DEGREE PROJECT IN MECHANICAL ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

PiChess

Voice Controlled Robotic Chess Player

OSCAR DE BRITO LINGMAN

AXEL SERNELIN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

PiChess

Voice Controlled Robotic Chess Player

OSCAR DE BRITO LINGMAN
AXEL SERNELIN

Bachelor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:51

Abstract

The purpose of this bachelor's thesis was to create a robot that could play chess through voice recognition and robotics. The two areas to be investigated were the robot's precision and speed. The reason for building a robot arm of the SCARA type was that it can easily pick up and place pieces with reach over the entire chessboard. The robot arm is controlled from a Raspberry Pi 4 and is moved by two Dynamixel AX-12a servomotors. To pick up chess pieces, a continuous 360-degree servomotor was used to lift an electromagnet which was mounted on a gear rack. A USB microphone was used to collect what move the player indicated. The Stockfish chess engine was used to generate moves for the robot.

The parts of the robot that had the greatest impact on precision were the stability of the aluminum profiles, the gear ratio between the gears that transmit torque to the arm, the gear mesh contact ratio and the size of the electromagnet. The time it took to complete a move could be reduced by increasing the speed of the motors when a chess piece was not attached to the electromagnet, and using a larger gear in the RC servo that raises and lowers the electromagnet.

Keywords— Mechatronics, Raspberry Pi, Chess, SCARA, Dynamixel

Referat

Röststyrd Robotisk Schackspelare

Syftet med det här kandidatexamensarbetet var att skapa en robot som genom röstigenkänning och robotik kunde spela schack. De två områden som skulle undersökas var robotens precision och hastighet. Anledningen till att bygga en robotarm av SCARA typ var att den enkelt kan plocka upp och ställa ner pjäser med räckvidd över hela schackbrädet. Robotarmen styrs från en Raspberry Pi 4 och drivs av två stycken Dynamixel AX-12a servomotorer. För att plocka upp schackpjäser användes en kontinuerlig 360-graders servomotor som lyfte en elektromagnet monterad på en kuggstång. En USB mikrofon användes för att samla in vad spelaren angav för drag. Schackmotorn Stockfish användes för att generera drag åt roboten.

De delar på roboten som hade störst inverkan på precision var stabiliteten i aluminiumprofilerna, utväxlingen mellan kugghjulen som överför moment till armen, anliggningsytan vid kuggingreppen och storlek av elektromagnet. Tiden det tog att genomföra ett drag gick att minska genom att öka hastigheten på motorerna då en schackpjäs inte satt fast på elektromagneten, samt använda ett större kugghjul hos RC servot som höjer och sänker elektromagneten.

Nyckelord— Mekatronik, Raspberry Pi, Schack, SCARA, Dynamixel

Acknowledgements

We would like to thank our supervisor Nihad Subasic for guiding and helping us throughout this project. We would also like to thank Staffan Qvarnström for helping us acquiring parts, especially the FTDI cable. A special thanks to Richard Sernelin for letting us use his personal workshop and helping us creating customized parts necessary for the project. Without any of them, this project would not be possible.

Oscar de Brito Lingman & Axel Sernelin
Stockholm, May, 2021

Contents

List of Figures

List of Abbreviations

FTDI	Future Technology Devices International
GPIO	General Purpose Input/Output
PWM	Pulse Width Modulation
RC	Radio Control
SCARA	Selective Complience Articulated Robot Arm
USB	Universal Serial Bus

Chapter 1

Introduction

This project considers the construction of a robotic arm that can be remotely controlled through voice activation. The goal is for the robotic arm to be able to play a game of chess, using the arm to execute moves by physically picking up and putting down chess pieces.

Today, society is constantly figuring out new ways to help and support people living with disabilities. New technology is being developed to tackle these problems, making life easier for disabled people. One of these technologies is robotics.

Robotics can be used to support those living with physical disabilities and more so, voice-controlled robotics can cover an even wider range of people.

1.1 Background

Inspiration for this project was initially taken from the 18th century chess-playing machine known as **The Turk**, **Mechanical Turk** or **Automaton Chess Player**. This machine was believed to be completely automated and defeated challengers all over the world. However, the machine was actually a hoax since a human player was hidden underneath the desk, executing the machines moves manually [1].

The following projects has been used as sources of inspiration for the construction of the arm.

- *Checkmate* [2] which uses frame-like construction to move along the x- and y-axis, magnetic sensors to detect the chess pieces and an electromagnet to grab them.
- *ChessPlayingRobot* [3] which uses a SCARA-arm construction to move along the x-, y- and z-axis, a camera for visual recognition to detect the chess pieces and an electromagnet to grab them.

1.2 Purpose

The following questions are to be investigated and answered in this thesis:

- Which physical parts of the robot have a noticeable impact on precision?
- How can the robot arm complete a move faster without losing precision?

1.3 Scope

Due to limited resources and time, some limitations have been formulated:

- The voice recognition will use an already existing software package for the Raspberry Pi.
- A virtual chess board package will be used for handling chess actions.
- The robot will use the open-source chess engine Stockfish [4] to calculate moves.

Since the purpose of this thesis was to research how well the robot arm can move chess pieces it was not of significant importance to do the above listed, from scratch.

1.4 Method

In order to answer the research questions, a robotic prototype was constructed, installed and programmed.

First, theoretical research was made to figure out how a robotic arm could be constructed and which components and parts were necessary. An early simulation of the robots lifting movement was made using Acumen [5] to understand how this was supposed to be accomplished practically (a picture of the simulation, Figure C.1, and the code necessary to produce the simulation can be found in Appendix C on page 35).

Second, the robotic arm was built, this was done using a trial and error - approach. Different parts were purchased and some custom made to test either parts limitation until their purpose was considered satisfied.

Third, programming of the robot, this was also done using a trial and error approach but some more theoretical research had to be made to finalize the code.

Fourth, testing, analyzing, fine tuning and result recording. To answer the research questions, some calibration first had to be made and later the robot was fine tuned whilst documenting the results of the robot. This was then used to answer the research questions.

Chapter 2

Theory

The following chapter consists of theoretical information which is necessary for the comprehension of the project. Here, the SCARA robot type as well as the components and software used in the project will be explained.

2.1 SCARA

SCARA, an acronym for Selective Compliance Articulated Robot Arm, is a solution for making the robot arm compliant in the X-Y axis. This means that the arm can move freely across an area decided by the length of two jointed arms.

Movement in the Z axis can be obtained either by altering the base of the arm or by using a rod, which can be altered, at the end of the second arm [6].

2.2 Hardware

In this section the hardware of the robot are explained.

2.2.1 Raspberry Pi 4

Raspberry Pi is a series of small single-boarded computers, the latest to the addition being the Raspberry Pi 4. The Raspberry Pi 4 runs on Raspberry Pi OS (previously known as Raspbian), a Linux based operating system, which gives it the capabilities to run more advanced tasks than that of a microcontroller. It can run programs developed for Linux operating system and programmed using advanced languages such as Python, Java, C++, etc [7].

A full overview of the Raspberry Pi 4 can be seen in Figure 2.2 on page 4.

2.2.2 ArbotiX-M Robocontroller

The ArbotiX-M Robocontroller is an advanced, Arduino based microcontroller, used primarily for DYNAMIXEL based actuators. It provides the servos with a 12V electrical current and handles data packages sent and received from the servo motors [8].

An overview of the ArbotiX-M Robocontroller can be seen in Figure 2.1.

Figure 2.1: ArbotiX-M overview [8].

