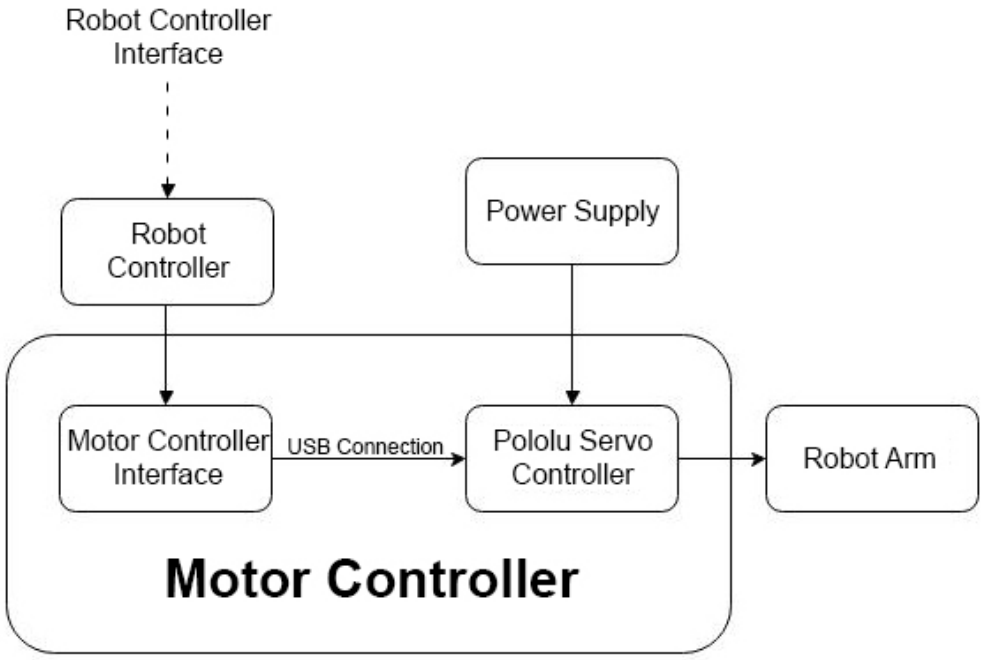


Figure 3.1: Robot Controller Overview

Gaining information about the robotic arm's environment is achieved by the sensory unit. The sensory unit uses a camera placed above the chess board to capture relevant image data. The image is then processed by computer vision algorithms to acquire segmented and readable information about the current chessboard state. Using this information in combination with previous game state knowledge allows the system to gain all the necessary information to make an informed decision on the actions it needs to take. The sensory unit is not directly involved in controlling the robotic arm, but rather serves to acquire information about its surroundings to support the decision making process.

The decision making process is carried out by Chenard [20], a chess engine that uses information about the current game state to predict and recommend future chess moves. The unit itself is an open source chess engine that has been adapted and integrated into the overall system. The control flow of the overall system is managed by a main controller unit. The main controller unit's task is to handle communication and coordination between modules as well as monitor the flow of the program to ensure that errors do not propagate between modules. The main controller unit handles communication with the robot controller through an interface that translates chess moves into instructions based on Cartesian coordinates.

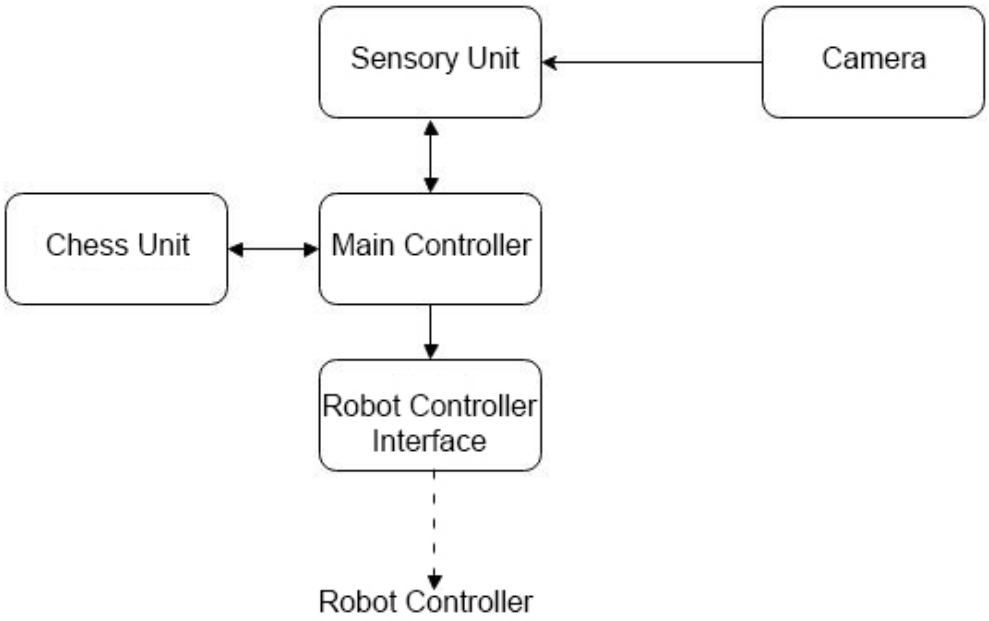


Figure 3.2: Sensory and Decision Overview

3.2 The Robotic Arm

The robotic arm is the main actuator of the chess playing system. Its main task is to provide stable and accurate manipulation of the chess environment. This task can be completed by a large variety of robots, therefore the solution proposed in this chapter is not necessarily the only suitable approach to manipulating chess pieces. The proposed solution for the robotic chess player is mainly focused on playing chess, however, the arm is designed to allow for possible adaptions to other game environments. To design a robotic arm suitable for chess, the arm would have to be made to fulfill the requirements listed in section 3.2.1.

3.2.1 Requirements

Workspace

The arm would need to be able to reach any targets on the chess board. In addition to the square sized chess board the resulting workspace will have to include some movement on the side of the board for the arm to be able to remove pieces.

End-effector

The end-effector would have to pick up and release pieces of varying sizes and forms. The pieces would need to be placed in the same pose as they were picked up to ensure that each piece reaches its destination with the base of the piece facing down. This means that the end-effector needs to grasp the pieces tightly enough so that they are not skewed while moving.

Accuracy

The accuracy of the arm is a measure of how close the arm can get to the instructed position. There is no exact measure of the accuracy needed, but any deviation from instructed position would need to be small enough for the arm to pick up the right pieces without disrupting other game elements.

Repeatability

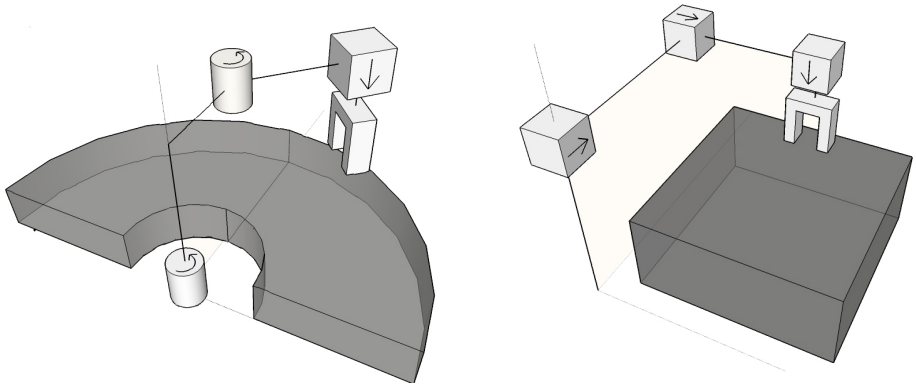
The repeatability of a robotic arm defines how close the arm can get to a position it has already been to, meaning how well the arm can repeat any movement. Repeatability is crucial in a robotic system as any controllers of the robotic arm needs to be certain that identical instructions are performed the same.

Price

The complete robotic system is mostly designed for entertainment and should thus be made with price in mind. There is usually a tradeoff between price and quality, so the robotic arm should be made from components that allow the lowest price that fulfills the other requirements.

3.2.2 General Design

When designing the mechanical arm the main focus was on fulfilling the workspace requirement as accuracy, repeatability and price would rely more on the choice of materials and actuators. Some arrangements that fulfilled the workspace requirement were considered. The SCARA and Cartesian manipulators depicted in figure 3.3 were both considered when designing the chess robot. The SCARA manipulator has two revolute joints to account for chess board position and a prismatic joint to account for object height. The Cartesian manipulator is somewhat simpler as it features three prismatic joints that correspond to movements in the three spatial dimensions.



(a) SCARA Manipulator (b) Cartesian Manipulator

Figure 3.3: Manipulators and workspace with positive direction of actuation

While both the manipulators fulfilled the workspace requirement, their movements would also be more intrusive as they would take up more space around them. The aim was to create a robot that fulfilled the workspace requirement, was unintrusive and left options open for added functionality in the future. The chosen design was an articulated manipulator that uses three revolute joints to control the position and a fourth revolute joint to control the end-effector direction, as seen in figure 3.4.

When designing a robotic arm one would like the least amount of joints that fulfills the requirements for the application at hand, as added joints will need more motors and more complex control algorithms. While adding an extra revolute joint before the end-effector would allow the arm to control the direction as well as orientation of the end-effector, it was deemed unnecessary for picking up chess pieces. The articulated manipulator has the advantage of mimicking a simplified human arm and can reach most points within the total length of the arm. The robot would be able to reach high above the ground, which could allow us to add improved functionality such as an end-effector camera that surveys game states and moves with the arm. The manipulator has a spherical workspace, the projection of this workspace onto the x-y plane can be seen in figure 3.5

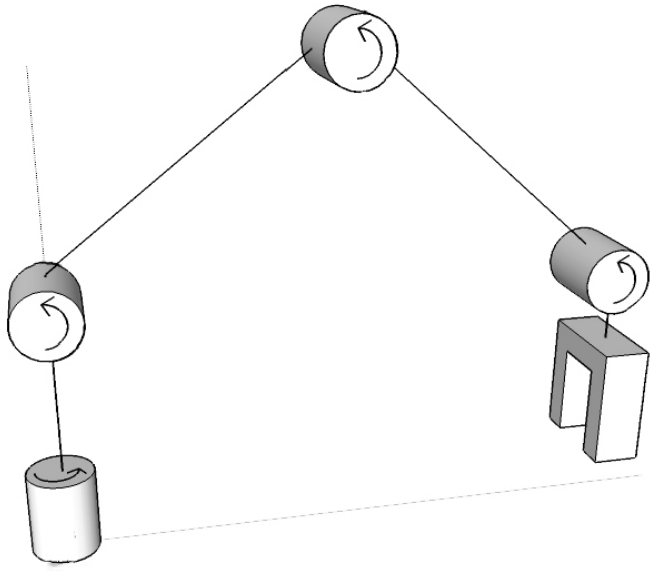


Figure 3.4: 4 DoF - Articulated Manipulator

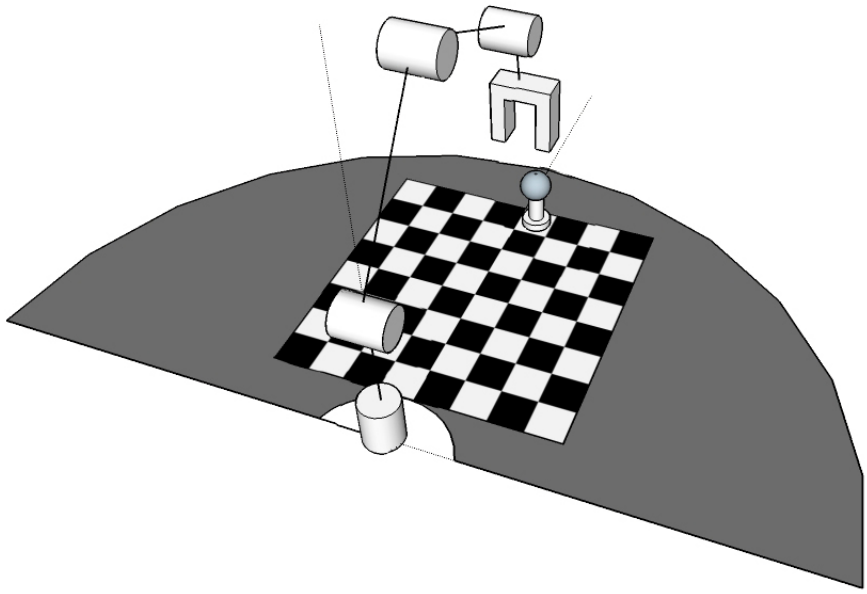


Figure 3.5: 4 DoF - Articulated Manipulator Workspace

3.2.3 Materials

When constructing a robotic arm one needs to ensure that the combined weight of links, motors and end-effector does not exceed the amount the motors can lift at each joint. Any increase in weight would result in either a decrease in functionality or a need for stronger motors, which would increase the overall price. Each link must also be strong enough to sustain acting forces without bending or breaking. Outside of the motors, the link strength has the largest mechanical impact on accuracy as any positional error that is not sensed by the controller, such as links bending, would decrease accuracy. Assuming a total arm length of 50cm, and the innermost link bending 2 degrees, the resulting end-effector error would be:

$$Error(2^\circ) = \sqrt{(50 - 50\cos 2)^2 + (50\sin 2)^2} = 1.75cm$$

An error this large would make the robotic arm unusable for chessboard manipulation, therefore the materials must provide a sufficient amount of strength to ensure a low end-effector error. In addition to being strong and light the material chosen for the prototype should be easy to work with, as one would expect the first prototype to need a lot of adjustments and improvements before the design is complete. Aluminum was the first considered material because of its bend strength to weight ratio. It was dismissed as each link would require a minimum amount of material to function with the given configuration, so even though aluminum is strong, the amount of weight would be excessive compared to the needed strength for lifting up chess pieces. The chosen material for the robotic arm was the plastic-like material Styrene acrylonitrile resin (SAN) which exhibits some of the characteristics of glass. The material had a relatively good bend strength compared to its mass density of $\rho = 1.08g/cm^3$ and allows for heat-bending, drilling and sawing.

3.2.4 End-effector

The end-effector is the device, usually attached at the end of the arm, which is used to interact with the environment. As the arm is being used for playing chess, the end-effectors task is to pick up and hold a piece, preferably without the piece moving while being held. The end-effector needs to be light, as it is placed at the point furthest away from the base of the arm, and will thus have the greatest impact on required motor torque. The end-effector requirements are loosely termed, which means a lot of different end-effector designs could fulfill the task.

One considered solution for the end-effector was by using a magnet and magnetic chess pieces. The first problem encountered with this approach is that there would be a need for specialized equipment in the form of magnetic chess pieces, which is not preferable when designing a general purpose chess robot. Another problem would be the changing pose of the chess piece, we wish to pick up and place the chess piece while retaining its original pose. By using a magnet the pose is likely to shift when picked up and placed. A problem arising from the inherent properties of magnetism is that the gripper would have no way of ensuring that undesirable

pieces are not pulled in or shifted towards the gripper. The idea of using a magnetic end-effector was scrapped as it was deemed an unnecessarily complex method for moving chess pieces.

The most common end-effector is designed as a gripper that actuates a force on the object through mechanical fingers. The simplest version of this is a two finger gripper that grabs an object by pinching its fingers together, as depicted in figure 3.6.

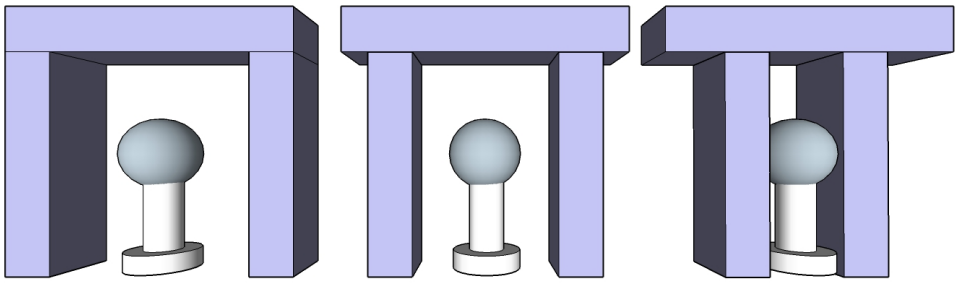


Figure 3.6: Two-finger gripper

One could design the gripper according to the shape of the object to increase lifting power; however, this would also lead to a very task-specific gripper that might not hold all varieties of shapes and forms of chess pieces. With a two finger gripper the force holding the object in place is based on the friction between the object and the fingers surface. By attaching a material with a high coefficient of friction to the fingers one can reduce the force necessary to hold the object in place.

The end-effector was chosen based on the simplest type that would fulfill the weight and strength requirements of moving chess pieces. The chosen end-effector was the Lynxmotion gripper depicted in figure 3.7 with an approximate weight of 25g and compatible with most standard-sized Hitec servos. The gripper opens to a maximum distance of 3.3cm, allowing it to grip any objects within this size.

3.2.5 Actuators

Servo motors were chosen as the main actuators of the robotic arm as they have gearing and control circuitry included, which would ease further implementation and motor control. The end-effector does not have a specific torque requirement as most standard-sized servos would provide the force necessary to lift chess pieces. To find the necessary motor torque for each joint, an approximate maximum acting force was calculated. Increased motor torque usually entails an increase in motor weight, which means that force calculations for each joint would have to begin at

Figure 3.7: Lynxmotion gripper [21]

the end-effector and backtrack to the base motor.

The maximum force acting on the shoulder, elbow and wrist motor can be found by assuming the weight attached to the motor is placed at an angle perpendicular to gravity as in figure 3.8.

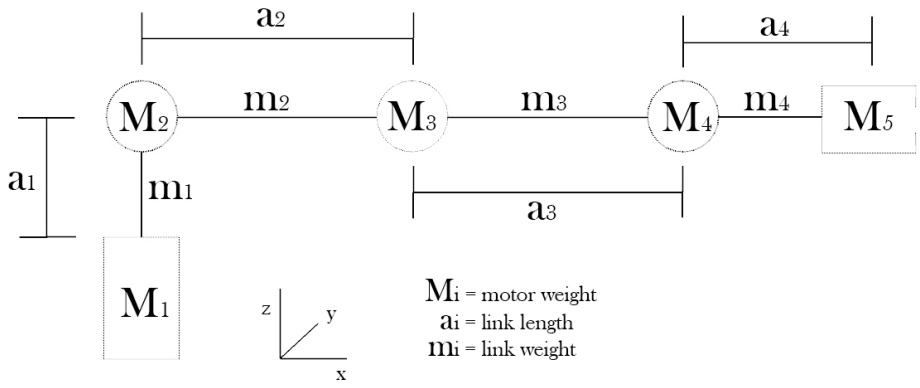


Figure 3.8: Force calculation

By simplifying the end-effector and motors to a single point with a given weight we can find the approximate maximum force acting on the shoulder, elbow and wrist motor as:

$$F_2 = \frac{a_2}{2}m_2 + a_2M_3 + (a_2 + \frac{a_3}{2})m_3 + (a_2 + a_3)M_4 \tag{3.1}$$
$$+(a_2 + a_3 + \frac{a_4}{2})m_4 + (a_2 + a_3 + a_4)M_5$$
$$F_3 = \frac{a_3}{2}m_3 + a_3M_4 + (a_3 + \frac{a_4}{2})m_4 + (a_3 + a_4)M_5 \tag{3.2}$$
$$F_4 = \frac{a_4}{2}m_4 + a_4M_5 \tag{3.3}$$

Type	Dimensions [cm]	Weight [g]
Link 1	N/A	N/A
Link 2	(30x5x0.4)	64.8
Link 3	(30x5x0.4)	64.8
Link 4	(5x3x0.4)	6.8
Gripper	N/A	25

Table 3.1: Weights and dimensions

The required torque of each motor would have to be greater than the maximum acting force to account for some loss due to inertia, friction and motor inefficiency. The initial calculations were done using the rectangular cuboid link values in table 3.1, the resulting maximum force at each joint and the chosen motors are listed in table 3.2. Required torque of the base motor cannot be calculated using the procedure above as it rotates on a plane perpendicular to gravity. The base motor would mainly have to combat friction and inertia which would be very time consuming to calculate. The base motor was chosen with a torque high enough to ensure it could easily rotate the arm.

