UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Project and Professionalism

(6CS020)

A1: Project Proposal

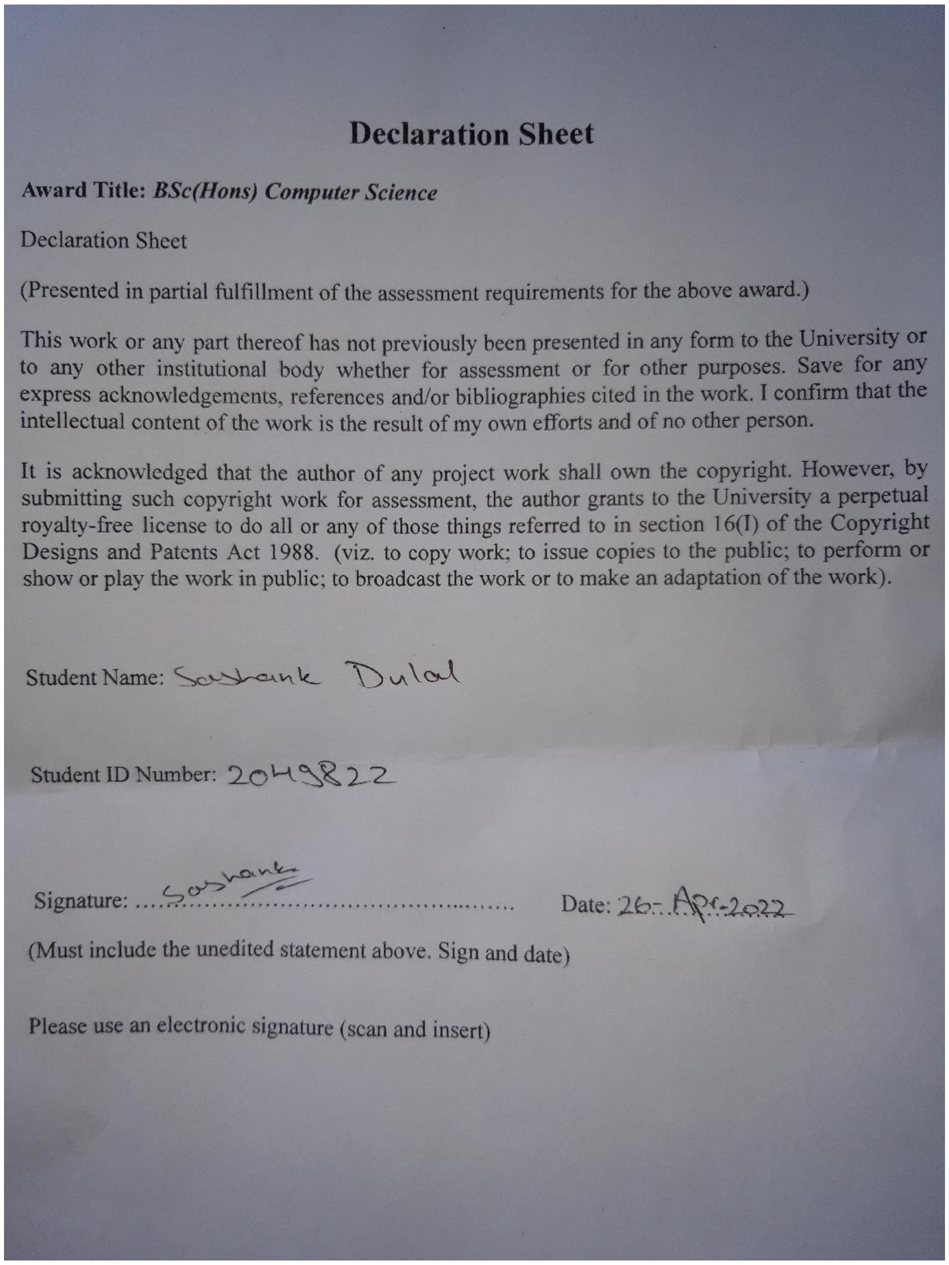
Student Id	: 2049822
Student Name	: Sashank Dulal
Supervisor	: Sangay Lama

Submitted on	: Tuesday, November 16,2021

Acknowledgements

My sincere gratitude goes towards University of Wolverhampton and Herald College Kathmandu for this programme and course availability. Secondly, I would like to thank module leaders Mr. Nirmal Thapa and Mr. Biraj Dulal for such experience regarding the project and for much needed guidance. Finally, special thanks to our supervisor Mr. Sangay Lama for his time and effort to make this project better with his knowledge and support.

Declaration Sheet



Declaration Sheet

Award Title: ***BSc(Hons) Computer Science***

Declaration Sheet

(Presented in partial fulfillment of the assessment requirements for the above award.)

This work or any part thereof has not previously been presented in any form to the University or to any other institutional body whether for assessment or for other purposes. Save for any express acknowledgements, references and/or bibliographies cited in the work. I confirm that the intellectual content of the work is the result of my own efforts and of no other person.

It is acknowledged that the author of any project work shall own the copyright. However, by submitting such copyright work for assessment, the author grants to the University a perpetual royalty-free license to do all or any of those things referred to in section 16(I) of the Copyright Designs and Patents Act 1988. (viz. to copy work; to issue copies to the public; to perform or show or play the work in public; to broadcast the work or to make an adaptation of the work).

Student Name: Sashank Dulal

Student ID Number: 2049822

Signature: Sashank .... Date: 26-Apr-2022

(Must include the unedited statement above. Sign and date)

Please use an electronic signature (scan and insert)

Abstract

In this age of robotics and Artificial Intelligence, much of our day to day lives have been directly or indirectly affected by automation. The exploration of capabilities of such robotic and intelligent systems helps humans get ready for the future scenarios where robots could be intelligent like them. Chessmate aims to analyze the different aspects of chess game, image processing techniques and robotics to build a functional chess-playing robot and explore the feasibility as well as the extent to which the system is efficient to aid in further exploration of game-playing robots in future. This report covers the research done on journals, findings from them, requirements of the project, breakdown and the findings of project done in detail.

Table of Contents

Table of Figures

1 Introduction

Chess is a 2-player strategy game played on an 8x8 board with 16 pieces for each player. With the aim of trapping the opponent's king piece (Also called "Checkmate"), the players devise their own strategies and moves to get to that aim. Since the invention of chess as a form of a mind game, people have been trying to assert their dominance in terms of mental abilities through chess (Chess.com, 2021). And with the advent of chess theories, the game have become more and more difficult for a normal person to understand. From a business standpoint, the global chess market has been generally rising in most of the countries. The North American industry is expected to hit 40 million US dollars by 2022. These all statistics point to the fact that the equipment and resources related to chess are on demand (Statista.com, 2021).

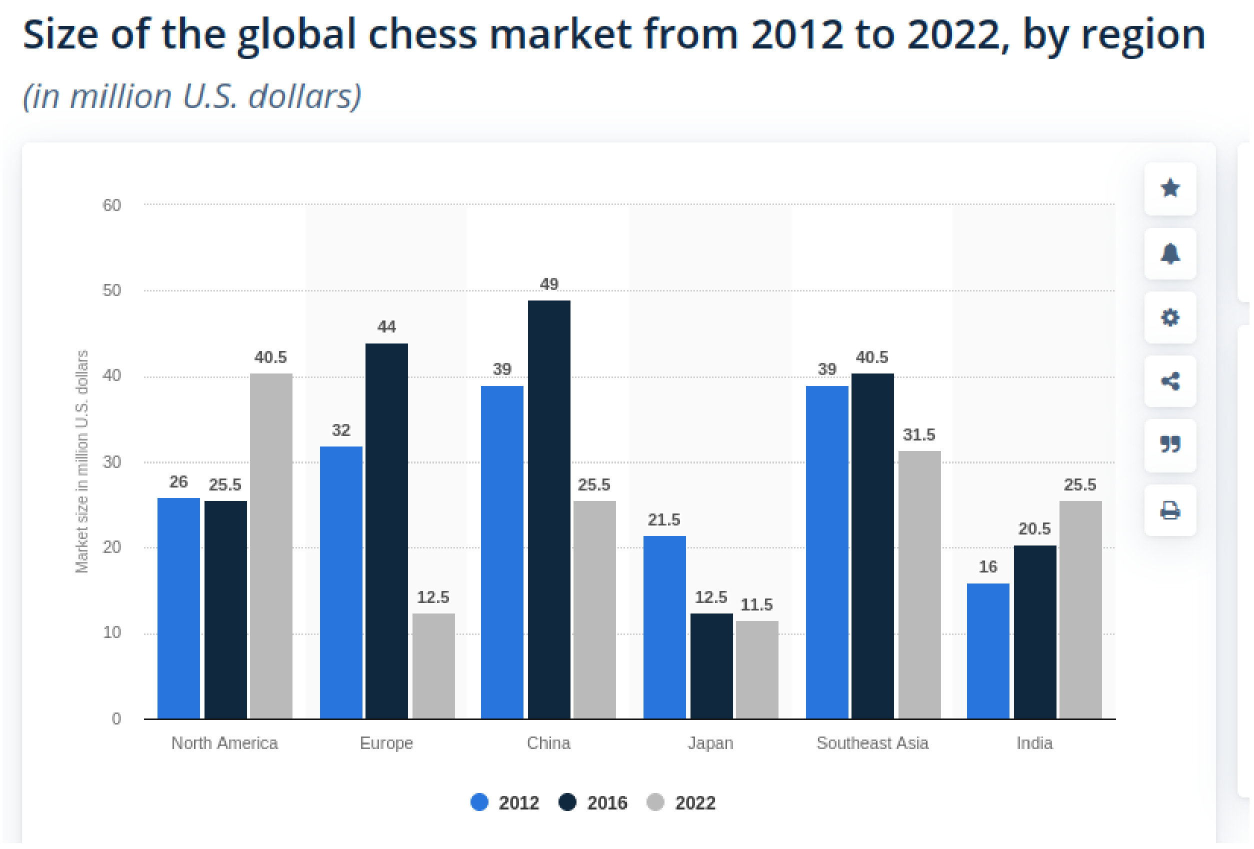


Figure 1: Global Chess Market Size Comparison from 2012 to 2022 (Statista.com, 2021)

With the increase in the whole industry, the availability of intelligent and specialized equipment for chess is very low. Similarly, the published resources are also vague to understand and expensive to manufacture. The main objective of this research is to perform an in-depth analysis on implementation of Artificial Intelligence (AI) to detect and recognize chess moves, analyze different methods to physically move the chess pieces with the help of a robotic system and determine the best way to move the pieces based on the analysis of different aspects of those methods and then construct the robot based on the research findings. At the end of this research, a fully functioning system capable of playing chess against a user will be constructed with the help of the research materials and findings.

Chessmate makes use of image processing library named opencv to carry out all the vision related tasks such as determining borders, squares and tracking chess movements inside the board. Similarly, the chess gameplay is handled by an open-source chess engine.

1.1 Problem Domain

I. No Proper method of training by playing physical match for newcomers

If a newcomer wants to gain proper experience at chess, he/she have to physically play that game and since they are not well-trained, playing with another professional can be discouraging for them and it is difficult to find the person who agrees to play with an unexperienced player

II. Costly trainings and coaches

In order to get good at chess, proper training and regular practice is required and the training programs and coaches are expensive to maintain.

1.2 Project as a Solution

I. Even the newest inexperienced players get to play with the standard opponent.

II. The cost of getting trained drastically decreases as it is a one-time investment. Once the bot has been bought, users can play whenever they like. Similarly, the bot would be constructed out of locally-sourced parts, repairing and even upgrading would not be difficult or expensive.

1.3 Aims and Objectives

1.3.1 Aims

- Investigate the feasibility of different robotic methods for movement of small objects in a confined space.
- Perform research on different algorithms used for moving the robots within a fixed area.
- Construct a robot using research findings that will be able to play chess against a user opponent.