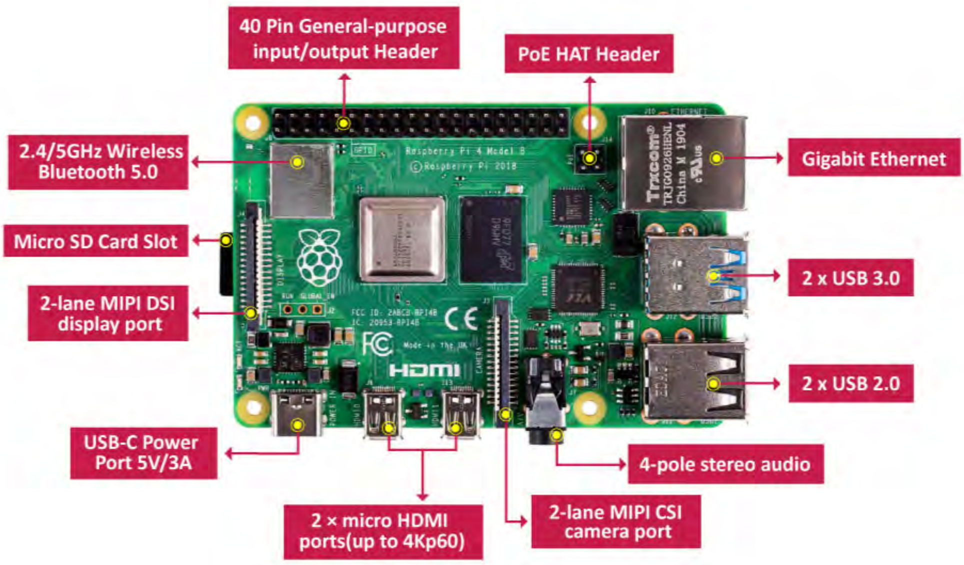


Figure 2.2: Raspberry Pi 4 hardware overview [9].

2.2.3 Dynamixel AX-12A Servomotor

A servomotor is a rotary actuator which allows for precise control of angular position, velocity and acceleration. It contains a motor connected to a sensor for position feedback [10]. The Dynamixel AX-12A servomotor has a rotation span of zero to 300 degrees. At 12V the servomotor has a stall torque of 1.5 N*m and returns feedback of its position, current motor temperature, load and input voltage back to the motor controller [11].

A picture of the Dynamixel AX-12A Servomotor can be seen in Figure 2.3.

Figure 2.3: Dynamixel AX-12A front and back [12].

2.2.4 Electromagnet

An electromagnet is a type of magnet in which the magnetic field is produced by an electric current [13]. Electromagnets can be switched on and off which allows for applications such as attracting a magnetic material on demand.

2.2.5 FTDI cable

The FTDI cable is a USB to Serial converter which allows for a simple way to connect transistor-transistor logic devices, such as an Arduino, to USB. The I/O pins of this FTDI cable are configured to operate at 5V [14].

2.2.6 RC Servo

RC Servos are small actuators commonly used in remote controls or robotics because of its ability to rotate and maintain a certain position or angle while still being small in size.

A common RC servo consist of a motor, a gearbox, a position sensor, an error amplifier, a motor driver and a circuit. A block diagram for a common RC servo can be seen in Figure 2.4.

The RC servo is connected through three wires; power (+5V), ground and a signal wire [15].

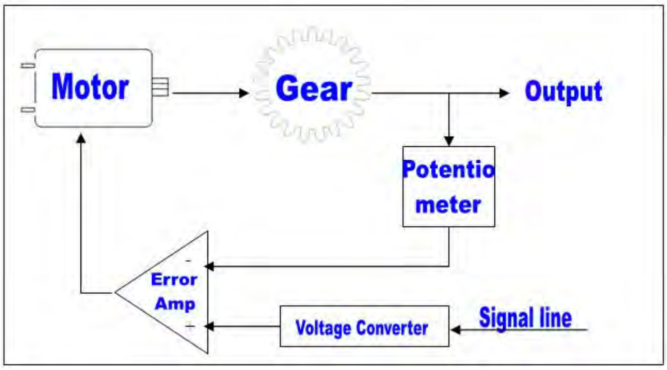


Figure 2.4: Block Diagram of a typical servo motor [15].

Pulse Width Modulation in Servos

An RC servo is controlled using PWM, Pulse Width Modulation. The angle at which the servo should be maintained is determined by the duration of a pulse that is applied to the signal wire.

The typical RC Servo expects a pulse every 20 ms, though this can vary from servo to servo, and applying a 1.5 ms pulse to the servo will result in the motor turning to the 90°position. Likewise, applying a 1.0 ms pulse turns the motor to the 0°position and 2.0 ms pulse to the 180°position, which is shown in Figure 2.5.

This is a very effective way to vary the power supply to a normally binary control device [15].

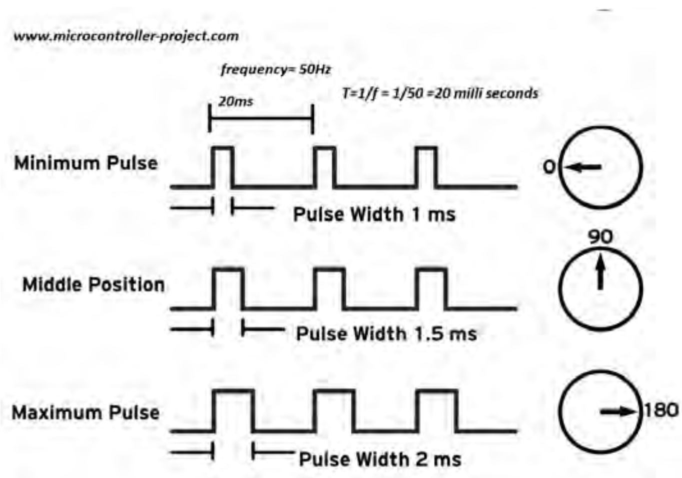


Figure 2.5: Servo motor PWN duty cycle and frequency requirement [16].

2.3 Software

In this section the software used will be explained.

2.3.1 Python

Python is an interpreted, general-purpose and high-level programming language. It is used for programming and software development such as web development, back end development, writing system scripts and data sciences [17].

2.3.2 Stockfish

Stockfish is a free and open-source chess engine, which evaluates chess positions by developing a tree of all legal moves for the user chosen depth. The engine then looks at all of these board positions and decides the best move considering things such as control of the center, vulnerability of the king to check, vulnerability of the opponent's queen, and a multitude of other parameters. Stockfish can be set to play with varying level of skill, which is done by limiting the computing time and search depth. If Stockfish is given enough time it will be unbeatable by any human chess player [4].

2.3.3 PyPose

Since the ArbotiX-M robocontroller is based on an Arduino, there is no way of controlling it with Python from a computer by default. The PyPose project is a program which provides the user a way of sending data packages to the ArbotiX-M from Python. By uploading the PyPose program to the microcontroller, it will act as a pass through so the Raspberry Pi can communicate directly with the servo motors. The PyPose project provides the Raspberry Pi with a driver which effectively allows you to get and set registers on any connected Dynamixel device via a FTDI-USB cable [18].

Chapter 3

Demonstrator

This chapter considers the construction of the robotic prototype used in this project.

3.1 Construction

The construction of the prototype consists of a robotic arm that stands on a wooden box which is attached to a plywood table. The chessboard is painted on the plywood table and is at a set distance from the robotic arm.

An overhead picture of the robot arm can be seen in Figure 3.1 on page 10.

3.1.1 Robotic Arm

The robot arm begins with a heavy aluminum cube which is attached to the wooden box by four screws. An axle is inserted vertically into the cube and held in place by a set screw. The bottom aluminum profile rides on the axle using two bearing housings, as seen in Figure 3.2a on page 11. The top aluminum profile is similarly attached at the rear end of the bottom profile, as seen in Figure 3.3a on page 11. This allows the arm to rotate freely around both axles.