Placement	Type	Weight [g]	Stall Torque (6V) [N-m]	Max Acting Torque [N-m]
Base	HS-805MG	197	2.42	N/A
Shoulder	HS-805MG	197	2.42	1.3
Elbow	HS-5645MG	60	1.19	0.488
Wrist	HS-22BB	27	0.47	0.0433
Gripper	HS-5645MG	60	1.19	N/A

Table 3.2: Chosen Hitec [22] servo motors

3.2.6 The Prototype

The prototype made in accordance with the findings in previous sections can be seen in figure 3.9. While the length of each link was made to fulfill the requirements in table 3.1, the effective link length, meaning the length from one actuating motor

to the next, was somewhat reduced as the motors could not be effectively attached at the extreme end of either link. The first and second motors are attached to wood, as it provided a cheap, robust and easily modified material for experimenting with prototype design. The entire arm is weighted down by a stone block to ensure that it does not tip over when fully stretched. If one was to produce an arm based on the prototype, the wood and stone could be changed to metal or hardened plastic to give it a more stylish look.

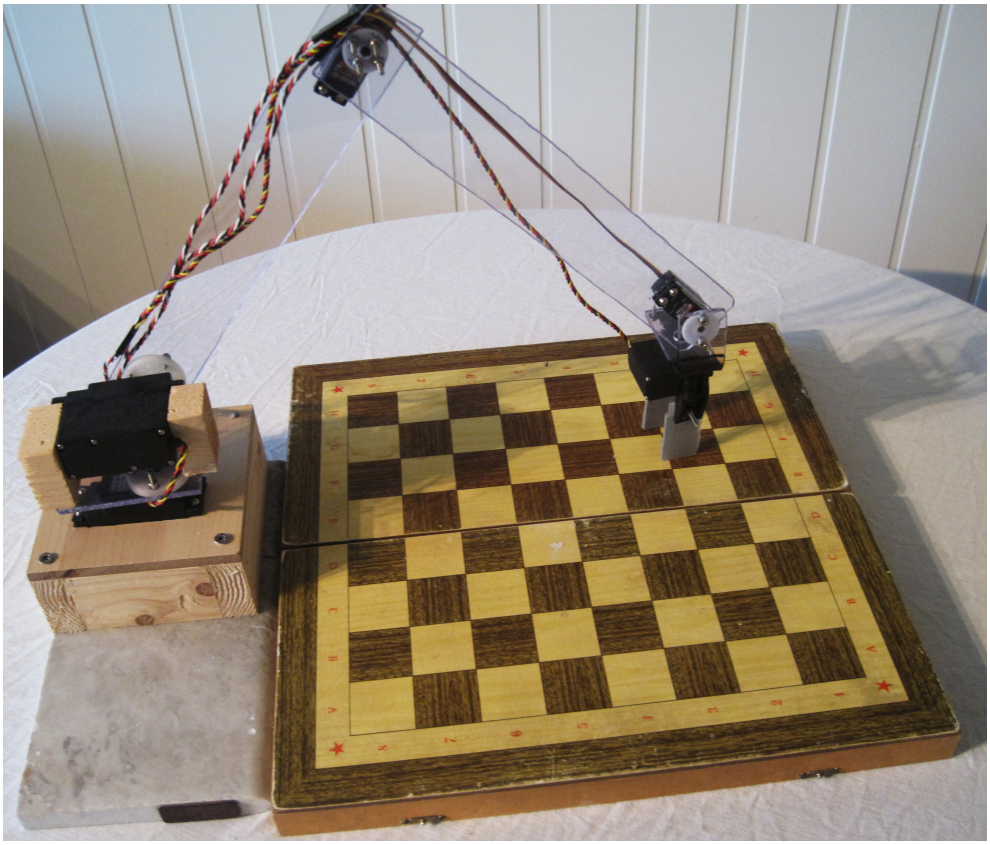


Figure 3.9: Robotic Arm Prototype

A problem with the end-effector occurred when it tried to pick up a small chess piece with large chess pieces nearby, as the motor attached to the end-effector would sometimes hit the nearby large pieces. A solution to this problem was devised by adding an additional wrist rotating joint to ensure that the gripper motor was facing away from large pieces, however, as there could be scenarios with large pieces completely surrounding a smaller one, this solution was deemed infeasible. The problem was solved by extending the grippers reach to ensure that the end-effector could pick up pieces of any size without crashing into other pieces. The added extensions were also bent outwards to increase the maximum end-effector

opening distance. The increase in maximum opening distance allows for some anticipated inaccuracies when approaching objects.

A geometric model of the arm with relevant dimensions is shown in figure 3.10. This model will be the basis of the controller implemented in section 3.4.

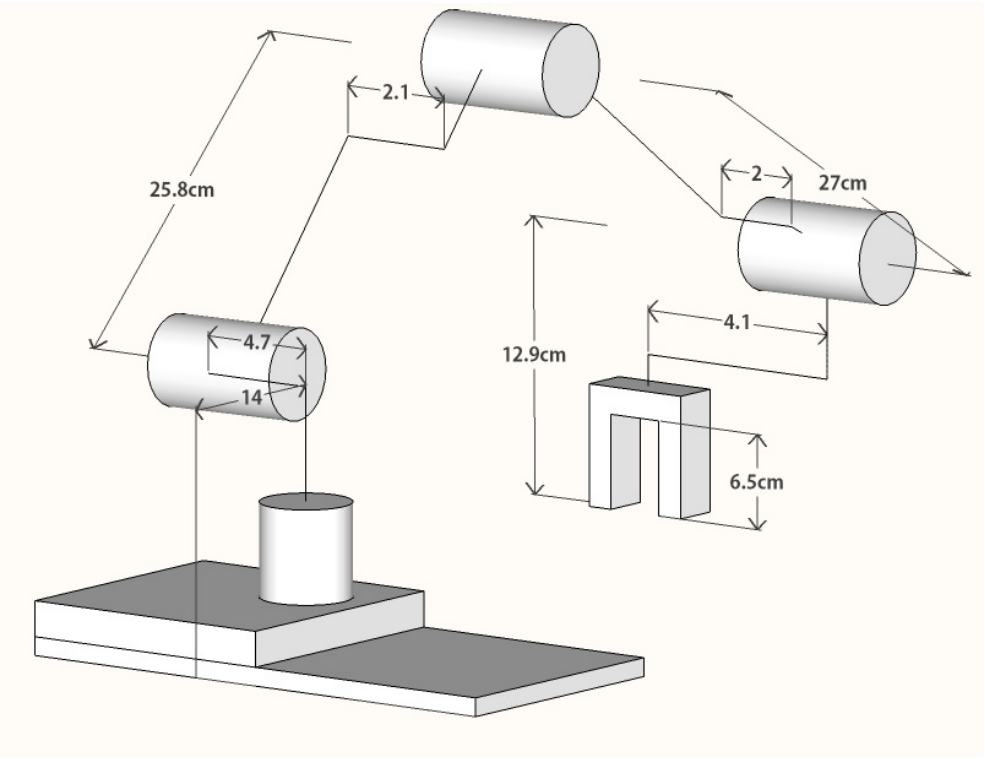


Figure 3.10: Robotic Arm Model

3.3 Motor Control

The motor controller task is to provide individual control of each motor as well as serve as an interface to translate angular positions into PWM signals for the servo motors. The advantage of using servo motors is that closed-loop control of the motors is kept inside each individual motor. The result is that the overall motor controller implemented does not need to concern itself with the inner functionality of a servo motor, but rather serve as an interface between a computer controller and the motors.

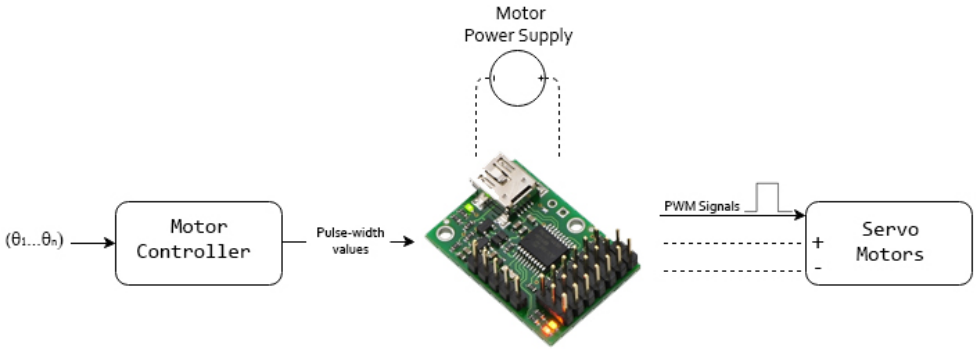


Figure 3.11: Motor Controller

The motor controller takes a joint angle as input and uses a Pololu micro maestro servo controller [18] to control each individual servo motor to the target angle by sending a series of PWM signals. The servo motors have a neutral position when receiving a PWM signal of width 1500μs and, depending on the servo, operate in a signal range of approximately 600-2400μs. The Pololu micro maestro includes a communication protocol that defines the rules for micro maestro's data exchange through USB. This protocol allows custom software to set the output to the servo motors, as well as allowing restrictions on speed and acceleration for increased control over the complete motion.

The computer is connected to the Pololu servo controller through USB. Communication between the motor controller and servo controller is achieved by using the libusb library [19] for controlling data transfers through USB. The motor controller connects to the micro maestro by searching USB-ports for connected devices and comparing each device to the known product- and vendor ID of the micro maestro.

```
vendorID = 0x1ffb;
productId[]={0x0089, 0x008a, 0x008b, 0x008c};
```

The subsequent transfer of data is performed as synchronous I/O, meaning the process waits for the transfer to complete, as opposed to asynchronous I/O where the process can continue while the data transfer is pending. The latter approach is generally better for data transferring as it does not block the process while transferring data, however in this case the data transfer means actuating the arm,

which will make the processing time gained by using asynchronous data transfer negligible compared to the added programming complexity.

The motor controller translates joint angles to pulse width as a function of the neutral position and an angular gain as:

$$PulseWidth = neutralposition + angle \times gain \tag{3.4}$$

This angular gain is defined as the increase in pulse width that would result in a 1° rotation. Initial testing of the servo motors revealed that the motors specified angular gain of $\frac{10\mu s}{1^\circ}$ was inaccurate, as an increase in pulse width of 900μs resulted in a rotation that did not reflect the expected 90° rotation. It was found necessary to calibrate the motor controller to ensure that the correct pulse width was sent to the motors.

3.3.1 Motor Calibration

The servo motors specification rated the neutral position at a pulse width of 1500μs and an angular gain of $\frac{10\mu s}{1^\circ}$. To find the correct gain and set a neutral position that reflects the neutral configuration in the robotic arm controller the motors had to be calibrated.

Calibration was done by using a chess board with known dimensions. The reference points were chosen to be evenly spaced across the chess board, and the corresponding joint angles that ensured the correct arm configuration for each point was found by using the inverse kinematic procedure in 3.4.2. The arm was controlled manually to each of these reference points, and the corresponding pulse width value for each motor at every point was stored. The pulse width values and their corresponding angles were compared between each pair of points that featured a significant angular change. By only comparing points where a motor has seen significant angular change, the error from manually controlling and reading pulse width values was reduced as the error was divided by a larger change in value. The calculated values were averaged across all compared points which resulted in the following neutral position and angular gain:

Motor	Motor Type	Neutral value	Gain
Motor 1	HS-805MG	1590	-9.39
Motor 2	HS-805MG	917	9.93
Motor 3	HS-5645MG	1319	-10.64
Motor 4	HS-22BB	973	-7.83

Table 3.3: Motor calibration values

The negative gain values come as a result of direction of rotation compared to the inverse kinematic configuration, when considering the gains of each motor the absolute value should be considered. The calculated angular gains were found to vary greatly from the specified gains supplied by the manufacturer. The calculated

gains are subject to human error, as measurements and control was performed manually. The calculated gains were briefly tested by instructing each motor to turn 90° and inspecting the resulting rotation. Using the new angular gains the resulting rotations was found to reflect the instructions. A more detailed discussion about the accuracy of the arm will be dealt with later in this paper.

The neutral pulse width position supplied by the manufacturer reflects the midpoint of rotation for a servo motor, meaning the motor can rotate an equal amount in either direction. This value was not found to be incorrect, however, offsetting the neutral pulse width to reflect the neutral inverse kinematic rotation simplifies the control and makes it more intuitively easy to understand. Offsetting the neutral value also helps with reducing the impact a potential error in angular gain would have. By choosing the neutral position as the midpoint between the maximum and minimum angle needed to cover the entire chess board we ensure that the maximum difference from target pulse width to neutral pulse width is minimized. As a result the total angular error resulting from an error in angular gains would be minimized as the pulse width changes linearly as a product of angle and gain. To see this consider a motor that needs to operate in the range of $0° - 90°$. The hypothetical correct angular gain for this motor is 9, but motor calibration resulted in a gain of 9.1. Using a neutral position of 0°, the error resulting from miss-calibration is:

$$|Error(90°)| = \frac{|90 \times 9.1 - 90 \times 9|}{9} = 1° \tag{3.5}$$

Offsetting the neutral to a 45° angle would result in a maximum error in either direction of:

$$|Error(45°)| = |Error(-45°)| = \frac{|45 \times 9.1 - 45 \times 9|}{9} = 0.5° \tag{3.6}$$

The error is minimized with respect to the calculated gain error by choosing the neutral pulse width at the center of the operational range of each motor. Choosing the right neutral position depends on knowing the operational range of each motor, which might be unknown depending on the position of the chess board relative to the arm. In this case the robotic arm's base has been assumed to have a position at the approximate center of the chessboard.

Motor calibration was done once for the chosen kinematic configuration, neutral position and angular gain was stored for future use. If one were to change the design of the arm or replace some of the motors a new calibration would need to be performed. The motor controlling the gripper was not calibrated in a similar manner as the others as it had no kinematic angle of reference. The accuracy of the gripper motor is also less important as it only relies on opening and closing, which does not require a lot of accuracy. With correct calibration, the control of the individual motors should provide the desired accuracy for the robotic arm. Any reduced accuracy could potentially be compensated for by the overall robotic arm controller.

3.4 Robot Control

The motor controller ensures stable control of each separate motor, however, there is still a need for a control scheme that considers the movement of the entire arm. The task of the robotic arm controller is to control the arm to a specified point or object while avoiding collision with the environment. The robot controller is implemented as an open-loop controller that accepts input in the form of coordinates and achieves its movement by instructing the motor controller to move each motor to the desired angle. The coordinates represent the desired position of the end effector relative to the base frame. The controller uses spatial domain knowledge about the chess board in combination with a known geometric model of the arm to calculate the desired angles based on an inverse kinematic approach.

The robotic arm controller is open-loop as it relies solely on its model of the robotic arm and the current state of the joints to calculate the series of joint angles that ensures a stable movement to the target location. The robotic arm controller receives no feedback in terms of its real-time angular positions or pose of the arm, as such it has no knowledge of its current position while moving. When standing still, the robotic arms position is assumed to be known and equal to the output position of the robotic arm controller. We can make this somewhat crude assumption as the closed-loop controllers in the servo motors ensures that they stop at, or close to, the correct angular value. In general, an open-loop control of a robotic arm can pose some problems:

1. Complex path planning and obstacle avoidance is not possible, as it requires knowledge of the arms configuration at all times

2. Information about the current state is unknown, the arm cannot be known to be idle or currently moving because of an unknown duration of motion.

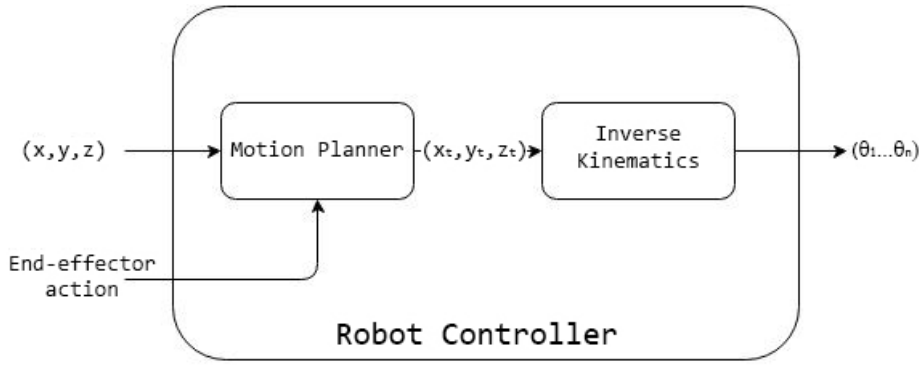


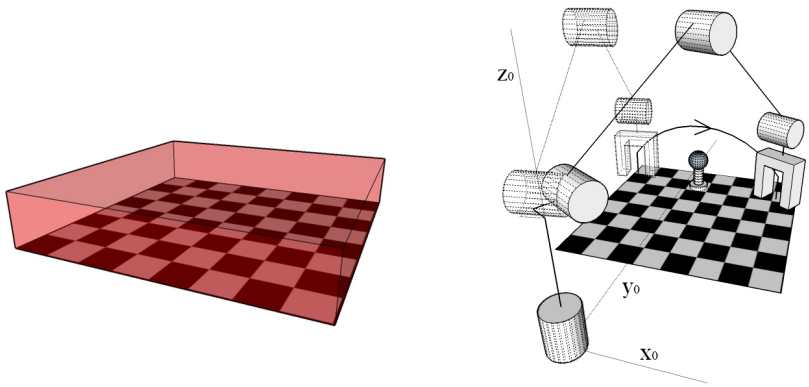
Figure 3.12: Robot Controller

The path planner breaks the desired movement down into multiple discrete movements and estimates the time at which they can be sent to the motor controller to ensure a collision free motion. The implemented path planner is described

in further detail in section 3.4.1. Before the instructions are sent to the motor controller they are translated from positions in Cartesian coordinates to motor angle instructions for the motor controller. The angles are calculated by using a geometric approach to inverse kinematics on the known model of the robotic arm, as describes in section 3.4.2.