1.3.2 Objectives

- A robotic system will be designed and developed using the analysis of research findings.
- Implement different image processing algorithms to detect and track movements.
- The robot will be made in such a way that it will be communicating with custom made/ open-source API that can generate chess moves for any given move of the user.

2 Artefacts to be developed

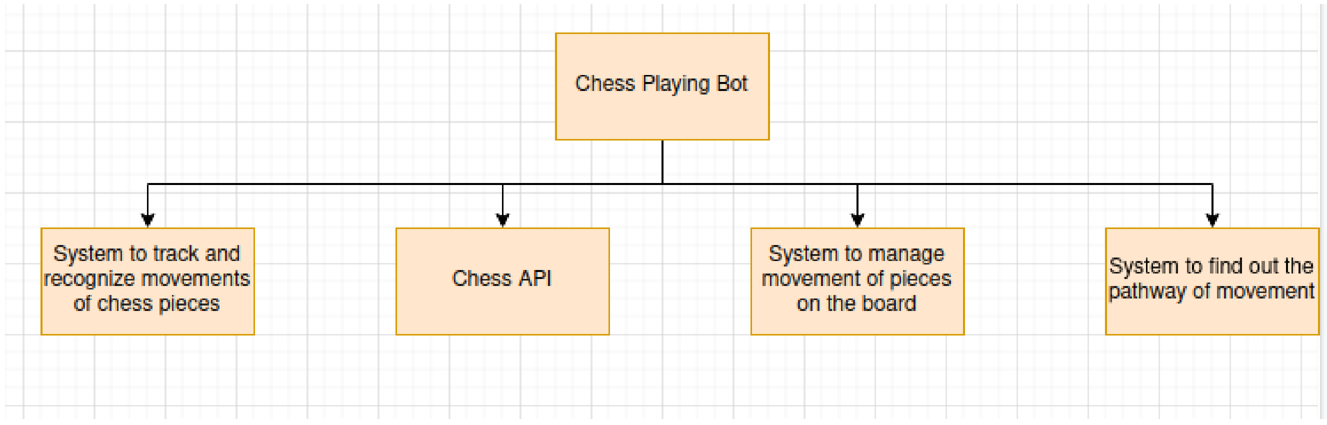

Chess Playing Bot
System to track and recognize movements of chess pieces
Chess API
System to manage movement of pieces on the board
System to find out the pathway of movement

Figure 2: Functional Decomposition Diagram (FDD) of the proposed system

2.1 System to detect movements of chess pieces placed by the user on the board.

This system is responsible for tracking the position of chess pieces on the board and detecting the moves of the user.

2.2 Chess API

This system generates best possible moves based on the moves done by the opponent user.

2.3 System for movement of pieces in the board.

This system takes in the recommendations made by the chess engine to physically move the chess pieces on the chess board.

2.4 System to find out the pathway of the movement.

This system finds out the optimal pathway for the chess piece to move from one place to another.

3 Academic Question

- How will the bot get the best possible movement for the move produced by the user?
- How will the chess pieces be moved using the arm?
- What will be used to track the movement of chess pieces?

4 Scope and Limitations

4.1 Scope

With the aim of creating a chess-playing bot capable of recognizing moves made by user and the bot, validating those moves, generating next moves for the bot and moving the chess piece on the board, this project has been initiated and will deliver a fully working robotic system in the end. The detailed scope statement of the project is provided below:

4.1.1 Project Aim

The aim of this project is to build a fully-functional chess playing robot with computer vision.

4.1.2 Resources

- One person team
- Around NRS 50000 budget

4.1.3 Project Roadmap

- Project approval and determining features by Nov 17, 2021
- Research on similar systems till Dec 19, 2021
- Chess engine by Feb 20, 2022
- Chess tracking system by March 5, 2022
- Arm movement system by April 15, 2022
- Integration and testing until Apr 25, 2022

4.1.4 Out of Scope

- Industry standard robotic arm
- Building hardware such as tripod stand, image processing kit from scratch

4.2 Limitations

The major limitations of the project are time, budget and skills. Since the whole project along with proper documentation and testing must be finished within 6 months of starting, maintaining quality along with speed of the development and testing is not possible. Similarly, budget is very limited which means hit and trial when it comes to hardware is also not an option. The main limiting factor is the skills required for the completion of the project which is somewhat lacking. Furthermore, unexpected outcomes such as malfunctioning hardware, bugs and errors can significantly affect the whole project flow. The project requirements might also change depending on future circumstances and it directly hampers the final deliverable. Similarly, there are risks related to hardware such as failing hardwares, malfunction and so on.

5 Report Structure

1. **Literature Review**

   In this section, published literatures that have covered the similar methodologies and principles have been thoroughly studied and analyzed. Different aspects of the project have been considered while studying the journals published.

2. **Research on Similar Systems**

   This part of the report explains a similar project like ours and the critical aspects of that project have been researched for aiding in the future roadmap of this project.

3. **Project Methodology**

   This section explains the methodology used for development of this project as well as the major milestones set for the project.

4. **Tools and Technologies Required**

   In this part, the required skills, hardware tools, softwares and libraries have been listed and their uses have been mentioned.

5. **Artefact Design**

   The overall breakdown of the system to the bare minimum for development and testing have been clearly explained in this section. Multiple subsystems and their respective functions have been detailed in this part.

6. **Conclusion**

   The major findings of the project along with the status of the finished product has been explained in this section.

7. **Critical Evaluation**

   Different factors of the project as well as their outcomes have been mentioned in this part of the report.

8. **Future Works**

   This heading explains the future possibilities of expansion as well as betterment of the system and new areas that can be used to make the system more versatile and strong.

9. **Evidence of Project Management**

   Under this heading, the screenshots of the meeting with the client (i.e. Supervisor) have been included.

10. **References**
11. **Appendices**

6 Literature Review

6.1 Gambit: An Autonomous Chess-Playing Robotic System

This paper presents a 6-DoF chess playing robot system named Gambit which is capable of playing chess on a physical board against humans. It includes a low-cost sensor for perception, custom-made robot arm and learning algorithms for detection and recognition of objects on the board. The authors of the project mainly view this project as a way of exploring the perception and manipulation in a “noisy, less constrained real-world environment”. Gambit does not require the chess pieces to be exactly modelled or instrumented as it monitors the state of the board and tracks the movement of the pieces that the opponent has made and communicates with the human opponent using spoken-language interface.

The perception system tracks the position of pieces on the board and the board is not fixed relative to the board and the board is calibrated continuously throughout the game. On the other hand, the system makes use of an open-source arm design as it was moderate in cost (about $18000 for parts) and their major aim was to enable smooth motion and interaction. It consisted of a 6-DoF arm with a gripper attached ad one end. As shown in figure. 1, the DoFs 1, 2 and 3 are used for positional control whereas 4, 5 and 6 are used for orientation control with the help of roll-pitch-roll spherical wrist.

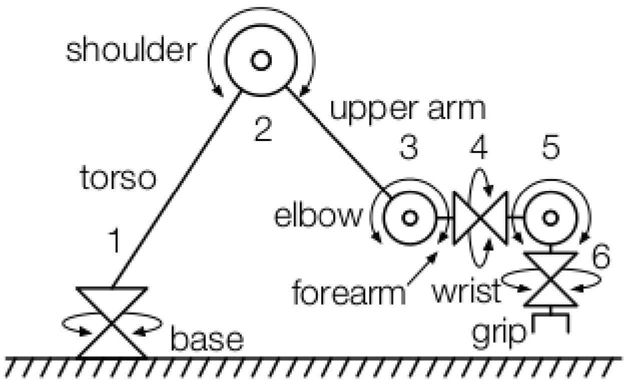


Figure 3: Schematic of the arm construction (Matuszek, et al., 2011)

For sensing purposes, the system consisted of a shoulder-mounted depth-sensing camera (similar to Xbox Kinect) along with a camera attached to the gripper. The depth-sensing camera provided color information along with depth of each pixel and worked within a range of 0.5m to 5m. Since the minimum working range was very high, it presented a problem that the camera had to be mounted high but the authors solved this issue by mounting the camera facing backwards of the torso of the arm so that the distance is increased.

The driver software for the system ran on a dedicated Intel Atom net-top PC with CAN and RS-485 PCI cards whereas controlling of the arm was done using a separate computer that handled ROS operations. And the system for detection and recognition of board consisted of four hierarchical classifiers mainly for detecting squares, pieces/backgrounds and two for recognition of types of chess pieces. Each piece was labelled out of pre-defined pieces {B, N, K, P, Q, and R}. The hierarchy of the vision algorithm is depicted in the figure below.

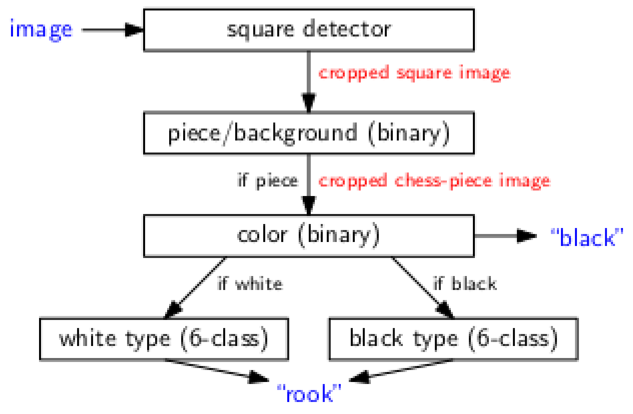


Figure 4: Hierarchy of the chess-piece recognition algorithms (Matuszek, et al., 2011)

(Matuszek, et al., 2011)

6.2 MarineBlue: A Low-cost Chess Robot.

MarineBlue was selected for literature review because it covers most of the areas of our project in detail. Ranging from chess piece detection using HSV space to using inverse kinematics through an analytical approach to get to the optimal solution without heavy computation. Although the system is old for now, most of the changes only happens on the gameplay side of the system so the concepts related to tracking chess pieces and moving of arm are still relevant till now.

This paper focuses on the details of a chess robot named MarineBlue. Emphasis has been given on the algorithms involving computer vision and robot manipulation in this paper. MarineBlue is an autonomous robot that can recognize moves placed by the user, generate moves for that scenario and move that particular piece with the help of an arm. Development of a compact robot on low cost was prioritized in this paper.

During the construction of the system, many factors such as chess board and construction of chess pieces were considered such that it could be easier to replicate. First the authors had thought of making the chess pieces uniform in dimensions in order to make it easier to move them but in the end, the gripper was constructed in such a way that it could pick and move every type of chess pieces. On the other hand, the chess board had to be modified in such a way that the length of the squares was 30 millimeters. Similarly, disjunctive colors were chosen to make sure that the algorithms detect which squares were occupied and track the objects.

For the vision side of the project, a Sony DFW-VL500 camera was used to gather high quality images and it was mounted 1 meter high above the chessboard. Next, for the movement of chess pieces, a Robix RC-6, a portable and highly configurable robot was used. The robot consisted of a set of separate units that could be put together to form a bigger unit. The units could move in conjunction to other units to form one smooth motion with the help of servos. The controllers controlled the servos and the controllers is linked to the parallel port of the chess computer and the servo controller gets the servo commands from the computer in the form of Robix-dependent scripting language. In order to make the robot sway less because of the heavy end, the gripper was made lighter and a degree of freedom (rotation) was removed from the gripper. Similarly, the gripper was made longer so that it would not collide with other pieces when a piece is being picked up and the gripper was made hemispherical to accommodate for small positioning errors.

For handling chess gameplay, an application that analyzes the situation of the board, recommend moves and execute them was developed in C/C++ using MS Visual Studio and was running on Windows 2000 computer.