On top of each steel axle, located at the rotation points, a cogwheel is attached to the aluminum profile. By controlling an intermeshing gear, which is set in place by attaching the Dynamixel AX-12A [11] to a custom made engine mount next to the cogwheel, movement can be generated in the arm. This can be seen in Figures 3.2b and 3.3b on page 11.

3.2 Hardware

The Raspberry Pi [7] acts as the brain of the robot. All necessary coding for controlling the servomotors, as well as running the chess engine and voice recognition programs, is done from here.

The microcontroller, ArbotiX-M [8], acts as a middleman between the Raspberry Pi and the Dynamixel AX-12A [11] servomotors. The microcontroller is required to control the servomotors as they run on arduino based programming and for its higher power demand (12V).

To position the arm at the desired x- and y-coordinate the Dynamixel Ax-12A servomotors are used. Because of their ability to receive and store position and speed data, they are very effective for calibrating the coordinates of each square on the chessboard, as well as regulating speed for achieving high precision.

For picking up and dropping off chess pieces, an electromagnet is attached to a gear rack which is intermeshed with the cogwheel being controlled by the RC servo.

The RC servo used is actually a continuous rotation servo. What this means is that instead of maintaining a position when applied with a pulse signal, the servo will continuously rotate clockwise or counterclockwise depending on the pulse signal. Because of this, measurements are made to adjust how long the servo should rotate for the electromagnet to reach a certain chess piece.

When a chess piece is to be lifted, the servo will rotate until a custom made limit switch receives signal. The limit switch is made using a screw that is connected to power and located on the gear rack, and a copperplate connected to ground and located on the bottom rear end of the upper aluminum profile. When the screw touches the copper plate, the circuit is complete and the switch receives signal which is programmed to stop the servo from rotating further.

The final component is a USB microphone connected directly to the Raspberry Pi. This is to be used by the human player to decide his/her move.

A complete list and wiring diagram, Figure 3.4, of the included electronic components can be found on page 12.

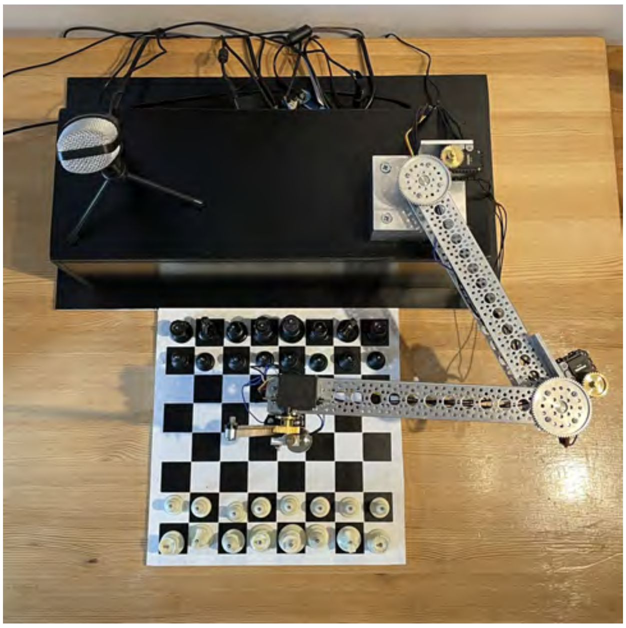
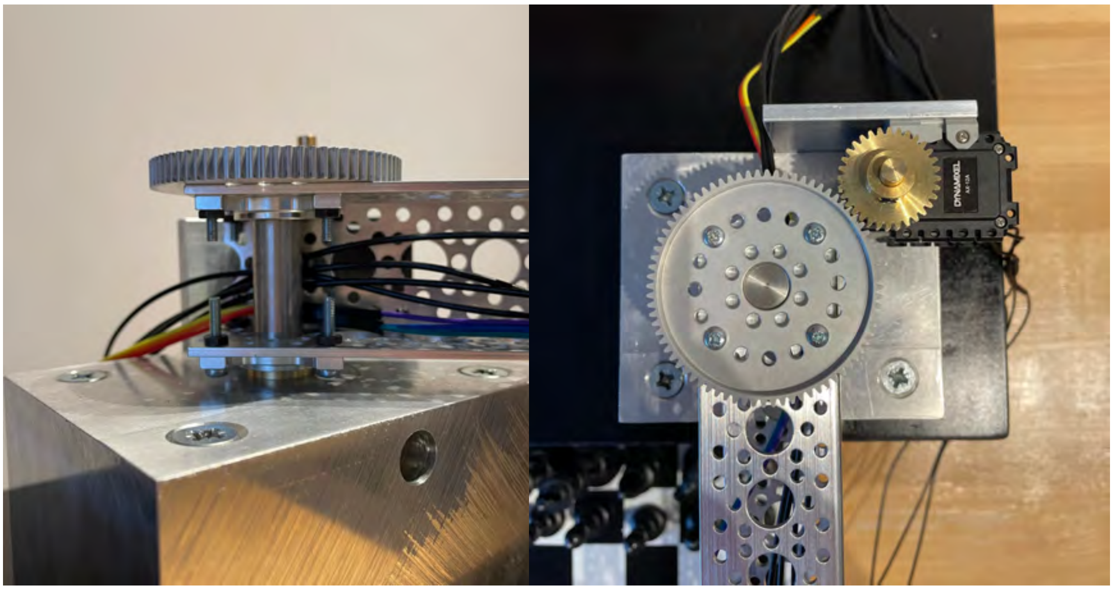
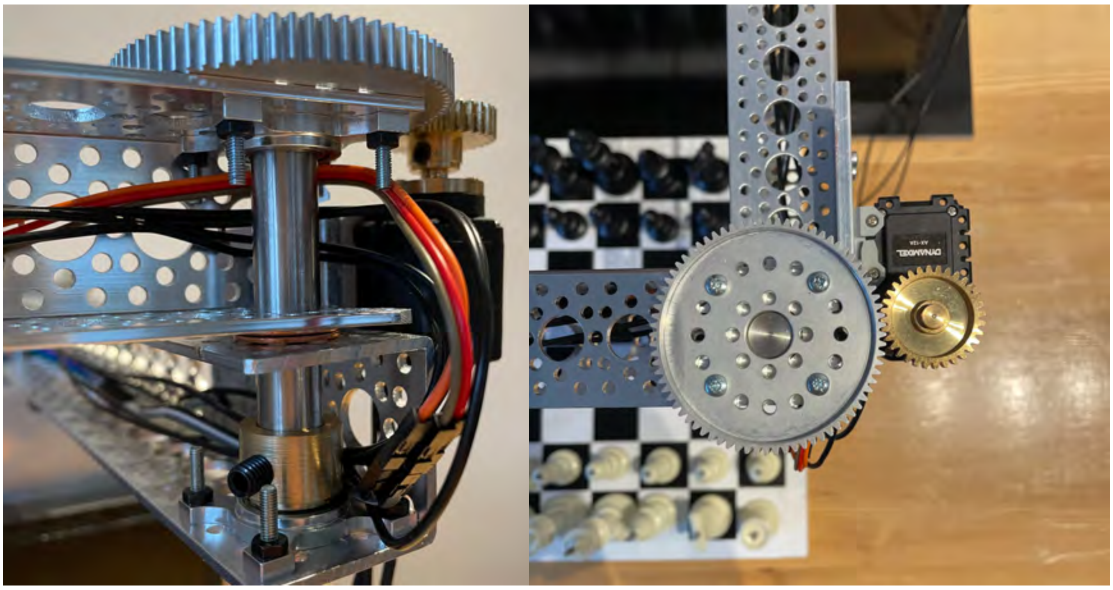


Figure 3.1: Overhead picture of robot.