3.4.1 Path planning

For a robot controller to perform online path and trajectory planning one would need feedback information in the form of obstacle positions and exact current state of the motors. As the controller only has information about the angles it has instructed the robotic arm of achieving, we have no real-time information about the current state of each joint. To find a path that avoids collision with chess pieces we can make some context specific simplifications about the obstacles. The arm is made to play chess, therefore one can make the simplified assumption that the area above the highest chess piece is obstacle free. The collision zone is represented by a square that covers the entire chess board, with a height of the highest chess piece, as seen in figure 3.13a. Avoiding collision with the chess pieces can be done by ensuring that any movement of the arm between chess board positions, i.e. on the base x-y plane, is done above the collision zone. An example of an obstacle free path between two points on the chess board can be seen in figure 3.13b.



(a) Collision Zone (b) Obstacle avoiding path

Figure 3.13: Obstacle Avoidance

For the arm to interact with chess pieces there is an obvious need for the arm to go into the collision zone. To avoid obstacles within the collision zone, pieces that should be picked up are approached from above. The end effector moves towards a piece parallel to the base z-axis, in a line perpendicular to the chess board. An example of a collision avoiding path followed when picking up a piece is shown in figure 3.14

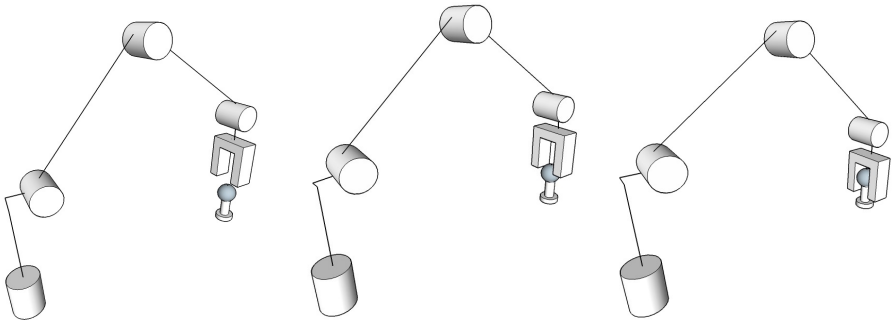
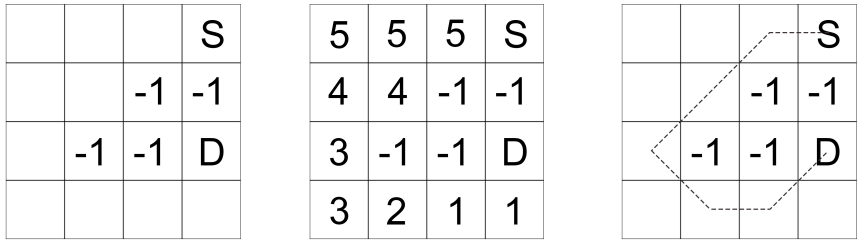


Figure 3.14: Movement towards a piece

The path planner is implemented as a wave front algorithm on a three dimensional map with known obstacle positions. The map is created by discretizing the robotic arm's workspace with assumed known obstacles represented by the collision zone and the robotic arm's base. Each discretized point is then mapped into a three-dimensional vector space with -1 representing obstacle points. The path is found by propagating the distance from the destination to each discretized position based on a breadth-first search. The collision avoidance problem is now reduced to the task of traversing the map from the source position (S) to the destination (D) by finding the path with lowest distance scores. This path will also ensure that the shortest route from source to destination is taken while avoiding obstacles. A simplified path planner for a two dimensional problem with obstacles represented by -1 can be seen in figure 3.15.



(a) Discretized Map (b) Distance Map (c) Path

Figure 3.15: Two Dimensional Path Planning

To ensure that the end-effector approaches pieces from above the discretized space above the target piece is mapped as obstacle free. The accuracy lost to the discretized map is regained by using the original source and destination position near the start and end of the path. The resulting path will be something akin

to an upside down "U". With this approach the path will be affected by the discretized map resulting in a non-smooth trajectory. A smooth path can be found by approximating the discretized path points with a Bézier curve. A Bézier curve can be calculated by recursively interpolating the path points using an increasing interpolation variable. A quadratic Bézier curve for three points is calculated as the interpolation of the interpolated points between P_1, P_2 and P_2, P_3:

$$B(t) = (1-t)[(1-t)P_0 + tP_1)] + t[(1-t)P_1 + tP_2] \tag{3.7}$$

To generalize this to higher order, the Bézier curve is found as the interpolation of previous Bézier curves:

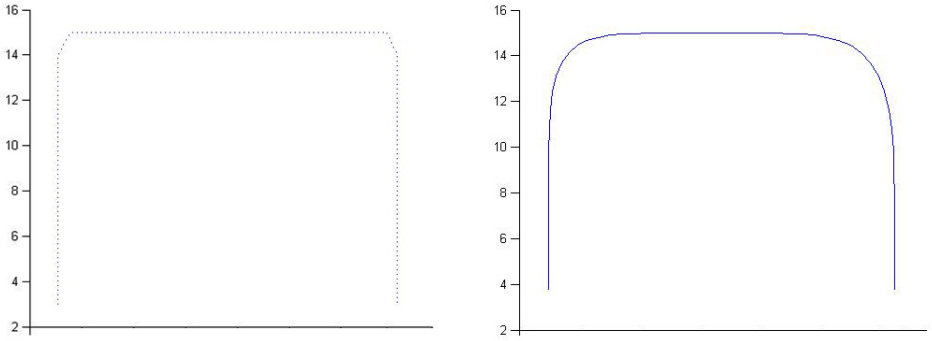
$$\begin{aligned} B_{P_0}(t) &= P_0 \\ B(t) &= (1-t)B_{P_0P_1\ldots P_{n-1}}(t) + tB_{P_1P_1\ldots P_n}(t) \end{aligned} \tag{3.8}$$

Bézier curves have some properties that are advantageous for the chess playing robot:

1. Endpoint interpolation, meaning the Bézier curve's start and endpoint will be the same as the original path's.
2. The start and end of the curve is tangent to the respective start and end of the original path.
3. The curve is represented by a straight line only if the original path is also a straight line.

The implications of (1) means the arm is assured to reach the correct position, so its start and endpoint accuracy is unaffected by the Bézier curve. (2) ensures that the approximated path will approach pieces from above, in a similar manner to the original path. These two properties of the Bézier curve is extremely important in our system as they ensure that accuracy is kept and the overall path does not collide with any objects. (3) is only useful if the discretized path is a straight line, as the resulting path should also be a straight line. An example path and its Bézier curve viewed in the direction straight from source to destination can be seen in figure 3.16.

While this simple obstacle avoidance ensures that the end effector does not collide with any obstacles, there is also the possibility of the intermediate links and joints colliding with objects. As we shall see in subsection 3.4.2 one can choose appropriate joint angles to ensure an upright pose of the arm, and thus avoiding collision with the table or pieces. The controller ensures that the arm follows a path free of collisions by instructing the arm to move to a series of intermediate points before reaching the desired position. This leads to problem (2) stated earlier, the controller does not know when a point has been reached and thus does not know when the arm is ready for new instructions. This problem is solved by using the arc length of each joint in combination with the acceleration and speed restrictions



(a) Original Path (b) Bézier Curve Approximation

Figure 3.16: Bézier Curve Path

of each separate motor to approximate the time needed for each motor to get in position. The approximate duration of the entire move is then found as the maximum of the durations for each individual motor.

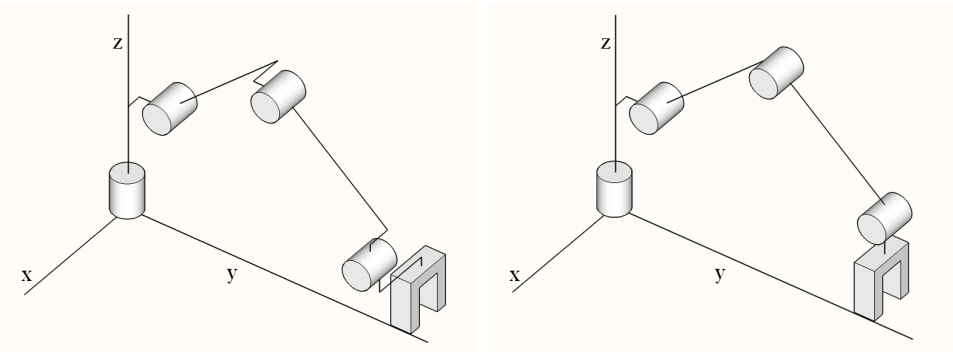
A possible solution to monitor whether the arm has completed a move is to incorporate vision data from the main controller unit. The vision data in the current application does not provide the accuracy needed for visual servoing, however, monitoring whether the arm is moving could be possible.

3.4.2 Inverse Kinematics

The robotic controller uses an inverse kinematic approach to calculating the desired angles for a given end-effector position, thereby providing the motor controller with the necessary positional instructions for each motor. To solve the inverse kinematic problem of a robotic manipulator one needs an accurate geometric model of the arm. The construction of a robotic arm often results in a series of offsets that need to be accounted for when calculating inverse kinematics for the arm. Some of these offsets can be reduced by choosing the correct placement for the motors. However, some joint offsets will usually remain where the motors are attached to the joints.

As seen in figure 3.17a the joint offsets result in the end effector being shifted along the x-axis. The only joint with an axis of rotation that is not parallel to the x-axis is the base motor, which allows us to simplify the configuration by combining all the joint offsets into one. The resulting end effector position and joint angles remain the same, but the simplified model shown in figure 3.17b allows for an easier geometric analysis.

The inverse kinematic problem consists of calculating the four joint angles $\theta_1 ... \theta_4$ from known link and joint paramters as well as the end effector position (x_c, y_c, z_c). By considering the simplified model with θ_i representing the angle of the i'th joint, we can make some observations:



(a) Manipulator arm with joint offsets (b) Simplified manipulator arm

Figure 3.17: Manipulator Arm Model

1. θ_1 is only dependent on joint offset and end effector position.
2. θ_4 ensures the end effector facing downwards, it can therefore be assumed to have a constant impact along the z-axis, and none along the y- or x-axis.
3. θ_2 and θ_3 are the variables that affect the total length of the arm, from base joint to end effector.

Under observation (1) we can ignore joint 2-4 when calculating θ_1, and project the base joint, end effector and link between them onto the x-y plane as in figure 3.18.

The angle θ_1 can be found from the end effector position x_c, y_c and joint offset d as:

$$\theta_1 = \alpha - \beta \tag{3.9}$$

By considering the triangle formed by x_c, y_c and r we find:

$$\alpha = Atan2(y_c, x_c) \tag{3.10}$$

And from the triangle formed by a, r, d

$$\beta = Atan2(d, a) = Atan2(d, \sqrt{x_c^2 + y_c^2 - d^2}) \tag{3.11}$$

Which results in:

$$\theta_1 = Atan2(y_c, x_c) - Atan2(d, \sqrt{x_c^2 + y_c^2 - d^2}) \tag{3.12}$$

Note that there are multiple solutions for θ_1 for every set of coordinates, however, as the joint rotations are restricted by the servo motors rotations, the solution from the left arm configuration has been chosen.

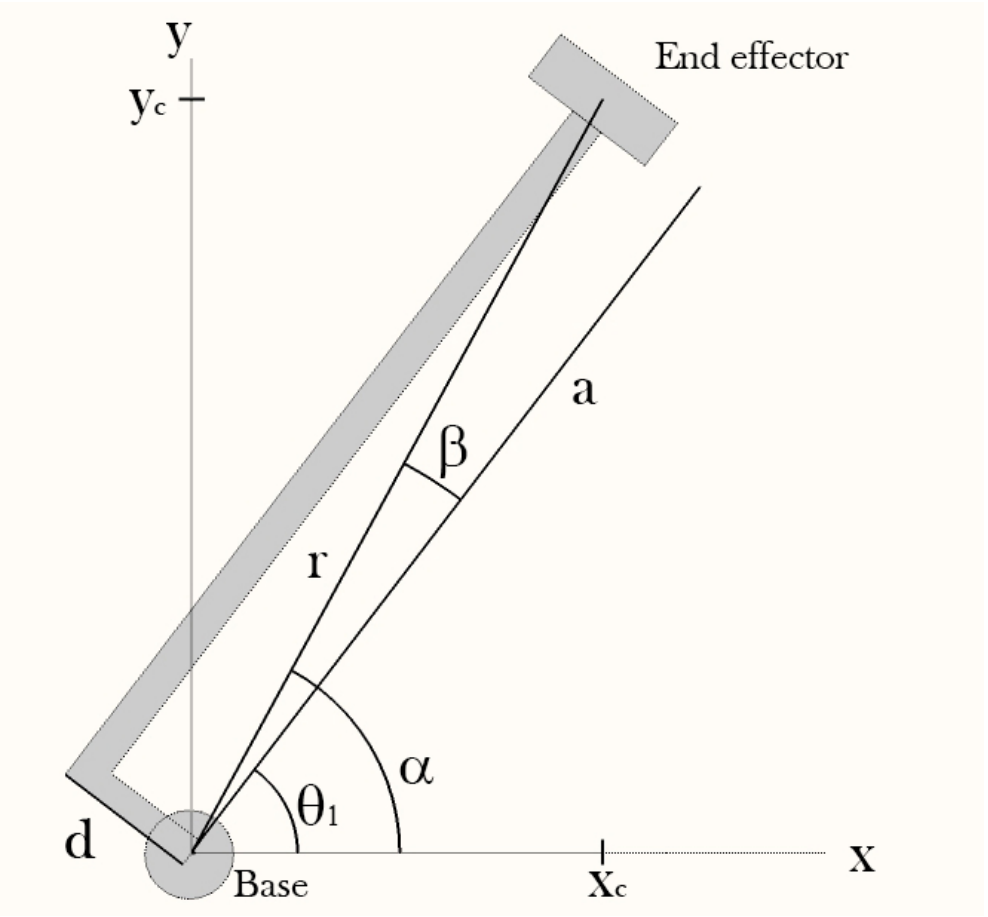


Figure 3.18: x-y plane projection

Given θ_1 we can find θ_2 and θ_3 by analyzing the plane formed by joint 2-4, shown in figure 3.19

Where h is the height from the second joint to the end effector, and a is the distance to the end effector from joint 2, along the direction of the arm. One can see that the distance from the center of joint 2 to the center of joint 4 can be found as:

$$d_{2,4} = \sqrt{a^2 + (h + a_4)^2} \tag{3.13}$$

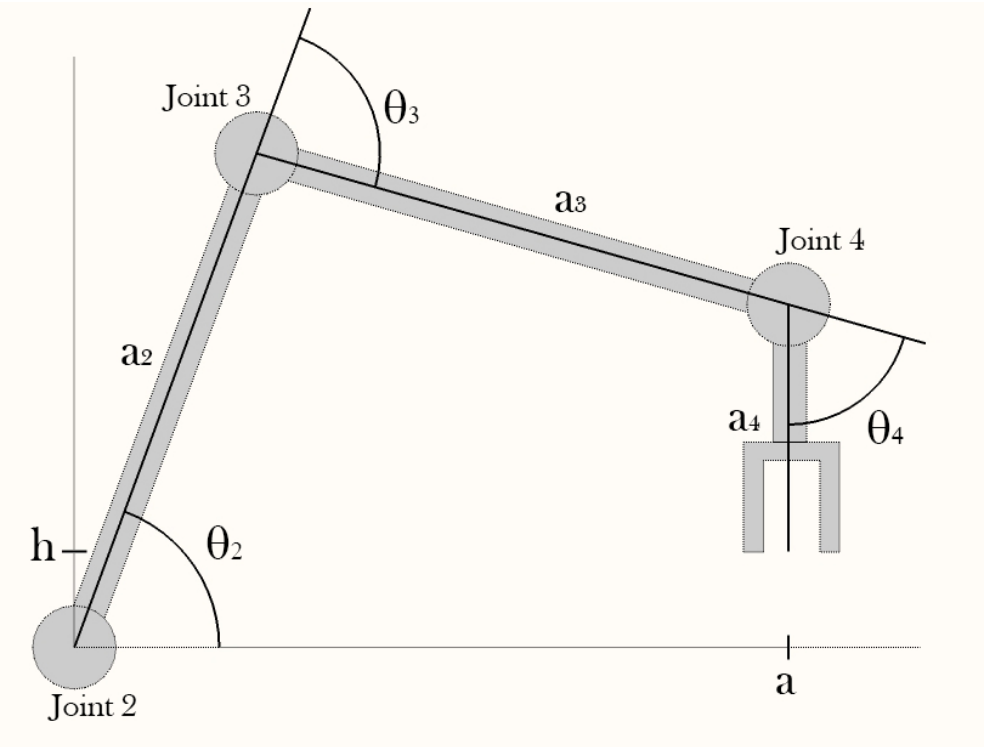


Figure 3.19: Projection on joint 2-4

By combining equation 3.13 with the law of cosines on the triangle formed by the centers of joint 2, 3 and 4, θ_3 can be calculated as:

$$\begin{aligned} d_{2,4}^2 &= a_2^2 + a_3^2 - 2a_2a_3cos(\pi - \theta_3) \\ cos(\theta_3) &= \frac{a^2 + (h + a_4)^2 - a_2^2 - a_3^2}{2a_2a_3} \end{aligned} \tag{3.14}$$

Considering figure 3.18, a can be found as:

$$\begin{aligned} a^2 &= r^2 - d^2 \\ a^2 &= x_c^2 + y_c^2 - d^2 \end{aligned} \tag{3.15}$$

Since joint 4 is always pointing straight down, the height from joint 2 to the end effector is calculated as:

$$h = z_c - d_1 \tag{3.16}$$

where d_1 represents the height along the z-axis from the base to joint 2. Combining equations 3.14 - 3.16 results in:

$$cos(\theta_3) = \frac{x_c^2 + y_c^2 - d^2 + (z_c + a_4 - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{3.17}$$

To get a general result for θ_3 that includes both solutions, one for elbow up and one for elbow down, one can note that,

$$\begin{aligned} cos(\theta_3) &= D \\ sin(\theta_3)^2 + cos(\theta_3)^2 &= 1 \\ sin(\theta_3) &= \pm\sqrt{1 - D^2} \\ \theta_3 &= Atan2(\pm\sqrt{1 - D^2}, D) \end{aligned} \tag{3.18}$$

Choosing the positive root funtion will result in θ_3 facing upwards, and consequently the negative root will give the solution where θ_3 is facing downwards. θ_2 can now be calculated as the sum of angles ω and γ in figure 3.20.