To detect the current game situations, the video feed was processed through three layers i.e. pixel classification layer, board layer and chessboard layer. The pixel classification layer determines the class of every pixels according to the color properties of the pixels. There were four defined classes: light square, dark square, light piece and dark piece. Since the white piece on white square could not be easily recognized by the algorithm, the authors used hue, saturation and brightness color space instead of RGB space.

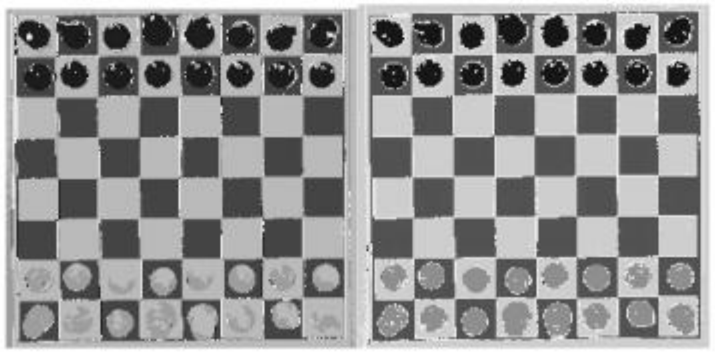


Figure 5: Comparison between RGB Space (on the right) and HSB (on the left)

Then the board layer uses the matrix containing the states of each pixel classes acquired from the previous stage to determine the position of board and pieces inside image and to find out which squares are occupied by the pieces or not. A reference image containing the color of dark square is used and the algorithm searches or the corners of the image by searching the ends of the pixels relating to the dark square category. Then interpolation is used to calculate the position of each square in the image. After that, the number of pixels in the classes "dark piece" and "light piece" is calculated and if the calculated number crosses predefined limit, then the algorithm declares that the square is filled. The ratio of surface area of the piece and the surface area of the square determines the threshold value. The chessboard layer is then fed with the output of previous stage: the list of squares occupied by a chess pieces of a color. This layer also keeps tracks of previous data which can be used to track the movements easily.

To compute the angle for the servos based on the destination of the gripper and the arm configuration, inverse kinematics was used.

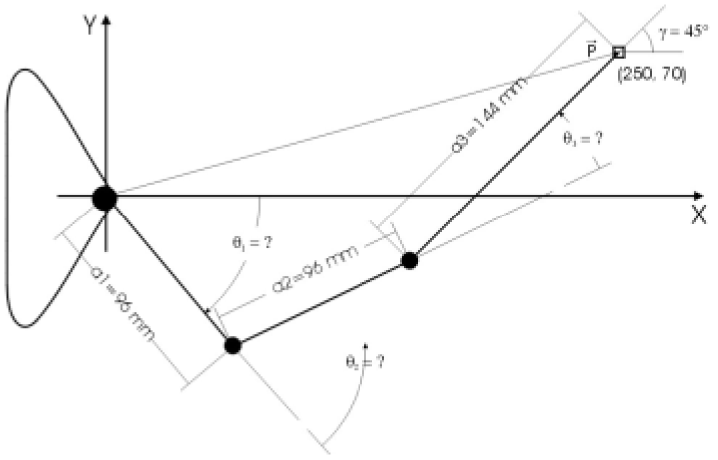


Figure 6: Top view of the Inverse Kinematics for the arm

Since solving the inverse kinematics involve heavy computational power, it is not practical to solve the algorithm for more than six joints. For this paper, the analytical approach was considered along with an assumption that the fourth segment was always in line with the third segment was made.

$$3.1: \quad \gamma = \theta_1 + \theta_2 + \theta_3$$
$$P = a1 + a2 + a3$$

The second formula can be rewritten as formula 3.2:

$$x = a_1 c_1 + a_2 c_{12} + a_3 c_{123}$$
$$y = a_1 s_1 + a_2 s_{12} + a_3 s_{123}$$

with

$$c_1 = \cos(\theta_1) \qquad s_1 = \sin(\theta_1)$$
$$c_{12} = \cos(\theta_1 + \theta_2) \qquad s_{12} = \sin(\theta_1 + \theta_2)$$
$$c_{123} = \cos(\theta_1 + \theta_2 + \theta_3) \qquad s_{123} = \sin(\theta_1 + \theta_1 + \theta_1)$$

Figure 7: Formula for the inverse kinematics

When the destination for gripper position (x, y, γ) is given, the objective is now to find out the three theta values by solving the above system of equations. As there can be infinite number of solutions for the equations if the γ is considered not important, the method used increases the angle γ in small increments and calculate two solutions to this angle if those solutions exist. After that, the best solution that requires the least movement for the arm is chosen.

(URTING & BERBERS, 2003).

6.3 Multi-Object Recognition and Tracking with Automated Image Annotation for Big Data Based Video Surveillance

In this paper, an improved region based scalable convolution neural network (IRS-CNN) based multiple object tracking model has been discussed. It uses Automatic Image Annotation (AIA) in order to improve the detection capacity and reduce computation time.

In the IRS-CNN method, the anomaly detection takes place in three stages namely AIA model to annotate images, RPN (Region Proposal Networks) and R-CNN. During the first stage, the image annotations are generated. Then a CNN creates regions in the image and finally at the last stage, the regions created in previous stage is used to detect the anomalies in the image. The IRS-CNN model is significant in size and consists of smaller sized sub-networks to detect anomalies. Similarly, the scaling awareness with AIA enables the IRS-CNN model to detect anomalies efficiently no matter the size and distribution.

Furthermore, CNN with WARP (Weighted Approximated Ranking) method was used to improve the time complexity of the annotation process. Five convolutional layers with 3 connected layers was used from CNN model. Similarly, the Region Proposal Networks (RPN) was made in such a way that it inputs images with varying dimensions and outputs groups of rectangular objects as proposals with their objectless values. In order to facilitate faster resource sharing with R-CNN, each network was made to share similar convolution layers and for region proposal, a smaller network gets an input of n*n spatial window where the sliding windows get mapped to lesser dimensional feature set. To train the RPN, a binary label (normal or abnormal) for each of the anchors (rank of every region box) was assigned.

Hence the paper concluded that the proposed IRS-CNN based MOT system was better than the existing RS-CNN method with the use of AIA to increase efficiency in detection and decrease computation complexity.

(Vijiyakumar, et al., 2021)

6.4 Analysis of the inverse kinematics for 5 DOF robot arm using D-H parameters

Inverse kinematics being an integral part of our system, failure to understand the importance of the algorithm, inputs and outputs as well as mathematics could prove fatal to the project. So an in-depth analysis of inverse kinematics algorithm was chosen for study.

This paper discusses on the drawbacks of using brute force method in iteration to solve the equations involving D-H parameters of the arm and focuses on reducing the RMSE (Root Mean Squared Error) to get accurate solution to the non-linear equations. Then the results was implemented to grip objects using DOF robotic arm and the methods used in this paper could be used for robotic arms up to 6 DOF. Since the location and orientation of the base of the arm are already known for every case, finding out the location of the end effector must be done by calculating the transformation (relationship between the base and the end effector). The transformation matrix $^{(i-1)}\mathrm{Ti}_{\mathrm{is}}$ calculated with the help of four parameters: a_i, α_i, θ_i and d_i where a_i is the distance between z_i and z_{i+1} along x_i, α_i is the angle between z_i and z_{i+1} about $x_{i,}$ d_i is the distance between xi_{-1} and x_i along z_i and θ_i is the angle between xi_{-1} and x_i about z_i.x. Then the system of equations to be explored is given below:

$${}^{i-1}T_i = R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\theta_i)D_z(d_i)$$

where,

$$R_z(\theta_i) = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D_x(a_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\alpha_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & di \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 8: Formula to get the transformation matrix

Hence, the formula for the transformation matrix is given by:

$${}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & -c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 9: Formula for the transformation matrix

Inverse kinematics could be achieved through two mainly two methods; analytical method which requires solving closed equations and numerical method where an iterative approach is used to get an approximation by solving he system of non-linear equations. The second method is computationally complex. The analysis is done on the numerical method in this paper which is conducted in 8 steps. First of all a line diagram with the joints and connections between joints is created, then the joints are assigned frames and D-H parameters are calculated, transformation matrix between two successive links is constructed, the transformation matrices are multiplied to get the total transformation matrix, in the next step, the numeric transformation matrix is constructed for the target location, then an equation is formed by equating the two obtained matrices. After that step, the equations are solved using iterative methods to get the angles for the joints and finally RMSE for the solutions is calculated using forward kinematics to get the most suitable solution.

The paper concluded that the test was done with over 20 test cases and the authors found the algorithm to be working efficiently and results can be used for the robotic arms upto 6 DOF.

(Patil, et al., 2017)

6.5 Analysis of Literature Review

The Gambit robot makes use of off-the-shelf equipments such as a camera that is similar to Xbox Kinect, a PC equipped with intel atom processor and an open-source arm design to build a capable yet slightly cheaper chess playing bot using inverse kinematics for generating movements of the bot. The system was able to keep track of the initial position of the board and then track the movement of the chess pieces as they were moved throughout the board so the chess pieces would not have to be modelled correctly.

The MarineBlue system presented all of the aspects to be considered while building a chess robot from scratch and with price considerations in mind. Since the paper was from the early 2000s, machine learning techniques found today were either not invented or not that sophisticated as it is now due to computing bottlenecks and so on. So the paper described using of surface areas to recognize chess pieces and track them on the chess board. For the movement of the arm, the inverse kinematics algorithm was used which can be found in similar systems till now. Similarly, the algorithm can detect all valid chess moves, even the castling move (which involves a displacement of the king and the rook), given that the user adheres to the chess playing rules. Some cheating actions, such as swapping two chess pieces of the same color would go unnoticed by this algorithm, since it will not detect any changes on the chessboard in that case. The reason for this is that the board layer only recognizes the color of a piece, not its type.

The “Multi-Object Recognition and Tracking with Automated Image Annotation for Big Data Based Video Surveillance” paper discusses use of multiple object tracking using technologies such as neural networks and Automatic Image Annotation in order to improve the accuracy of detection of objects in real time. Similarly, use of big data analytics in video surveillance brings in a variety of new areas of application. Being better than the pre-existing RS-CNN model, and using AIA to improve detection with the least requirement of computation, the paper concluded that the method had shown promising results.

Finally the “Analysis of the inverse kinematics for 5 DOF robot arm using D-H parameters” paper presented with the unique aspect of using analytical approach with iterative method and avoiding errors while calculating inverse kinematics for DOF robot arms up to 6 joints. While testing the research results, the authors tested the algorithm on 20 sample cases and they stated that the algorithm worked efficiently.

With the analysis of these different algorithms and techniques, it can be deduced that inverse kinematics is one of the most important algorithm while working with robotic arms that involves the use of various DOFs.

7 Research on Similar Systems

7.1 Raspberry Turk

Raspberry Turk is an open-source chess playing bot that uses Raspberry Pi as the main computer. The robot was inspire by the Mechanical Turk, a chess playing machine from the 18th century. It uses computer vision technique to keep track of the situation of the board and then makes moves based on the changes on the board. A raspberry pi camera is used as the sensor to track the movements and in order to train the computer vision algorithm, the author wrote a custom script and then captured every possible move on the board manually and he took about 4 images of a chess piece in each square varying the position and orientation within the square and changing the lighting scenario. Then the image were processed in a series of steps starting from creating individual folders for each iteration, then breaking the images and then creating a labelled folder containing 60x60 pixel images and then preparing the dataset out of it.