(a) Lower arm axle and bearing housings. (b) Overhead picture of lower arm gears.

Figure 3.2: Lower arm point of rotation.



(a) Upper arm axle and bearing housings. (b) Overhead picture of upper arm gears.

Figure 3.3: Upper arm point of rotation.

Below follows a list of all hardware components used, thse are connected as seen in Figure 3.4.

1. 12V Power Supply
2. 5V Power Supply
3. Dynamixel AX-12A Servo Motors
4. USB Microphone
5. Limit Switch
6. RC Servo
7. Electromagnet
8. ArbotiX-M Robocontroller
9. FTDI Cable
10. Breadboard with a Solid State Relay
11. Raspberry Pi 4

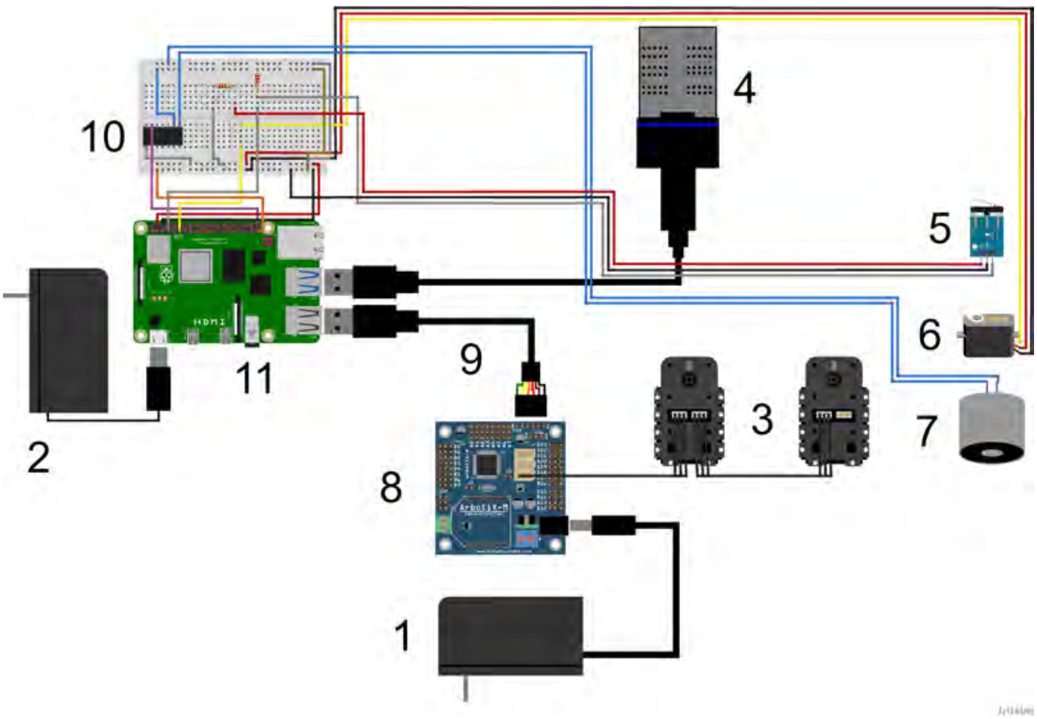


Figure 3.4: A wiring diagram of all the hardware components, made using Fritzing [19].

3.3 Software

All software is written in Python on the Raspberry Pi [7].

3.3.1 Main program

The main file consists of a loop which keeps iterating as long as the chess game is still in play. Five python classes were created for different purposes, taking care of arm movement, gripper movement, chess engine handling and speech recognition. These are integrated in the main file for a more organized structure and keeps the main file short and concise. Figure A.1 on page 22 in Appendix shows a state diagram of how the main file operates.

3.3.2 Chess

To be able to play chess versus a computer we imported Stockfish [4]. Given any chess position Stockfish will return the best possible move. This is used to determine the robots move which is then sent to the movement class. To be able to set up a virtual chess board a chess package was imported. This allowed for an easy way to handle chess related actions such as a check for legal moves and look for a check mate.

3.3.3 Speech

Using the USB microphone, the Raspberry Pi [7] listens to what the human player says and then searches the text string for a chess piece and a chess board coordinate. This is then later translated into a legal chess move, such as ”g1f3” using the imported chess package. If no legal move is detected, the program listens for a new input by the player.

3.3.4 Movement

Given a chess move, this part will determine if its a capture, normal move or a castling move and then move the arm to the correct position. All the motor values for each position are stored in a dictionary which makes them easy to retrieve.

3.3.5 Gripper

To pick up a piece, the program lowers the gripper to the corresponding piece height and then turns the electromagnet on. To drop off a piece the electromagnet turns off. The electromagnet is switched on using a control signal from a Raspberry Pi GPIO pin which turns on a solid state relay. This allows for a higher voltage than what the Raspberry Pi can output.

Chapter 4

Results

The parts of the robot that had the most impact on precision were the following:

- Sturdiness of aluminum profiles.
- Ratio between the gears used for transmitting rotational motion from servo to arm.
- Gear mesh contact ratio.
- Area of magnetized surface on the electromagnet.

The speed of the robot arm was increased whilst not carrying any chess piece to improve the overall speed. For white to castle on the king side it took 33 seconds to fully complete the move compared to 35 seconds before. When a larger gear was installed on the RC servo it increased the pickup speed and thus decreasing the time required by 2 seconds making it a final 31 seconds to castle the king and rook.

Chapter 5

Discussion & Conclusion

This chapter discusses the results and research questions, concludes the project and suggests future development.

5.1 Discussion

The aluminum profiles used in this project had one flaw which quite severely reduced the precision, it had one face open. Because two bearing housings had to be attached on a face surface of the profile, it was necessary to turn the profile with the open side facing sideways. This made the arms, especially the lower one, twist and bend when more weight was added across the robot arm. This resulted in swaying and the electromagnet not having a perfect vertical facing.

The cogwheel attached to the top of each axle was too big when compared to the gear on the servos. The high gear ratio led to less precise movement as a small rotation from the servo leads to a rather big motion in the arm.

The gear mesh contact ratio, meaning how much of the cogs are in touch with each other, had a great impact on precision. Low ratio would lead to sway and a much less smooth movement in the arm. This was resolved using slide-able engine mounts for the Dynamixel AX-12A servos so the gears could first be pushed in place before tightening the screws holding the mount.

Even though the coordinates of each chessboard square was finely calibrated, it was discovered that the pin on the chess pieces would attach to the electromagnet a little bit different every time. This led to the placement of chess pieces would get worse for every additional move. This was, however, quite easily resolved using a plastic cone at the tip of the electromagnet. The cone would help guide the chess piece toward the center of the magnet, resulting in consistent pick up and drop off.

To increase the speed, or rather decrease the time taken, of a move at no cost in precision, two applications proved to be especially effective.

First, increasing the Dynamixel AX-12A rotation speed by 100% when it was not currently moving a chess piece. This is important since if the speed was increased too high when a chess piece was being lifted, the electromagnet would loose its grip and the piece would drop. If the speed was increased by more than 100%, the torque would be too high, leading to the gears slipping, displacing the arm and lowering precision significantly.

Second, changing the cogwheel of the RC servo to a larger one. Since the RC servo is simply used to raising or lowering the electromagnet, it did not matter as much to have precise movement. A larger cogwheel leads to a higher linear speed in the gear rack and thus decreasing its travel time.

5.2 Conclusion

The research questions were answered and the project can be considered successful. The robot was able to recognize a move and physically execute it, however, there are room for improvement.