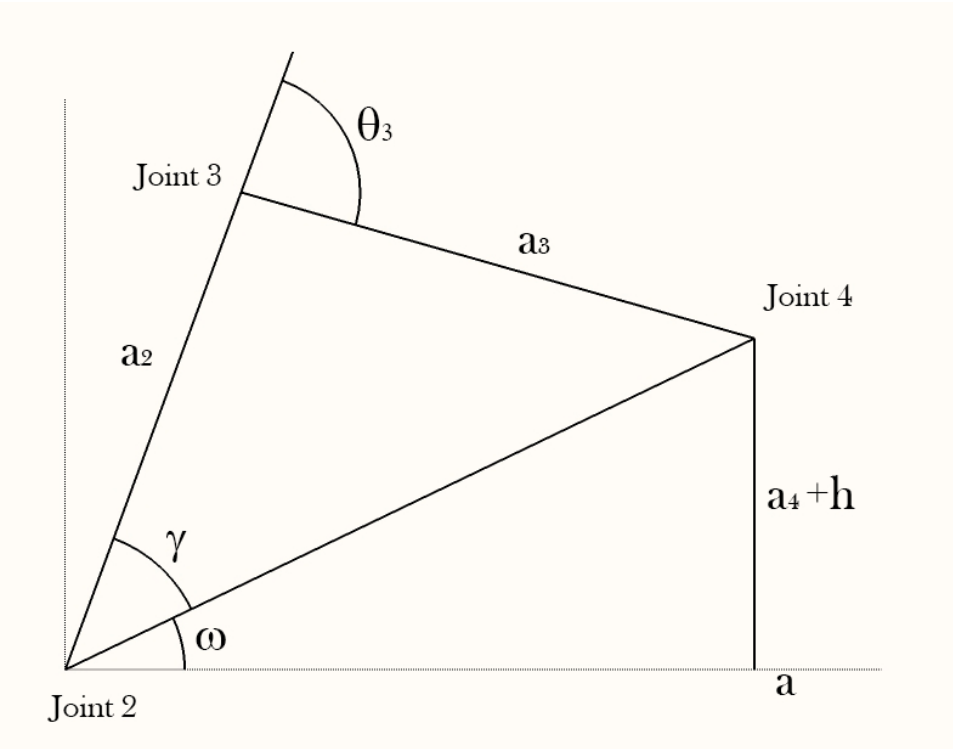


Figure 3.20: Angles ω and γ

ω can be found by the right traingle formed by joint 2, a and joint 4 as:

$$\omega = Atan2(a_4 + h, a) = Atan2(a_4 + z_c - d_1, \sqrt{x_c^2 + y_c^2 - d^2}) \tag{3.19}$$

By considering the the axis along a_2 and its normal axis towards joint 4, γ is calculated as:

$$\gamma = -Atan2(a_2 + a_3\cos(\theta_3), a_3\sin(\theta_3)) \tag{3.20}$$

and as a result:

$$\theta_3 = Atan2(a_4 + z_c - d_1, \sqrt{x_c^2 + y_c^2 - d^2}) - Atan2(a_3\sin(\theta_3), a_2 + a_3\cos(\theta_3)) \tag{3.21}$$

To ensure that the end effector is pointing straight downwards, θ_4 can be found from θ_2 and θ_3 as:

$$\theta_4 = -\pi/2 - \theta_2 - \theta_3 \tag{3.22}$$

The calculated angles represent the complete configuration of the robotic arm. Assuming the initial model was accurate, complete control of the robotic arm is performed by using the path planning and calculated angles as instructions to the motor controller.

3.5 Sensory unit

Any fully autonomous unit that wishes to interact with its environment needs information about its surroundings. Retrieving usable information that is accurate enough for a robotic arm proved a challenging task. The sensory unit was set up with a camera placed above the table and facing downwards to capture an image containing all the relevant data about the chess game. The camera used for this project was a Microsoft LifeCam Studio [23], however, any standard webcam would suffice as neither a high resolution nor special properties is necessary. The camera position needs to fulfill the following requirements:

1. Observe the entire chessboard
2. View the chessboard in a relatively downright manner

The second requirement is loosely termed, however, its main purpose is to reduce the amount of chess piece occlusion and perspective distortion. The unit would need to not only capture the image data but also interpret it into information that would be usable for a chess processing unit. In general, chess engines accept input that represents current game state and movement of pieces.

Computer vision processing was done using the OpenCV [17] library which contains a large amount of functions to aid in capturing, storing and processing images and real time video. The process behind capturing and analyzing images in any programming language is in itself complex. Using OpenCV allows us to shift the focus from implementing trivial functions towards integrating and adapting existing functions into a complex program that can process chess games in real time.

3.5.1 Capturing the image

The camera is located above the chessboard, as shown mounted on the arm in figure 3.21. The captures images will therefore exhibit less perspective distortion and occlusion than if the camera was mounted on any side of the chessboard.

The VideoCapture class enables the opening of a channel to any supported video format or camera connected to the computer, and storing the connection as a VideoCapture object. A video stream is merely a series of images, to analyze the video one needs to process the images it contains. OpenCV reads and stores the image data as a two dimensional matrix for positional image data, with a third dimension for representing pixel values.

```
Mat image;
VideoCapture cap(webcam ID);
Image = imread(cap);
```

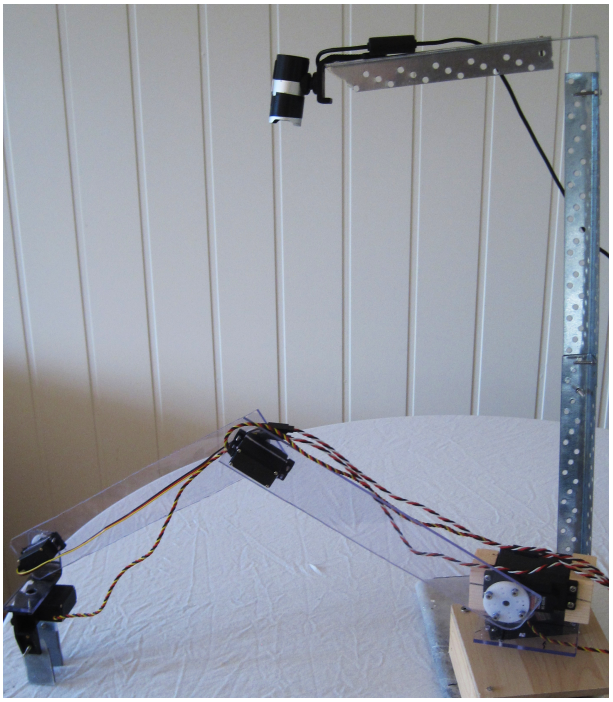


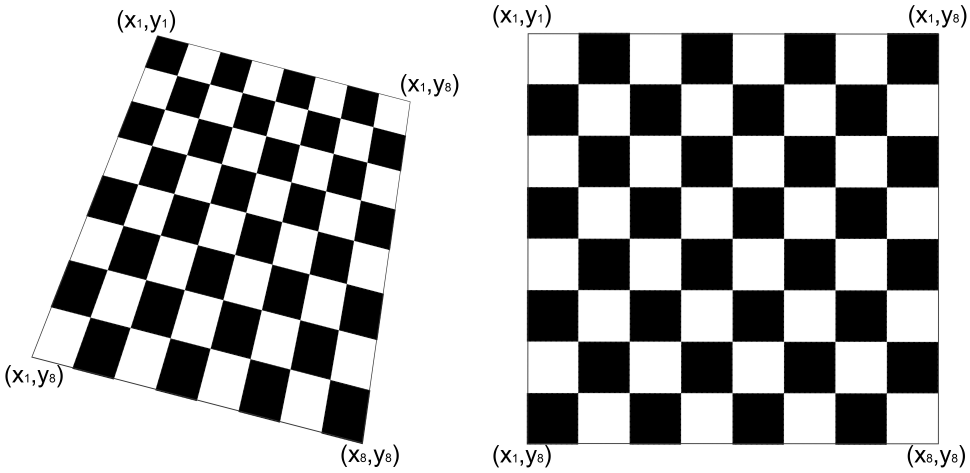
Figure 3.21: Camera Position

3.5.2 Locating the chess board

A lot of chess recognition methods use some sort of assisting mechanism to recognize the location of the chessboard in an image. Some use pre-placed markers of specific colors that stand out, others need manual input from the user to locate the board. Other methods used for camera calibration makes some assumptions about the chessboard such as the camera having an unobstructed view of the board or the board being placed at a certain rotation relative to the camera. The method presented in this section attempts to locate and segment a populated chessboard of arbitrary size, rotation and color.

The goal is for the image to be segmented so the chessboard region can be compared to previous instances. As the board might shift any time during the game, the chessboard region must be extracted and represented in a fashion that is invariant to board rotation and translation. A typical chessboard model from a given perspective can be seen in figure 3.22a. The goal is to segment the chessboard and transform the region so its corner points align with a square region as in figure 3.22b.

Locating and transforming the chessboard region in this way enables detection methods to be made without considering board rotation or translation. Each square can then be accessed by assuming a perfectly square board with chess square length and width equal to 1/8 of the board size.



(a) Chessboard model (b) Perspective transformed chessboard

Figure 3.22: Chessboard Perspective

As all chessboards are based on the same concept, some assumptions can be made:

1. The chessboard has squares of alternating light and dark color
2. The board is placed correctly, with the top left and bottom right squares of light color.
3. The camera is positioned above the table, but one cannot guarantee that pieces are not obstructing the lines and corners of the chessboard.

One approach to finding the chessboard is to perform a corner detection to locate both the extreme corners of the chessboard as well as the corners of each chessboard square. The board could be located by a maximum probability alignment of the located corners. A problem occurs when false corners are detected while some real corners are missing as a result of assumption (3). False corners could come as a result of image noise, variable lighting conditions or chessboard texture. The resulting chessboard detection would also rely on finding a good corner threshold to locate the correct corners. While a corner detection based chessboard locater would have some success, the result was found to give varying results in segmenting the board which would impact the detection of pieces on the chessboard.

The chessboard detection scheme in this paper is inspired by a probabilistic approach presented in [24]. The general idea is to find a set of possible square-like regions based on image edges and transforming the regions to fit a perfect square. A chessboard template is generated and projected with perspective transform onto the square region. The method in [24] generates the template colors by sampling

each square for its average intensity value and uses a variety of masks depicted in figure 3.23 to sample around potential pieces. A drawback with such a method is the assumption of pieces being centered in their respective chess squares, relative to the image frame, a condition that might not hold when a human places the pieces. The method presented in this paper generates the template colors based on domain knowledge about chessboard colors.

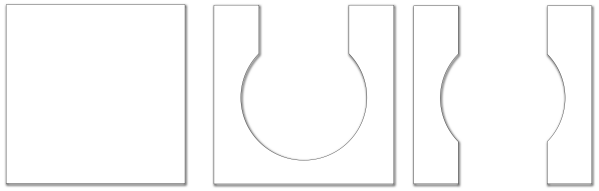
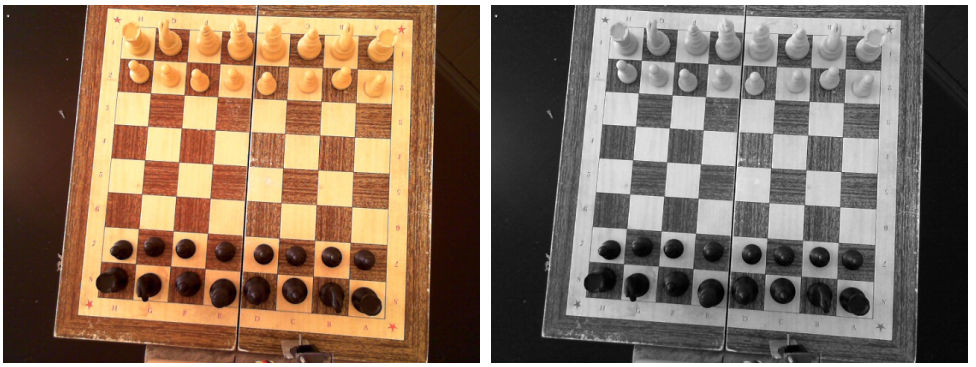


Figure 3.23: Sampling Masks

3.5.3 Chessboard detection: A Hough lines based template matching approach

The first step in locating the chess board is image pre-processing. The colored image is captured in the RGB color spectrum and is usually influenced by some noise or unimportant details. To get a proper edge detection result the image is converted to a gray scale image with pixel values ranging from 0 (black) to 255 (white). A gray scale mathematical morphology operation is performed using a circular structuring element to erode then dilate the image. Figure 3.24 represents a gray scale noise-reduced image and the originally captured image.

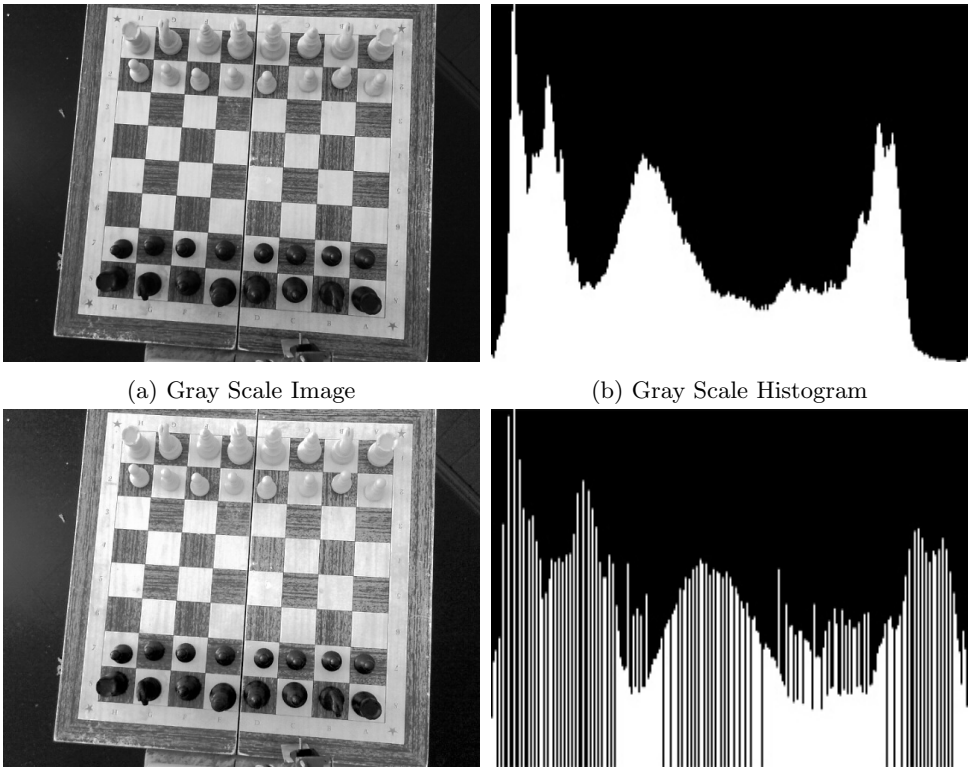


(a) Original Image (b) Gray Scale Image

Figure 3.24: Chessboard Image

To allow for robust line detection there is a need for a robust edge detection scheme. Edge detection often relies on stable lighting conditions to ensure that

results are relatively consistent over time. A typical solution to ensure a robust result is to perform histogram equalization. The histogram equalization increases the contrast of the image so the edge detector can provide more stable results independent of lighting. The resulting image is based on stretching the histogram to expand across all possible values as seen in figure 3.25.



(a) Gray Scale Image

(b) Gray Scale Histogram

(c) Histogram Equalized Image

(d) Equalized Histogram

Figure 3.25: Histogram Equalization

As the chessboard will be located based on image edges the accuracy of the chessboard detection will be influenced by the accuracy of the edge detection scheme. The chosen method for edge detection is the Canny edge detector. The Canny edge detector is chosen based on the requirements:

- Low error rate
- Good edge localization
- Single edge response

Good edge localization is crucial to ensure an accurate chessboard detection that fits the actual chessboard as closely as possible. The single edge detection ensures

that each edge only votes once on a line in a given direction and thus increases the accuracy and reduces the number of duplicate lines. The edge detection resulting from the gray scale image can be seen in figure 3.26.

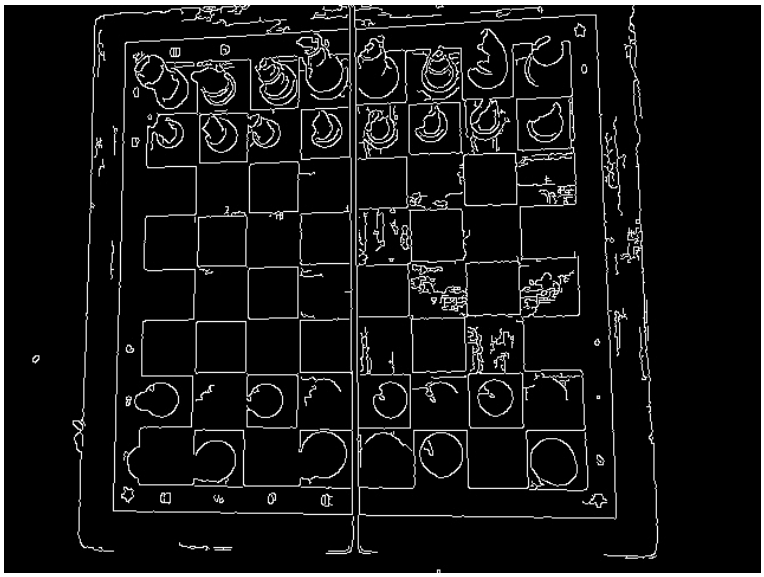


Figure 3.26: Canny Edge Image

As image edges can be partially occluded or influences by noise a more robust line estimate is found by using the Hough transform to detect image lines. The Hough transform is implemented by modifying the existing OpenCV algorithm to get lines more relevant for detecting the chessboard. As the image is searched for a chosen number of lines, the detected lines should be relevant to locating the chessboard and are thus subject to the following two constraints:

1. Two lines may not be of similar angle and position.

2. Two lines that are not normal on each other may not intersect within the image.

In either case, if there are two conflicting lines, the line with the smallest number of votes is discarded. Constraint (1) is necessary to ensure a true local maxima is found when detecting lines as two lines in close proximity has no relevance to detecting a chessboard. Constraint (2) ensures that detected lines are either normal or relatively parallel to each other, and thereby representing potential chessboard lines. The intersection between two polar lines is found by calculating the length along one line to the intersection as:

$$x_1 = r_1 cos(\theta_1)$$
$$y_1 = r_1 sin(\theta_1)$$
$$x_2 = r_2 cos(\theta_2)$$
$$y_2 = r_2 sin(\theta_2)$$
$$b = \frac{(y_2 - y_1)\sin\theta_1 + (x_2 - x_1)\cos\theta_1}{\cos\theta_1 \sin\theta_2 - \cos\theta_2 \sin\theta_1} \tag{3.23}$$

The intersection's x and y coordinates relative to the image frame is then:

$$x_I = x_2 - b\sin\theta_2 \tag{3.24}$$
$$y_I = y_2 + b\cos\theta_2 \tag{3.25}$$

If an intersection is found it is compared to the image size to verify if the intersection occurs within the image frame. The reason for restricting the intersection to the image range is that detected lines may not be perfectly parallel as a result of perspective transform or Hough transform's inaccuracy. The modified Hough transform finds the strongest lines that fulfill the two requirements. The strength of modifying the Hough transform to fit the application at hand is reflected in increased robustness and reduced processing time as irrelevant lines are not accepted for further processing. An edge image with a requested 35 lines drawn in blue can be seen in figure 3.27.