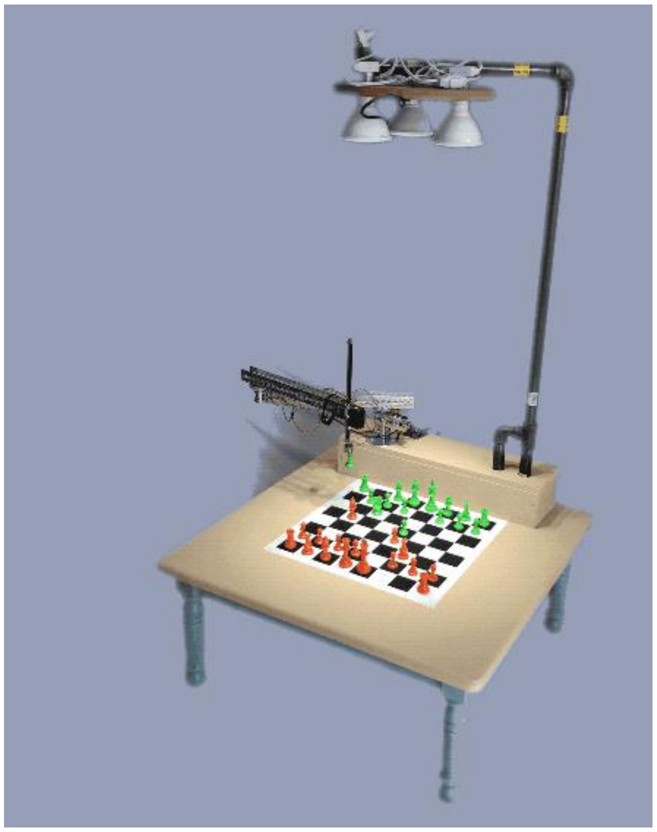


Figure 10: Picture of the Raspberry Turk System (Meyer, 2017)

In order to move the chess pieces, a SCARA (Selective Compliance Articulated Robot Arm) was used and it could move only in x and y axes. Then for picking and placing the chess piece, an electromagnet was attached to horizontal beam that could move up and down and the chess pieces contained a piece of metal attached on a drilled hole on top of them. For handling the chess gameplay, stockfish engine was used.

7.2 Analysis of the system

As the image processing techniques could not easily identify the pieces on the board if the white piece is on the white square and black piece is on the black square so the author used green and orange as the color of the pieces. That was one of the main highlights of the system. Similarly, to allow for promotion of rook to other pieces, the author has trained a convolutional neural network to train the system classifying pieces (rook, bishop or knight) in the image to find out which piece the player promoted the pawn to. So this made the end result comparable to a chess game against real human opponent.

(Meyer, 2017)

8 Project Methodology

Agile scrum is selected as a development methodology to be followed for this project. Since the requirements may change over time and agile approach allows the project to cope up with the changing requirements. Similarly, the major advantage of using scrum methodology is that it focuses on creation of deliverables at the end of each sprint and prioritizes incremental development method. So at the end of every sprint, we are left with a working version of the project. As the project will go through various iterations throughout the development cycle, the chance of it failing or having problems significantly goes down. The use of sprints and pre-set goals for each sprint means that the project is completed within the timeframe and prioritize the important aspects of the project for each sprint.

For our purpose, the sprint will last about 8-10 weeks and the whole project will be completed in 3 sprints. At the beginning of the project, the product backlog will be prepared and sprint backlog for each sprint will be prepared during the start of each sprint. Instead of daily standup, the meeting will be done on a weekly basis with the product owner/Supervisor and at the end of every sprint, sprint review and retrospective will be conducted.

9 Tools and Technologies

9.1 Hardware Requirements

- Laptop with at least intel i5 10th gen or similar processor, 8 GB RAM and 256 GB storage

  Laptop is required to program, manage and debug the scripts running on raspberry pi and arduino. Similarly, it acts as a bridge between the developer and the system in case of any errors.
- Raspberry Pi 3 B+ or better (1GB RAM or more with at least 16 GB UHS-1 SD card)

  Raspberry Pi is used as the main computing part of the system. It is be responsible for image processing and chess engine related tasks. It performs tasks such as board detection, moves generation, game validation and so on.
- Raspberry Pi Camera

  The camera is paired with raspberry pi as the main sensor of the system for gameplay tracking purposes. A raspberry pi camera module v2 was used for this project.
- Tripod Stand

  Tripod stand acts as a platform for the camera used with the raspberry pi.
- Chess Board

  A typical chess board with standard chess square size was used.
- Servos

  Servos are used as the actuators for moving the chess pieces on the board with the construction of the arm.
- Robot arm kit

  Robot arm kit that can be easily assembled was used. The kit was acquired from mindsieducation.com.
- Arduino

  Arduino was used as the controlling mechanism for the robotic arm. The arduino gets location of the movement from the raspberry pi and controls the robtic arm to get to that particular position.
- Servo Driver

  Since multiple servos are required for a robotic arm, controlling servos using arduino only makes it complex in terms of power consumption and processing. So a I2C based servo driver was used.

- Battery/Power Supply

  A power supply to power raspberry pi, arduino and the servo driver was required.

9.2 Software Requirements

- Raspbian OS

  Raspbian OS or Raspberry PI OS is a debian based operating system for raspberry pi. It supports all the raspberry pi models. Raspbian OS is required for this project as a main OS running on raspberry pi to execute the scripts to track the gameplay and make necessary calculations to move the arm based on inverse kinematics.
- Arduino IDE

  Arduino IDE is used in a separate computer in order to program the arduino in such a way that when it receives the destination of the gripper and the action it needs to take (i.e. make move, capture piece, castling etc.), the arduino can execute set of commands to the servo on the robotic arm to perform such actions.

- VNC Server/ VNC Viewer

  Since we will be using a separate device for programming, we need VNC for programming in headless mode on raspberry pi and when the project is finished, VNC helps a lot in debugging wirelessly.

- SSH

  SSH is used to access raspberry pi through terminal of another device to execute commands

  Swiftly during debugging or deployment. Similarly, when VNC server is enabled on raspberry pi, another device can access its contents easily through VS Code SSH connection to make changes to program code on the go.

- Pycharm

  Pycharm is free-to-use IDE for python with community version and it provides version control for easy integration with GitHub and it also provides PEP8 checking which helps maintain code standard. As we have python as most of the portion of our project, pycharm is the IDE of choice for python development for the project.
- OpenCV

OpenCV is an open-source computer vision library focused mainly on real-time applications. It is one of the best libraries for image processing on the go. Similarly, it supports variety of OS along with support on various programming languages. It is used in our project to identify chess board and track movements on the board.

- GitHub

  The use of version control is very important for projects like this. There are many versions of code with step changes made over time. So to keep track of all code is a difficult task to do. Similarly, there will always be a backup copy of our project at all times. GIT is the technology that enables version control and GitHub provides services related to GIT.

- VSCode

  VSCode is an open-source IDE supporting multiple languages along with support for variety of extensions for making development easier. The extension we will be using is SSH extension provided by Microsoft which enables us to program raspberry pi over SSH from another device.

10 Artefacts

As explained in the introduction, the important task of the project is to identify the board, detect the squares, track changes occurring inside the chess board, generate moves according to the moves placed by the user and move the chess pieces for the computer, the whole project is divided into 3 major subsystems based on the types of tasks. The Functional Decomposition Diagram (FDD) of the system can be seen below:

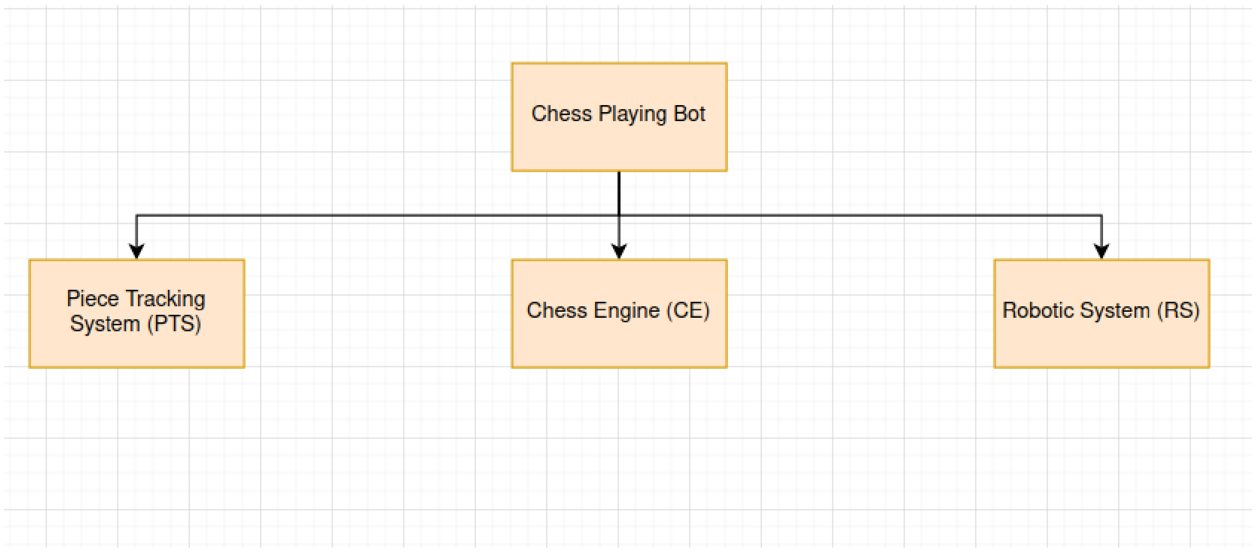


Figure 11: Proposed Artefacts of the System

Note that the artefacts provided earlier is different from the final artefact as the moving pieces and finding pathway for movement can be done by one single system and the chess API is changed with chess engine as we are using a pre-built chess engine for generating moves.

10.1 Chess Engine

This subsystem is responsible for handling all the gameplay related tasks such as validating moves, keeping track of the game, generating moves, detecting if the game has ended or not and so on.

10.1.1 SRS

CE - F - 1.0

Types of Requirements:

F : Functional Requirements

NF : Non-Functional Requirements

UR : Usability Requirements

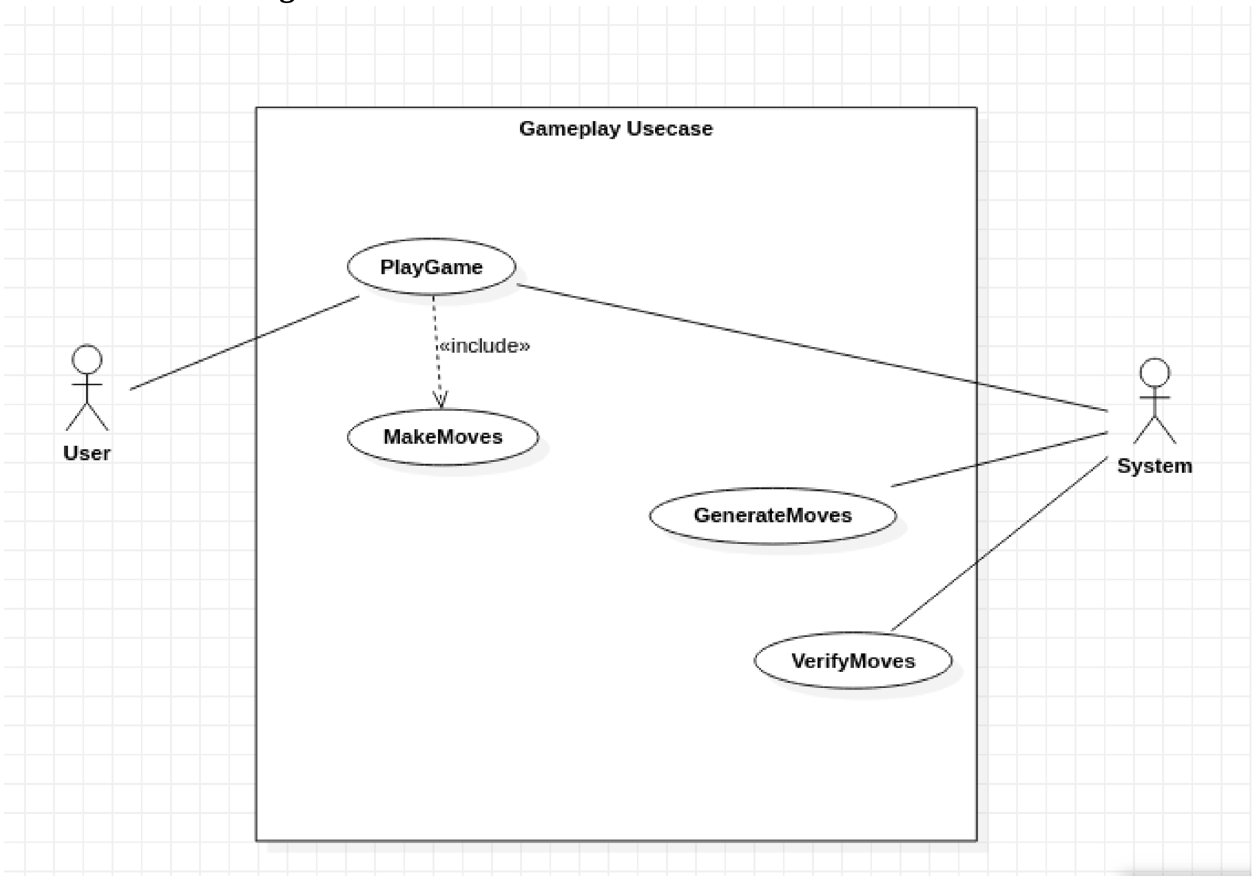
Sub System

CE : Chess Engine

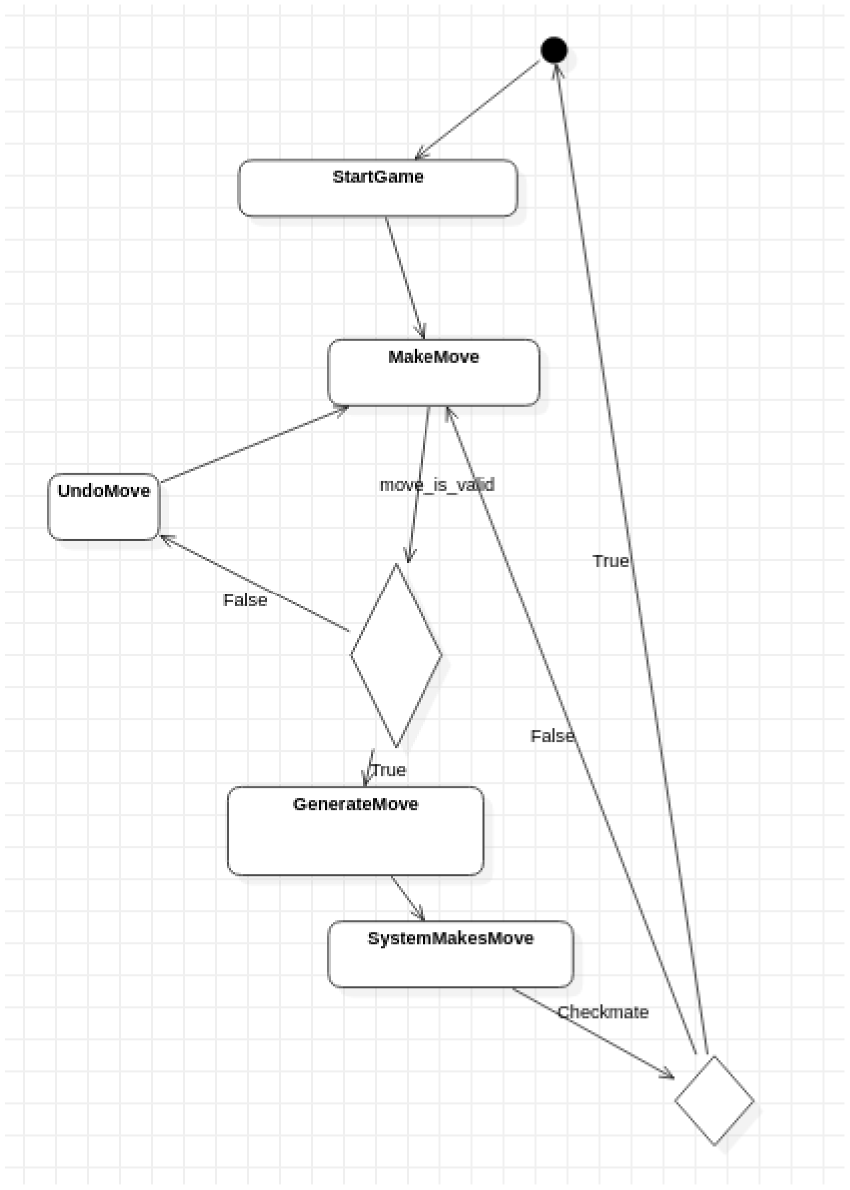
Req. code	Requirements Description	Use Case
CE-F-1.0	The user should be able to play games against the system and the moves should be validated.	Chess Engine
CE-NF-1.1	Castling functionality should be available for both the user and the bot.	
CE-U-1.2	Once the castling has been done, the user and bot should not be able to castle again.	
CE-U-1.3	Whenever there is an invalid move placed by the user, the system should inform the user that they have placed invalid moves.	

CE-F-1.4	The system should have level selection functionality.	
CE-NF-1.5	The difficulty of gameplay should be based on the level selected.	

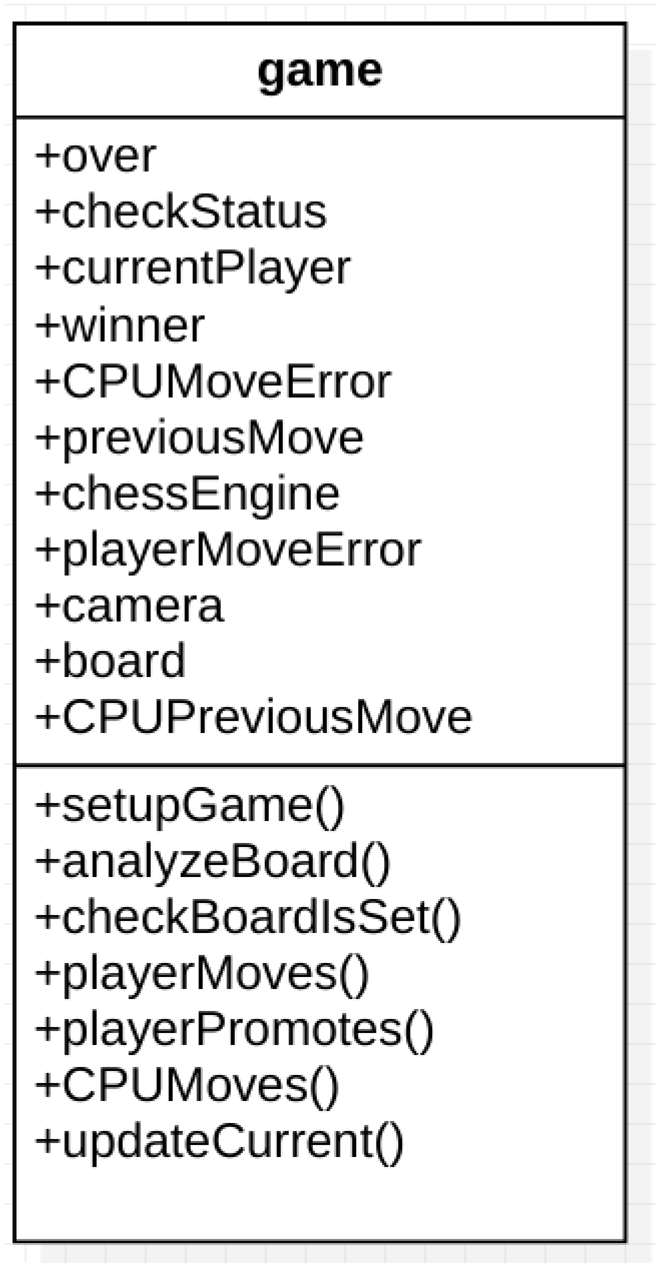
10.1.2 Use Case Diagram



10.1.3 Activity Diagram



10.1.4 Class Diagram



The chess engine part contains different methods for tasks such as initializing game, analyzing game situation, checking if the board is setup correctly or not, entering the player move to the system, entering the player promotion to the system, generating the move for the CPU and updating the current status of the board. These all methods have been derived from the stockfish superclass where all of the definitions have already been provided. Only the functions have been modified for our own purposes.

10.1.5 Introduction to Stockfish

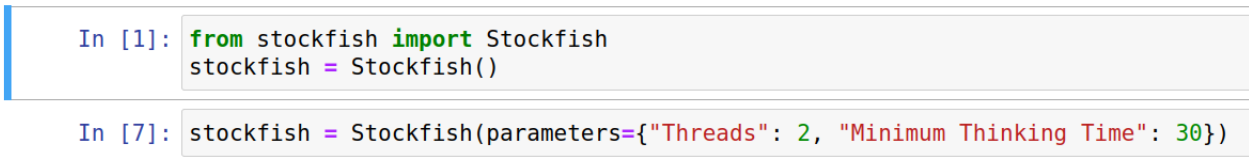
An open-source chess engine “stockfish” is used for this project. It handles all of the tasks such as checking and validating moves, generating new moves for the bot, handling gameplay scenarios, detecting whether the game has ended or not and so on.

Stockfish is an open-source chess engine based on “Glaurung Engine” which was started by Tord Romstad. Later, Marco Costalba developed stockfish 1.0 forking the Glaurung 2.1 engine in 2008. After that the founder of Glaurung along with Joona Kiiski also joined the project and the project slowly rose to popularity and became one of the strongest chess engine of the world (Stockfishchess.org, 2021). The project is available under the GNU General Public License V3.

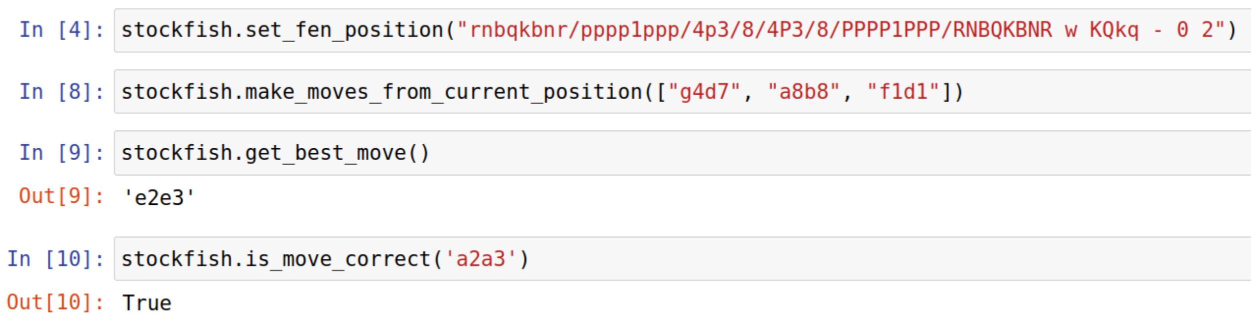
10.1.6 Testing

In order to test the functionalities of this subsystem, a separate notebook was made implementing the stockfish library and all the functionalities were tested before implementing it in our project. Since the stockfish library provides all the functions for the chess gameplay, the required methods for our project are shown in the screenshots below.

First of all, the stockfish library is imported and the stockfish class is instantiated along with setting the parameters for running the program. In the line 4 of our code below, the chess board is initialized using the FEN (Forsyth-Edwards Notation). The line after that represents making of multiple moves from the current board position. And the best move for the CPU is given by the get_best_move function.