The points brought up in the discussion would lead to a much faster and precise movement, if resolved. Also, it was found that increasing the surface area of the pins located at the top of each chess piece reduced sway of the piece and risk of dropping due to heavy load.

5.3 Future Work

A possible addition to the robot could be a LED display which would be used to show the user what the robot is doing at that very moment, whether it's listening for a move, calculating a move or executing a move. The display could also be used as a menu if one would like to change the difficulty of the chess engine or maybe change mode to two humans playing against each other.

Another addition could be a speaker which would be used for multiple purposes such as telling the current location of a chess piece, the move the robot is currently doing or any other information the player would be interested in. This would be particularly useful if designing the robot for people with eye disorders.

Bibliography

[1] P. Hitlin, L. Rainie, N. Hatley, A. Smith, P. van Kessel, B. Broderick, M. Duggan, A. Perrin, M. Porteus, S. Greenwood, D. Page, and O. O'Hea. (2016). *Research in the crowdsourcing age, a case study, 1. what is mechanical turk?*, Available at: `https://www.pewresearch.org/internet/2016/07/11/what-is-mechanical-turk/` [Accessed 01/20/2021].

[2] J. Ericson and A. Westermark. (2020). *Checkmate, Remote arduino powered chess*, Degree Project in Technology, First Cycle, Available at: `https://www.diva-portal.org/smash/get/diva2:1462109/FULLTEXT01.pdf` [Accessed 01/21/2021].

[3] F. Baldhagen and A. Hedström. (2020). *Chess playing robot, Robotic arm capable of playing chess*, Degree Project in Technology, First Cycle, Available at: `https://www.diva-portal.org/smash/get/diva2:1462118/FULLTEXT01.pdf` [Accessed 01/21/2021].

[4] S. Nicolet, U. Corzo, S. Kiminki, nodchip, and J. VandeVondele. (2021). *Stockfish/readme.md, Overview*, Available at: `https://github.com/official-stockfish/Stockfish/blob/master/README.md` [Accessed 02/20/2021].

[5] *Acumen*, Available at: `http://www.acumen-language.org/` [Accessed 03/24/2021].

[6] *Scara robots, The right choice for your application*, Available at: `https://www.fanuc.eu/de/en/robots/robot-filter-page/scara-series/selection-support` [Accessed 01/28/2021].

[7] *Raspberry pi, Faqs*, Available at: `https://www.raspberrypi.org/documentation/faqs/` [Accessed 01/30/2021].

[8] *Trossen robotics, Arbotix-m robocontroller*, Available at: `https://www.trossenrobotics.com/p/arbotix-robot-controller.aspx` [Accessed 01/30/2021].

[9] *Raspberry pi 4 computer model b 4gb, Product details*, Available at: `https://www.seeedstudio.com/Raspberry-Pi-4-Computer-Model-B-4GB-p-4077.html` [Accessed 04/20/2021].

[10] D. Sawicz. (). *Hobby servo, Fundamentals*, Available at: `http://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf` [Accessed 02/12/2021].

[11] *Robotis, Dynamixel ax-12a*, Available at: `https://www.robotis.us/dynamixel-ax-12a/` [Accessed 02/12/2021].

[12] *Trossen robotics, Dynamixel ax-12a robot actuator*, Available at: https://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx [Accessed 04/20/2021].

[13] R. Nave. *Hyperphysics, Electromagnet*, Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/elemag.html [Accessed 02/13/2021].

[14] *Ftdi chip, Ttl-232r-5v*, Available at: https://ftdichip.com/products/ttl-232r-5v/ [Accessed 02/13/2021].

[15] kh86. (2011). *Cytron technologies, How rc servo works?*, Available at: https://tutorial.cytron.io/2011/09/19/how-rc-servo-works/ [Accessed 03/15/2021].

[16] (2020). *Engineers garage, Controlling servo motor with stm32f103 microcontroller using stm32cubemx code configurator by stmicroelectronics and keil uvision 5 ide for cortex m1 series microcontrollers*, Available at: https://www.engineersgarage.com/electronic-projects/interfacing-servo-motor-with-stm32/ [Accessed 03/15/2021].

[17] *Python, What is python? executive summary*, Available at: https://www.python.org/doc/essays/blurb/ [Accessed 02/10/2021].

[18] *Arbotix robocontroller, Pypose introduction*, Available at: http://vanadiumlabs.github.io/pypose/ [Accessed 02/20/2021].

[19] *Fritzing, Electronics made easy*, Available at: https://fritzing.org/ [Accessed 04/25/2021].

[20] *Creatly, Your visual workspace*, Available at: https://creately.com [Accessed 04/25/2021].

Appendix A

PiChess State Diagram

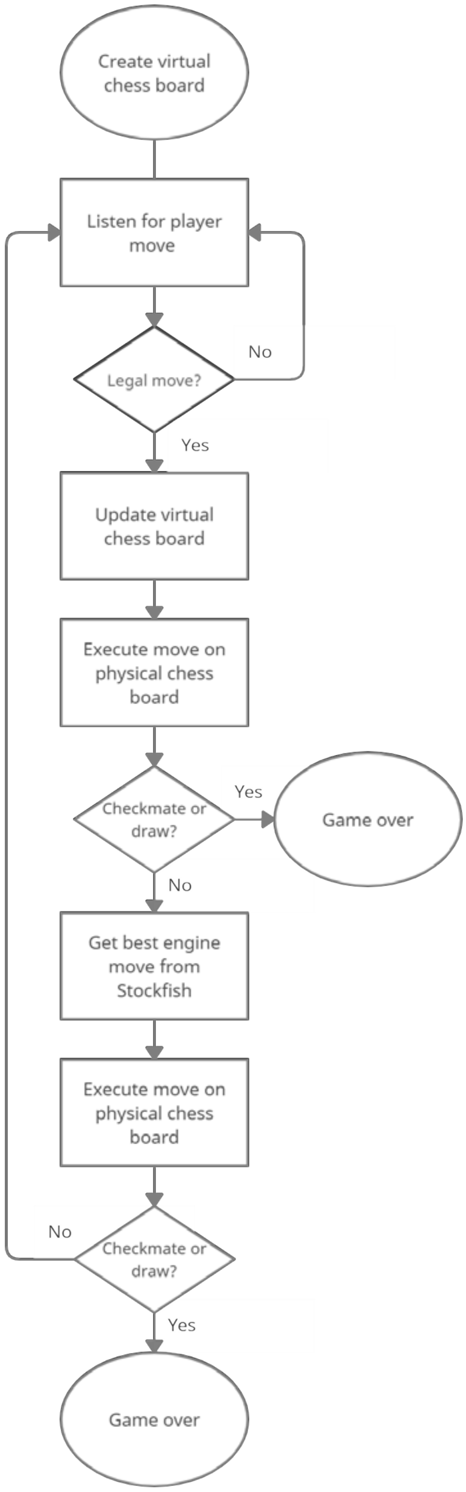


Figure A.1: State diagram of PiChess, made using Creatly [20].