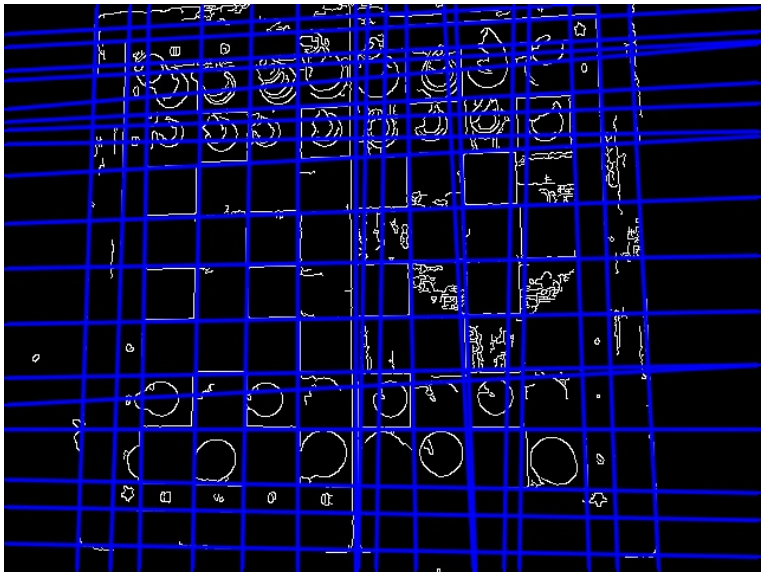


Figure 3.27: Detected Lines

To reduce the amount of detected lines one could request fewer lines or increase the threshold. Fewer lines would reduce processing time, however, it would result

in a lower probability of detecting the correct four chessboard lines. The choice of line detection parameters will involve a tradeoff between processing time and accuracy. Since a low processing time is not the focus for this application, the method is implemented to allow for some redundant lines to ensure more accurate results.

The detected lines are processed to find all combinations of four intersecting lines that form a square-like region. A region is classified as square-like if two vertices have at most a 10% error from forming a square. This restriction is possible to reduce the amount of squares found given the current camera position and orientation, however, it may need to be relaxed if camera pose is changed considerably. To further reduce computation time the four square corners are required to be located in the four image quadrants. This assumption comes as a result of the inherent restriction of the robotic arm that requires for the board to be located relatively close to the center of the arm, and thus near the center of the captured image. Each square-like region now represents a region of interest in the original image.

To find the most likely chessboard region a chessboard template is generated for each region based on sampled intensity values within that region. The gray scale intensity values of the chessboard template can be found by making two assumptions:

1. The squares are of alternating light and dark color, i.e. high and low gray scale intensity values

2. The color frequency of a populated chessboard will be dominated by the colors of the chess squares

Assumption (2) can be made as a fully populated chessboard contains 32 pieces, 16 of each color. As there are 16 unpopulated squares of each color one can assume that the chess square colors will dominate any gray scale histogram of the region as the squares make up a larger portion of the image. This assumption is strengthened when assuming that most pieces do not cover an entire square, and thus some chess square color will be present on populated squares as well.

To locate the chessboard colors each pixel is sampled and a gray scale histogram that represents the frequency of each intensity value is generated. The gray scale value of the chess squares can be found as the two largest local maximum intensity values. A chessboard template can now be generated, however, the regions populated by chess pieces may give a substantial error when compared to the unpopulated template. As the region surrounding the chessboard squares often consists of a color similar to that of the chess squares there is a possibility that the error resulting from the populated squares is large enough to give inconclusive matching results. This problem can be solved by using knowledge about chess piece positions from previous game states to generate a populated chessboard template. The chess piece colors are found in a similar manner as earlier but now by searching for two local maximum intensity values that are distinct from the previously calculated chess square values. The search begins on intensity values outside a region around each of the previously found chess square values. The excluded region

shrinks if no distinct local maxima can be found, resulting in the chess piece colors eventually defaulting as the respective chess square colors if no distinct colors are found. This would occur more frequently as the game progresses and the number of pieces is reduced. In this case the error that ensues from using an unpopulated chessboard template is also reduced. An example chessboard template generated from the intensity value histogram is shown in figure 3.28.

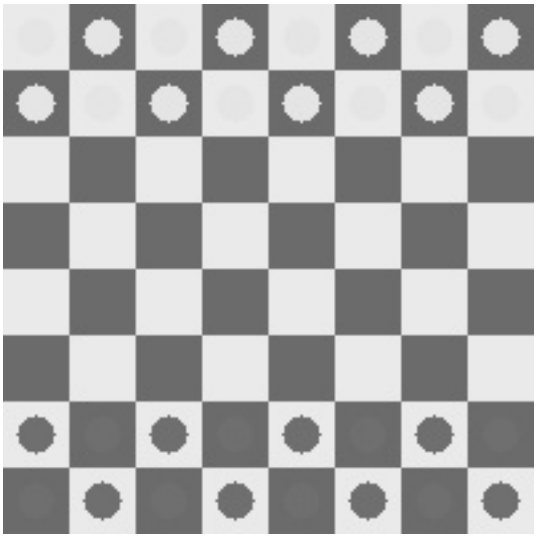


Figure 3.28: Chessboard template

The pieces are assumed to be located near the center of the chess square, however, the total error will be more robust to non-centered pieces as the error will be averaged out across the entire board.

To compare the located square regions with the template image one needs to perform a perspective transform to ensure that both images are viewed in the same perspective. A perspective transform is calculated so that the four corner points of the square region are shifted to form a perfect square of the same size as the chessboard template. The transformation is performed by finding the perspective matrix M that matches the four points (x_i^*, y_i^*) to four square points (x_i, y_i) as:

$$\begin{bmatrix} t_i x_i^* \\ t_i y_i^* \\ t_i \end{bmatrix} = M \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{3.26}$$

Perspective transform is applied to the image by mapping every pixel to a chessboard image as:

$$Chessboard(x,y) = Image(\frac{M_{1,1}x + M_{1,2}y + M_{1,3}}{M_{3,1}x + M_{3,2}y + M_{3,3}}, \frac{M_{2,1}x + M_{2,2}y + M_{2,3}}{M_{3,1}x + M_{3,2}y + M_{3,3}}) \tag{3.27}$$

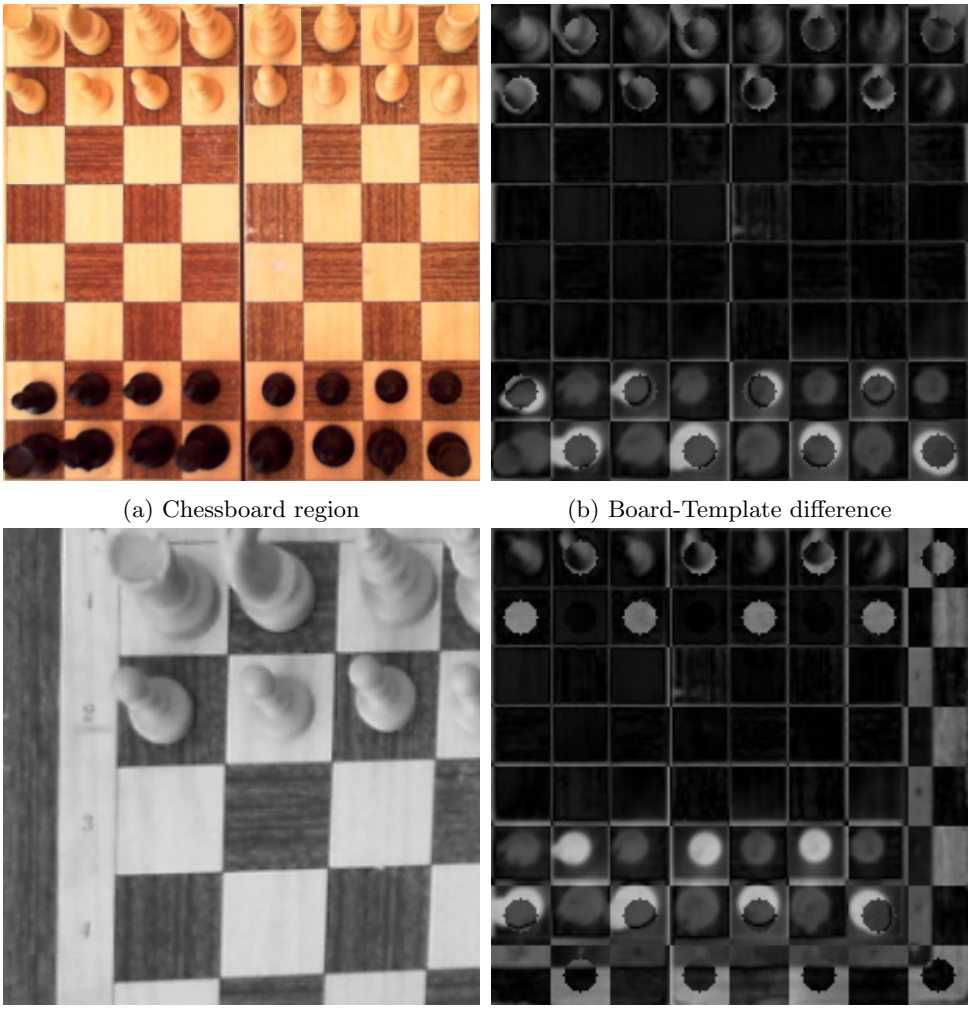
For each square region the perspective transform is calculated and applied to the original image region. The probability is derived by comparing each region to its respective template by calculating the sum of absolute differences between the images. Since both images are of equal size, the difference can be found by taking the absolute difference between each square region pixel value and its corresponding template pixel value:

$$SAD = \sum_{i=1}^{n}\sum_{j=1}^{m} |Image(i,j) - Template(i,j)| \tag{3.28}$$

The square region representing the chessboard is found as the region with the lowest total absolute difference from the template. Based on a given board situation the two most likely square regions and the absolute difference to their respective templates can be seen in figure 3.29. The correct chessboard region had a lower sum of absolute difference, and was therefore chosen as the segmented chessboard region.

The template matching is only performed on extracted line regions, therefore the approach is only as accurate as the line detection method. By properly pre-processing each image the number of false positives and undetected chessboards resulting from inaccurate line detection can be kept small. The approach taken to chessboard detection performed well in a variety of lighting conditions and chess piece positions, further results will be analyzed in chapter 4.

The perspective transformed and extracted chessboard region is stored for each processing time step. It is important to note here that the stored chessboard region is of the original image, hence not the one that was histogram equalized. The reason for storing the original image is that the images will be compared to previous time steps and pre-processing such as histogram equalization can give variable results depending on strong variations in lighting. The advantage of such an approach is that the stored chessboard image stays the same for each time step regardless of the board being shifted or rotated during gameplay. This allows the detection of chess moves to occur without considering movement of the board. Some changes in the region extracted will occur as the detected lines cannot be guaranteed to be the completely identical for each time step. Therefore the subsequent chess move detection would have to be robust to small changes in chessboard region detection.



(a) Chessboard region

(b) Board-Template difference

(c) False chessboard region

(d) False board-template difference

Figure 3.29: Detected Regions

3.5.4 Chess move detection

A robust chess move detection scheme is necessary to update the game state based on the moves of a human player. The method must fulfill some requirements:

1. Robust to changes in lighting and noise.
2. Must detect movement of pieces of arbitrary size, position and orientation.
3. Robust to variations of chessboard detections

A general game state could be found by performing a set of object recognition on each individual piece. This would however require a different camera placement as objects would need to be viewed from the site to perform robust detection. As the camera stands it would be near impossible to provide stable object recognition as the pieces are viewed from above and thus the program lacks information such as object height. A piece viewed from above and the side is shown in figure 3.30.

(a) Queen Above (b) Queen Side (c) Bishop Above (d) Bishop Side

Figure 3.30: Chess Piece Perspective

Placing the camera at a more tilted angle would give rise to a new problem as a fully populated chessboard will result in primarily occluded and partially visible chess pieces. An extreme example of this is shown in figure 3.31 with blue rectangles marking completely occluded pawns. To ensure a robust detection scheme one would need to create a more complex camera system with multiple cameras viewing the board from different angles. Note that the problems of an object recognition approach to chess move detections has so far assumed that all chess sets are the same. Chess pieces come in a large variety of shapes, colors and sizes which makes it highly doubtful that a single piece recognition scheme could perform robustly on any given chess set. The approach taken in this paper therefore assumes no knowledge about the shape or size of the chess pieces, but rather relies on game state knowledge to locate chess moves.

The implemented detection of chess moves is based on using information about previous game states to compare the before and after images of the respective game states. Since each game of chess starts in the same initial state, the position and appearance of each piece is known based on its initial position and the history of moves. The chess squares are formed as an 8x8 grid of equally sized squares on the previously segmented chessboard image with applied perspective transform. The grid of chess squares from the previous segmented image can be seen in figure 3.32.

By considering a new game state where a move has occurred, the task of locating the move is reduced to comparing each respective chess square from the two states. With a perfect segmentation result and no changes in lighting between the

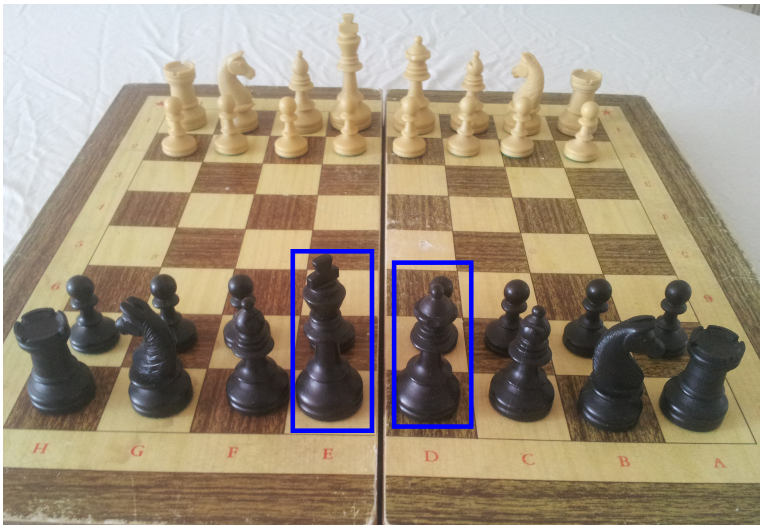


Figure 3.31: Occluded View

two time steps, the task is trivial as the move can be found by comparing each region based on absolute difference. The move can be found by calculating the regions with the highest difference. Such a simple approach may face problems when lighting conditions change between game states or the segmentation result experiences minor shifts in translation or rotation.As a general absolute difference approach will not take into account pieces moving within their own square, the approach is also prone to errors resulting from human players accidentally bumping into other pieces. Generally the difficulty of robustly comparing chess squares is increased in cases where piece colors are similar to the underlying chess square colors. To see this consider the two different scenarios when an unoccupied light square is populated by a dark and light piece. By comparing the two scenarios by absolute difference of a perfectly segmented square the sum of difference is much larger for the scenario with the dark piece although both represent a possible move, as seen in figure 3.33.

The comparison of the light piece on the light square may be too weak compared to the difference resulting from changes in lighting conditions, shade caused by moving pieces or variable segmentation results. Some approaches will be presented and tested with the aim of finding a method that provides robust results regardless of piece colors, lighting conditions or human error. The presented methods make the assumption that any normal valid chess move will consist of changes occurring in two different positions. Therefore a normal chess move is located by finding the two chess squares that have the highest probability being involved in the move.

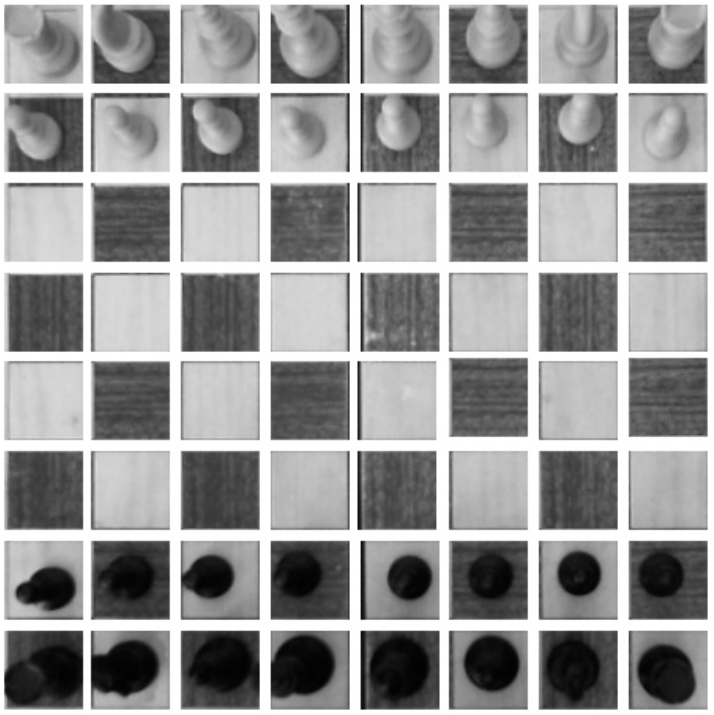
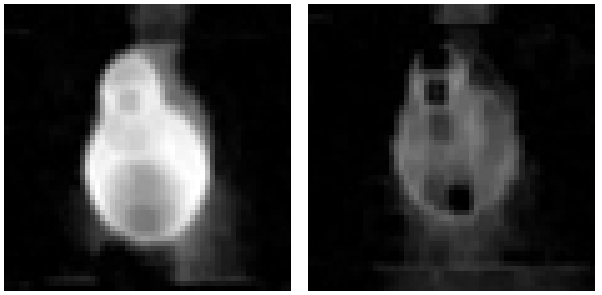


Figure 3.32: Grid of Chess Squares

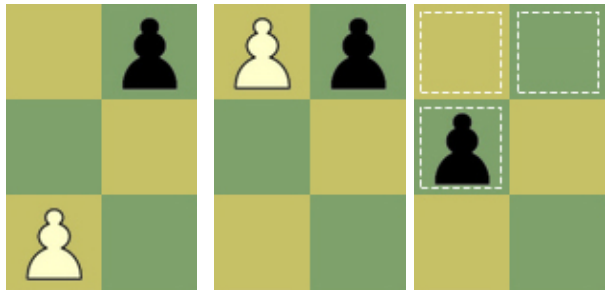


(a) Difference Dark Piece (b) Difference Light Piece

Figure 3.33: Absolute Difference of Chess Move

Some special moves will result in the move being characterized by changes in more than two chess squares. Specifically:

En passant: Special move that results in a move being characterized by the source and destination of the piece, as well as the position of the pawn that is hit. In this case, three chess squares marked in figure 3.34c must be detected.



(a) (b) (c)

Figure 3.34: En Passant

Castling: A special move where a castle and king are swapped and placed on two new squares. The move is characterized by the four squares in figure 3.35, two source and two destination squares.