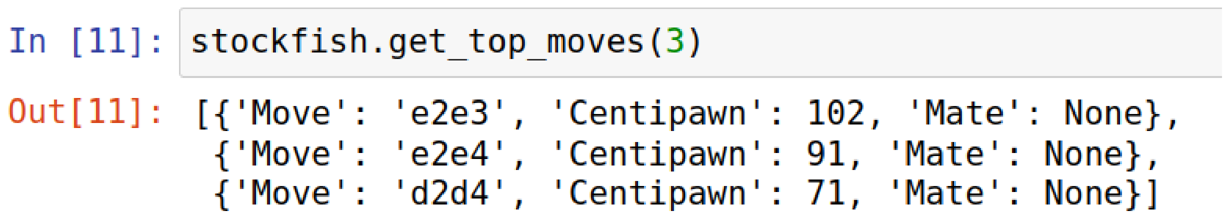


```
In [1]: from stockfish import Stockfish
        stockfish = Stockfish()

In [7]: stockfish = Stockfish(parameters={"Threads": 2, "Minimum Thinking Time": 30})
```

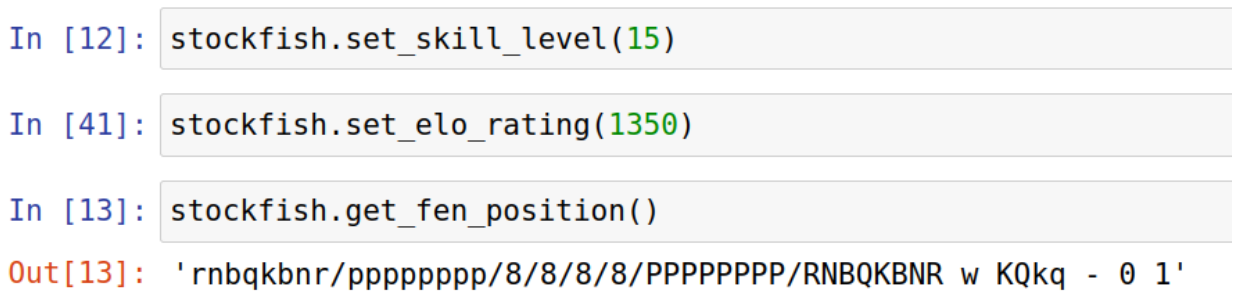


```
In [4]: stockfish.set_fen_position("rnbqkbnr/pppp1ppp/4p3/8/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2")

In [8]: stockfish.make_moves_from_current_position(["g4d7", "a8b8", "f1d1"])

In [9]: stockfish.get_best_move()
Out[9]: 'e2e3'

In [10]: stockfish.is_move_correct('a2a3')
Out[10]: True
```

Similarly, checking if the move is valid or not can be done through is_move_correct function.

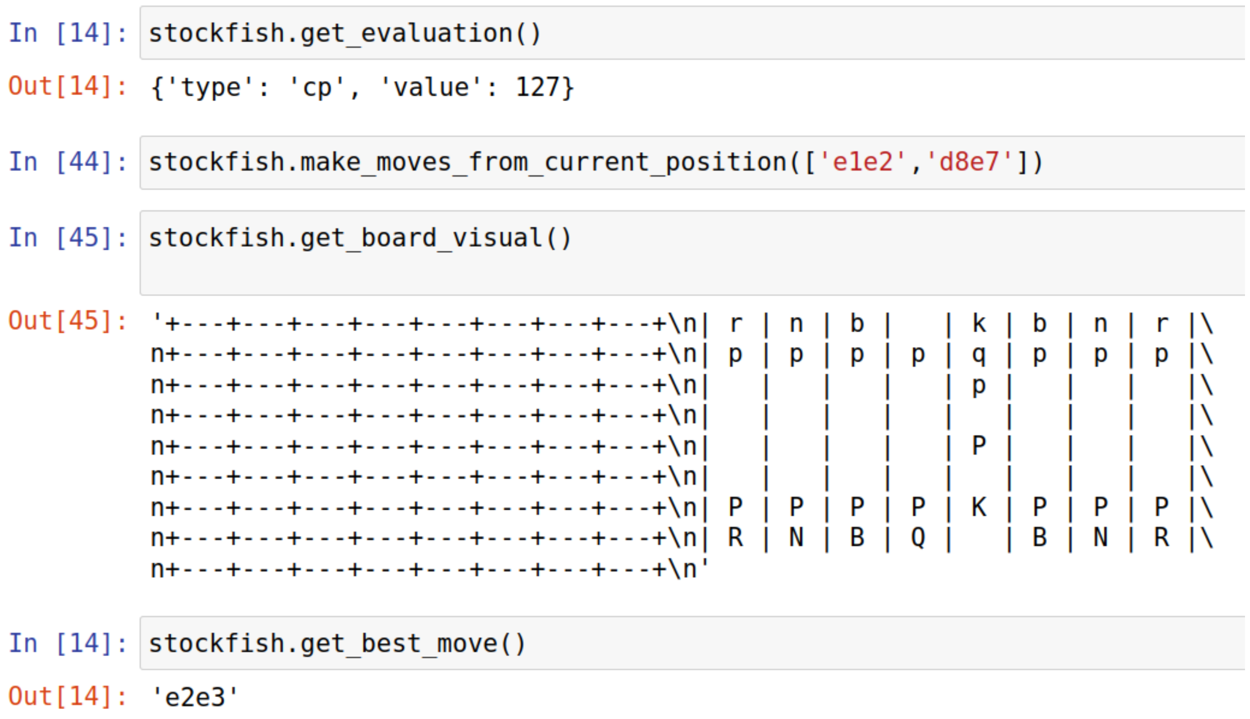


```
In [11]: stockfish.get_top_moves(3)
Out[11]: [{'Move': 'e2e3', 'Centipawn': 102, 'Mate': None},
          {'Move': 'e2e4', 'Centipawn': 91, 'Mate': None},
          {'Move': 'd2d4', 'Centipawn': 71, 'Mate': None}]
```



```
In [12]: stockfish.set_skill_level(15)

In [41]: stockfish.set_elo_rating(1350)

In [13]: stockfish.get_fen_position()
Out[13]: 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1'
```

In the block above, we have done multiple of things such as getting top 3 moves from the system, setting skill level, setting rating for the user and getting the current FEN position. All of these functions are required for a game to be better and customizable.



```
In [14]: stockfish.get_evaluation()

Out[14]: {'type': 'cp', 'value': 127}

In [44]: stockfish.make_moves_from_current_position(['e1e2','d8e7'])

In [45]: stockfish.get_board_visual()

Out[45]: '+---+---+---+---+---+---+---+---+\n| r | n | b |   | k | b | n | r |\
         n+---+---+---+---+---+---+---+---+\n| p | p | p | p | q | p | p | p |\
         n+---+---+---+---+---+---+---+---+\n|   |   |   |   | p |   |   |   |\
         n+---+---+---+---+---+---+---+---+\n|   |   |   |   |   |   |   |   |\
         n+---+---+---+---+---+---+---+---+\n|   |   |   |   | P |   |   |   |\
         n+---+---+---+---+---+---+---+---+\n|   |   |   |   |   |   |   |   |\
         n+---+---+---+---+---+---+---+---+\n| P | P | P | P | K | P | P | P |\
         n+---+---+---+---+---+---+---+---+\n| R | N | B | Q |   | B | N | R |\
         n+---+---+---+---+---+---+---+---+\n'

In [14]: stockfish.get_best_move()

Out[14]: 'e2e3'
```

Now in the block above, the functions to get the evaluation of the board, getting visualization of the board on the CLI and to get the best move have been implemented. So from these series of images representing the codes, it can be deduced that the error-prevention and customizability is already provided by the chess engine and we are only required to invoke those methods in our code. After integration, only integration testing will be done for this subsystem.

10.2 Piece Tracking System

10.2.1 SRS

PTS - F - 1.0

Types of Requirements:

F : Functional Requirements

NF : Non-Functional Requirements

UR : Usability Requirements

Sub System

PTS: Pieces Tracking System

Req. code	Requirements Description	Use Case
PTS-F-1.0	The system should be able to detect squares and their colors.	Detect Squares
PTS-NF-1.1	The system should keep track of the squares, their color and their co-ordinate within the image.	
PTS-F-1.2	The system should be able to identify chess pieces.	Detect Pieces
PTS-F-1.3	The system should be able to detect changes occurred in the chess board and identify what changes have occurred since the last time.	Track Pieces
PTS-NF-1.4	Depending on the difference in the images, the system should be able to track the movements of the pieces on the chess board.	Track Pieces

PTS-U-1.5	The system should also track the pieces taken by the user which is usually placed outside of the chess board.	

10.2.2 Class Diagram

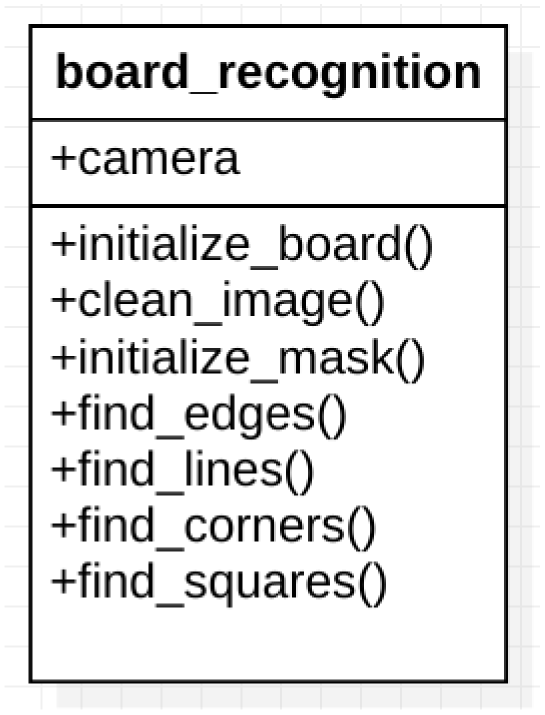


Figure 12: Class diagram

10.2.3 Algorithm Explanation

Image processing is the integral part of the system. Many techniques such as edge-detection, line-detection, color-conversion and color tracking are used in order to be able to determine the borders, squares inside of the chess board and track changes based on color changes. OpenCV library will be implemented in python for its simplicity in performing all of the above tasks. The vision aspect of the project will be carried out in the following steps:

I. **Board Calibration**
II. **Pieces Tracking**

10.2.4 Board Calibration

In this step, the essential parameters such as the position and orientation of the board, edges of the board and inner squares are found with the help of edge-detection, line-detection and contour-detection techniques in a serial order. First of all, an image is taken by the raspberry pi camera to start the calibration process. Due to the position and orientation of the camera on the stand, the image is slightly tilted and the image is needed to be flattened for further operations so the four end co-ordinates of the chess board on the image is found by manual process and is plugged into the perspective transformation function to get a flat image of the chess board.