Appendix B

Python Code

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

################# Main File #################

from stockfish_engine import StockfishEngine
from speech_engine import SpeechEngine
from movement_engine import MovementEngine
from arm_movement import ArmMovement
import chess


board = chess.Board()
stockfish = StockfishEngine()
mic = SpeechEngine()
movement = MovementEngine()
arm = ArmMovement()

f = open("last_position.txt", "w") # saves the last position in case of a crash
arm._is_moving()# needs to call this function for the arms to be able to set speed
move = None
run = True
while run:
    while move is None: # looks for a legal move until found
        [piece, square] = mic.get_square_and_piece() # gets the piece and square from microphone
        if piece and square is not None: # if both were found it tests if there is a legal move with those two conditions
            move = stockfish.get_player_move(board, piece, chess.WHITE, square)
    movement.execute_move(move, board) # executes the move with robot arm when a legal one is found
    board.push(move) # updates the virtual board
    f.write(board.fen()) # writes last position to file
    # print(board) if you wanna se the board in the console
    move = stockfish.get_best_engine_move(board) # gets stockfish response move
    movement.execute_move(move, board) # executes the move with robot arm
    board.push(move) # updates virtual board
    f.write(board.fen()) # writes last position to file
    # print(board)
    move = None # resets move to None
    if board.is_game_over() # if game is over, the program shuts off
        run = False
f.close()
```

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

from driver import Driver
from ax12 import *

servo_1 = 1 # id for dynamixel servos
servo_2 = 2
servos = [servo_2, servo_1]
default_speed = [50, 50]

def _register_bytes_to_value(register_bytes): # takes the bytes given from dynamixel and returns it as a value
    return register_bytes[0] + (register_bytes[1]<<8)

class ArmMovement: # handles movement of the dynamixel servos which controls the arm
    def __init__(self, port="/dev/ttyUSB0"):
        self.coordinates_dict = {"a8" : [790, 570], "b8" : [735, 635],
                                 "a7" : [760, 530], "b7" : [705, 595],
                                 "a6" : [745, 480], "b6" : [685, 545],
                                 "a5" : [740, 420], "b5" : [665, 485],
                                 "a4" : [735, 355], "b4" : [665, 425],
                                 "a3" : [750, 270], "b3" : [680, 340],
                                 "a2" : [765, 175], "b2" : [700, 250],
                                 "a1" : [800, 55],  "b1" : [715, 155],
                                 "c8" : [680, 705], "d8" : [610, 765],
                                 "c7" : [645, 660], "d7" : [570, 720],
                                 "c6" : [620, 605], "d6" : [550, 665],
                                 "c5" : [610, 550], "d5" : [540, 600],
                                 "c4" : [610, 480], "d4" : [540, 535],
                                 "c3" : [620, 400], "d3" : [550, 450],
                                 "c2" : [635, 315], "d2" : [570, 370],
                                 "c1" : [665, 210], "d1" : [600, 270],
                                 "e8" : [535, 825], "f8" : [440, 880],
                                 "e7" : [495, 775], "f7" : [410, 815],
                                 "e6" : [475, 715], "f6" : [395, 755],
                                 "e5" : [470, 650], "f5" : [400, 685],
                                 "e4" : [475, 575], "f4" : [405, 610],
                                 "e3" : [485, 500], "f3" : [425, 530],
                                 "e2" : [505, 415], "f2" : [450, 445],
                                 "e1" : [540, 315], "f1" : [480, 345],
                                 "g8" : [330, 920], "h8" : [200, 945],
                                 "g7" : [310, 850], "h7" : [205, 875],
                                 "g6" : [305, 790], "h6" : [215, 805],
                                 "g5" : [315, 715], "h5" : [240, 730],
                                 "g4" : [335, 640], "h4" : [270, 655],
                                 "g3" : [360, 555], "h3" : [300, 575],
                                 "g2" : [390, 470], "h2" : [330, 490],
                                 "g1" : [425, 370], "h1" : [375, 390]
                                 } # stores all the servo motor values for each individual chess board square

```

```
        self.driver = Driver(port=port) # opens up the driver on the given usb port via the FTDI cable. Uses serial communication

    def get_coordinate(self, square): # returns the servo motor values of a given square
        return(self.coordinates_dict.get(square, None)) # returns None if invalid square

    def move_to_square(self, square, speed): # moves the arm to a given square
        self.move(self.get_coordinate(square), speed)

    def home_position(self):
        speed = [100, 100] # moves the arm to the home position
        self.move([80, 680], speed)

    def drop_position(self): # moves the arm to the drop piece position
        self.move([900, 150], default_speed)

    def startup_calibrate(self): # turns the motors to its end position for calibration use
        self.move([0, 1023], default_speed)

    def current_position(self): # returns both servos positions
        return self._values_for_register(P_PRESENT_POSITION_L) # register id from ax12

    def _values_for_register(self, register): # returns two values given what register id you request
        return [_register_bytes_to_value(self.driver.getReg(index, register, 2)) for index in servos]

    def _is_moving(self): # returns True if any of the servos is moving, else False
        return any([self.driver.getReg(index, P_MOVING, 1) == 1 for index in servos])

    def set_speed(self, speed): # sets the moving speed of both servos
        for i in servos:
            self.driver.setReg(i, P_GOAL_SPEED_L, [speed[i%2]%256, speed[i%2]>>8])

    def move(self, goal_position, speed): # moves the servos to a given position with a default speed
        self.set_speed(speed)
        for i in servos:
            self.driver.setReg(i, P_GOAL_POSITION_L, [goal_position[i%2]%256, goal_position[i%2]>>8])

```

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

import RPi.GPIO as GPIO
from time import sleep
import chess

electromagnet_pin = 21 # GPIO pin id's
servo_pin = 17
sensor_pin = 4

pieces = {
          'p': 2.6,
          'r': 2.7,
          'n': 2.4,
          'b': 2.2,
          'k': 1.9,
          'q': 2
          } # corresponds to the height of the piece (seconds)

class Gripper(object): # handles movement of the gripper
    def __init__(self):
        GPIO.setmode(GPIO.BCM) # uses Broadcom SOC channel pin numbering, use GPIO.BOARD for "normal numbering"
        GPIO.setwarnings(False) # disables consol warning messages
        GPIO.setup(servo_pin, GPIO.OUT) # sets pins to outputs
        GPIO.setup(electromagnet_pin, GPIO.OUT)
        GPIO.setup(sensor_pin, GPIO.IN) # sets sensor pin as input

    def move_down(self, t): # moves the gripper down for t seconds
        pwm = GPIO.PWM(servo_pin, 50) # 50 Hz
        pwm.start(0) # starts pwm signal
        pwm.ChangeDutyCycle(1.5) # down motion
        sleep(t) # waits for t seconds
        pwm.ChangeDutyCycle(0) # stops servo
        pwm.stop() # stops pwm signal

    def move_up(self): # moves the gripper up until the sensor at pin 4 is HIGH
        pwm = GPIO.PWM(servo_pin, 50) # 50 Hz
        pwm.start(0) # starts pwm signal
        pwm.ChangeDutyCycle(12.5) # up motion
        while GPIO.input(sensor_pin) == GPIO.LOW: # waits for signal
            # print("Waiting for sensor...")
            pass
        pwm.ChangeDutyCycle(0) # stops servo
        pwm.stop() # stops pwm signal

    def electromagnet(self, on): # turns on/off magnet
        output = GPIO.HIGH if on else GPIO.LOW # if true then high otherwise low. Controls the solid state relay which is wired up to the electromagnet
        GPIO.output(electromagnet_pin, output)

    def pickup(self, piece): # picks up a piece
```

```python
        t = pieces[piece] # gets the height of the piece to pickup (seconds)
        self.move_down(t) # moves down
        sleep(0.2)
        self.electromagnet(True) # turns on the electromagnet
        sleep(0.2)
        self.move_up() # moves all the way up

    def dropoff(self, piece): # drops off a piece
        t = pieces[piece] # gets the height of the piece to pickup (seconds)
        self.move_down(t) # moves down
        sleep(0.2)
        self.electromagnet(False) # turns off the electromagnet
        sleep(0.2)
        self.move_up() # moves all the way up

    def discard_piece(self): # discards a captured piece
        self.move_down(0.5) # moves down for 0.5s
        sleep(0.2)
        self.electromagnet(False) # turns off electromagnet
        self.move_up() # moves all the way up