(a) Step 1 (b) Step 2

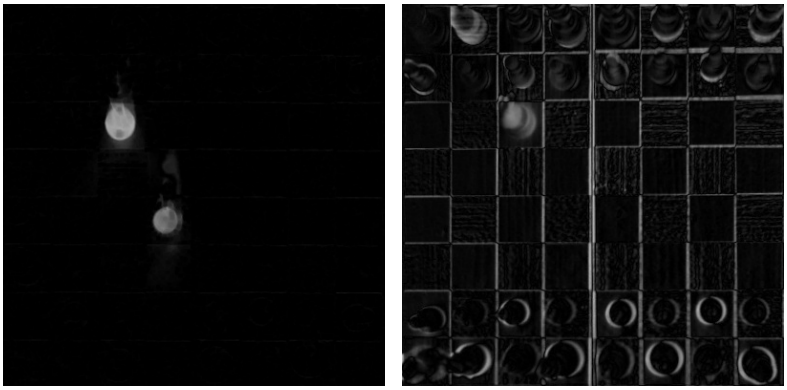
Figure 3.35: Castling

For the reasoning behind these cases see the introduction of special moves section 2.5.2. Both of these special moves can be detected by using information about the previous game state to determine if the most probable chess square changes were a result of a special move. For the sake of simplicity the presented methods will assume that a normal move has occurred. The concept of detecting the two special moves is the same as for a regular move with the difference of using previous game state information to check for the possibility of a special move. The subsequent methods are presented as possible solutions to comparing the various chess square regions and thereby detecting the two regions that experience the largest amount of change.

Sum of absolute differences

The sum of absolute differences approach, as mentioned earlier, locates moves by finding the areas that experienced the largest amount of pixel by pixel change. To account for some inaccuracies as a result of similar colors the absolute difference of pieces on similarly colored chess squares receive a larger weight than other squares. Consider figure 3.36a showing the absolute difference between two identically segmented game state images. The move can easily be located by locating the two grid locations that experience the largest amount of difference. Figure 3.36b is a

result of the absolute difference between two game state images with a small change in segmentation. Locating the correct grid position of the chess move is no longer quite as trivial. This result highlights the weakness of using a pixel-by-pixel based comparison method as it relies on a perfect segmentation result and no human-error in the form of shifted pieces.



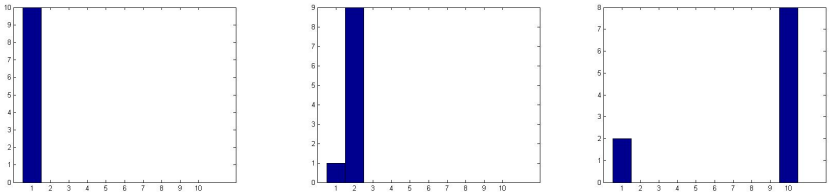
(a) Absolute Difference (b) Shifted Absolute Difference

Figure 3.36: Absolute Difference of Chessboard

Histogram matching

For every chess square two histograms are generated, one for the previous time step and one for the current time step. These histograms contain the frequency of colors in the chess square regions and thus do not depend on the position of colors. An advantage with this approach is the fact that pieces can be positioned arbitrarily within a chess square without its position impacting the result. To compare two histograms one needs to define a robust comparison metric. The simplest approach is finding the sum of the absolute difference of the histograms, however, small shifts in lighting may cause large errors as histograms may shift in either direction. To see this consider the histograms of figure 3.37, histogram 1 is clearly most similar to histogram 2, however, as the absolute differences does not distinguish between histogram position, histogram 1 and histogram 3 would be deemed the most similar.

With variable lighting, a shift like the one presented in 3.37 may occur, therefore a robust histogram comparison method must account for both the value and position of a given histogram color. Additionally the comparison must be performed quickly as a large number of histograms will be compared. The Wasserstein metric, also known as the Earth Mover's Distance(EMD), is a method that defines the difference between two probability distributions of equal size by accounting for both the value and position of each histogram bin. A histogram containing the frequency of color in an image region can also be viewed as the probability distribution of color. Informally EMD can be defined as the minimum amount of work necessary to turn one distribution into the other. By using a popular analogy of the



(a) Histogram 1 (b) Histogram 2 (c) Histogram 3

Figure 3.37: Example Histograms

distributions being piles of dirt, the work is defined as the amount of dirt times the distance it needs to be moved for the two piles of dirt to be equal. For a discrete distribution on a one-dimensional array the EMD can be calculated by iterating through the array and keeping track on how much work is performed between each consecutive step:

```
EMD[0] = 0;
for i:=0 to histogramSize do
begin
 EMD[i+1] = (H1[i] + EMD[i]) - H2[i]
end;
Distance = s
```

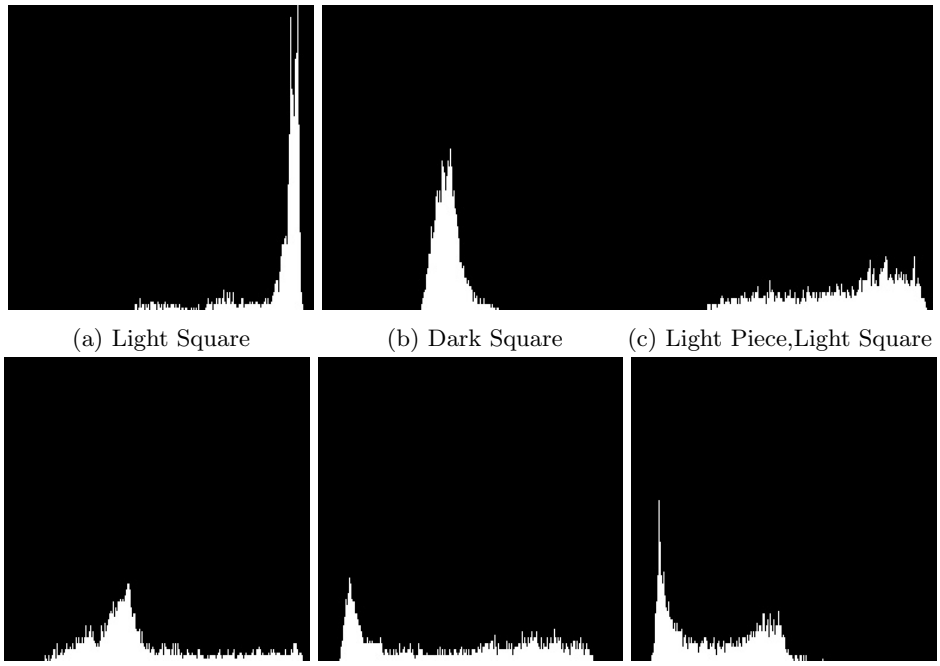
Where EMD_i is the earth movers distance at each step and *Distance* the total difference between the two histograms H_1 and H_2. An advantage of using color frequency to compare squares is that the pixel position of color, and hence position of chess pieces, is insignificant as long as the piece is positioned on a square. By using the earth movers distance the result is also more robust to small changes in lighting or noise.

Template matching

A template based approach to locating chess moves is implemented by generating a template based on previous known square configurations. For a chess game this involves creating six templates, one for each possible situation:

- Light piece on light square
- Light piece on dark square
- Dark piece on light square
- Dark piece on dark square
- Empty light square
- Empty dark square

The purpose of creating a template at each interval is that it allows the move detector to adapt the templates to account for changes in lighting. Two template-like approaches can be taken, one based on the absolute difference between a template and the underlying square, another based on generating histogram templates and comparing the histogram templates to the underlying square's histogram based on earth mover's distance. This approach would locate the most likely content of each chess square, and thus locate a move based on observing a change in square contents relative to the previous game state. For a histogram based template approach, the histogram template of the various gray scale square contents can be viewed in figure 3.38.



(a) Light Square (b) Dark Square (c) Light Piece,Light Square

(d) Light Piece, Dark Square (e) Dark Piece, Light Square (f) Dark Piece, Dark Square

Figure 3.38: Histogram Templates

3.5.5 Move detection testing

An intuitive decision could be hard to make based on the comparison methods presented. To decide on which method should be used, they were all tested in parallel on the same images. Four variations of the histogram matching approach were tested:

- Gray scale histogram
- Gray scale histogram with histogram equalization
- YC_BC_R color histogram
- YC_BC_R color histogram with equalized luma (brightness)

The YC_BC_R color representation is similar to the HSV representation represented in section 2.4.1, in that it separates the color components, C_B and C_R, from the brightness Y. The advantage of using such a color representation is that the brightness component can be manipulated separately to make the recognition process more robust to lighting changes without altering the color components.

Each method is tested based on the number of correctly classified instances and the strength of which these classifications are made. The strength of a classification is defined as the difference of the weakest detected square divided by the difference of the strongest undetected square. The strength of a classification shows how well the correctly detected moves are separated from other squares with the various chosen methods. As the methods will see a limited amount of testing, a method that classifies with a stronger conviction may in the long run outperform a method that has a higher number of correct classifications on the test set. The strength will also serve to separate methods that correctly classify the same amount of moves.

The classification of moves is separated into four categories ranging from best to worst scenarios:

- **Positive:** Correctly classified instances
- **Weak positive:** Correctly classified instances with a weak strength
- **Weak false:** Incorrectly classified instances with a weak strength
- **False positive:** Incorrectly classified instances.

Weakly classified instances result in an inconclusive matching as the gap between correct and false moves is too small. Weak classifications are preferable to false positives as the classification is recognized to be inconclusive and a new attempt can be made on a new image or by using another classification method. The threshold for weak classifications was chosen to be 1.1, meaning the strength of the correct move must be atleast 10% larger than the largest incorrect move. Reducing this threshold would result in a potential larger amount of true positive and false positive classifications, while an increased threshold would result in more classifications being inconclusive. The threshold was chosen to ensure a low false positive rate without classifying a large amount of positive instances as inconclusive.

To compare the methods a normal test and stress test was performed. During the normal test, the testing environment was ensured to fulfill the following requirements:

- Stable lighting conditions
- Stationary chessboard
- Unmoved pieces remain stationary
- No obtrusive shadows

The resulting classifications of each method is shown in figure 3.39. The template matching approach was not included in the graphs as initial testing proved it incapable of providing the robust results necesarry for the chess robot. Based on the normal test, the absolute difference and equalized color histogram approach performed best, reaching a positive classification rate of 99% with 1% weak classifications which were correctly classified on a second attempt. The strength of each method shown in figure 3.40 reflects the mean strength of each correctly classified instance. By using the same threshold as earlier, a strength bellow 1.1 would result in a weak, and therefore inconclusive, classification. Combining the detection strength and classification results proved the absolute difference approach to be the stronger method on a normal test.

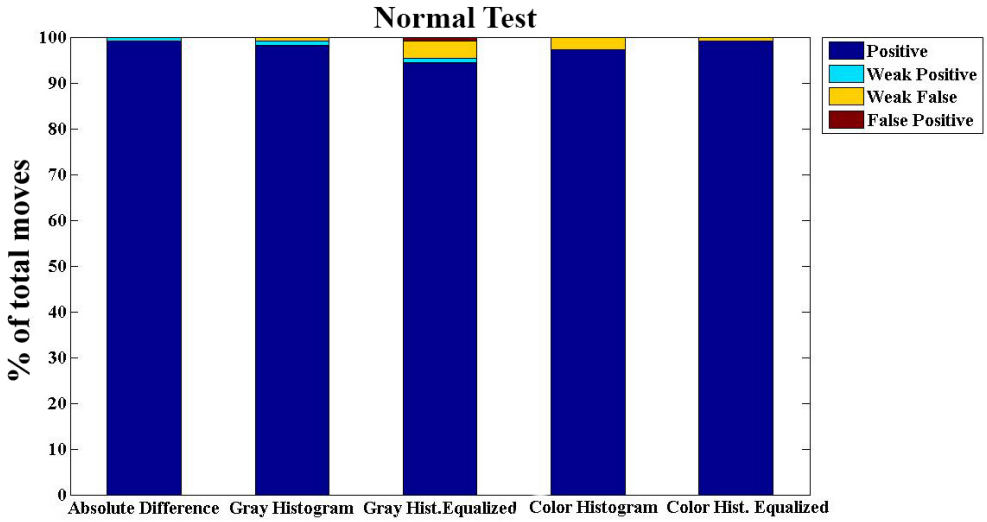


Figure 3.39: Normal test: Accuracy of move detectors

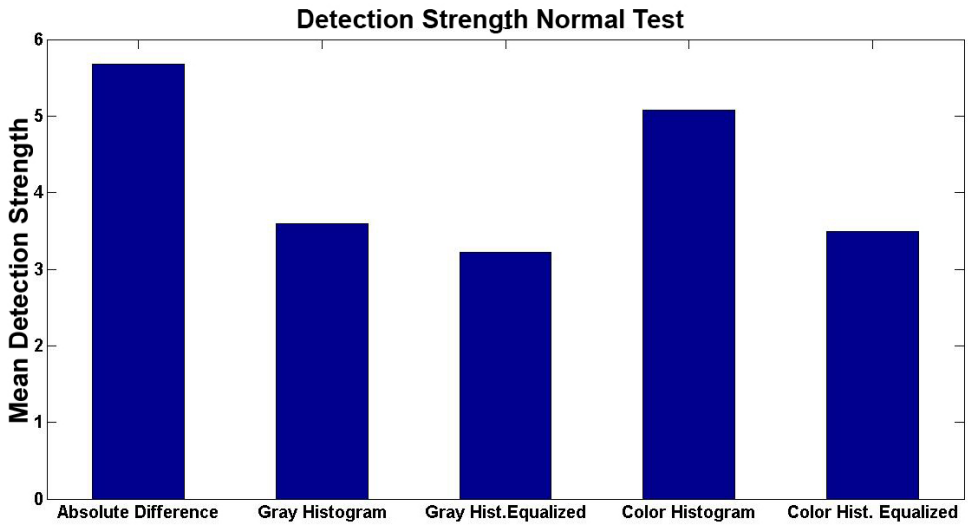


Figure 3.40: Normal test: Strength of move detectors

The requirements for the normal test cannot be guaranteed during regular use, therefore a stress test was performed. The stress test was performed by altering each of the previously assumed static variables between each move, meaning the chessboard was subject to:

- Varying lighting conditions
- Rotated and shifted chessboard
- Shifted unmoved chess pieces
- Shadows from multiple light sources

The stress test's classification results is shown in figure 3.41 and mean strength in figure 3.42. The absolute difference approach as well as the gray scale and color histogram approaches are shown to perform considerably worse during the stress test. The absolute difference approach struggled to correctly classify instances where inter-square shifting occurred at the location of unmoved pieces, and instances where the chessboard segmentation result shifted as a result of varying lighting conditions. Both equalized histogram approaches performed reasonably well on the stress test with a small reduction in correctly classified instances when compared to the normal test. The biggest difference is the occurrence of a false positive as a result of shadows on the chessboard and was incorrectly classified with every tested method.

The comparison method that uses a YC_BC_R color histogram with equalized brightness was chosen for the overall detection method with the gray scale equalized histogram approach being used as backup in the case of a weak detection.

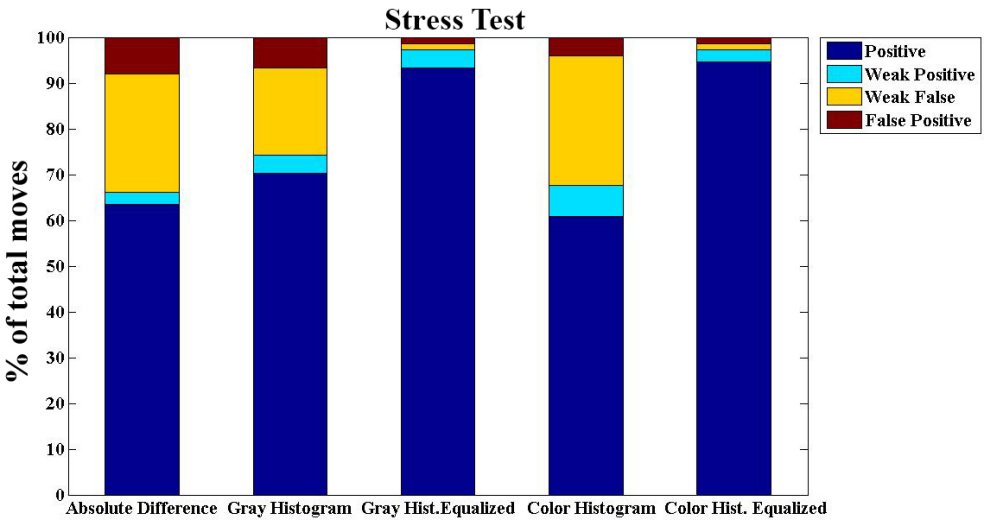


Figure 3.41: Stress test: Accuracy of move detectors

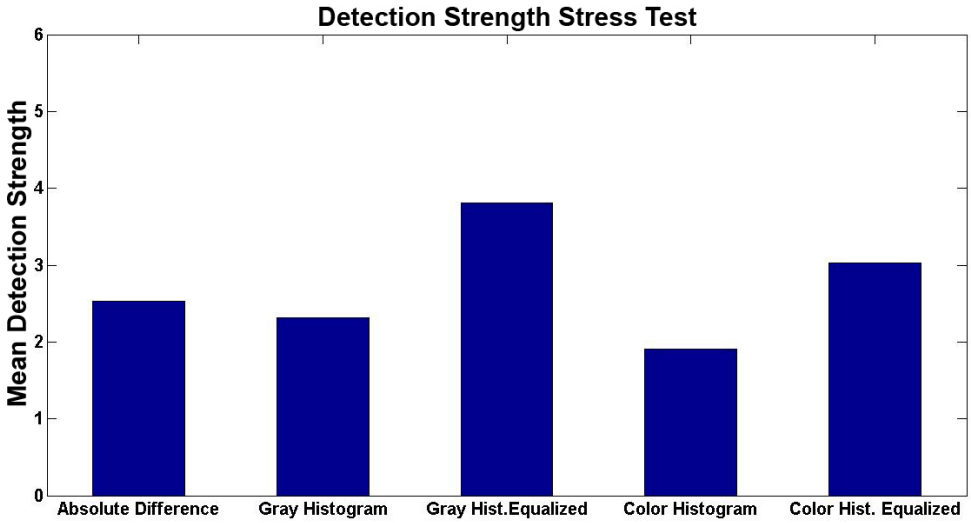


Figure 3.42: Stress test: Strength of move detectors

The results obtained with the detection methods are considered robust enough to be used to update game state information without the need for manual instructions. With the chess unit calculating moves based on the sensory information, the robotic arm can play a game of chess fully autonomously.

3.6 Main Controller Unit

The main controller unit serves as the systems central controller that handles communication and coordination between the various modules of the system. Its main tasks are to keep track of the current game state and ensure that the various modules execute their desired functions at the correct time. The MCU also serves as an interface between the chess and vision unit by ensuring that each module receives appropriate instructions.