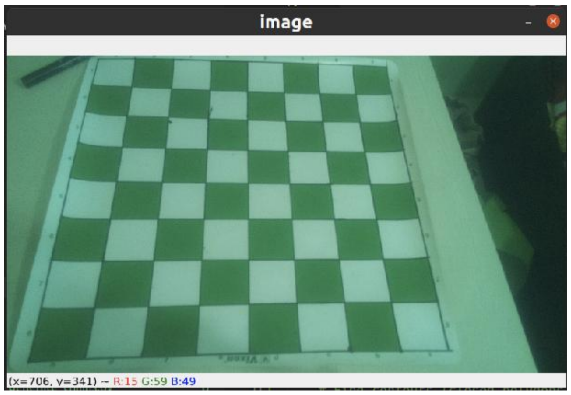


Figure 13: Tilted Image

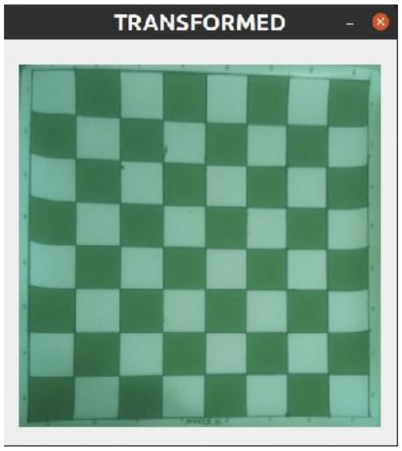


Figure 14: Flattened Image

Then the flattened image is converted to grayscale for easiness in performing operations. Since the chess board being used in colored green and white, we will not get better results in future tasks if we used that same image so the image is converted to black/white using adaptive thresholding technique which calculates the threshold for the image on a regional basis. For an example, as it can be seen in the image below, the Sudoku image consists of varying levels of brightness and contrast. So using one global threshold value to convert the image seems like a bad choice since the image where there is a shadow will automatically be mapped to black and vice-versa. The use of adaptive thresholding helps mitigate this issue by varying the threshold on a regional basis. For an instance, the threshold for the shadowed part will be different from the bright part and there are various methods to find out the threshold value for the region such as using average of the neighbourhood area minus the constant ‘c’ or using the weighted Gaussian sum minus the constant ‘c’. The adaptive thresholding technique provides an intelligent solution to dynamically threshold the image with varying lighting conditions (Opencv.org, 2022).

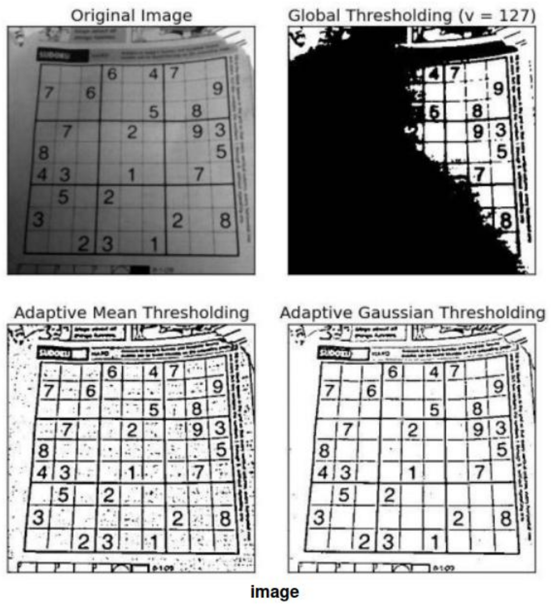

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding
image

Figure 15: Various thresholding techniques (opencv.org, 2022)

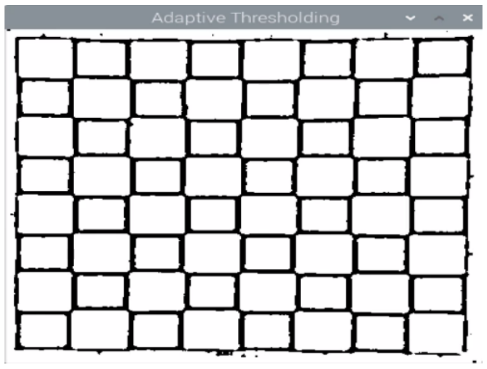


Figure 16: Adaptive thresholding in our chess board

Then the grayscale image is used to find the external border of the chess board using contour-detection function provided by OpenCV and the remaining pixels except the inside of that border is blackened. In simple words, contours are the boundaries of similar-appearing intensity in the images. For an example, a boundary of white square in our context can be taken as one of the contours. In order to find out the external border, the contour with the biggest area is considered as the chess board and the external parts are blackened out in the next step.

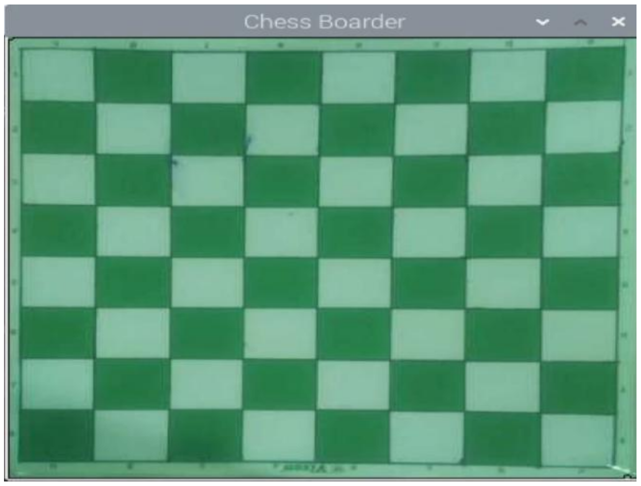


Figure 17: Chess boarder drawn on the image

As it can be seen on the image above, there is a small black border drawn on the image of the chess board by the contour-detection. This image is used to mask out all the other unwanted artifacts from the image and replace them with black color. The masked image of the chess board is shown below and in that image, a small black line can be seen on the top left corner of the image that the function found out to be remaining portion of the image except the chess board.

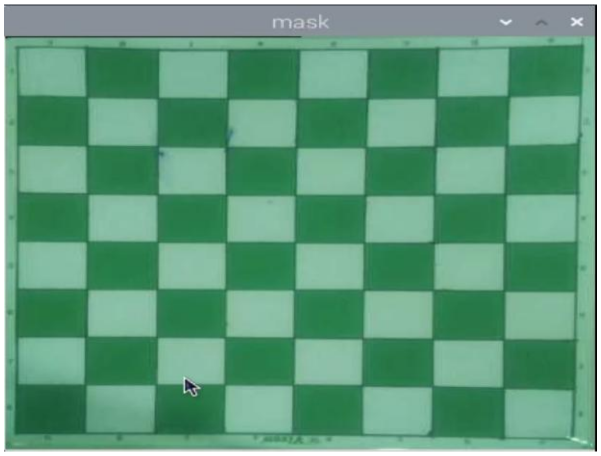


Figure 18: Masked chess board

After that, the image containing only the chessboard is used to find out the edges/borders of individual chess squares and the edges are used to draw lines on the chess board programmatically to find the intersection of the lines inside the chess board. The intersection points are be used to determining the boundaries for each squares in every operations happening after this. Then the calibration process is complete. The edges are found out by the canny edge detection function of OpenCV where the input image is the previously generated masked image and the output is the edges of the squares.

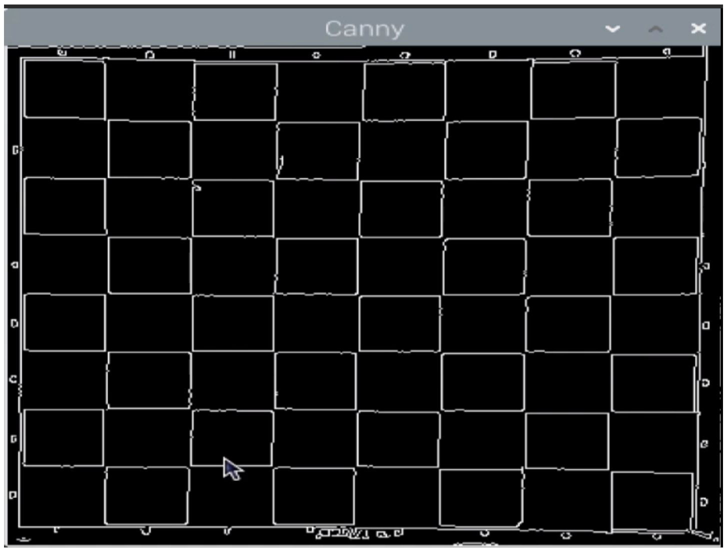


Figure 19: Edge detection on the thresholded image

After the edges have been detected somewhat successfully, line detection algorithm is applied on the edged image acquired above and the lines are drawn on the image. The Hough Lines method is used to find lines in the image.

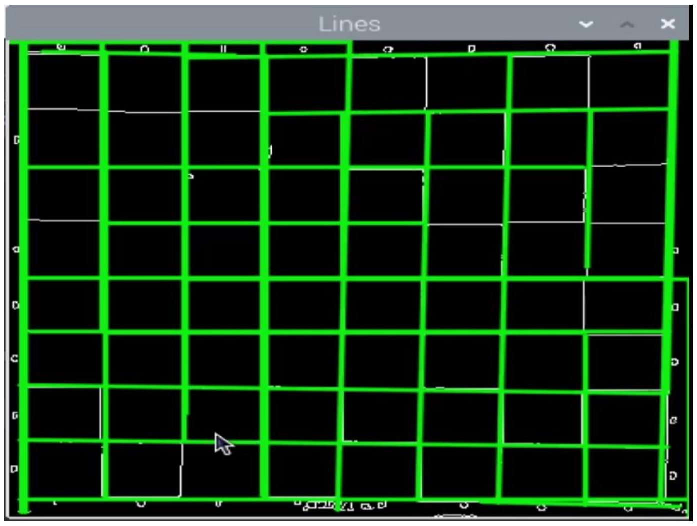


Figure 20: Line-detection on the edged image

As it can be seen in the image above, the algorithm fails to detect some lines as they are not properly straight, it will not be a problem for us as the non-detected lines are not in some way touched by another detected line and it can be used to determine the points of intersection. It is done by iteratively going through all the horizontal and vertical lines drawn on the image and finding out the intersection and placing them inside one global corners variable. Then it is once again checked if there are any duplicate entries by calculating the distance of every corners individually.

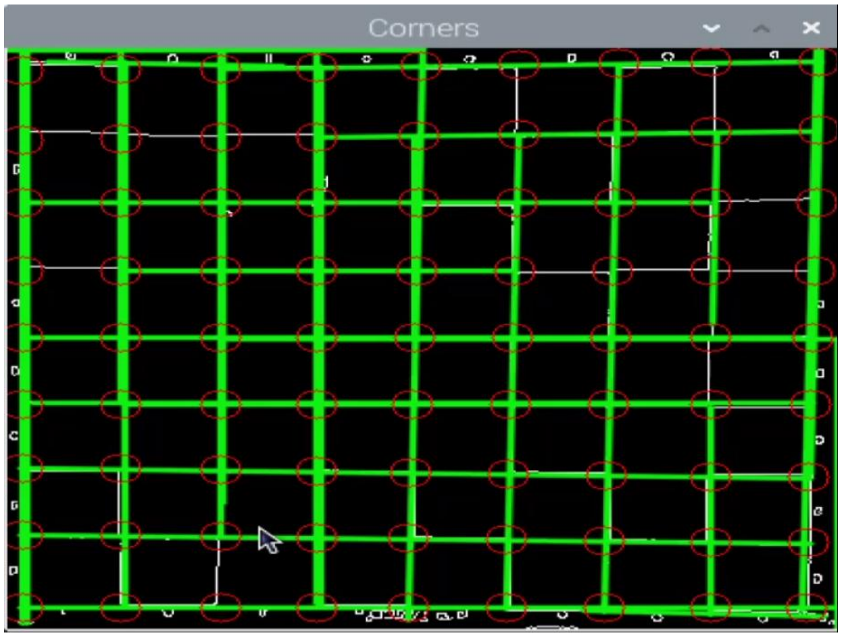


Figure 21: Detection of squares and their corners

Now the corners can be used to determine the position of the squares in the image, draw them and label them.