    def go_to_idle(self): # puts the gripper at a distance closer to the pieces to save time
        move_down(1) # moves down for x seconds

    def cleanup(self): # clears the gpio pins
        GPIO.cleanup()
```

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

from arm_movement import ArmMovement
from gripper import Gripper
import chess

default_speed = [40, 40] # slow speed
fast_speed = [80, 80] # fast speed

class MovementEngine: # handles logic to decide how the arm should move given a move
    def __init__(self):
        self.gripper = Gripper()
        self.arm = ArmMovement()

    def convert_to_square(self, square): # converts a square string to a chess.SQUARE integer
        file = ord(square[0]) - 97 # converts the file string to the corresponing file id
        rank = ord(square[1]) - 49 # converts the rank string to the corresponing rank id
        return chess.square(file,rank) # returns the chess square id given the file and rank

    def piece_at(self, square, board): # returns the current piece at a given square in a given board position
        piece = board.piece_at(self.convert_to_square(square)) # gets the chess.PIECE on given square
        if piece is not None:
            return str(piece).lower() # returns as a lowercase string if not None
        else:
            return None # returns None if piece is None

    def capture(self, move, board): # handles the movement if the chess move is a capture
        goal_square = move[-2:] # gets the square where the captured piece is
        piece = self.piece_at(goal_square, board) # gets the piece which is captured
        self.arm.move_to_square(goal_square, fast_speed) # moves to the captured piece
        while self.arm._is_moving(): # waits until its at its destination
            # print("Moving to: " + goal_square)
            pass
        self.gripper.pickup(piece) # picks up the captured piece
        self.arm.drop_position() # moves the arm to the discard piece position
        while self.arm._is_moving(): # waits until its at its destination
            # print("Moving to drop position...")
            pass
        self.gripper.discard_piece() # gripper drops piece
        self.make_move(move, board) # after piece is discarded it makes a normal move.
        pass

```

```python
    def make_move(self, move, board): # handles the movement if the chess move is a normal move
        start_square = move[:2] # gets the square where the piece is
        goal_square = move[-2:] # gets the square where the piece is going to
        piece = self.piece_at(start_square, board) # gets the piece which is to be moved
        self.arm.move_to_square(start_square, fast_speed) # moves the arm to the piece
        while self.arm._is_moving(): # waits until its at its destination
            # print("Moving to: " + start_square)
            pass
        self.gripper.pickup(piece) # gripper picks up the piece
        self.arm.move_to_square(goal_square, default_speed) # arm moves to the square where it is to be placed
        while self.arm._is_moving(): # waits until its at its destination
            # print("Moving to: " + goal_square)
            pass
        self.gripper.dropoff(piece) # gripper drops off the piece
        self.arm.home_position() # moves back to home position

    def castle(self, move, board):
        move_string = str(move) # converts move to a string
        if board.is_kingside_castling(move):
            if board.turn: # if its white and kingside castle
                self.make_move(move_string, board) # makes the king move
                rook_move = "h1f1"
                self.make_move(rook_move, board) # makes the rook move
            else: # if its black and kingside castle
                self.make_move(move_string, board) # makes the king move
                rook_move = "h8f8"
                self.make_move(rook_move, board) # makes the rook move

        elif board.is_queenside_castling(move):
            if board.turn: # if its white and queenside castle
                self.make_move(move_string, board) # makes the king move
                rook_move = "a1d1"
                self.make_move(rook_move, board) # makes the rook move
            else: # if its black and queenside castle
                self.make_move(move_string, board) # makes the king move
                rook_move = "a8d8"
                self.make_move(rook_move, board) # makes the roob move

    def is_capture(self, move, board): # checks if the chess move is a capture or not
        goal_square = move[-2:] # gets the destination square of the move
        return True if self.piece_at(goal_square, board) is not None else False #if the square where the piece is to be placed is not empty, it is a capture

    def execute_move(self, move, board): # decides what type of move it is and then executes it
        move_string = str(move) # converts move to a string
        if board.is_castling(move): # if castle
            self.castle(move, board)
        elif self.is_capture(move_string, board): # if capture
            self.capture(move_string, board)
        else:
            self.make_move(move_string, board) #else normal move
```

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

from chess import uci

squares = ['a1', 'b1', 'c1', 'd1', 'e1', 'f1', 'g1', 'h1',
           'a2', 'b2', 'c2', 'd2', 'e2', 'f2', 'g2', 'h2',
           'a3', 'b3', 'c3', 'd3', 'e3', 'f3', 'g3', 'h3',
           'a4', 'b4', 'c4', 'd4', 'e4', 'f4', 'g4', 'h4',
           'a5', 'b5', 'c5', 'd5', 'e5', 'f5', 'g5', 'h5',
           'a6', 'b6', 'c6', 'd6', 'e6', 'f6', 'g6', 'h6',
           'a7', 'b7', 'c7', 'd7', 'e7', 'f7', 'g7', 'h7',
           'a8', 'b8', 'c8', 'd8', 'e8', 'f8', 'g8', 'h8'] # list which stores all squares

class StockfishEngine(object): #handles chess moves and positions
    def __init__(self):
        self._engine = uci.popen_engine('stockfish') #opens up an instance of the stockfish engine
        self._engine.uci()

    def get_best_engine_move(self, board): # gets stockfish engine's best move given the current position
        self._engine.position(board) # feeds the board to stockfish
        result = self._engine.go(movetime=1000) # lets stockfish calculate all the possible lines for 1000ms
        return result.bestmove # returns the best move in the current position

    def get_current_piece(self, board, square): # returns what piece is at requested square and converts it to a lowercase lette.
        return str(board.piece_at(square)).lower()

    def try_legal_move(self, board, suggested_move): # returns the move if its legal, otherwise returns None
        legal_moves_dict = {str(move) : move for move in board.legal_moves } # creates a dictionary with moves in a string format as key and chess moves as item
        move = legal_moves_dict.get(suggested_move, None) # returns None if non legal move
        return move # retuns the move

    def square_name(self, square): # given a squares id (0-63, integer), it returns the name of the square as a string
        return squares[square]

    def get_pieces(self, board, piece_type, piece_color): # gets all pieces and their positions on the board and returns a list
        return [square for square in board.pieces(piece_type, piece_color)]

    def get_player_move(self, board, piece_type, piece_color, destination_square): # takes a piecetype and a destination and returns the legal move which it can do
        moves = []
        pieces_on_board = self.get_pieces(board, piece_type, piece_color) # gets the pieces of the given types positions on the board
```

```
        for square in pieces_on_board:
            square = self.square_name(square) # gets the square names of the squares
            move = self.try_legal_move(board, square+destination_square) # tries which of the moves is a legal move
            if move is not None:
                moves.append(move) # appends the move to a list if its not None
        return moves[0] if moves else None #returns the first legal move, else None
```

```python
# Voice controlled robotic chess player
# R'o'ststyrd robotisk schackspelare
# Datum : 2021 - 05 - 09
# Written by : Axel Sernelin and Oscar de Brito Lingman
# Examinator : Nihad Subasic
# TRITA-ITM-EX 2021:51
# Kurskod : MF133X

# Bachelor's Thesis in Mechatronics at KTH

import speech_recognition as sr
import chess
import re # regular expression package

pieces_dict = {
                "knight"   : chess.KNIGHT,
                "knights"  : chess.KNIGHT,
                "nights"   : chess.KNIGHT,
                "night"    : chess.KNIGHT,
                "nite"     : chess.KNIGHT,
                "light"    : chess.KNIGHT,
                "wright"   : chess.KNIGHT,
                "height"   : chess.KNIGHT,
                "fight"    : chess.KNIGHT,
                "sight"    : chess.KNIGHT,
                "site"     : chess.KNIGHT,
                "horse"    : chess.KNIGHT,
                "whores"   : chess.KNIGHT,
                "9th"      : chess.KNIGHT,
                "nike"     : chess.KNIGHT,
                "might"    : chess.KNIGHT,