The MCU initializes the vision unit, robot arm and the chess unit, with the chess unit running on a separate thread that communicates with the MCU through shared variables and thread locks. A chess round starts with a player moving a piece and informing the MCU that a piece has been moved. The MCU then instructs the vision unit to locate and store the current chessboard image as well as detect the chess move by comparing the image to the previous game state. If the previous step fails an error prompts the user to ensure that a correct move was performed. If the chess move is correct, the user may allow the system another attempt at detecting the move or manually enter the move. Once a valid move has been found the game state is updated and the chess unit is prompted for its move. The round is completed once the robot arm has completed the chess unit's move and a subsequent chessboard image has been captured. The sequence of a round of chess from the human player's move to the corresponding chess unit move can be seen in figure 3.43

3.6.1 Robot Arm Interface

A typical chess move is categorized by the source and destination coordinates in the form of *C2E3* which translates to Cartesian coordinates as (x_1, y_1, z_1) and (x_2, y_2, z_2). A chess robot interface is necesarry to translate chess moves into instructions for the robotic arm. These chess instructions are separated into five categories based on the necessary movement of the robot arm:

- **Regular move:** Move piece from (x_1, y_1, z_1) to (x_2, y_2, z_2)
- **Hit move:** Remove piece from (x_2, y_2, z_2), move piece from (x_1, y_1, z_1) to (x_2, y_2, z_2)
- **Casteling move:** Move king from (x_1, y_1, z_1) to (x_2, y_2, z_2) and tower from (x_3, y_3, z_3) to (x_4, y_4, z_4)
- **Pawn promotion:** Remove piece from (x_1, y_1, z_1), place promoted piece at (x_2, y_2, z_2)
- **En passant:** Remove piece from (x_3, y_3, z_3), move piece from (x_1, y_1, z_1) to (x_2, y_2, z_2).

Where (x_1, y_1, z_1) and (x_2, y_2, z_2) represent the Cartesian coordinates of the respective source and destination chess coordinates. In the case of a casteling or en passant move, the remaining chess coordinates can be calculated based on move

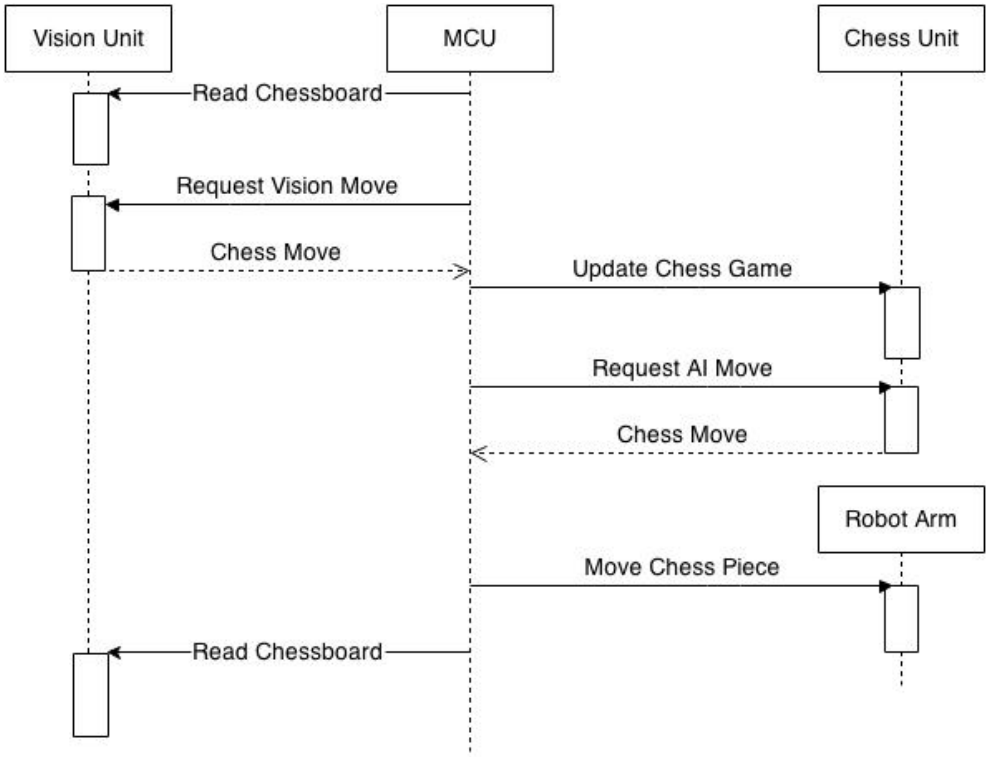


Figure 3.43: MCU Sequence Diagram

type. Chess coordinates are translated to Cartesian coordinates by assuming the chessboard's size and orientation, as well as the distance from the robot arm to the chessboard, is known. For the robot arm to move pieces on a chessboard of arbitrary size and orientation without the need for manual readings, one would need a computer vision method that either calculates these chessboard sizes accurately or a robot arm control that uses vision data for direct feedback control. These approaches will be discussed more thoroughly in section 5.3.

The placement of removed chess pieces is instructed by the chess robot interface by storing a 2x8 grid along both sides of the chessboard where removed pieces can be placed. Previously placed pieces are recorded so the arm retains the information and can place future pieces on available locations alongside the chessboard. The drawback of this approach is that it would require the area on the sides of the chessboard to remain free of any external objects. A more robust solution that involves creating a custom box to room the chess pieces is briefly discussed in section 4.3.

The main controller unit constitutes the last necessary module that ties all the previously implemented systems together.

Chapter 4

Testing and results

This chapter will provide a discussion of the results obtained by testing the various modules, as well as a video test of the complete system. Section 4.1 describes the results obtained from the robotic arm prototype. Section 4.2 provides a discussion on the main results of the chosen computer vision methods as well as a brief discussion of the overall vision system. A discussion of the complete system and some possible improvements will be covered in section 4.3.

4.1 The Mechanical Arm

The robotic arm was tested by instructing the arm to move a variety of chess pieces on a chessboard. The test chessboard was square with a length of 47cm and the robotic arm was placed at the center of one side with a 6.3cm distance from the center of the arm to the edge of the chessboard. Initially, all chess pieces were placed at the center of their respective chess squares. Initial testing proved that the robotic arm managed to move pieces from and to any positions on the chosen chessboard. Some inaccuracies were observed, however, by using a gripper of adequate size, the inaccuracies had negligible impact on the total move.

If the robotic arm is to be used reliably it needs to exhibit a high degree of repeatability. To test the repeatability of the arm, some pieces were instructed to be moved back and forth between two points. Previous movement was repeated with no observable difference. The precision of the robotic arm was tested more thoroughly by instructing the arm to move to a set of test points of known coordinates and measuring the distance from the end effector to the desired test points. The mean error in length and width direction can be seen in figure 4.1.

The error in height was measured somewhat more arbitrarily, ranging from 2-8mm, with larger error occurring further away from the robot arm's base. With the chosen gripper, a height error of less than the object piece would result in a successful move, therefore the height error is less important than the error in width and length direction. Some of the inaccuracies are attributed to the necessary manual calibration of the servo motors. With a maximum chess piece width of

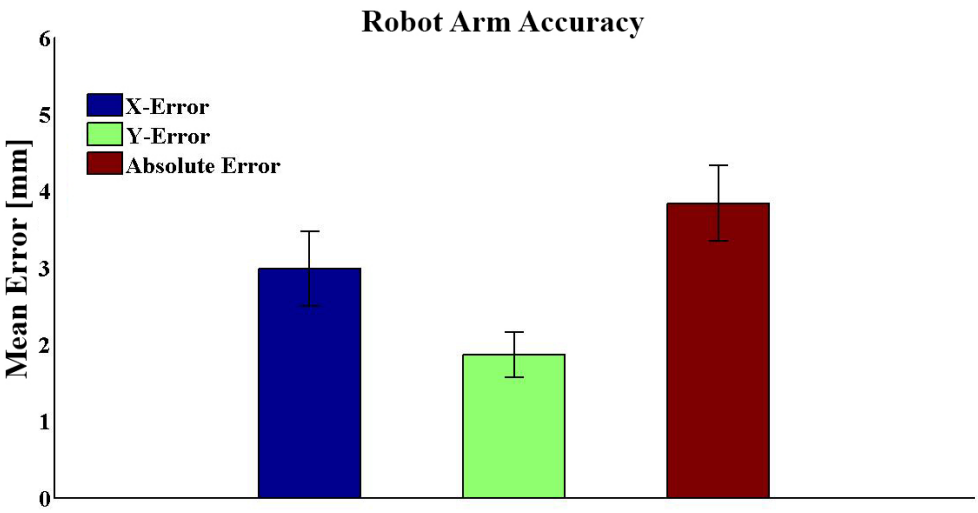


Figure 4.1: Robot Arm Accuracy

2cm and a gripper that extends to a maximum of 3.8cm, the maximum absolute error is required to be less than 9mm. With these requirements, the accuracy of the robotic arm is sufficient for the current application. The most notable errors occurred at the far extreme positions of the arm. The arm was found to resolve small changes in position better in close proximity to the arm's base. The reduction in accuracy is suspected to come as a result of the arm struggling to hold accurate positions when fully stretched. Even though the robotic arm moves sufficiently accurate for the current application, increased accuracy may be possible by using servo motors that are better suited by providing a higher amount of torque. Within the desired price range, no servo motors were found that was better suited for the current application. The chosen servo motors provide a larger amount of angular velocity than what is necessary for the system, therefore one could apply heavier gears, thereby decreasing speed and increasing maximum torque. This would involve either modifying the existing servo motors, or building a new set of custom actuators from base DC motors. This road was not taken for the current project, however, it does stand as a potential improvement by using custom-made motors that are specifically designed for the robotic chess player.

4.2 The sensory unit

4.2.1 Chessboard Detection

Chessboard detection is the core of the vision unit as all subsequent processing is performed on the segmented region. The chessboard detection method was tested under a variety of lighting conditions with various piece placements. By searching the image for 35 lines, the approach performed well during both a normal and

stress test, as can be seen by the detection strength in figure 4.2. The stress test was performed by moving pieces within their respective chess squares and altering the lighting conditions. During a stress test with the sum of absolute differences approach, the correct board was separated by false regions by an average of approximately 35%, reflecting the board detection strength of 1.35. The method was found to segment the chessboard region well during both the stress test and normal test, however, some small variations in the detected region occurred during both tests. These variations were expected as the line detections experienced small shifts under varying lighting conditions.

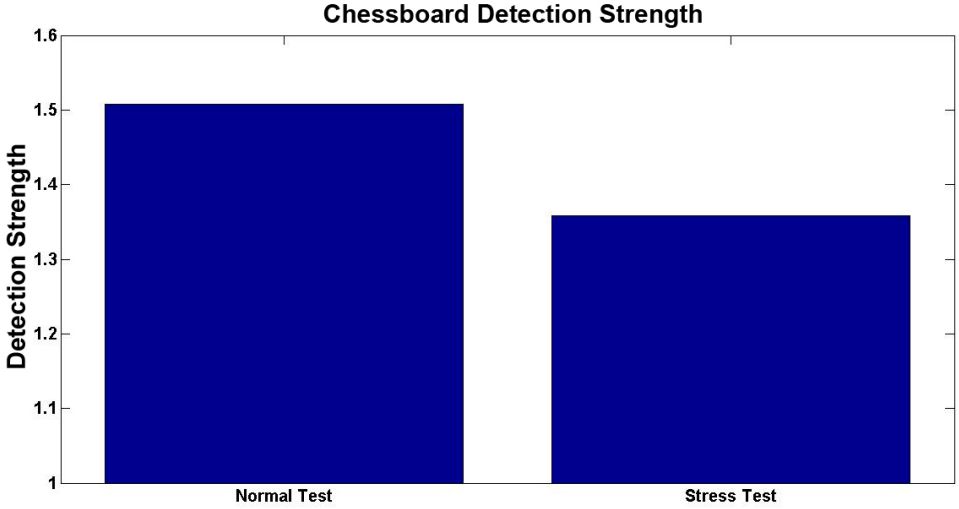


Figure 4.2: Chessboard Detection Strength

The biggest drawback of the chessboard detection method was the need for accurate line detection results. Detection was only performed on regions located by the image lines, therefore the four edges of the chessboard would have to be detected for the chessboard detection to succeed. The change of detecting the four chessboard lines increases with the total number of detected lines, which is chosen as a maximum number of lines. The drawback of detecting a large number of lines is the increase in processing time as more regions are compared to the chessboard template. Processing time with various number of lines is shown in figure 4.3.

As no requirement has been set on how fast the system must react to the opponents move, no correct solution exists on how many lines should be chosen. In general one should seek to use the least number of lines necessary for a robust result. A chessboard will contain at least 18 lines, with multiple additional false lines located at the edges of the chessboard. With the current test environment, 30 lines were shown to be sufficient for a robust chessboard detection scheme, resulting in an average processing time of approximately 1 second. If multiple foreign objects are present around the chessboard, some lines could be shifted to these objects and thus resulting in a wrongly detected chessboard region. Even with the maximum

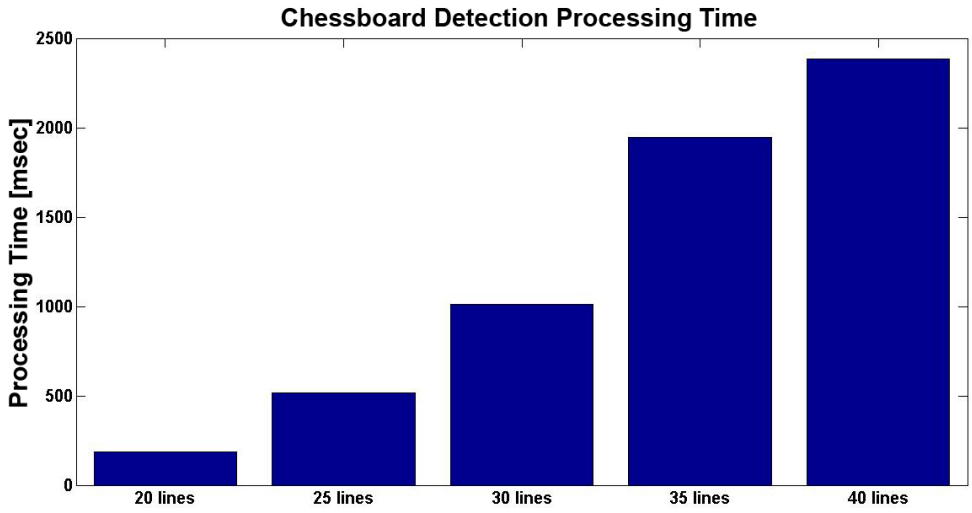


Figure 4.3: Processing time: Chessboard recognition

number of lines chosen at 40, the average processing time does not exceed 2.5 seconds, which should be sufficient for a chess detection scheme. Note that no time has been spent on optimizing the processing time of the current solution, therefore one could expect there to be some room for improvements.

4.2.2 Chess Move Detection

A robust move detector would need to recognize moves on any kind of chess set with a high accuracy. A chess move detection approach based on color histogram equalizing was chosen based on the tests in section 3.5.4. Based on some differences in weak positive and weak false classifications, a gray scale equalizing approach was chosen in case of an inconclusive result in the former approach. The methods were tested on a chessboard without requiring disjunctive piece colors or known chess piece shapes. A normal test was run with no distinct lighting changes between moves, and a stress test was run by changing the lighting conditions between moves as well as shifting pieces arbitrarily within their respective chess squares. The results of the two approaches can be seen in figures 4.4 and 4.5.

During normal testing the color based detector reached a positive classification rate of 99%, with 1% inconclusive classifications. If no conclusive classification can be made with either approach, the detection is rerun on a new image. With this approach, all moves during normal testing were correctly classified. The only drawback of an inconclusive classification is the necessity of processing another image, thus increasing processing time.

The stress test highlighted a problem with the current detection approaches as one move was misclassified. The misclassification occurred as a result of a light source placed by the side of the chessboard and therefore casting dark shadows

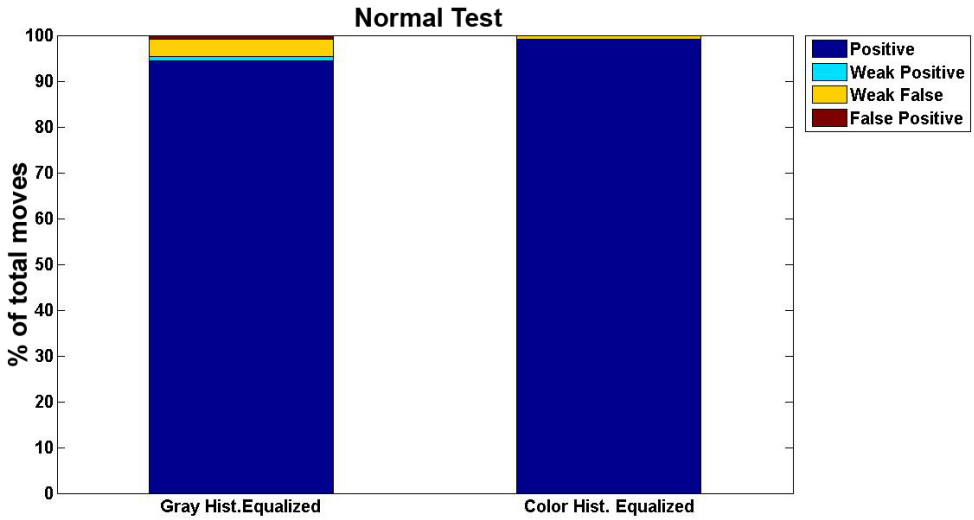


Figure 4.4: Accuracy of Move Detectors

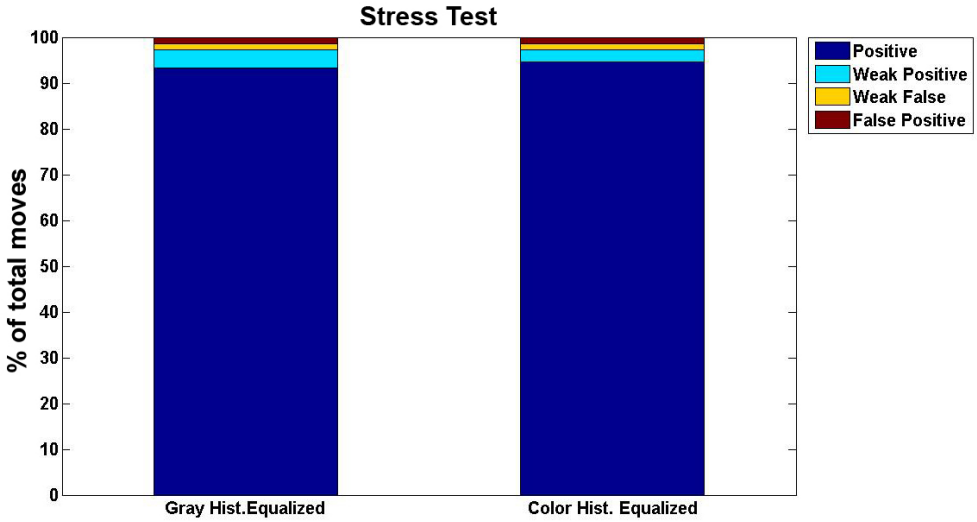


Figure 4.5: Stress Test: Accuracy of Move Detectors

on various places on the chessboard. The error occurred as the chosen chess piece colors were similar to the underlying chessboard colors, therefore a square that experienced a dark shadow also experienced a stronger change in color than the square containing the moved piece. No tested method was found to be robust to these types of shadows with the chosen chess piece colors, however, the misclassification did not occur on a test run with black chess pieces on brown squares and light chess

squares. To ensure a robust detection result one would either have to use chess pieces colors that differ from the chessboard colors or ensure stable lighting with few obtrusive shadows. As discussed briefly in section 3.5.4, object recognition using standard cameras was deemed infeasible due to camera placement and the large variations of chess pieces. A possible improvement to the current solution is to add a time-of-flight camera that provides depth information in the image. Using depth information the current system could be more robust to various dark shadows and changes in lighting. A depth camera was not pursued as an option because of a substantial increase in total cost, nonetheless, future advances in depth camera technology may provide cheap alternatives that could make it a viable option.