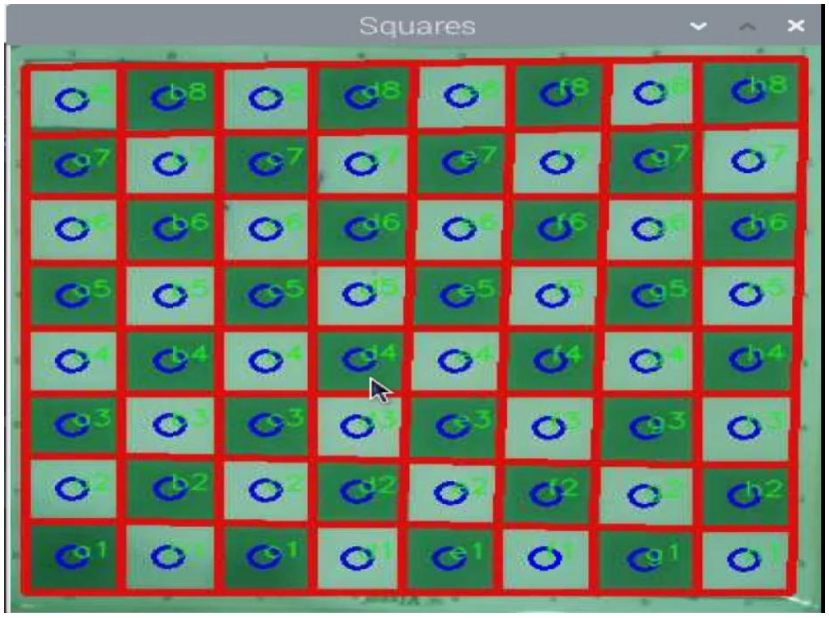
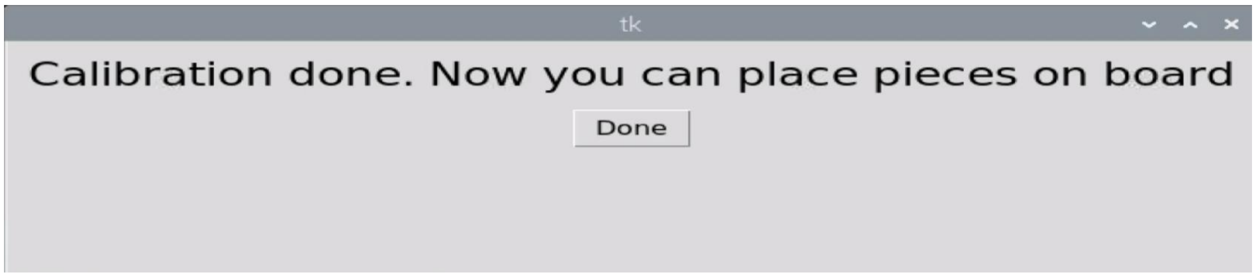


Figure 22: Putting labels to the squares



Calibration done. Now you can place pieces on board

Figure 23: Message showing the completion of calibration

After the calibration has been complete, the user is prompted with the message indicating the success of the calibration phase and the user is told to place the chess pieces on the board.

10.2.5 Pieces Tracking

The major concept behind the pieces tracking used in this project is the fact that if a piece is moved from one square to the other, there is definitely a change in color values happening in those squares where the pieces have been taken from and moved to. For example, if a pawn have been taken from a square and placed on another square, there is a significant change in those particular squares that can be used to identify the moves based on the highest change in colors.

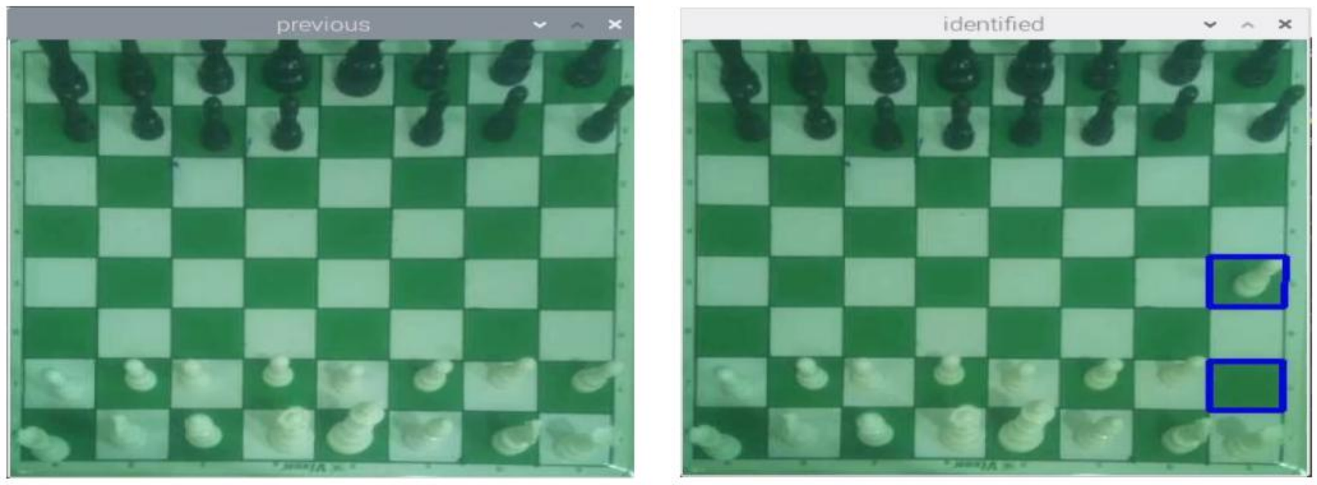


Figure 24: Chess movement tracking by comparing color values from previous and current image

As shown in the image above, the left-most image consists of all the chess pieces in their own initial position, the picture is already recorded in the system for reference and when a move has been made by the user and confirmed to the system, the camera takes a picture and compares the color values of each individual squares. The highest changes in the squares become the candidate for the most probable movement and the system is able to detect movements. The system is capable of detecting most of the moves given a better lighting condition and a medium-quality camera. Similarly, computation wise, it is not that computationally expensive and can run in medium-grade hardware such as raspberrypi as it does not make use of any deep learning and neural networks for tracking changes.

10.3 Robotic System

10.3.1 SRS

RS - F - 1.0

Types of Requirements:

F : Functional Requirements

NF : Non-Functional Requirements

UR : Usability Requirements

Sub System

PTS: Pieces Tracking System

Req. code	Requirements Description	Use Case
RS-F-1.0	The arm should be able to move to provided destination	Move
RS-F-1.1	The arm should be able to pick up pieces and drop them	pick
RS-F-1.2	When provided with piece capturing scenario, it must be able to capture the pieces	capture
RS-F-1.3	The system should be able to handle castling	castle
RS-F-1.4	The system should allow promotion of the pieces	Promotion

10.3.2 Robotic System in Detail

The subsystem consists of a robotic arm, consisting of 4 Degrees of Freedom (DOF) made up of 5 servos one on base which rotates around the z-axis, 2 servos on the shoulder position which rotates about the y-axis, the third one is on the elbow position which also rotates about the y- axis and finally the wrist joint consists of a servo that controls the orientation of the end-effector which is an electromagnet. The whole pick-and-place mechanism is controlled by an Arduino UNO paired with a PCA9685 servo driver module. Similarly, the electromagnet is switched on and off with the help of relay which is fed 12 volts that go through a boost converter to boost the 12V to 24V required by the electromagnet. The servos used are Hitec HS 755 mg, HS 645 mg and HS 540 mg servos which all contain metal gears for added strength. The angles required by the servos is pre-recorded in the Arduino code so that very little calibration is required during the time of testing and operation.

At first, the plan was to make use of inverse kinematics to efficiently and accurately determine the servo angles for the individual positions bud due to time constrains and errors while modelling the robot in Unified Robot Description Format (URDF), the results obtained by this method were very inconsistent and unreliable so the decision to utilize a hardcoded method had to be done.

10.3.3 Algorithm Definition

This subsystem works under the assumption that the board is already calibrated with respect to the arm. To do this, the measurements of the distance between the chess board and the arm base is clearly recorded for the future reference. Then the next step is to record the respective servo angles of all 4 servos for each of the 64 squares of chess board manually and store it programmatically. Then the chess move for the CPU acquired from the chess engine is used to navigate to the squares with the help of the pre-recorded servo angles. Once all the servos have reached the desired angle denoting the successful transition of the arm to a particular square, the electromagnet present at the end position of the arm is toggled on or off programatically based on whether it is a source square or a destination.

11 Conclusion

Critically evaluating the project to answer the academic questions asked in the earlier part of the report, the tracking of the chess pieces and gameplay was done through the OpenCV python library that provides functions such as identifying the borders, lines and shapes and comparing images and finding changes between them. Secondly, the predictions of movements for the bot along with various tasks such as move validation, keeping track of the game and all the functionalities required for a smooth gameplay was handled through the stockfish chess engine and python chess library. The python chess library was used to keep track of the game and the stockfish was used to get better predictions for the bot. Thirdly, the aim of using inverse kinematics to achieve smooth and accurate movement of the robotic arm was not successful but the angles for servos were pre-programmed to give a static method of operation for the arm. In general, the pieces tracking system was developed from scratch and it performed smoothly given the perfect lighting conditions. Similarly, the chess engine part was flawless as it was pre-built library and the robotic arm performed according to the moves provided. Overall aspect of the projects along with the aims and objectives set earlier were mostly achieved.

Finally, the project concluded with a functional system but the functionalities were not refined from any perspective. For an instance, the manual input of the servo angles was tedious as well as error-prone. This could have been easily mitigated by the use of algorithmic approach such as inverse kinematics but was not possible since it required the arm to be modelled in 3D modelling tools to get the perfect measurements of the arm joints and servos. It was not feasible due to strict deadline. Similarly, the chess engines could also be modified to fit our requirements and to make it more secure for the end user. Hardware board uses could be minimized to using raspberry pi only as it is more than capable of handling all the tasks on its own.

12 Critical Evaluation

Through this project, it can be deduced that the construction of arm from scratch is not feasible and it is not smooth in terms of movement. The project directly depended on the 5 servos present on the arm and the quality of servos used determined the outcome of the project. The longer the servos were used, the more the inaccuracies began so in future projects, it can be mitigated by the use of high-efficiency servos or stepper motors. Similarly, the Servo Easing library used for smooth movement of servos provided an efficient way of controlling multiple servos smoothly and with desired speed but it was not fully dependable as while attaching the servo, it did not take care of the speed aspect and in most of the cases, the arm moved from its initial position when the Arduino is turned on. Another hectic task of the project was repeated calibration of the board that was needed to be done in every game.

13 Future Works

For those who are thinking of replicating the project and making it better,

14 Evidence of Project Management

15 References

Chess.com, 2021. *What Russia Taught the World About Chess.* [Online] Available at: https://www.chess.com/article/view/russia-world-chess

Matuszek, C. et al., 2011. Gambit: An Autonomous Chess-Playing Robotic System. *2011 IEEE International Conference on Robotics and Automation*, pp. 1-7.

Meyer, J., 2017. *Raspberry Turk.* [Online] Available at: http://www.raspberryturk.com/ [Accessed 10 12 2021].

Opencv.org, 2022. Opencv Documentation. 20 03, pp. 1-20.

opencv.org, 2022. *Opencv.org.* [Online] Available at: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html [Accessed 20 03 2022].

Patil, A., Kulkarni, M. & Aswale, A., 2017. Analysis of the inverse kinematics for 5 DOF robot arm using D-H parameters. *IEEE*, 1(1), pp. 1-6.

Statista.com, 2021. *Global Chess Market.* [Online] Available at: https://www.statista.com/statistics/809953/global-chess-market-size/

Stockfishchess.org, 2021. *About - Stockfish - Open Source Chess Engine.* [Online] Available at: https://stockfishchess.org/about/ [Accessed 01 03 2022].

URTING, D. & BERBERS, Y., 2003. MarineBlue: A Low-Cost Chess Robot. pp. 1-7.

Vijiyakumar, K., Govindasamy, V. & Akila, V., 2021. 4. Multi-Object Recognition and Tracking with Automated Image Annotation for Big Data Based Video Surveillance. *IEEE*, pp. 1-5.

16 Appendices