                "pawn"     : chess.PAWN,
                "pan"      : chess.PAWN,
                "vaughn"   : chess.PAWN,
                "paw"      : chess.PAWN,
                "prawn"    : chess.PAWN,
                "lawn"     : chess.PAWN,
                "dawn"     : chess.PAWN,
                "shaun"    : chess.PAWN,
                "spawn"    : chess.PAWN,
                "yawn"     : chess.PAWN,
                "pawns"    : chess.PAWN,
                "poem"     : chess.PAWN,
                "home"     : chess.PAWN,
                "on"       : chess.PAWN,

                "queen"    : chess.QUEEN,
                "wien"     : chess.QUEEN,
                "queens"   : chess.QUEEN,

                "rook"     : chess.ROOK,
                "rooks"    : chess.ROOK,
                "route"    : chess.ROOK,
                "rock"     : chess.ROOK,
                "roof"     : chess.ROOK,
                "brook"    : chess.ROOK,
                "brooke"   : chess.ROOK,
                "crook"    : chess.ROOK,

```

```python
                "king"     : chess.KING,
                "kings"    : chess.KING,
                "kong"     : chess.KING,
                "kill"     : chess.KING,
                "cling"    : chess.KING,
                "kin"      : chess.KING,

                "bishop"   : chess.BISHOP,
                "bishops"  : chess.BISHOP,
                "shop"     : chess.BISHOP,
                "chop"     : chess.BISHOP,
                "shops"    : chess.BISHOP
                } #dictionary with similair sounding words

class SpeechEngine: #handles voice input and identifying the player move
    def __init__(self):
        self.r = sr.Recognizer()
        self.m = sr.Microphone()

    def listen_for_move(self):
        text_string = None
        with self.m as source: self.r.adjust_for_ambient_noise(source) # adjusts for ambient noise
        print("Listening...")
        with self.m as source: audio = self.r.listen(source)
        print("Got it! Analyzing move...")
        try:
            value = self.r.recognize_google(audio) # recognizes speech using Google Speech Recognition
            if str is bytes:  # special handling to correctly print unicode characters to standard output
                text_string = format(value).encode("utf-8")
            else:
                text_string = format(value)
        except sr.UnknownValueError: # error handling
            print("Oops! Didn't catch that move, try again")
        except sr.RequestError as e:
            print("Uh oh! Couldn't request results from Google Speech Recognition service; {0}".format(e))
        return(text_string.lower() if text_string else None) # returns a lowercase string if its not None

    def piece_finder(self, text_string): # gets the chess piece from the recorded text string
        string_split = text_string.split() # splits the string into a list
        for word in string_split: # for each word in the string it checks if its in the dictionary
            piece = pieces_dict.get(word, None)
            if piece is not None: # if theres a match, it returns the piece
                return piece

    def square_finder(self, text_string): # gets the square where the piece is to move to
        square = re.search(r'[a-h]+[1-8]', text_string) # re expression to find a coordinate in the string
        if square is not None:
            return square.group() # returns the square if its found
        else:
            return None
```

Appendix C

Acumen

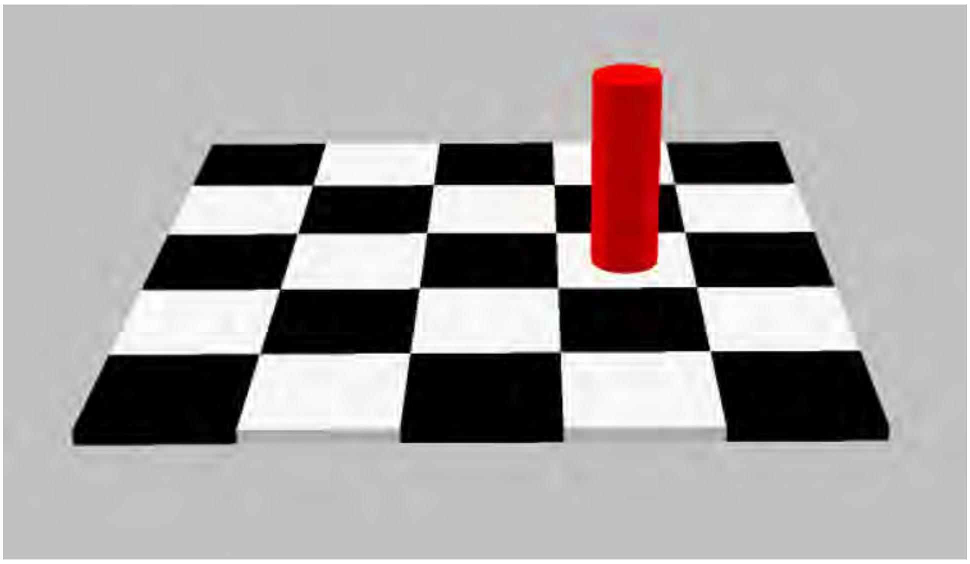


Figure C.1: Simulation of lifting movement, made using Acumen [5].

The following pages include the simulation code which was used in Acumen [5].

```
//Acumen simulation av hur en schackpjäs rör sig i vår robot
//Skapad av Axel Sernelin och Oscar de Brito Lingman
//ME133X Grupp 33
//Vi vet inte hur man ska få ett object att rotera kring en fix punkt så vi kunde inte göra robotarmarna.
model Main(simulator) =
initially
c1 = create Cyl((0,0,0)), //skapar upp cylindern
c2 = create Chess(), //skapar upp brädet
x1=0, x1'=0, //sätter start position och hastigheter för de olika riktningarna
y1=0, y1'=0,
z1=2.75, z1'=0
always
if x1<3 //när x1 positionen så stannar den
then x1' = 1
else
x1' = 0,
if x1>=3 && y1 < 2 //när x1 positionen är uppnådd så startar y1 och kör tills y1 är i position 2
then y1' = 1
else
y1' = 0,
if x1>=3 && y1 >= 2 && z1 > 0.75 //när x1 och y2 är i rätt position startar z1 hastigheten tills z1 positionen är på 0.75
then z1' = -1
else
z1' = 0,
c1.pos = (x1,y1,z1) //sätter cylinderns position
model Cyl(pos) =
initially
_3D = (),_Plot=()
always
_3D = (Cylinder center = pos + (0,0,0) //skapar cylinder med olika attribut och position pos som justeras från huvudprogrammet
color = red
length = 1.5
radius = 0.25
rotation = (pi/2,0,0)
)
model Chess() =
initially
_3D = (
(Box center = (0,0,0) //skapar rutor som på ett schackbräde, ett block per ruta.
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (1,0,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (2,0,0)
color = black
length = 1
width = 1
height = 0.1
)
```

```
(Box center = (3,0,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (4,0,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (0,1,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (0,2,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (0,3,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (0,4,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (1,1,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (1,2,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (1,3,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (1,4,0)
color = white
length = 1
width = 1
height = 0.1
)
```

```
(Box center = (2,1,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (2,2,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (2,3,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (2,4,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (3,1,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (3,2,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (3,3,0)
color = black
length = 1
width = 1
height = 0.1
)
(Box center = (3,4,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (4,1,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (4,2,0)
color = black
length = 1
width = 1
height = 0.1
)
```

```
(Box center = (4,3,0)
color = white
length = 1
width = 1
height = 0.1
)
(Box center = (4,4,0)
color = black
length = 1
width = 1
height = 0.1
))
```

TRITA-KTH-ITM 2021:51