The total processing time for the complete sensory unit with 25 and 30 line detections is shown in figure 4.6. Move detection takes a negligible amount of processing time when compared to the three methods involved in chessboard detection. This leaves room for further improving the robustness of the move detector by adding more functionality, such as by adding a depth camera, or running multiple different methods in parallel and choose a result based on weighted voting.

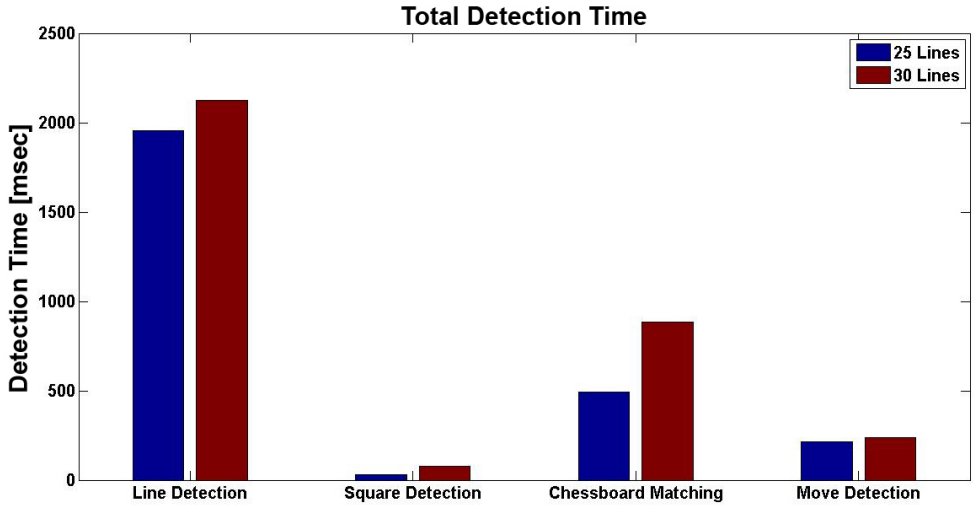


Figure 4.6: Processing time

4.3 The Robotic Chess Player

This section will cover the combined system as a whole and discuss some potential improvements to the overall system. The complete system was tested under relatively stable lighting conditions. A short video demonstrating the performance of the chess playing robot has been added as a digital attachment to the paper. The video serves as the basis for discussing the results of the robotic chess player, therefore any reader of this section is advised to view some of the added video footage.

Note that the video is taken to reflect the ability of the robotic chess player, and therefore does not represent the author's own chess ability.

As seen in the videos, the robotic chess player manages to play a game fully autonomously, with the only human interaction occurring to signal the turn-taking. The sensory unit manages to recognize chess moves with a high degree of accuracy, however, the video shows a significant amount of time being spent on processing the images between moves. The chessboard detection scheme in the video is slower than an improved version as the method used 40 detected lines while storing some data for troubleshooting. The findings of section 4.2.1 proved that this could be reduced to 30 lines, thereby decreasing the processing time by approximately 1.5 seconds. The total processing time of the updates sensory unit is in the region of 3-3.5 seconds, which was deemed sufficient for a chess playing robot.

The robotic chess player performed well when both picking up and placing pieces. The pieces were all approached in a relatively downright manner, and picked up without significantly bumping into nearby pieces. The placement of pieces occurred in a similar manner, with most of the chess pieces being completely contained within the target chess square. Later inspection of the source code revealed a rounding error of the coordinates that resulted in a small shift of the end effector after a piece was placed. This small shift can be seen in the video, and while it had no significant impact on the overall result, it was fixed after the video was made. The robotic arm remained consistent over a significant amount of time, with no observed deteriorating behavior as a result of prolonged run time.

Both speed and acceleration of each motor is limited to ensure that no unnecessary strain is put on the robotic arm. Increasing these limits would result in the arm performing its actions quicker, at the expense of reduced stability. As displayed in the video, the link between the third and fourth motor experiences some instability during rotation of the base motor. This comes as a result of poor quality of the servo motor's attachment horn. The system compensates for this instability by waiting for the link to adjust itself back into the correct position. For a more stable solution it is possible to experiment with different servo motors or design a custom-made motor attachment that ensures a more rigid connection.

The biggest drawback of the current system is the lack of accurate time control of the motors. The control circuit is integrated into the servo motors, therefore the overall motor controller and robot controller has no feedback information about the current position of each motor. The only known positional information exists in the form of the position each motor has been instructed to achieve. The lack of accurate time management is shown in the video as the robotic arm pauses at certain checkpoints based on an estimate of the duration of the previous move. At some points the estimated time exceeds the actual travelling time, resulting in a non-smooth movement as the arm waits longer than what is necessary. In contrast a lower estimated time results in the arm potentially moving into areas with obstacles. While the robotic arm is capable of playing chess, feedback information from the motors would result in smoother movement and a reduction in total time necessary for the arm to move a piece. Some potential improvements to the current solution will be discussed in chapter 5.

Chapter 5

Discussion

A detailed discussion on specific results and modules was presented in chapter 4. This chapter will serve as a more general discussion about the project and the work process used to develop the robotic chess player.

5.1 Goal and Method

The goal was to create a fully autonomous chess playing robot, with all necessary software and hardware. The focus was on creating a functioning system by using a breadth-first approach, meaning each module would be developed up to its simplest working point before focusing on further improvement to any specific part of the system. This approach was used to focus on creating the overall system without going into too much detail on one specific module, but rather create the basis for each module and improve them as time progressed.

Given the author's lack of experience with creating a mechanical arm, a complete robotic arm was initially going to be bought and the focus shifted to adapting and creating a system around it. Unfortunately the robotic arms currently on the market were either too expensive or did not fulfill the requirements in section 3.2.1. Consequently the decision was made to build the mechanical arm as well. The process of designing and creating the mechanical arm was very time consuming, but it was interesting to see if one could create a robust robot arm by using relatively cheap motors and equipment. Developing an arm for the project also allowed a large degree of freedom when it came to adapting and altering the arm to better suit the application.

In hindsight it may have been a bit ambitious to create a complete chess playing robot given the time restrictions. While working with such a large project from start to finish provided some interesting results, it may have been better to focus on a smaller portion of the overall chess robot. With a more narrow focus on some modules, the results obtained from each module could have been more interesting in regards to a fully commercialized product. Even so, designing the entire system was an interesting task that resulted in a basis for further work to improve the

chess playing robot. This paper also has the advantage of providing a guide on the complete design of a fully autonomous robot, including mechanical construction, sensory equipment and robot control.

With practical projects there will always be a significant amount of time spent searching for errors and making the modules of different theoretical fields compatible with the rest. To reduce the amount of time spent implementing, the project could have been approached in a more theoretical fashion with simulations replacing real world testing. While this would free some time for researching each part of the system more thoroughly, it would also result in a less satisfactory outcome as no actual development would have been made.

When it comes down to playing chess against a computer opponent one could envision simpler approaches than the one taken in this paper. One such approach would be an electronic chess board with integrated sensors and chess unit that uses moving magnets inside the board to move pieces. While an electronic chess board could exhibit better results, it would also suffer from needing expensive sensors and specialized equipment. For entertainment purposes it is not likely that people will want to spend a large amount of money on a board with one single purpose. By using a robotic arm and camera to provide movement and sensory data the system serves as a more general purpose entertainment system that could play any number of games by swapping the decision-making and computer vision software. With the approach taken in this paper each part of the system is created to be general and self-sustaining, so any one module could be replaced or modified as long as new modules respect the current communication protocols.

5.2 Main Result

The goal was to create an affordable chess playing robot, capable of playing chess using a variety of different chess boards and pieces. The resulting system managed to play chess with a sufficient amount of accuracy and stability in the testing environment. With a total prize of approximately 2000 NOK, the goal of creating an affordable robot has been achieved. A further reduction in price could be possible by using mass produced parts and custom-made motors. The largest obstacle that remains unsolved is the increase in stability and robustness that would be necessary for a fully commercialized product.

The robotic arm performed its actions well, however, it did exhibit some lack of accuracy when moving pieces situated far away from its neutral position. While the cheap materials, fasteners and manual measurements can account for some of these errors, there was no doubt that the servo motors lacked the accuracy to achieve near zero error. Choosing other servo motors that fulfill the torque requirements could improve the system, but in general the added accuracy would lead to more expensive servo motors. The focus when building the arm was to buy components at a level that provided adequate control of each component, however, in the case of the servo motors it could have proven advantageous to build the motors from scratch. While it is no guarantee that the result would have been better it would have given better control and allowed for the implementation of closed-loop control

from of the robotic arm. This in turn would allow for more complex path and trajectory planning. The reason servo motors were chosen as the main actuators comes down to the same reasoning as stated earlier, too much focus on one specific module would not give an adequate amount of time to explore the full system solution.

The obstacle of using computer vision to calculate exact board orientation and chess square positions relative to the robotic arm was not overcome. The system as it stands needs manually measured chess board dimensions to function, a feature that is not desirable for a commercial product. Needing chess board dimensions came as a result of the open-loop robot controller relying on accurate spatial coordinates and inverse kinematics to provide control. Another solution would be to use vision-based robot control, also known as visual servoing, where the error between the end-effector and desired object is processed based on visual data and fed directly into the robots feedback loop. This approach could result in a more general solution for control of the robot manipulator, however, it would require an extensive amount of research and work to accomplish, and given the scope of the task it would most likely have diminished other parts of the project. An approach based on visual servoing may give rise to new problems as it introduces another module prone to error. It is believed that further work in the chosen direction could result in a more general purpose game playing robot, while still retaining its ase functionality.

When designing the robot manipulator most of the focus was on achieving a certain degree of functionality, therefore the arm was made in a way that allowed for easy modification and testing. The downside with this approach is that the arm did not end up being very aesthetically pleasing. If one were to commercialize such a product, the constructed arm would serve as more of a concept platform than an actual visual representation of the product.

The results obtained in this paper can be used for many purposes. A robotic chess player could see its use as a replacement for human interaction in scenarios where a human player is necessary. While some of this need has been replaced by computers and gaming consoles, playing on a physical board would provide a more natural feel and a more intuitive interface as most people are familiar with a large variety of board games. In a retirement home, a chess playing robot could provide good entertainment for lonely elderly people who are not as comfortable with using computers. Moreover, a physical game playing robot could provide entertainment for hospital patients that spend a large amount of time by themselves. With people spending a large amount of time in front of computer and TV screens, the author believes that most people would find it more enjoyable to play a game of chess on a physical board.

With the increase in popularity of chess in Norway, a robotic chess player is a good way of inspiring Norwegian students to take an interest in the field of robotics. The robotic chess player presents the large range of fields necessary to create a fully autonomous robot, thereby providing an intuitive and interesting introduction to robotics.

5.3 Future Work

Although the robotic chess player achieved good results, there is still room for improvement in some areas. Some ideas for improvement has been mentioned in previous chapters, thus this section will provide a summary of previously mentioned ideas as well as present some new possibilities for improving the overall system.

5.3.1 The Robotic Arm

The mechanical construction of the robotic arm could be improved by altering the choice of materials. The materials were chosen to provide a high degree of flexibility in case of modifications, once a complete prototype has been designed, easily modifiable materials is no longer necessary. Using more suitable materials and making the arm more aesthetically pleasing is the last step in finishing up the mechanical arm.

Recall that most of the instability and lack of accurate control of the robotic arm was attributed to the motors. Given that the motors did not operate as specified by the manufacturer, there may be a need to design custom-made motors that are more suited for the robotic arm. This would allow increased control over the torque and speed of each motor, as well as ensuring that the gears are equipped to handle prolonged strain. It was also shown that a change of the current motors is needed to include positional feedback information and thereby provide necessary information for smooth and stable motion. The existing solution could be improved by extracting the positional feedback information contained within the servo motors and feed it back to the robot controller. In the case of custom-made motors, one would need to ensure that the robot controller can read the positional feedback device of each motor.

5.3.2 The Sensory Unit

The current sensory unit requires some modifications for it to be used on chessboards of arbitrary size and orientation without manually reading chessboard dimensions. By using known intrinsic camera parameters combined with a known camera position it may be possible to provide size estimates that are accurate enough to be used by the robotic arm. Such a method would require extensive research as a high degree of accuracy and robustness is necessary. A further improvement of the sensory unit is possible by including a time-of-flight camera to provide depth information. This, in combination with a camera mounted inside the gripper could assist both the chess detection methods as well as the robot controller.

5.3.3 Human Interaction

Human-robot interaction is one of the areas that did not receive any attention during this thesis. The current system requires the user to notify the system of a completed move by pressing a button on the computer. Information is displayed to

the user in the form of console commands, and while a proper computer interface could display the information in a more readable way, this would still require using the computer to communicate with the robotic chess player. A suggested approach to solve the interaction between the user and the system is by implementing a voice interface based on natural language. The idea is that necessary information about system status and the notification of turn-taking could be shared through the voice interface. No real research was done to progress this approach to human-robot interaction, it is therefore left open as a potential improvement to the chess playing robot.

5.3.4 Pawn Promotion

While it did not occur during normal testing of the robotic chess player, the current system is unable to perform unassisted pawn promotion. This is because of the problem occurring with the pieces removed by the human player as the robotic chess player receives no information about the placement of the removed pieces. This results in the chess robot being unable to perform a proper pawn promotion, as it is unable to locate and place the desired piece back on the table. A proposed solution to this is to include a box that is shaped to give room to the various chess pieces at specific locations. With this approach both the chess robot and human player would place removed pieces into the box, thereby allowing the robotic arm to properly perform pawn promotion by inserting previously removed pieces. This is more of a conceptual idea, as no progress was made towards creating such the box that would allow for unassisted pawn promotion.

Chapter 6

Conclusion

This thesis has investigated the possibility of creating a low-cost and robust chess playing robot. A variety of robotic arm designs has been considered, resulting in the design of a four degrees of freedom elbow manipulator arm. Standardized servo motors were chosen as the manipulator's actuators, providing closed loop control contained within the motors. Each motor was calibrated and control of the motors was interfaced to a computer through a dedicated servo controller. A look-and-move vision based open-loop controller was implemented as the main controller of the robotic arm. The robotic arm was combined with a vision unit that interpreted chess moves and a chess unit that served as the main decision-maker in the system.

The robotic chess player presented in this thesis was nicknamed the Magnus, named after the famous Norwegian chess player Magnus Carlsen. The system is fully autonomous, in the sense that it can perceive moves, decide on a countermove and execute the chosen move. The robotic arm was found to have a relatively high performance considering the use of cheap and standardized motors. The system managed to complete multiple games of chess without human intervention, proving the possibility of creating a relatively robust and affordable autonomous small scale robot. While there is still some room for improvement, the largest obstacles of building a chess playing robot is believed to have been overcome.

The goal of this project was not only to create a chess playing robot, but to contribute to bringing robotics into our everyday lives. This project proved the possibility of using cheap robots for entertainment in average households. As the technology necessary gets cheaper more robots will emerge in our everyday lives, some in the form of household assistants, others in the form of entertainment systems. With further research in the area of small scale robotics, the future could hold an endless amount of possibilities that involve human-robot interaction.

The robotic chess player developed in this thesis could serve as both a great entertainment robot as well as a good inspiration for students to seek a future in robotics.

Bibliography

[1] G. M. Levitt, *The Turk, chess automaton.* McFarland & Company, 2006.

[2] IBM, "Deep blue overview."

[3] P. K. N. Dantam and M. Stilman, "The motion grammar for physical human-robot games," *IEEE International Conference on Robotics and Automation*, pp. 5463–5469, 2011.

[4] C. M. et al., "Gambit: A robust chess-playing robotic system," *ICRA*, pp. 4291–4297, 2011.

[5] D. Urting and Y. Berbers, "Marineblue: A low-cost chess robot."

[6] S. H. M. W. Spong and M. Vidyasagar, *Robot Modeling and Control.* John Wiley & Sons, 2006.

[7] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower pair mechanisms," *Applied Mechanics*, vol. 22, pp. 215–221, 1955.

[8] F. Chaumette, "Potential problems of stability and convergence in image-based and position-based visual servoing," 1998.

[9] C. Vebner, "Image-based visual servo control of robot manipulators," 2002.

[10] G. D. H. S. Hutchinson and P. I. Corke, "A tutorial on visual servo control," *Transactions on robotics and automation*, vol. 12, no. 5, 1996.

[11] M. B. G. Palmieri, M. Palpacelli and M. Callegari, "A comparison between position-based and image-based dynamic visual servoings in the control of a translating parallel manipulator," *Journal of Robotics*, 2012.

[12] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd ed. Cengage Learning, 2008.

[13] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, 1986.

[14] I. Sobe, "History and definition of the sobel operator," 2014.

[15] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15:1, 1972.

[16] C. E. Shannon, "Programming a computer for playing chess," *Philosophical Magazine*, vol. 41, no. 314, 1950.

[17] Intel, W. Garage, and Itseez, "Open source computer vision library," 2000.

[18] Pololu, "Pololu micro maestro." [Online]. Available: http://www.pololu.com/product/1350

[19] E. Software. [Online]. Available: http://www.libusb.org/

[20] D. Cross, "Chenard a freeware chess program." [Online]. Available: http://cosinekitty.com/chenard

[21] Lynxmotion, "Gripper kit overview." [Online]. Available: http://www.lynxmotion.com/p-375-a4wd1-gripper-kit.aspx

[22] Hitec, "Hitec website." [Online]. Available: http://hitecrcd.com/

[23] Microsoft, "Microsoft lifecam studio overview."

[24] J. E. Neufeld and T. S. Hall, "Probabilistic location of a populated chessboard using computer vision," *Circuits and Systems*, pp. 616–619, 2010.

Appendix A

Appendix

A.1 Digital Attachment Contents

The digital attachment includes the following directories:

- "Test Video" that contains a video of the robotic chess-player in action.
- "Images" that contains various images of the robotic arm that was not included in the project
- "Source Code" that contains the entire source code of the project as well as necessary external library files.